

USER GUIDE

Essential Studio for WPF

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

SfDomainUpDown.....	51
Getting Started with WPF Domain Updown (SfDomainUpDown).....	51
Assembly deployment.....	51
Creating Application with SfDomainUpDown control	51
Creating project	51
Adding control via designer	51
Adding control manually in XAML.....	51
Adding control manually in C#	52
Populating by DataBinding.....	53
Spin button alignment	54
Theme	55
Populating Data in WPF Domain Updown (SfDomainUpDown).....	55
ItemsSource	55
ContentTemplate	56
AutoReverse in WPF Domain Updown (SfDomainUpDown)	56
Spin Button Alignment in WPF Domain Updown (SfDomainUpDown)	57
Right	57
Left	57
Both.....	58
Gestures in WPF Domain Updown (SfDomainUpDown)	58
Mouse Wheel	58
Appearance and Styling in WPF Domain Updown (SfDomainUpDown).....	58
Spin animation	58
Accent brush	58
DoubleTextBox.....	59
WPF Double TextBox Overview	59
Control structure.....	59
Features	59
Getting Started with WPF Double TextBox.....	59
Assembly deployment.....	59
Adding WPF DoubleTextBox via designer	59
Adding WPF DoubleTextBox via XAML	60
Adding WPF DoubleTextBox via C\#	60
Setting Value	61
Value Changed Notification	62

Min Max Value Restriction.....	63
Step Interval to increase or decrease the value	63
Formatting the value.....	64
Setting the Culture	64
Theme	65
Changing Double Value in WPF Double TextBox	65
Change double value by pasting the clipboard's text	66
Show UpDown Button.....	67
Value Changed Event	67
Setting the Null value.....	68
Setting Watermark Text.....	69
Step Interval in WPF Double TextBox	70
Change Value on Up, Down arrow key	70
Change Value on Mouse Wheel.....	70
Change Value on Click and Drag	71
Allow or restrict selection on focus	71
Restriction or Validation in WPF Double TextBox.....	72
Restrict the value within minimum and maximum value	72
Restrict number of decimal digit.....	73
Read only mode	74
Culture and Formatting in WPF Double TextBox	75
Culture based formatting.....	75
NumberFormatInfo based formatting	75
Formatting with dedicated properties.....	76
Appearance in WPF Double TextBox	77
Setting the Foreground	77
Setting the Background.....	78
Setting the Corner Radius	79
Apply Background for Selection.....	79
Align Value	80
Setting ToolTip	80
Theme	80
Range Adorner in WPF Double TextBox	81
Changing background of range-adorner	81
DropDownButtonAdv.....	81

WPF Dropdown Button (DropDownButtonAdv) Overview	81
Key features	81
Getting Started with WPF Dropdown Button (DropDownButtonAdv)	82
Control structure.....	82
Assembly deployment.....	82
Creating simple application with Dropdown Button	82
Setting label	84
Setting size mode	85
Setting icon template.....	86
Setting icon template selector	88
Setting image	90
Setting icon width and height	91
Adding items to Dropdown Button.....	92
Theme	94
Data Binding in WPF Dropdown Button (DropDownButtonAdv)	94
Creating model.....	94
Creating view model	95
Bind data from view model.....	95
Bind command from view model.....	96
Command Binding in WPF Dropdown Button (DropDownButtonAdv)	100
Dropdown Menu Items in WPF Dropdown Button (DropDownButtonAdv)	103
Setting icon for dropdown menu items.....	103
Setting icon bar visibility	104
Setting scrollbar visibility	105
Resizing dropdown menu	107
Checkable dropdown menu items	108
Adding custom dropdown menu items	109
Setting icon bar visibility for custom dropdown menu items.....	110
Dropdown Direction in WPF Dropdown Button (DropDownButtonAdv)	112
Multiline Text in WPF Dropdown Button (DropDownButtonAdv)	114
Events in WPF Dropdown Button (DropDownButtonAdv)	115
DropDownOpening	115
DropDownOpened	115
DropDownClosing	115
DropDownClosed	116

Events for dropdown menu items	116
Styles and Templates in WPF Dropdown Button (DropDownButtonAdv).....	117
Edit appearance in Expression Blend	117
Edit appearance in Visual Studio.....	119
Themes in WPF Dropdown Button (DropDownButtonAdv)	121
Gantt	122
WPF Gantt control Overview	122
Real World Scenario.....	122
Getting Started with WPF Gantt	122
Appearance and structure of Gantt	122
Class diagram	126
Feature summary	126
Adding GanttControl to an application.....	127
Adjusting chart and grid size	131
Schedule padding.....	131
ScheduleType	132
Showing date with time in GanttGrid	132
see also.....	133
Data Binding in WPF Gantt	133
TaskDetails Binding	133
External Property Binding	135
Dependency Relationship in WPF Gantt.....	140
Dynamic Predecessors and Resources.....	143
Predecessor Validation	143
Auto Update Hierarchy in WPF Gantt.....	145
Use Case Scenario	145
Properties.....	145
Using the Auto Update Hierarchy Support in an Application	145
Default Scenario.....	145
Using Custom Logics in Business Objects.....	149
Sample Link	154
Baseline Support in WPF Gantt.....	155
Baseline Table View	155
Use Case Scenario	155
Adding Baseline Table View to an Application	155

Baseline Chart View	156
Baseline Customization	157
Samples	157
On-Demand Baseline Column Inclusion.....	157
Samples Link.....	159
Project Statistics.....	159
CustomToolTip in WPF Gantt.....	161
Properties.....	161
Adding CustomToolTip to Gantt	162
Samples Link.....	163
Custom Schedule in WPF Gantt	164
Custom Numeric	164
Custom DateTime	164
GanttScheduleCell Class.....	165
Adding Custom Schedule to an Application.....	165
Samples Link.....	167
Adding CustomDateTime Schedule to an Application	167
Samples Link.....	169
ScheduleCellCreatedEventArgs Class.....	170
Calendar Customization in WPF Gantt.....	170
Use Case Scenarios.....	170
Properties.....	170
Adding Calendar Customization to an Application	171
Holidays Customization in WPF Gantt	173
Zooming in WPF Gantt	175
Built-in Zooming.....	175
Custom Zooming	177
Tables for Zooming Properties and Events	180
Highlighting Tasks in WPF Gantt	181
Use Case Scenario	181
Properties.....	181
Adding Highlighting Tasks to an Application.....	181
Samples Link.....	183
see also.....	183
Custom Node Style in WPF Gantt	183

Use Case Scenarios.....	183
Adding Custom Node Style to an Application.....	183
Samples Link.....	188
Visual Style in WPF Gantt.....	189
Properties.....	189
AddingVisualStyle to Gantt Control	189
Samples Link.....	191
DateTime Indicator Customization in WPF Gantt.....	191
Use Case Scenarios.....	192
Adding the DateTime Indicator to an Application	192
Sample Link	193
Resource View Gantt Inline Items in WPF Gantt	194
Use Case Scenarios.....	194
Adding Inline Items to an Application.....	194
Data Structure.....	197
Information Displayed in Gantt.....	197
Sample Link	197
Strip Lines in WPF Gantt	198
Strip lines in Essential Gantt support the following features:	198
Types of strip lines	198
Properties.....	198
Adding strip lines to application	200
see also.....	203
Import and Export Support in WPF Gantt.....	203
Properties.....	203
Methods.....	203
Import/Export Task Details from/to XML	203
Flow Direction in WPF Gantt.....	205
Localization in WPF Gantt	206
How to.....	208
GridControl	209
WPF GridControl Overview.....	209
Key Features.....	209
Feature Summary.....	210
Getting Started with WPF GridControl	213

Adding the Grid Control to a WPF Application	213
Populating the Grid control with Data	217
Appearance in WPF GridControl	218
Cell Styles	218
Cell Types in WPF GridControl	228
Setting Cell Type	229
Header cell type	229
Static cell type	229
CheckBox cell type	230
Button cell type	230
Image cell type	231
ComboBox cell type	231
DropDownList cell type	235
DateTimeEdit cell type	238
IntegerEdit cell type	239
DoubleEdit cell type	241
CurrencyEdit cell type	243
PercentEdit cell type	245
MaskEdit cell type	247
UpDownEdit cell type	248
RichText cell type	250
DataTemplate cell type	251
Nested grid cell type	252
Creating custom cell type	262
Inserting Images into Grid Cells	271
Formula Cells in WPF GridControl	272
Defining a FormulaCell	272
Using the Formula Library	273
Adding Formulas to the Formula Library	274
Function Reference Section	276
Editing in WPF GridControl	320
Clipboard Support	320
Undo/Redo	325
See also	330
Managing Rows and Columns in WPF GridControl	330

Rows and columns count	330
Row heights and column widths	331
Hiding rows and columns	331
Freeze rows and columns	332
Header rows and columns	333
Resize rows and columns	334
Inserting rows and columns	334
Moving rows and columns	335
Removing rows and columns	336
Cell Layout Customization in WPF GridControl	337
Covered Cells	337
Banner Cells	338
Overlapping Cells	339
Graphic Cell	341
Covered Ranges in WPF GridControl	351
Creating covered cells	351
Find whether a cell in covered range	352
Remove covered range at run time	352
Extend covered range at run time	354
Selection in WPF GridControl	355
Selection modes	355
Range selection	356
Record selection	357
Selecting rows and columns programmatically	359
Selection appearance customization	359
Excel-like selection frame	361
CurrentCell	361
Excel-like CurrentCell	362
Highlighting row and column header based on selection	362
See also	363
Autofit Cells in WPF GridControl	364
Autofit row height	364
Autofit column width	364
Autofit Options	365
Autofit Cells based on Wrap Text	365

How to avoid rendering issues due to fractions when auto fit the cells	365
Change the size of row height and column width to fit all cells in View	366
See also	366
Virtualization in WPF GridControl.....	366
Virtual Mode	366
Virtual Cells	368
Performance in WPF GridControl	370
Sample Architecture	370
Sample Features.....	371
Interactive Features in WPF GridControl.....	371
Drag-and-Drop columns.....	371
Excel like - Cell drag and drop.....	372
Excel like - Resizing.....	374
Events in WPF GridControl.....	376
QueryCellInfo and CommitCellInfo	377
QueryBaseStyles	378
QueryCoveredRange	379
QueryCellSpanBackgrounds.....	380
ResizingRows and ResizingColumns.....	381
GridResizingRowsEventArgs.....	383
GridResizingColumnsEventArgs	383
RowsInserted and ColumnsInserted	384
RowsMoved and ColumnsMoved	385
RowsRemoved and ColumnsRemoved	385
ClipboardCanCopy.....	386
ClipboardCanCut	387
ClipboardCanPaste.....	388
ClipboardCopy.....	388
ClipboardCut	389
ClipboardPaste	390
CellButtonClick.....	390
CellClick.....	391
Cell Mouse Events.....	392
CurrentCellActivating.....	394
CurrentCellActivated.....	395

CurrentCellDeactivating	395
CurrentCellDeactivated	395
CurrentCellStartEditing	396
CurrentCellEditingComplete	396
CurrentCellValidated	397
CurrentCellChanging	397
CurrentCellChanged	398
ToolTip in WPF GridControl	398
ToolTip for specific cell	398
ToolTip for row and column	399
Set ToolTip in QueryCellInfo event	400
Show or hide the ToolTip	401
Setting ToolTip delay	401
Handling ToolTip opening event	401
Hide ToolTip for disabled cell	402
Identify whether cell has ToolTip	402
Customize the ToolTip	403
Remove the ToolTip	405
Comment Tip in WPF GridControl	405
Comment Tip for specific cell	405
Comment Tip for row and column	406
Change comment indicator position	407
Set CommentTip using QueryCellInfo	408
Handling CommentTip opening event	408
Customize the CommentTip	409
Input Message Tip in WPF GridControl	410
Input Message Tip for particular cell	410
Input Message Tip for row and column	410
Set Input Message Tip using QueryCellInfo event	411
Show or hide Input Message Tip	412
Identify whether cell has Input Message Tip	412
Customize the Input Message Tip	413
Remove the Input Message Tip	414
Export to Excel in WPF GridControl	414
Exporting to Excel	414

Exporting to CSV.....	419
Import from Excel in WPF GridControl	422
Use Case Scenarios.....	422
Tables for Properties, Methods, and Events.....	422
Adding Excel Importing to an Application.....	425
How To	427
Printing in WPF GridControl.....	428
Print Dialog Options	429
Printing Header and Footer	433
Serialization in WPF GridControl.....	435
Adding Serialization to an Application.....	435
Methods.....	436
Zooming in WPF GridControl	437
Use Case Scenarios.....	437
Change Zoom Scale of the Grid Control.....	437
Real Time Applications in WPF GridControl.....	439
Applications with High Frequency Updates	439
Excel-like UI Applications	439
Testing in WPF GridControl.....	441
Grid Control UI Automation Support	442
VS 2010 Coded UI Testing	446
SfGridSplitter.....	452
WPF GridSplitter (SfGridSplitter) Overview	452
Features of SfGridSplitter.....	453
Getting Started with WPF GridSplitter (SfGridSplitter)	453
Structure of SfGridSplitter	454
Visual representation.....	454
Assembly deployment.....	454
Adding control manually in XAML.....	455
Add control manually in C\#	455
Resize the grid rows	456
Resize the grid columns	457
Resizing the grid rows and columns with specific pixel	458
Show or hide the grid row and columns	461
Custom UI for Collapse buttons	462

Custom UI for expander gripper	464
Deferred resizing.....	466
Grid splitter for the merged columns or rows	467
Theme	469
Appearance in WPF GridSplitter (SfGridSplitter)	470
Setting the Background.....	470
Theme	472
GroupBar	473
WPF Navigation Pane (GroupBar) Overview	473
Control Structure	474
Features	474
Navigation Pane Features	475
Getting Started with WPF Navigation Pane (GroupBar)	475
Assembly deployment.....	475
Create a simple application with GroupBar	475
Create a project	475
Add control through designer.....	475
Add control manually in XAML	476
Add control manually in C\#	476
Add items using GroupBarItem.....	477
Bind data	478
Add content to GroupBar Item	480
Expand one or more items.....	481
Theme	482
Adding GroupBar Item to GroupBar in WPF Navigation Pane (GroupBar).....	483
Adding Content to GroupBar Item in WPF Navigation Pane (GroupBar)	484
Adding GroupView to GroupBar Item.....	484
Adding GroupView Item to GroupView	485
Adding Panel to GroupBar Item	486
Events to handle with GroupBar Item	486
Rotating the GroupBar in WPF Navigation Pane (GroupBar)	487
Rotating the Content in WPF Navigation Pane (GroupBar)	488
Collapsing the GroupBar in WPF Navigation Pane (GroupBar).....	488
Events to handle with AllowCollapse.....	489
Adjusting the GroupBar width in WPF Navigation Pane (GroupBar).....	490

Setting the FlowDirection for GroupBar in WPF Navigation Pane.....	491
Closing the GroupBar in Stack Mode in WPF Navigation Pane (GroupBar).....	492
Maximum number of Visible Items in OverFlowPanel	492
Changing the orientations of GroupBar in WPF Navigation Pane	494
Events to Handle Orientation of Groupbar	497
OrientationChanged Event.....	497
OrientationChanging Event.....	497
Changing Header Height in WPF Navigation Pane (GroupBar).....	497
Height of the Container that hosts the Selected Header.....	498
Setting Animation for GroupBar expansion collapse in WPF	499
Animation Speed.....	499
Animation Type.....	500
Setting ToolTip for GroupBar in WPF Navigation Pane (GroupBar)	501
Using Context menu in GroupBar in WPF Navigation Pane (GroupBar).....	502
Toolbars in GroupBar in WPF Navigation Pane (GroupBar)	503
Navigation Pane in GroupBar in WPF Navigation Pane (GroupBar)	504
Navigation Pane Gripper	505
Adding Groupview in ListView Mode in WPF Navigation Pane (GroupBar)	505
Default Image for the GroupView Item in WPF Navigation Pane (GroupBar).....	506
State Persistence in WPF Navigation Pane (GroupBar)	507
GroupBar Items in WPF Navigation Pane (GroupBar)	508
Status of the GroupBar Item.....	508
Cursor Type for the GroupBar Item	509
Content Length of the GroupBar Item.....	510
Hidden GroupBar Item.....	511
Setting Corner Radius for the GroupBar Item.....	511
Appearance in WPF Navigation Pane (GroupBar)	512
Visual Mode	512
Theme	516
Item Container Style	517
Item Container Style Selector	518
Drag Marker Color.....	521
Custom Template for the Collapse Button	522
Collapse Button Color	522
Customizing Group Bar Header	523

Populating-Data in WPF Navigation Pane (GroupBar).....	524
Through XAML.....	524
Through C#.....	524
Data-Binding in WPF Navigation Pane (GroupBar).....	525
Data-Binding to Objects.....	527
Data-Binding with XML.....	529
Customizing-Data-Templates in WPF Navigation Pane (GroupBar).....	530
Item Template.....	531
Item Template Selector.....	531
Header Template.....	533
Content Template.....	534
Header Template Selector.....	535
Content Template Selector.....	536
Drag-and-Drop in WPF Navigation Pane (GroupBar).....	538
Details.....	538
Localization in WPF Navigation Pane (GroupBar).....	541
Grouping.....	543
WPF Grouping Overview.....	543
Introduction to Essential Grouping.....	543
Prerequisites and Compatibility.....	544
Getting Started with WPF Grouping.....	545
Creating a WPF Application.....	545
Deploying Essential Grouping.....	546
Data Binding in WPF Grouping.....	547
Creating an ArrayList of Objects.....	547
Setting a Datasource In the Grouping Engine.....	551
Iterating Through the Data.....	552
Using Grouping in WPF Grouping.....	552
Grouping a Table.....	553
Accessing a Particular Group.....	556
Adding a Summary.....	558
Retrieving Summary Values for a Particular Group.....	559
Data Manipulation in WPF Grouping.....	560
Filters.....	560
Expressions.....	564

Sorting	566
Custom Sorting	568
Algebra Supported In Expressions Filters in WPF Grouping	572
Custom Functions	573
HierarchyNavigator	573
WPF Breadcrumb (HierarchyNavigator) Overview	573
Features	573
Getting Started with WPF Breadcrumb (HierarchyNavigator)	574
Assembly deployment	574
Create a simple application with HierarchyNavigator	574
Create a project	574
Add control through designer	574
Add control manually in XAML	575
Add control manually in C\#	575
Add items using HierarchyNavigatorItem	575
Bind data	576
Theme	578
Populating Data in WPF Breadcrumb (HierarchyNavigator)	578
HierarchyNavigator Initialization	578
Data binding	580
See Also	586
Mouse Support in WPF Breadcrumb (HierarchyNavigator)	586
Keyboard Support in WPF Breadcrumb (HierarchyNavigator)	587
Edit Mode in WPF Breadcrumb (HierarchyNavigator)	587
Refresh Button in WPF Breadcrumb (HierarchyNavigator)	588
Template Customizing in WPF Breadcrumb (HierarchyNavigator)	589
HierarchicalDataTemplate	589
Progress Bar in WPF Breadcrumb (HierarchyNavigator)	590
Command Binding in WPF Breadcrumb (HierarchyNavigator)	591
Skins in WPF Breadcrumb (HierarchyNavigator)	593
Theme	593
Tile Control	594
WPF Tile Control Overview	594
Key Features	594
Getting Started with WPF Tile Control	594

Assembly deployment.....	594
Creating simple application with Hub Tile and Pulsing Tile	594
Setting title, header and image in tile.....	597
Theme	598
Hub Tile in WPF Tile Control	599
Setting header content	599
Setting title content	600
Setting image	601
Setting secondary content	602
Animation.....	604
Transitions.....	605
Grouping	607
Freezing/Unfreezing.....	612
Notifications.....	618
Appearance and styling.....	620
Pulsing Tile in WPF Tile Control	624
Setting header content	624
Setting title content	626
Setting image	627
Animations	628
Translations.....	630
Grouping	632
Freezing/Unfreezing.....	633
Notifications.....	639
Appearance and styling.....	641
Themes in WPF Tile Control.....	643
Theme	643
IntegerTextBox.....	643
WPF IntegerTextBox Overview	643
Control structure.....	643
Features	644
Getting started with WPF IntegerTextBox.....	644
Assembly deployment.....	644
Adding WPF IntegerTextBox via designer	644
Adding WPF IntegerTextBox via XAML	644

Adding WPF IntegerTextBox via C\#	645
Setting Value	646
Value Changed Notification	647
Min Max Value Restriction.....	648
Step Interval to increase or decrease the value	648
Formatting the value.....	648
Setting the Culture	649
Theme	649
Changing Integer Value in WPF IntegerTextBox.....	650
Change integer value by pasting the clipboard's text.....	651
Show UpDown Button.....	652
Value Changed Event	652
Setting the Null value.....	652
Setting Watermark Text.....	653
Step Interval in WPF IntegerTextBox	654
Change Value on Up, Down arrow key	654
Change Value on Mouse Wheel.....	655
Change Value on Click and Drag	655
Allow or restrict selection on focus	656
Restriction or Validation in WPF IntegerTextBox	656
Restrict the value within minimum and maximum value	656
Read only mode	657
Customize the behavior for invalid value	658
Culture and Formatting in WPF IntegerTextBox.....	659
Culture based formatting.....	659
NumberFormatInfo based formatting	660
Formatting with dedicated properties.....	661
Appearance in WPF IntegerTextBox	661
Setting the Foreground	661
Setting the Background.....	663
Setting the Corner Radius	663
Apply Background for Selection.....	663
Align Value	664
Setting ToolTip	664
Theme	664

Range Adorner in WPF IntegerTextBox	665
Changing background of range-adorner	665
SfImageEditor.....	666
WPF ImageEditor (SfImageEditor) Overview	666
SfImageEditor.....	666
Key features	666
Getting Started with SfImageEditor	667
Adding ImageEditor reference	667
Initialize ImageEditor	667
Loading image in ImageEditor.....	667
Theme	669
See also	669
Text in SfImageEditor	669
Toolbar	669
Customization	669
Adding text programmatically	670
Text settings	670
See also	671
Shapes in SfImageEditor	671
Adding shapes using toolbar	671
Free hand sketch (Path)	672
Customization	672
Adding shapes programmatically.....	673
Pen settings.....	673
Shape resizing	675
Events.....	675
See also	676
Transformation in WPF ImageEditor (SfImageEditor) control.....	676
Flip.....	676
Rotate.....	678
Crop support in SfImageEditor.....	679
Toolbar cropping.....	679
Programmatic cropping	682
See also	685
Save and Reset functionality in Image Editor	685

Saving an image using the toolbar	686
Saving an image programmatically	686
Reset	686
Events.....	687
See also	688
Undo and Redo Supports	688
Undo.....	688
Redo	689
Zooming and Panning in WPF ImageEditor (SfImageEditor) control	689
Zooming	689
Toolbar	689
Mouse wheel.....	689
Panning	690
With toolbar	690
Without toolbar (Programmatically).....	690
Toolbar Customization	691
Customization	691
Events.....	693
Image picker support	694
Commands	695
Z ordering in WPF ImageEditor (SfImageEditor)	696
BringToFront	696
BringForward	696
SendToBack.....	696
SendBackward.....	697
Localization in WPF ImageEditor (SfImageEditor)	697
CustomView in SfImageEditor	700
Add a custom view on image editor	700
Customize the custom view	701
SfKanban	702
WPF Kanban (SfKanban) Overview	702
Key features	703
Getting Started with WPF Kanban(SfKanban)	703
Adding assembly reference.....	703
Create a simple Kanban	704

Theme	709
Column in WPF Kanban (SfKanban) control.....	710
Customizing column size	710
Categorizing columns.....	710
Headers	711
Column Tags.....	712
Expand/Collapse Column	712
Enable/Disable Drag & Drop	713
Cards in WPF Kanban (SfKanban) control	713
Customizing kanban cards	715
Template	715
Events in WPF Kanban (SfKanban) control	717
CardTapped	717
CardDoubleTapped	717
CardDragStart.....	717
CardDragEnd	718
CardDragEnter.....	718
CardDragLeave	718
CardDragOver.....	719
ColumnsGenerated	719
ColumnGenerated.....	719
Placeholder in WPF Kanban (SfKanban) control.....	719
Workflow configuration.....	721
Work In-Progress limit	722
Swim lanes in WPF Kanban (SfKanban) control	723
Customization	724
SfLinearGauge	725
WPF LinearGauge (SfLinearGauge) Overview.....	725
Key features	726
Getting Started with SfLinearGauge	726
Adding gauge references	726
Initialize gauge	726
Configuring scale.....	727
Adding a symbol pointer	728
Adding a bar pointer	728

Adding ranges	729
Theme	731
See also	732
Scale in SfLinearGauge	732
Setting minimum and maximum values for a scale	732
Setting interval for a scale	733
Scale customization	734
Setting scale direction.....	735
Setting position for a scale.....	736
Ticks support in WPF Linear Gauge (SfLinearGauge) with customization	737
Tick customization	737
Setting position for tick.....	737
Setting minor ticks per interval.....	738
Labels support in WPF Linear Gauge (SfLinearGauge).....	739
Label color customization	739
Label font customization.....	739
Setting position for labels	740
Setting postfix and prefix for labels	741
Labels visibility	742
Ranges support in WPF Linear Gauge (SfLinearGauge)	743
Setting start and end values for range.....	743
Range customization.....	744
Binding range stroke to ticks and labels	745
Setting range position	746
Pointers support in WPF Linear Gauge (SfLinearGauge)	749
Bar pointer	749
Symbol pointer in linear scale.....	751
Positioning symbol pointer	753
Setting animation for pointer	755
Orientation customization in WPF Linear Gauge (SfLinearGauge)	756
SfLinearProgressBar	757
WPF Linear ProgressBar Overview.....	757
Key features	758
Creating a simple application with Linear ProgressBar	758
Assembly deployment.....	759

Adding control through designer	759
Adding control manually in XAML.....	759
Adding control through code behind.....	760
Theme	760
Different states in Linear ProgressBar	760
Determinate	760
Indeterminate	761
Buffer	761
Segments in Linear ProgressBar.....	761
Segment	762
Segment with corner radius.....	762
Appearance in Linear ProgressBar	762
Corner radius.....	762
Padding	763
Color Customization.....	763
Range Colors	764
Gradient	765
AnimationDuration	765
AnimationEasing	766
SfMap	767
WPF Maps (SfMap) Overview	767
Use case scenario	767
Key Concepts of Map	767
Structure of Map.....	767
Getting Started with WPF Maps (SfMap).....	768
Adding SfMap reference	768
Create SfMap control.....	768
Initializing maps	770
Adding layers.....	771
Configure the SfMap Control	771
Attach the Shape file with Map	772
Data Binding in Map.....	773
Adding markers	774
Color mapping.....	775
Adding legends.....	776

Theme	779
See also	779
Populate Data in WPF Maps (SfMap).....	779
Data binding.....	779
Custom Data Binding in WPF Maps (SfMap).....	783
Rule for Specifying the Latitude	783
Rule for Specifying the Longitude	783
See Also	787
Layers in WPF Maps (SfMap)	787
Imagery layer	787
Shape file layer.....	787
Appearance customization	789
ZOrder index for layers	791
Events.....	794
see also.....	795
Shape Types in WPF Maps (SfMaps).....	795
Add shapes using map element collection	795
Add shapes using point collection	803
Bubbles in WPF Maps (SfMap).....	809
Adding Bubbles to a Map.....	810
Customizing Bubble Symbol.....	812
Range Color Mapping.....	815
Shapes Color Customization in WPF Maps (SfMap)	817
Properties.....	817
Tree map-like support.....	818
Range Color Mapping.....	819
ColorPalette	821
CustomColorPalette	822
Annotations in WPF Maps (SfMap).....	824
Positioning a MapAnnotation	826
Map Selection in WPF Maps (SfMap)	827
Single Selection	828
Multi Selection	829
MapPopup in WPF Maps (SfMap).....	830
Map Shape Labels in WPF Maps (SfMap)	833

Multilayer Support in WPF Maps (SfMap)	835
Loading Multiple Shapefiles in a Single Container	835
Adding Multiple Layers in the Map	835
SubShapeFileLayers	835
Map Points in WPF Maps (SfMap)	840
Customizing the MapPoint	840
About MapPointTemplate Property	840
MapPointIcon	840
MapPointPopup	842
Legend in WPF Maps (SfMap)	845
Legend visibility	845
Legend Position	845
Legend header	845
Legend categories	846
KML Format in WPF Maps (SfMap)	849
KML Shapes Support in ShapeFileLayer	849
KML Shapes Support in SubShapeFileLayer	850
Map Providers in WPF Maps (SfMap)	851
Open Street Map	851
Bing Map	855
See also	859
User interaction in WPF Maps (SfMap)	859
Tooltip	859
Tooltip for shapes	859
Tooltip for bubbles	863
Tooltip for markers	866
Tooltip for markers in imagery layer	867
Zooming and panning	869
Zooming	869
Zooming in ShapeFileLayer	870
Panning the map	871
Zooming in ImageryLayer	872
Markers in WPF Maps (SfMap)	878
Adding the marker	878
Add a custom marker	879

Customizing marker icons	881
Customizing labels	883
Marker Alignment	884
Selection Mode	887
Events.....	888
See also	889
Commands in WPF Maps (SfMap)	889
ZoomIn Command.....	890
ZoomOut Command.....	890
Pan Command.....	890
ZoomReset Command.....	890
PanReset Command.....	890
Events in WPF Maps (SfMap)	891
How to.....	892
SfMaskedEdit	893
WPF MaskedTextBox (SfMaskedEdit) Overview.....	893
Key features	894
Getting Started with WPF MaskedTextBox (SfMaskedEdit)	894
Control Structure	894
Assembly deployment.....	894
Adding WPF SfMaskedEdit via designer	895
Adding WPF SfMaskedEdit via XAML.....	895
Adding WPF SfMaskedEdit via C\#.....	896
Restrict the user to enter valid data	896
Setting the value	897
Get the value in various formats.....	897
Indicates error on invalid input.....	898
Indicates the missed input.....	898
Value changed notification	899
Theme	899
Input Restriction in WPF MaskedTextBox (SfMaskedEdit)	900
Restrict the user to enter valid data	900
Setting the value	903
Get the value in various formats.....	903
Input validation.....	906

Get the validation result	907
Indicates error on invalid input.....	907
Indicates the missed input	908
Setting the watermark	909
Value changed notification	910
Appearance in WPF MaskedTextBox (SfMaskedEdit).....	910
Setting the background.....	910
Setting the foreground.....	911
Setting the border color.....	911
Change flow direction	911
Theme	912
MenuAdv.....	912
WPF Menu (MenuAdv) Overview	912
Getting Started with WPF Menu (MenuAdv).....	912
Structure of the MenuAdv Control	913
Assembly deployment.....	913
Creating Application with MenuAdv control	913
Creating project	913
Adding control via designer	914
Adding control manually in XAML.....	918
Adding control manually in C#	918
Set icon for Menu item	919
Theme	921
Data Binding in WPF Menu (MenuAdv).....	922
Data-Binding to Objects.....	922
Data-Binding with XML	923
Customizing Data Templates in WPF Menu (MenuAdv)	925
Item Template.....	925
Binding and DataTemplate Support in WPF Menu (MenuAdv).....	926
Adding the Binding and DataTemplate Support to an Application.....	926
Command Binding and Command Target Support in WPF Menu (MenuAdv)	928
Using the Command Binding Support in an Application.....	929
Orientation Support in WPF Menu (MenuAdv)	931
Adding the Orientation Support to an Application.....	932
Expand Modes Support.....	933

Icon Support in WPF Menu (MenuAdv).....	934
Use Case Scenarios.....	934
Adding the Icon Support to an Application.....	934
Check Box and Radio Button Support in WPF Menu (MenuAdv)	935
Using the Check Box and Radio Button Support in an Application.....	935
InputGestureText Support in WPF Menu (MenuAdv)	936
Adding the InputGestureText Support to an Application	936
MenuItemSeparator Support in WPF Menu (MenuAdv).....	937
Adding the MenuItemSeparator Support to an Application.....	937
Boundary Detection in WPF Menu (MenuAdv)	938
Scroll Support in WPF Menu (MenuAdv).....	940
Using the Scroll Support in an Application.....	940
Animation Support in WPF Menu (MenuAdv)	941
Adding the Animation Support to an Application.....	941
Keyboard Navigation Support in WPF Menu (MenuAdv)	942
Blendability Support in WPF Menu (MenuAdv).....	942
Styling in WPF Menu (MenuAdv).....	944
Theme	944
SfMultiColumnDropDown.....	945
WPF Multi Column Dropdown (SfMultiColumnDropDown) Overview.....	945
Getting Started with WPF Multi Column Dropdown (SfMultiColumnDropDown)	945
Assembly deployment.....	946
Creating simple application with SfMultiColumnDropDownControl.....	946
Creating Data Model for sample application	948
Binding to Data	950
Defining DisplayMember and ValueMember	951
Defining Columns	951
Editing and filtering.....	952
Theme	952
Data Binding in WPF Multi Column Dropdown (SfMultiColumnDropDown)	952
Binding with complex and indexer properties.....	954
Columns in WPF Multi Column Dropdown (SfMultiColumnDropDown)	956
Automatically generating columns	956
Manually defining columns.....	956
Column sizing	960

Editing and AutoComplete in WPF Multi Column Dropdown.....	961
Auto Completion of Text.....	961
Auto Increment.....	962
Null Value Support.....	962
Open popup while editing.....	962
Open popup while loading.....	962
Keyboard Interactions.....	962
Getting editor text	963
Selecting text when editor got focus	964
Selection in WPF Multi Column Dropdown (SfMultiColumnDropDown)	964
Multi-Selection.....	965
Events.....	969
Filtering in WPF Multi Column Dropdown (SfMultiColumnDropDown).....	969
Case-Sensitive Filtering	970
Ignore Diacritic Sensitivity.....	970
How to filter SfMultiColumnDropDownControl based on various column values.....	971
Filtering Delay	973
Limitations.....	973
Styles and Templates in WPF Multi Column Dropdown	973
Edit Appearance in Expression Blend.....	973
Edit Appearance in VisualStudio	975
Popup customization in WPF Multi Column Dropdown	976
Popup sizing	977
Resizing popup	978
Keep DropDownPopup as StaysOpen	978
Events.....	978
UIAutomation.....	979
SfNavigationDrawer	981
WPF Navigation Drawer (SfNavigationDrawer) Overview	981
Getting Started with WPF Navigation Drawer (SfNavigationDrawer)	983
Assembly Deployment	983
Creating simple application with SfNavigationDrawer	983
Adding content to the control	985
Adding sidebar menu items	985
Different display modes in WPF Navigation Drawer (SfNavigationDrawer).....	989

Compact display mode.....	990
Expanded display mode	991
Auto display mode change.....	993
Collapsible drawer mode	995
Populating data in WPF Navigation Drawer (SfNavigationDrawer).....	997
Populating using built-in items	997
Data Binding.....	1004
Hierarchical Data Binding.....	1007
IndentationWidth.....	1010
Popup support	1013
Handling Selection in WPF Navigation Drawer (SfNavigationDrawer)	1013
Header and Footer in WPF Navigation Drawer (SfNavigationDrawer).....	1014
Customizing the Toggle button and Header	1014
Customizing the Footer.....	1016
Creating a custom header and footer.....	1018
Keyboard Support in WPF NavigationDrawer (SfNavigationDrawer).....	1018
Commands and events in SfNavigationDrawer	1019
Opening event.....	1019
Opened event	1020
Closing event.....	1022
Closed event.....	1023
ItemClicked event	1024
ItemCollapsed event	1026
ItemExpanded event.....	1028
Commands	1030
Custom views in WPF Navigation Drawer (SfNavigationDrawer).....	1034
Populating the with custom views.....	1034
Customizing the drawer.....	1037
Customize Panel size.....	1037
Navigation Pane Sides.....	1039
Swipe gesture and sensitivity.....	1050
Customize animations.....	1052
NotifyIcon.....	1062
WPF Notify Icon Overview	1062
Features	1062

Getting Started with WPF Notify Icon.....	1062
Assembly deployment.....	1062
Creating an application with NotifyIcon control.....	1062
Creating a project.....	1062
Adding control via designer	1062
Adding the control manually in XAML	1063
Adding control manually in C#	1063
Show the notify icon	1064
Text	1064
Tooltip	1065
Theme	1065
Interactive Features in WPF Notify Icon	1065
BalloonTip	1065
Animation.....	1067
Setting the position of the Notify Icon.....	1068
Setting the Shape of the Notify Icon.....	1068
Layout Related Features in WPF Notify Icon	1069
Theme	1069
Customizing the Header of the NotifyIcon	1069
Events.....	1070
OlapChart	1073
WPF Olap Chart Overview.....	1073
Key features	1073
Getting Started with WPF Olap Chart	1073
Through Visual Studio	1073
Through expression blend	1077
Through code-behind.....	1081
Design-time binding.....	1084
Data Binding in WPF Olap Chart	1089
Binding OLAP chart to offline cube	1089
Binding OLAP chart to cube in local SQL Server.....	1090
Binding OLAP chart to cube in online SQL Server	1090
Binding OLAP chart to cube in online Mondrian Server	1090
Binding OLAP chart to cube in online ActivePivot Server	1090
XAML Configuration in WPF Olap Chart	1091

KPI in WPF Olap Chart.....	1092
Drill Operation in WPF Olap Chart	1094
Drill position	1095
Show/hide expanders	1095
Chart Types in WPF Olap Chart.....	1096
Column chart.....	1097
Stacking column chart.....	1098
Stacking column 100 chart.....	1098
Bar chart.....	1099
Stacking bar chart	1100
Stacking bar 100 chart	1101
Area chart.....	1102
Stacking area chart.....	1103
Spline area chart	1104
Step area chart.....	1105
Line chart	1106
Spline chart	1107
Rotated spline chart.....	1108
Step line chart	1109
Scatter chart.....	1110
Pie chart	1111
Area in WPF Olap Chart	1112
Adding chart header	1113
Area customization	1113
Legend in WPF Olap Chart	1115
Show/hide legend	1116
Visibility customization	1117
Dock position	1118
Row/column setting.....	1118
Series in WPF Olap Chart	1120
Point label	1120
Color customization	1122
Border customization.....	1122
Custom data templates.....	1123
Event	1124

Chart animation	1126
Pie chart customization.....	1127
Axes in WPF Olap Chart	1128
Grid lines customization	1129
Format settings	1130
Label font settings.....	1132
Primary axis label visibility	1133
Paging in WPF Olap Chart	1134
Zooming and Scrolling in WPF Olap Chart	1138
Palette in WPF Olap Chart	1140
Custom palette.....	1141
Excel-like palette.....	1141
Appearance in WPF Olap Chart.....	1144
Chart style and legends.....	1144
Chart border and background style	1146
Chart points labels	1147
Chart axis labels	1148
Tooltip in WPF Olap Chart.....	1149
Watermark in WPF Olap Chart	1151
Chart Type for Specific Series in WPF Olap Chart	1153
Exporting in WPF Olap Chart	1154
Exporting as an image	1154
Exporting to Word document	1155
Exporting to a PDF document	1156
Printing in WPF Olap Chart	1158
Theming in WPF Olap Chart.....	1158
Localization in WPF Olap Chart.....	1159
RTL support	1161
OlapClient.....	1162
WPF Olap Client Overview	1162
Key features	1162
Getting Started with WPF Olap Client	1163
Through Visual Studio	1163
Through expression blend	1165
Through code-behind.....	1167

Data Binding in WPF Olap Client.....	1169
Binding OLAP client to offline cube	1169
Binding OLAP client to cube in local SQL Server	1170
Binding OLAP client to cube in online SQL Server.....	1170
Binding OLAP client to cube in online Mondrian Server	1171
Binding OLAP client to cube in online ActivePivot Server	1171
OlapClient: Elements in WPF Olap Client.....	1172
Cube Selector	1172
Cube dimension browser	1172
Axis element builder	1173
Elements editor.....	1175
Toolbar	1176
Report manipulation	1176
OLAP grid and OLAP chart.....	1180
Calculated Members in WPF Olap Client.....	1180
Named Set in WPF Olap Client.....	1181
Drill Through in WPF Olap Client	1182
Serialization in WPF Olap Client.....	1185
Paging in WPF Olap Client.....	1187
Filtering in WPF Olap Client	1191
Filtering by member.....	1191
Filtering by value.....	1192
Subset filter	1193
Sorting in WPF Olap Client.....	1194
Exporting in WPF Olap Client	1195
Exporting OLAP chart	1195
Exporting OLAP grid	1198
Theming in WPF Olap Client	1201
Localization in WPF Olap Client	1202
RTL support	1203
OlapGrid	1204
WPF Olap Grid Overview	1204
Key features	1204
Getting Started with WPF Olap Grid.....	1204
Through Visual Studio	1204

Through Expression Blend	1207
Through code-behind.....	1210
Design-time binding	1212
Data Source in WPF Olap Grid	1217
Binding OLAP grid to offline cube	1217
Binding OLAP grid to cube in local SQL Server	1218
Binding OLAP grid to cube in online SQL Server	1218
Binding OLAP grid to cube in online Mondrian Server	1218
Binding OLAP grid to cube in online ActivePivot Server	1218
XAML Configuration in WPF Olap Grid	1219
KPI in WPF Olap Grid.....	1220
Drill Down/Up in WPF Olap Grid.....	1221
Drill Position in WPF Olap Grid	1222
Grid Layout in WPF Olap Grid	1222
Paging in WPF Olap Grid	1225
Conditional Formatting in WPF Olap Grid	1229
Cell Selection in WPF Olap Grid	1231
Freeze Headers in WPF Olap Grid.....	1232
Hyperlink Cells in WPF Olap Grid	1233
Tooltip in WPF Olap Grid.....	1235
Header tooltip.....	1235
Value cell tooltip	1236
Grid Style Dialog in WPF Olap Grid	1236
Style dialog.....	1236
Configuring the properties of cell style.....	1237
Member Properties in WPF Olap Grid	1239
"All" - Level Member.....	1241
Exporting in WPF Olap Grid	1242
Theming in WPF Olap Grid.....	1246
Localization in WPF Olap Grid.....	1247
RTL support	1248
OlapGauge.....	1249
WPF Olap Gauge Overview	1249
Key features	1249
Getting Started with WPF Olap Gauge	1250

Control initialization.....	1250
Data Binding in WPF Olap Gauge	1257
Binding an OLAP gauge to offline cube.....	1257
Binding OLAP gauge to cube in local SQL Server	1257
Binding OLAP gauge to cube in online SQL Server	1258
Binding OLAP gauge to cube in online Mondrian server	1258
Binding OLAP gauge to cube in online ActivePivot Server.....	1259
XAML Configuration in WPF Olap Gauge	1259
Properties.....	1259
KPI in WPF Olap Gauge	1260
Appearance in WPF Olap Gauge	1261
Gauge radius	1261
Built-in frame types.....	1262
Skins	1264
Gauge Customization in WPF Olap Gauge	1269
Layout customization	1269
Gauge header.....	1271
Gauge label	1271
Gauge factor.....	1272
Tooltip in WPF Olap Gauge	1273
Pointer tooltip.....	1273
Marker tooltip.....	1274
Localization in WPF Olap Gauge	1275
Translation	1275
Resource file and file name conventions.....	1275
Culture specification	1276
RTL.....	1277
OLAP Common	1278
Business Intelligence BI in WPF OLAP Common	1278
What is BI?	1278
Why to use BI?	1278
What's new in BI?	1278
Multi-dimensional data.....	1279
Online Analytical Processing (OLAP) in WPF OLAP Common	1279
ADOMD.NET.....	1279

ADOMD.NET assembly and setup files information	1279
Syncfusion OLAP Architecture in WPF OLAP Common	1280
OLAP base	1281
OLAP Silverlight base	1282
OLAP Silverlight base wrapper	1282
OlapDataProvider in WPF OLAP Common	1283
AdomdDataProvider	1284
OlapDataManager in WPF OLAP Common	1285
Establishing connection with the SSAS server	1286
Establishing connection with the SSAS server through data provider	1286
Establishing connection with the offline cube	1286
Establishing connection with XMLA Server	1286
Connecting to Mondrian Server	1287
Connecting to Active Pivot Server	1287
Connecting to SAP Business Warehouse Server	1287
UseWhereClauseForSlicing	1291
Drill through	1291
OlapReport in WPF OLAP Common	1292
Properties	1292
Elements of OLAP report	1293
Measure element	1295
Key Performance Indicator (KPI) element	1296
NamedSet element	1296
Sort element	1297
Calculated member	1297
Subset element	1299
Filtering slicer elements by range	1299
Creating the OLAP report	1301
Binding OlapReport to OlapDataManager in WPF OLAP Common	1319
Virtual KPI Element in WPF OLAP Common	1319
Adding virtual KPI element to the OlapReport	1319
Drill Position in WPF OLAP Common	1322
Drill Replace in WPF OLAP Common	1322
Paging in WPF OLAP Common	1322
QueryBuilderEngine in WPF OLAP Common	1323

MDX query specification	1323
Steps in query generation	1324
MDX Query Parsing in WPF OLAP Common	1325
MDX Query binding with drill-up and drill-down operations	1325
Adding MDX Query binding with drill-up and drill-down operations to an application.....	1326
OLAP Data Processing in WPF OLAP Common.....	1326
Steps in processing OLAP data	1326
In Silverlight.....	1327
PdfViewer.....	1327
WPF Pdf Viewer Overview	1327
Key features	1328
Getting Started with WPF Pdf Viewer.....	1329
Assemblies required.....	1329
Create a simple PDF Viewer application	1329
Display PDF file.....	1331
Theme	1332
PDF Rendering Engines in WPF Pdf Viewer	1333
PDFium	1333
SfPdf	1335
Viewing PDF Files in WPF Pdf Viewer	1335
Open PDF file from the local disk using toolbar	1335
View PDF file using the file path	1336
View PDF file from stream	1337
View PDF file using the ItemSource property.....	1337
View PDF files without using the toolbar.....	1338
Obtain the PDF file information	1339
Viewing Password protected PDF Files in WPF Pdf Viewer	1340
View password protected PDF files in run time.....	1341
Hide the built-in password dialog	1341
Printing PDF Files in WPF Pdf Viewer.....	1342
Silent Printing.....	1342
Customizing print size	1343
Printing PDF document with orientation settings	1344
Set the custom document name to be displayed when printing.....	1345
Hide Print Status dialog when printing the PDF files	1346

Customizing Paper source in Silent Printing	1346
Exporting PDF pages in WPF Pdf Viewer.....	1347
Exporting a single page	1347
Exporting a specific range of pages.....	1348
Exporting with a custom image size.....	1349
Exporting with a custom image resolution	1350
Extract Text from PDF Files in WPF Pdf Viewer	1350
Extract text from a particular page	1351
Extract text from an entire file.....	1351
Extract text with bounds	1352
Form Filling in PDF Files in WPF Pdf Viewer	1353
Supported form fields	1354
Retrieve the details of text box form field	1354
Import and Export Form Data	1354
Working with PDF Layers in WPF Pdf Viewer	1356
Toggling the visibility of a PDF layer	1356
Toggling the visibility of the group of layers.....	1356
Disabling the layers.....	1357
Redaction in WPF Pdf Viewer	1358
Enable redaction mode	1358
Mark region to redact	1358
Customizing redaction appearance	1359
Appearance of the included redaction	1364
Open pop-ups.....	1364
Delete redaction marked region.....	1365
Events.....	1366
Keyboard shortcuts	1366
Organize Pages in WPF Pdf Viewer	1367
Rotating PDF page(s).....	1369
Acquiring page rotation	1370
Rearranging PDF pages	1371
Removing PDF page(s)	1371
Get the selected page indexes.....	1372
Disabling page organizer	1372
Keyboard shortcuts	1373

Handwritten Signature in WPF Pdf Viewer	1373
How to set the opacity of the handwritten signature?	1374
How to set the color of the handwritten signature?	1374
How to flatten handwritten signature on save?	1374
Working with included handwritten signatures	1375
Track the changes in a handwritten signature.....	1377
Add handwritten signature from code	1379
Keyboard shortcuts	1381
Interaction Modes in WPF Pdf Viewer	1381
Selection mode	1381
Panning mode	1381
Marquee zoom mode	1382
Working with PDF Pages in WPF Pdf Viewer	1382
Page Border.....	1382
Saving PDF Files in WPF Pdf Viewer	1384
Events.....	1384
Keyboard shortcuts	1386
Adjust the magnification of PDF documents using the WPF PDF Viewer.....	1386
Hide the zoom tools in the toolbar	1386
Customization using APIs	1386
Zoom modes	1387
Define the minimum and maximum zoom percentage.....	1387
Zoom changes notification.....	1388
Active view port rendering at higher zoom percentages	1388
Searching Text in WPF Pdf Viewer	1388
Select and Copy Text in WPF Pdf Viewer	1390
Detecting the completion of text selection	1390
Copying the selected text	1390
Localization in WPF Pdf Viewer.....	1391
Adding Resource file	1391
Localize Resource File with Different Assembly or Namespace	1392
Navigation	1392
Thumbnail Navigation in WPF Pdf Viewer	1392
Bookmark Navigation in WPF Pdf Viewer	1394
Page Navigation in WPF Pdf Viewer.....	1396

Hyperlink Navigation in WPF Pdf Viewer	1397
Annotations in WPF PDF Viewer	1400
Keyboard shortcuts	1400
Working with Commands in WPF Pdf Viewer	1400
Magnification operations	1400
Page Navigations	1401
Printing PDF	1403
AnnotationCommand	1403
Save Document Command	1405
How to	1405
Load a specific page	1405
Disable toolbar items	1405
Acquire total number of pages in WPF Pdf Viewer	1408
Change the color of the Loading Indicator in WPF Pdf Viewer	1409
Change the text displayed in the loading indicator in WPF Pdf Viewer	1409
Toggle visibility of the tool bar in WPF Pdf Viewer	1409
Toggle visibility of the scroll bar in WPF Pdf Viewer	1410
Acquire current page being displayed in WPF Pdf Viewer	1411
Enable and Disable Notification bar in WPF Pdf Viewer	1411
PercentTextBox	1412
WPF Percent TextBox Overview	1412
Control structure	1412
Features	1412
Getting Started with WPF Percent TextBox	1412
Assembly deployment	1412
Adding WPF PercentTextBox via designer	1412
Adding WPF PercentTextBox via XAML	1413
Adding WPF PercentTextBox via C\#	1413
Setting Value	1414
Value Changed Notification	1415
Min Max Value Restriction	1415
Step Interval to increase or decrease the value	1416
Formatting the value	1416
Setting the Culture	1417
Theme	1417

Changing Percent Value in WPF Percent TextBox	1418
Change percent value by pasting the clipboard's text.....	1419
Show UpDown Button.....	1420
Value Changed Event	1420
Setting the Null value.....	1421
Setting Watermark Text.....	1421
Step Interval in WPF Percent TextBox	1422
Change Value on Up, Down arrow key	1423
Change Value on Mouse Wheel.....	1423
Change Value on Click and Drag	1423
Allow or restrict selection on focus	1424
Restriction or Validation in WPF Percent TextBox.....	1424
Restrict the value within minimum and maximum value	1424
Restrict number of decimal digit.....	1426
Read only mode	1427
Culture and Formatting in WPF Percent TextBox	1427
Culture based formatting.....	1427
NumberFormatInfo based formatting	1428
Formatting with dedicated properties.....	1429
Positive Value Pattern.....	1430
Negative Value Pattern	1431
Appearance in WPF Percent TextBox	1432
Setting the Foreground	1432
Setting the Background.....	1433
Setting the Corner Radius	1434
Apply Background for Selection.....	1434
Align Value	1434
Setting ToolTip	1435
Theme	1435
Range Adorner in WPF Percent TextBox.....	1435
Changing background of range-adorner	1436
PivotGrid	1436
WPF Pivot Grid Overview.....	1436
Getting Started with WPF Pivot Grid	1436
Adding pivot grid through designer	1436

Adding pivot grid through XAML.....	1437
Adding pivot grid through code-behind.....	1437
Adding pivot grid through expression blend	1438
Binding a data source to pivot grid control	1439
Data Binding in WPF Pivot Grid.....	1443
Binding pivot grid to list	1443
Binding pivot grid to data table	1443
PivotItem in WPF Pivot Grid.....	1445
Defining pivot item in XAML	1445
Defining pivot item in code-behind	1445
PivotComputationInfo in WPF Pivot Grid	1446
Defining PivotComputationInfo in XAML.....	1447
Defining PivotComputationInfo in code-behind.....	1447
Summary Types in WPF Pivot Grid.....	1448
DisplayIfDiscreteValuesEqual summary type in pivot grid	1449
Custom Summary in WPF Pivot Grid.....	1451
Defining custom summary in load time	1452
Defining custom summary in runtime	1454
Custom Calculations in WPF Pivot Grid	1454
Providing expression field calculation for summaries	1456
Expression Fields in WPF Pivot Grid.....	1458
Creating the expression fields.....	1459
Normal Mode	1461
Filtering in WPF Pivot Grid	1461
Excel-Like Filtering in WPF Pivot Grid	1463
Sorting.....	1466
Grouping Bar in WPF Pivot Grid	1471
Grouping Bar Context Menu in WPF Pivot Grid.....	1472
Calculated Field in WPF Pivot Grid.....	1473
PivotGrid Field List in WPF Pivot Grid	1474
Restrict Resizing of Row Header in WPF Pivot Grid	1476
Expand/Collapse Headers in WPF Pivot Grid	1477
Display Options in WPF Pivot Grid	1481
Grid Layout in WPF Pivot Grid.....	1483
State Persistence in WPF Pivot Grid	1484

Sub-Totals for Child Elements in WPF Pivot Grid	1486
RowPivotsOnly Mode in WPF Pivot Grid	1488
Multi-Column Sorting in WPF Pivot Grid	1489
Multicolumn sorting in normal mode	1489
Multicolumn sorting in row pivots only mode.....	1491
Show/Hide Sub-Totals in WPF Pivot Grid	1492
Hiding all the subtotals	1492
Hiding only the row subtotals.....	1494
Hiding only the column subtotals	1495
Hiding the subtotals for the specific pivot item.....	1497
Asynchronous Data Loading in WPF Pivot Grid	1498
Virtualized Binding (Performance Improvement) in WPF Pivot Grid	1500
Improved (Deferred) Scrolling in WPF Pivot Grid	1502
Resize to Fit in WPF Pivot Grid	1503
Editing in WPF Pivot Grid	1505
Enable editing for value cells	1505
Enable editing for total cells	1506
Custom editing manager.....	1507
Updating in WPF Pivot Grid	1509
Single Calculation Header in WPF Pivot Grid	1511
GetRawItem in WPF Pivot Grid	1512
Conditional Formatting.....	1513
Formatting using data conditions in WPF Pivot Grid	1513
Formatting using gradient color scales in WPF Pivot Grid	1515
Cell Style and Template in WPF Pivot Grid	1522
Cell styles	1522
Cell templates	1524
Cell Selection in WPF Pivot Grid.....	1526
Cell selection	1526
Cell selection with headers	1527
Freeze Headers in WPF Pivot Grid	1529
Hyperlink Cells in WPF Pivot Grid	1530
Defining the property in pivot grid	1531
Tooltip in WPF Pivot Grid.....	1532
Adding tooltip for entire pivot grid	1532

Adding tooltip to specific areas in pivot grid	1533
Adding custom tooltip for pivot grid.....	1535
Skin Customization in WPF Pivot Grid	1536
Serialization and Deserialization in WPF Pivot Grid.....	1538
Using the serialization/deserialization in pivot grid	1539
Exporting in WPF Pivot Grid.....	1540
Export to Excel	1540
Export to Word.....	1541
Export to PDF	1542
Export to CSV	1542
Threshold limitations	1543
Printing in WPF Pivot Grid.....	1543
Print preview options.....	1545
Localization in WPF Pivot Grid	1545
Localization using resource file	1545
Localization using satellite assembly	1546
PivotSchemaDesigner	1547
Overview in WPF Pivot Grid	1547
Binding PivotSchemaDesigner in WPF Pivot Grid	1548
Setting Caption in PivotTableFieldList in WPF Pivot Grid	1550
Pivot Computation Information Dialog in WPF Pivot Grid.....	1553
Hiding Fields in WPF Pivot Grid.....	1554
Displaying Calculations in WPF Pivot Grid	1556
How to.....	1557
How to load an existing PivotEngine into PivotGrid?	1557
How to improve loading and scrolling performance in PivotGrid?	1558
How to expand and collapse entire group in PivotGrid?	1558
How to get the count of records in WPF PivotGridControl	1559
How to hide the Grand Total present in the PivotGrid?	1559
How to change the GridLine color and thickness?	1560
How to customize the appearance of expanders in PivotGrid?	1560
How to set defer layout update in PivotGrid?	1561
How to resizing the columns and rows in PivotGrid?	1561
PropertyGrid	1567
WPF PropertyGrid Overview.....	1567

Control structure.....	1567
Features	1568
Getting Started with WPF PropertyGrid	1568
Assembly deployment.....	1568
Adding WPF PropertyGrid via designer	1568
Adding WPF PropertyGrid via XAML.....	1569
Adding WPF PropertyGrid via C#	1570
Populating the properties	1571
Custom Editor as value editors	1572
Selected property item changed notification	1574
Disable animation on loading selected object.....	1575
Tooltip support.....	1576
Override the property items	1577
Override editor.....	1578
Property item value changed notification	1579
Theme	1579
Nested Properties in WPF PropertyGrid	1580
Explore the nested properties	1580
Enable or disable nested properties, for a specific property.....	1582
ReadOnly Properties in WPF PropertyGrid	1586
ReadOnly properties using attributes	1586
Change properties as readonly at runtime	1587
Attached Properties in WPF PropertyGrid.....	1589
Show or hide attached properties of SelectedObject.....	1589
Update value on lost focus in WPF PropertyGrid	1590
Binding modes	1590
Collection Editor in WPF PropertyGrid	1592
How to add or remove items in collection using collection editor.....	1592
Edit a selected object, which is of type collection	1593
Edit a selected object, which has a property of type collection.	1594
Collection Editor for nested collection property	1596
Readonly mode for collection type properties	1598
Readonly mode for specific property of collection type.....	1600
Restrict the collection editor from opening.....	1602
Built-in Editor in WPF PropertyGrid	1604

Built-in mask to restrict user input	1606
Mask attribute with custom mask to restrict user input	1607
MaskEditor to restrict user input	1608
Custom Editor in WPF PropertyGrid	1610
Best practice to follow	1610
Creating the Custom Editor.....	1610
Creating Custom Editor for a dynamic property.....	1611
Assigning a Custom Editor using Editor Attribute.....	1612
Assigning a Custom Editor using Collection	1613
Assigning a Custom Editor to the specific property	1614
Assigning a Custom Editor based on the property type	1615
Assigning a Custom Editor by the editor type.....	1616
Use constructor with parameters in custom editor	1618
Create custom editor control for the property item	1621
Attach the custom editor with property item	1622
Dispose the custom editor	1623
Category Editor in WPF PropertyGrid	1623
Adding Category Editor to PropertyGrid.....	1623
DisplayName of Property in WPF PropertyGrid.....	1626
Change property display name using attributes.....	1626
Change property display name at runtime	1627
Change width of property's name column	1629
Property Description in WPF PropertyGrid.....	1630
Property Description using attributes.....	1630
Change Property Description at runtime.....	1632
Setting description panel height.....	1633
Custom UI for description panel	1635
Different custom UI for specific property's description panel.....	1636
Custom Property Definition in WPF PropertyGrid.....	1638
Define PropertyItem manually.....	1638
Add or remove PropertyItem at runtime.....	1639
Custom definition of PropertyItem.....	1641
Manually add own value editor and categorize the properties	1642
Manually define nested properties for PropertyItem	1644
Manually customize the UI of description panel	1646

Grouping in WPF PropertyGrid	1648
Grouping using attributes	1648
Grouping the Properties at runtime	1650
Expand or Collapse Category group	1652
Show or Hide the Group Button	1653
Sorting in WPF PropertyGrid	1654
Sorting the Properties	1655
Disable the Sorting	1657
Show or Hide the Sort Button	1659
Ordering in WPF PropertyGrid	1660
Ordering using Attribute	1660
Ordering based on properties defined in class	1662
Change Property order at runtime	1665
Virtualization in WPF PropertyGrid	1666
Filtering and Searching in WPF PropertyGrid	1667
Hide the Properties using Collection	1667
Hide the Properties using Attributes	1669
Hide the Properties at runtime	1671
Searching the Properties	1673
#	1675
Keyboard Navigation in WPF PropertyGrid	1675
Keyboard Navigation between property items	1675
Handling focus of the editors	1676
Appearance in WPF PropertyGrid	1678
Setting the Foreground	1678
Setting the Background and FontWeight	1678
Category Header's foreground and background	1681
Customize the height of PropertyViewItem and PropertyCategoryViewItem	1682
Tooltip support	1683
Theme	1684
Localization in WPF PropertyGrid	1685
SfRadialMenu	1688
WPF Radial Menu (SfRadialMenu) Overview	1688
Getting Started with WPF Radial Menu (SfRadialMenu)	1688
Theme	1689

Populating Items in WPF Radial Menu (SfRadialMenu).....	1689
Items Source	1689
Display Member Path.....	1690
Command Path	1691
Item Template.....	1691
Icon in WPF Radial Menu (SfRadialMenu)	1692
Populating Color Palette in WPF Radial Menu (SfRadialMenu).....	1692
ToolTip in WPF Radial Menu (SfRadialMenu)	1693
Layout Types in WPF Radial Menu (SfRadialMenu)	1694
Appearance and Styling	1695
Radius.....	1695
CenterRimRadiusFactor	1696
RimBackground	1696
RimActiveBrush.....	1696
RimInactiveBrush	1697
RimHoverBrush	1698
IsExpanderVisible	1698
RimRadiusFactor	1699
Navigation Button Style	1700
MenuBackgroundColor	1700
MenuMouseOverBackgroundColor	1701
SfRadialSlider	1702
WPF Radial Slider (SfRadialSlider) Overview	1702
Key features	1702
Getting Started with WPF Radial Slider (SfRadialSlider)	1702
Structure of SfRadialSlider	1703
Assembly deployment.....	1703
Adding WPF SfRadialSlider via designer	1703
Adding WPF SfRadialSlider via XAML.....	1704
Adding WPF SfRadialSlider via C#	1704
Select tick value	1705
Display selected value.....	1706
Change min-max tick value	1707
Change start and end position	1707
Change tick display interval	1708

Step interval	1709
Change inner and outer rim radius	1709
Change tick direction	1710
Text formatting	1711
Value changed notification	1712
Theme	1713
Working With RadialSlider in WPF Radial Slider (SfRadialSlider)	1713
Select tick value	1713
Display selected value	1714
Custom UI for display content	1715
Change min-max tick value	1716
Change start and end position	1716
Change tick display interval	1717
Step interval	1718
Change tick direction	1718
Display maximum value	1719
Change tick radius	1720
Change tick label radius	1721
Change inner rim radius.....	1722
Change outer rim radius	1724
Change selection pointer radius	1725
Text formatting	1727
Value changed notification	1728
Appearance in WPF Radial Slider (SfRadialSlider).....	1729
Setting the foreground.....	1729
Setting the background.....	1729
Change flow direction	1730
Theme	1730

SfDomainUpDown

Getting Started with WPF Domain Updown (SfDomainUpDown)

This section provides you an overview of working with [SfDomainUpDown](#) for WPF and provides a walk through to configure the [SfDomainUpDown](#) control in a real time scenario.

Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package needs to be added as a 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 to the SfDomainUpDown's dependent assemblies.

Creating Application with SfDomainUpDown control

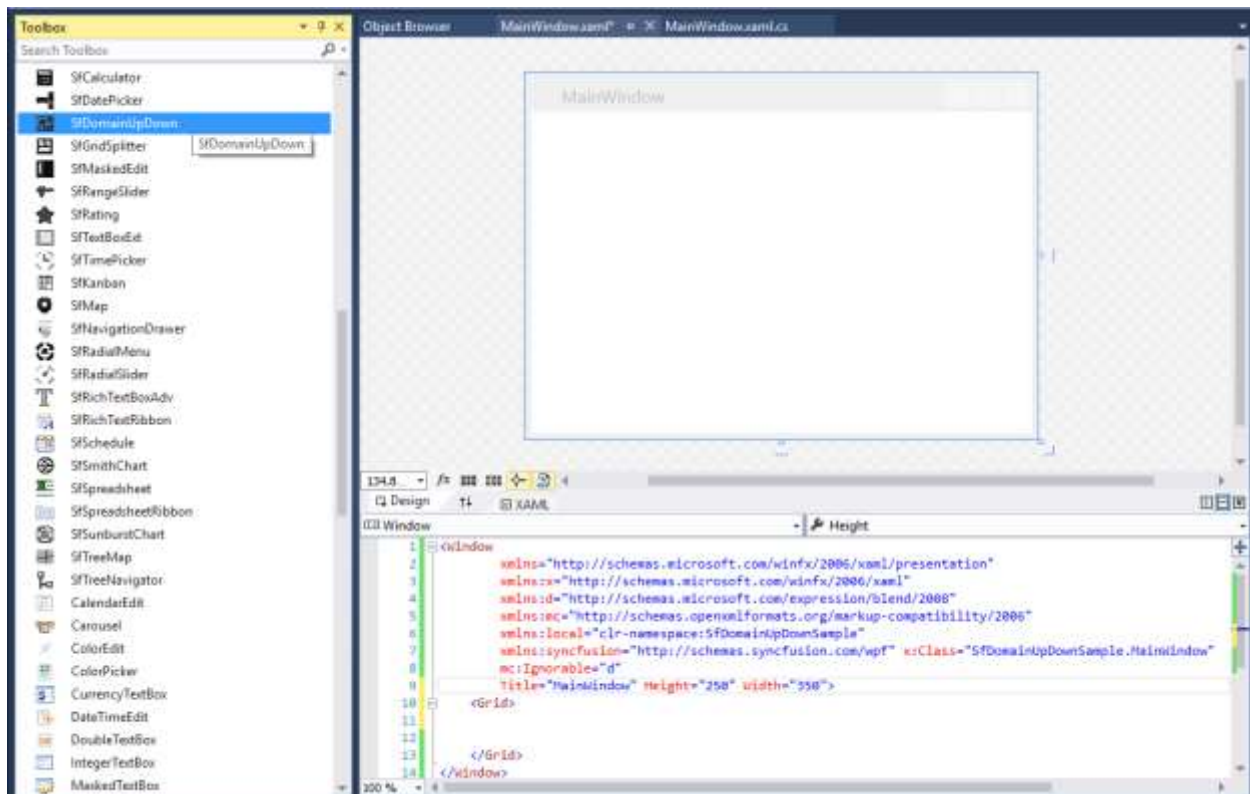
In this walk through, you will create a WPF application that contains the [SfDomainUpDown](#) control.

Creating project

Create a new WPF project to show the [SfDomainUpDown](#) control in Visual Studio.

Adding control via designer

The [SfDomainUpDown](#) 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 control manually in XAML page, follow the given steps:

1.Add the following required assembly references to the project, *Syncfusion.SfInput.WPF*
 Syncfusion.SfShared.WPF 2.Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page or Syncfusion.Windows.Tools.Controls namespace. 3.Declare [SfDomainUpDown](#) 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:SfDomainUpDown Height="30" Width="150" Value="40" />
</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.SfInput.WPF*
 Syncfusion.SfShared.WPF 2.Import the **SfDomainUpDown** namespace
Syncfusion.Windows.Controls.Input. 3.Create the **SfDomainUpDown** control instance and add it to the page.

C#

```
using System.Windows;
using Syncfusion.Windows.Controls.Input;
namespace ComboBox
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfDomainUpDown sfDomainUpDown1 = new SfDomainUpDown();
            this.Content = sfDomainUpDown1;
            sfDomainUpDown1.Height = 30;
            sfDomainUpDown1.Width = 150;
            sfDomainUpDown1.Value = 40;
        }
    }
}
```




Populating by DataBinding

You can populate the [SfDomainUpDown](#) control using the [ItemsSource](#) property.

1.You can create a data object class named **Employee** as Model and declare properties as shown below,

C#

```
public class Employee
{
    private string name;
    private string email;
    public string Email
    {
        get { return email; }
        set { email = value; }
    }
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
```

2.You can create a **ViewModel** class with several data objects in constructor.

C#

```
public class ViewModel
{
    private List<Employee> employees;
    public List<Employee> Employees
    {
        get { return employees; }
        set { employees = value; }
    }
    public ViewModel()
    {
        Employees = new List<Employee>();
        populateItem();
    }
    private void populateItem()
    {
        Employees.Add(new Employee { Name = "Lucas", Email = "lucas@syncfusion.com" });
        Employees.Add(new Employee { Name = "James", Email = "james@syncfusion.com" });
        Employees.Add(new Employee { Name = "Jacob", Email = "jacob@syncfusion.com" });
    }
}
```

3. You can bind the Employees property from the ViewModel class to the ItemSource property of SfDomainUpDown control and the control is set to display content based on the Name inside the [ContentTemplate](#) property.

XML

```
<syncfusion:SfDomainUpDown x:Name="domainUpDown"
HorizontalAlignment="Center" VerticalAlignment="Center" Width="200"
ItemsSource="{Binding Employees}">
<syncfusion:SfDomainUpDown.ContentTemplate>
<DataTemplate>
<StackPanel Orientation="Horizontal">
<TextBlock Text="{Binding Name}"/>
</StackPanel>
</DataTemplate>
</syncfusion:SfDomainUpDown.ContentTemplate>
</syncfusion:SfDomainUpDown>
```



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

Spin button alignment

You can customize the position of the spin button in the [SfDomainUpDown](#) control using the [SpinButtonsAlignment](#) property.

XML

```
<syncfusion:SfDomainUpDown x:Name="domainUpDown" Height="35" Width="150"
SpinButtonsAlignment="Right" Value="James" />
```

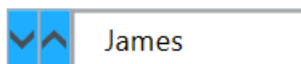
C#

```
SfDomainUpDown domainUpDown1 = new SfDomainUpDown();
domainUpDown1.Height = 30;
domainUpDown1.Width = 150;
domainUpDown1.SpinButtonsAlignment =
Syncfusion.Windows.Controls.SpinButtonsAlignment.Right;
```

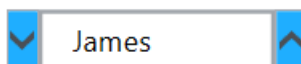
1.Right



2.Left



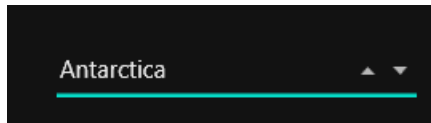
3.Both



Theme

SfDomainUpDown supports various built-in themes. Refer to the below links to apply themes for the SfDomainUpDown,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Populating Data in WPF Domain Updown (SfDomainUpDown)

The DomainUpDown control can be populated with a predefined list of items.

For example, in the following code, the DomainUpDown populates a list of employees:

C#

```
public class Employee
{
    public string Name { get; set; }
    public string Email { get; set; }
}
```

Create a collection attribute:

C#

```
private List<Employee> employees;
public List<Employee> Employees
{
    get { return employees; }
    set { employees = value; }
}
```

Populate the collection with items:

C#

```
Employees = new List<Employee>();
Employees.Add(new Employee{Name = "Lucas", Email = "lucas@syncfusion.com"});
Employees.Add(new Employee { Name = "James", Email = "james@syncfusion.com"
});
Employees.Add(new Employee { Name = "Jacob", Email = "jacob@syncfusion.com"
});
```

ItemsSource

Bind the Employees collection to the ItemsSource property of DomainUpDown:

C#

```
<Page xmlns:editors="clr-
namespace:Syncfusion.Windows.Controls.Input;assembly=Syncfusion.SfInput.Wpf"
>
```

```
<Grid>
<editors:SfDomainUpDown x:Name="domainUpDown"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="200"
ItemsSource="{Binding Employees}" >
</editors:SfDomainUpDown>
</Grid>
</Page>
```

Note: When the ContentTemplate property of the DomainUpDown control is not set, Items will be displayed as business objects in the control.

ContentTemplate

ContentTemplate helps the user decorate the content with visual elements. At this point, the control is populated with list of employees, and the Employee model contains two properties: Name and Email. In this example, the control is set to display content based on Name.

XML

```
<editors:SfDomainUpDown x:Name="domainUpDown"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="200"
ItemsSource="{Binding Employees}">
<editors:SfDomainUpDown.ContentTemplate>
<DataTemplate>
<StackPanel Orientation="Horizontal">
<Image Height="24" Width="24" Source="Image.png"/>
<TextBlock Text="{Binding Name}"/>
</StackPanel>
</DataTemplate>
</editors:SfDomainUpDown.ContentTemplate>
</editors:SfDomainUpDown>
```



AutoReverse in WPF Domain Updown (SfDomainUpDown)

Incrementing the value starts from the maximum value once it has reached the minimum value and starts from the minimum value once it has reached the maximum value.

XML

```
<editors:SfDomainUpDown x:Name="domainUpDown"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="200"
AutoReverse="True"
ItemsSource="{Binding Employees}">
</editors:SfDomainUpDown >
```

Spin Button Alignment in WPF Domain Updown (SfDomainUpDown)

The spin button's position in the DomainUpDown control can be changed using `SpinButtonsAlignment`. It contains three modes for positioning spin buttons:

1. Right
2. Left
3. Both

Right

Spin buttons will be aligned on the right side of the control.

XML

```
<editors:SfDomainUpDown x:Name="domainUpDown"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="200"
SpinButtonsAlignment="Right"
ItemsSource="{Binding Employees}">
</editors:SfDomainUpDown>
```

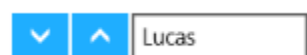


Left

Spin buttons will be aligned on the left side of the control.

XML

```
<editors:SfDomainUpDown x:Name="domainUpDown"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="200"
SpinButtonsAlignment="Left"
ItemsSource="{Binding Employees}">
</editors:SfDomainUpDown>
```



Both

The spin button's decrement button will be aligned on the left side of the control and the increment button is aligned on the right side of the control.

XML

```
<editors:SfDomainUpDown x:Name="domainUpDown"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Width="200"
    SpinButtonsAlignment="Both"
    ItemsSource="{Binding Employees}">
</editors:SfDomainUpDown>
```



Gestures in WPF Domain Updown (SfDomainUpDown)

Mouse Wheel

The current item moves up or down when mouse wheel is scrolled.

Appearance and Styling in WPF Domain Updown (SfDomainUpDown)

Spin animation

Items will spin up or down with smooth transition. The transition can be disabled using the EnableSpinAnimation property.

XML

```
<syncfusion:SfDomainUpDown x:Name="domain"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Width="200" EnableSpinAnimation="True"/>
```

Accent brush

The AccentBrush property is used to decorate the hot spots of a control with a solid color.

XML

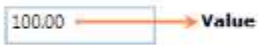
```
<Page xmlns:editors="clr-
namespace:Syncfusion.Windows.Controls.Input;assembly=Syncfusion.SfInput.Wpf"
>
<Grid>
<editors:SfDomainUpDown x:Name="domainUpDown"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Width="200"
    AccentBrush="Black"
    Value="James">
</editors:SfDomainUpDown >
</Grid>
</Page>
```

DoubleTextBox

WPF Double TextBox Overview

The [DoubleTextBox](#) control restricts input to double values with support for data binding, watermark, null value and support for culture. It provides various customization options to improve its appearance and suit an applications.

Control structure



Features

The core features of the [DoubleTextBox](#) 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 Null Value support.
- Provides culture support.

Getting Started with WPF Double TextBox

This section explains how to create a WPF [DoubleTextBox](#) 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.

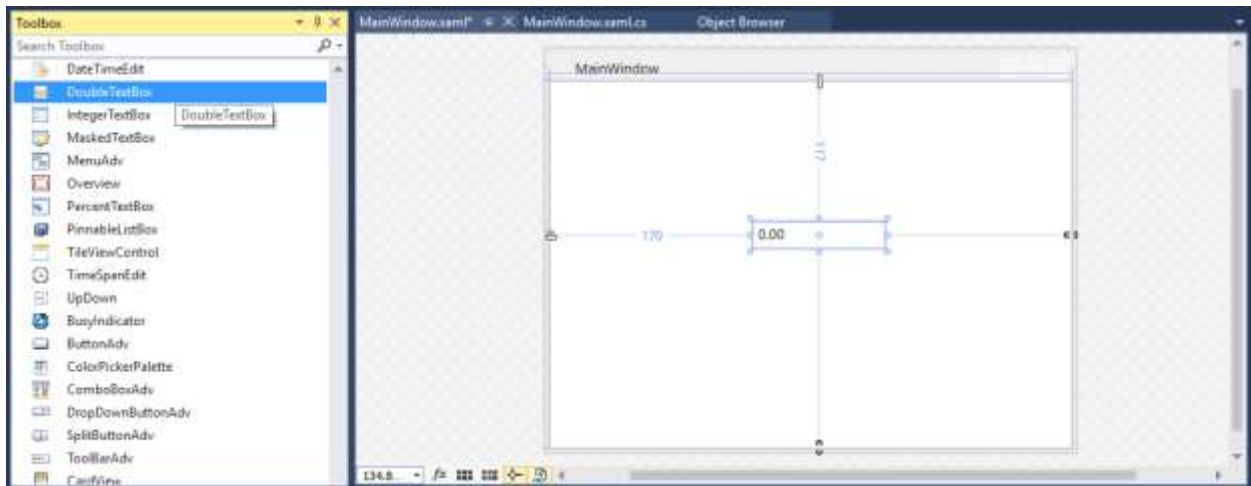
You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

Adding WPF DoubleTextBox via designer

You can add the [DoubleTextBox](#) 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 DoubleTextBox via XAML

To add the DoubleTextBox 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 **DoubleTextBox** 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="DoubleTextBoxSample.MainWindow"
Title="DoubleTextBox Sample" Height="350" Width="525">
<Grid>
<!--Adding DoubleTextBox control -->
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="100" Height="25"
VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
</Window>
```

Adding WPF DoubleTextBox via C#

To add the DoubleTextBox 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.

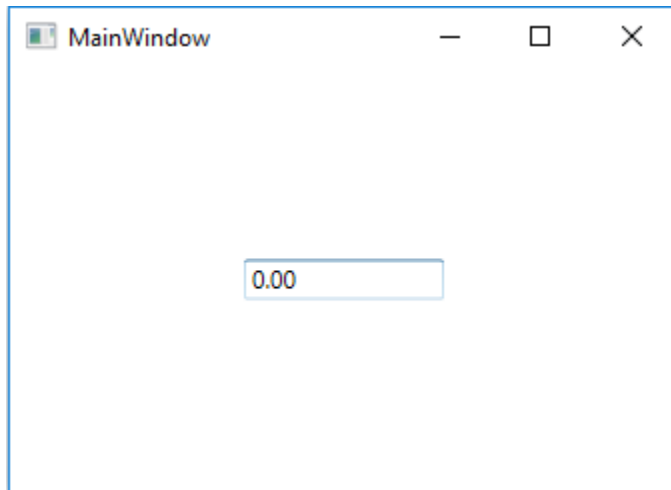
C#

```
using Syncfusion.Windows.Shared;
```

4. Create an instance of DoubleTextBox and add it to the window.

C#

```
//Creating an instance of DoubleTextBox control
DoubleTextBox doubleTextBox = new DoubleTextBox();
// Setting height and width to DoubleTextBox
doubleTextBox.Height = 25;
doubleTextBox.Width = 100;
//Adding DoubleTextBox as window content
this.Content = doubleTextBox;
```



Setting Value

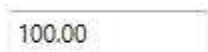
The value of the DoubleTextBox can be set by using the [Value](#) property.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="100" Height="23"
Value="100"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 23;
doubleTextBox.Value = 100;
```



Note: Do not use the [Text](#) property to set the value for the DoubleTextBox. 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 DoubleTextBox using the [Value](#) Property.

The following code snippets illustrate the value binding from one DoubleTextBox to another.

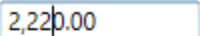
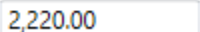
XML

```
<StackPanel>
<syncfusion:DoubleTextBox x:Name="doubleTextBox1" Height="25" Width="100"
Value="{Binding myValue,UpdateSourceTrigger=PropertyChanged}"/>
<syncfusion:DoubleTextBox x:Name="doubleTextBox2" 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 **DoubleTextbox** 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:DoubleTextBox ValueChanged="DoubleTextBox_ValueChanged"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.ValueChanged += new
PropertyChangedCallback(DoubleTextBox_ValueChanged);
```

You can handle the event as follows:

C#

```
private void DoubleTextBox_ValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    // Get old and new value
    var newValue = e.NewValue;
    var oldValue = e.OldValue;
}
```

Min Max Value Restriction

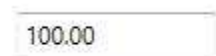
The **Value** of **DoubleTextBox** can be restricted within maximum and minimum limit. You can define the minimum and maximum values by setting the [MinValue](#) and [MaxValue](#) properties. It allows the user to enter the value between **MinValue** and **MaxValue**.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="100" Height="25"
Value="100" MaxValue="999.99" MinValue="-999.99"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
//Setting minimum value
doubleTextBox.MinValue = -999.99;
//Setting maximum value
doubleTextBox.MaxValue = 999.99;
doubleTextBox.Value = 100;
```


Step Interval to increase or decrease the value

The **DoubleTextBox** 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:DoubleTextBox x:Name="doubleTextBox" Width="150" Height="25"
Value="8"
IsScrollingOnCircle="True" ScrollInterval="4"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 150;
doubleTextBox.Height = 25;
doubleTextBox.MinValue = 0;
doubleTextBox.MaxValue = 100;
doubleTextBox.Value = 8;
```

```
doubleTextBox.IsScrollingOnCircle = true;
doubleTextBox.ScrollInterval = 4;
```

12.00

Formatting the value

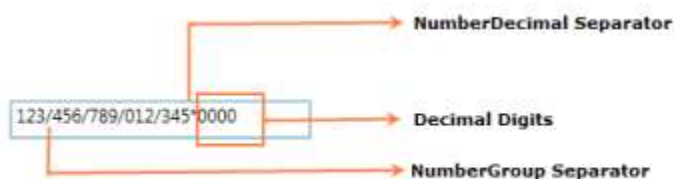
You can customize the number format by either setting the [NumberFormat](#) property or the [NumberGroupSeparator](#), [NumberGroupSizes](#), [NumberDecimalDigits](#), and [NumberDecimalSeparator](#) properties of `DoubleTextBox`.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Height="25" Width="200"
Value="123456789012345">
<syncfusion:DoubleTextBox.NumberFormat >
<numberformat:NumberFormatInfo NumberGroupSeparator="/"
NumberDecimalDigits="4" NumberDecimalSeparator="*" />
</syncfusion:DoubleTextBox.NumberFormat>
</syncfusion:DoubleTextBox>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 200;
doubleTextBox.Height = 25;
doubleTextBox.Value = 123456789012345;
doubleTextBox.NumberFormat = new NumberFormatInfo()
{
    NumberGroupSeparator = "/",
    NumberDecimalDigits = 4,
    NumberDecimalSeparator = "*"
};
```



Setting the Culture


The `DoubleTextBox` provides support for globalization by using the [Culture](#) property. The `Culture` is used to format the decimal separator and group separator of the `DoubleTextBox` value based on the respective culture.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Height="25" Width="150"
Culture="en-US" Value="1234567"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 150;
doubleTextBox.Height = 25;
doubleTextBox.Value = 1234567;
doubleTextBox.Culture = new System.Globalization.CultureInfo("en-US");
```




Note: When you use both `NumberFormat` and `Culture`, the `NumberFormat` will have a higher priority.

Theme

DoubleTextBox supports various built-in themes. Refer to the below links to apply themes for the DoubleTextBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Changing Double Value in WPF Double TextBox

The `DoubleTextBox` allows the user to change the value using the `Value` property.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Height="25"
Width="150" Value="10"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 150;
doubleTextBox.Height = 25;
doubleTextBox.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 `DoubleTextBox` value.

The following code snippets illustrate the value binding from one `DoubleTextBox` to another.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox1" Value="{Binding
MyValue, UpdateSourceTrigger=PropertyChanged}" Height="25" Width="100"/>
<syncfusion:DoubleTextBox x:Name="doubleTextBox2" Value="{Binding
MyValue, UpdateSourceTrigger=PropertyChanged}" Width="100" Height="25" />
```

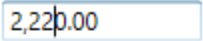
ViewModel.cs

C#

```

class ViewModel : NotificationObject
{
    private double myValue;
    public double MyValue
    {
        get
        {
            return myValue;
        }
        set
        {
            myValue = value;
            RaisePropertyChanged("MyValue");
        }
    }
}

```




Change double value by pasting the clipboard's text

By default, **DoubleTextBox** 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:DoubleTextBox PasteMode="Advanced"
Value="12345.67"
Name="doubleTextBox"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.PasteMode = PasteMode.Advanced;
doubleTextBox.Value = 12345.67;
```

Pasting "89.56"

**Show UpDown Button**

You can increment or decrement the double value of **DoubleTextBox** by setting the **ShowSpinButton** property value as **true**. Click UpButton to increment or DownButton to decrement the double value. The default value of **ShowSpinButton** property is **false**.

XML

```
<syncfusion:DoubleTextBox Height="30" Width="150" ShowSpinButton="True" />
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.ShowSpinButton = true;
```

**Value Changed Event**

The **DoubleTextbox** 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:DoubleTextBox ValueChanged="DoubleTextBox_ValueChanged"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.ValueChanged += new
PropertyChangedCallback(DoubleTextBox_ValueChanged);
```

You can handle the event as follows:

C#

```
private void DoubleTextBox_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 `DoubleTextBox` 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 double value. It will display only on setting the `UseNullOption` property is set to `true`.

NullValue = Null**XML**

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Height="25"
Width="100" UseNullOption="True" NullValue="{x:Null}"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.NullValue = null;
doubleTextBox.UseNullOption = true;
```

**NullValue = 10****XML**

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Height="25"
Width="100" UseNullOption="True" NullValue="10"/>
```


C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.NullValue = 10;
doubleTextBox.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 **DoubleTextBox** allows you to set the desired brush as a foreground for **WaterMarkText** using [WaterMarkTextForeground](#) property. The default color of **WaterMarkTextForeground** is Black.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="100"
Height="25" UseNullOption="True" WatermarkText="Type here"
WatermarkTextIsVisible="True" WatermarkTextForeground="Red"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.UseNullOption = true;
doubleTextBox.WatermarkText = "Type Here";
doubleTextBox.WatermarkTextIsVisible = true;
doubleTextBox.WatermarkTextForeground = Brushes.Red;
```


Setting Watermark Template

You can customize the Visual appearance of the **WatermarkText** by using the [WatermarkTemplate](#) property.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="100" Height="25"
WatermarkText="Type Here" CornerRadius="3"
WatermarkTextIsVisible="True" WatermarkOpacity="0.5"
UseNullOption="True">
  <syncfusion:DoubleTextBox.WatermarkTemplate >
    <DataTemplate>
      <Border Background="Red">
```

```
<TextBlock Text="{Binding}" VerticalAlignment="Center" Margin="5,0,0,0"/>
</Border>
</DataTemplate>
</syncfusion:DoubleTextBox.WatermarkTemplate>
</syncfusion:DoubleTextBox>
```



Note: The `UseNullOption` property must be enabled if you want to see `NullValue` or `WaterMarkText` in `DoubleTextBox` control.

Note: If both `NullValue` and `WaterMarkText` are specified, you will only see `NullValue` but not `WaterMarkText`.

Step Interval in WPF Double TextBox

The `DoubleTextBox` 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 `DoubleTextBox` 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 `DoubleTextBox` control allows you to increase or decrease the `Value` of `DoubleTextBox` based on the `ScrollInterval` by pressing the up arrow and down arrow keys on the keyboard.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="150"
Height="25" Value ="10" ScrollInterval="2"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 150;
doubleTextBox.Height = 25;
doubleTextBox.Value = 10;
doubleTextBox.ScrollInterval = 2;
```



Change Value on Mouse Wheel

The `DoubleTextBox` 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:DoubleTextBox x:Name="doubleTextBox" Width="150" Height="25"
Value ="34"
IsScrollingOnCircle="True" ScrollInterval="3"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 150;
doubleTextBox.Height = 25;
doubleTextBox.Value = 34;
doubleTextBox.IsScrollingOnCircle = true;
doubleTextBox.ScrollInterval = 3;
```

**Change Value on Click and Drag**

The **DoubleTextBox** allows you to increase or decrease the value based on the **ScrollInterval** by clicking and dragging the mouse when the **EnableExtendedScrolling** property is **true**. **DoubleTextBox** value increases when you click and drag the mouse to the right or the top of the screen and decreases when the mouse moves to the left or the bottom of the screen. Before that, the control should be in an unfocused state.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="120" Height="25"
Value="10"
ScrollInterval="5" EnableExtendedScrolling="True"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 120;
doubleTextBox.Height = 25;
doubleTextBox.Value = 10;
doubleTextBox.ScrollInterval = 5;
doubleTextBox.EnableExtendedScrolling = true;
```

**Allow or restrict selection on focus**

DoubleTextBox 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

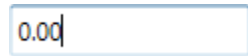
```
<syncfusion:DoubleTextBox x:Name="doubleTextBox"
TextSelectionOnFocus="False"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
```

```
doubleTextBox.TextSelectionOnFocus = true;
```

TextSelectionOnFocus = "False"



TextSelectionOnFocus = "True"



Restriction or Validation in WPF Double TextBox

This section explains how to validate or restrict the `DoubleTextBox` control value.

Restrict the value within minimum and maximum value

The [Value](#) of the `DoubleTextBox` 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 `DoubleTextBox` 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 `DoubleTextBox` is validated, when the `DoubleTextBox` loses the focus. That is, the `DoubleTextBox` will accept any value, validation will only take place after the `DoubleTextBox` has lost its keyboard focus. After validation, when the value of the `DoubleTextBox` 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 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:DoubleTextBox x:Name="doubleTextBox" Width="150" MaxValue="100"
MinValue="10"
MinValueOnExceedMinDigit="True" MaxValueOnExceedMaxDigit="True"
MinValidation="OnKeyPress" MaxValidation="OnLostFocus"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 150;
doubleTextBox.Height = 25;
doubleTextBox.MinValue = 10;
doubleTextBox.MaxValue = 100;
doubleTextBox.MinValidation = MinValidation.OnKeyPress;
doubleTextBox.MaxValidation = MaxValidation.OnLostFocus;
doubleTextBox.MinValueOnExceedMinDigit = true;
doubleTextBox.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 digit

You can format the decimal digits in the `DoubleTextBox` control using `NumberDecimalDigits` property. You can restrict the decimal digits of the text within maximum and minimum limits in `DoubleTextBox` control using `MinimumNumberDecimalDigits` and `MaximumNumberDecimalDigits` properties. The default value of `MinimumNumberDecimalDigits`, `MaximumNumberDecimalDigits` and `DoubleDecimalDigits` properties is `-1`.


Note: If the value of `MinimumNumberDecimalDigits` property is greater than the value of `MaximumNumberDecimalDigits` property, the text of `DoubleTextBox` will be updated based on the value of `MinimumNumberDecimalDigits` property.

XML

```
<syncfusion:DoubleTextBox Value="125.32545" HorizontalAlignment="Center"
VerticalAlignment="Center"
MaximumNumberDecimalDigits="4"
MinimumNumberDecimalDigits="1" />
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Value = 125.32545;
doubleTextBox.MaximumNumberDecimalDigits = 4;
doubleTextBox.MinimumNumberDecimalDigits = 1;
```



When value of `MinimumNumberDecimalDigits`, `MaximumNumberDecimalDigits` and `NumberDecimalDigits` properties are specified, `NumberDecimalDigits` property takes high precedence and updates the text of `DoubleTextBox` property.

XML

```
<syncfusion:DoubleTextBox Value="125.32545" HorizontalAlignment="Center"
VerticalAlignment="Center"
MaximumNumberDecimalDigits="4"
MinimumNumberDecimalDigits="1"
NumberDecimalDigits="3"
/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Value = 125.32545;
doubleTextBox.MaximumNumberDecimalDigits = 4;
doubleTextBox.MinimumNumberDecimalDigits = 1;
doubleTextBox.NumberDecimalDigits = 3;
```



Read only mode

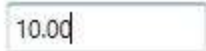
The `DoubleTextBox` cannot allow the user input, edits when `IsReadOnly` property is sets to `true`. The user can still select text and display the cursor on the `DoubleTextBox` by setting the `IsReadOnlyCaretVisible` property to `true`. However, value can be changed programmatically in readonly mode.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" IsReadOnly="True"
Value="10" IsReadOnlyCaretVisible="True"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Value = 10;
doubleTextBox.IsReadOnly = true;
doubleTextBox.IsReadOnlyCaretVisible = true;
```



Culture and Formatting in WPF Double TextBox

Value of `DoubleTextBox` can be formatted in following ways:

- Culture
- NumberFormatInfo
- Dedicated properties (NumberGroupSeparator, NumberGroupSizes, NumberDecimalDigits, NumberDecimalSeparator)

Culture based formatting

The `DoubleTextBox` provides support for globalization by using the `Culture` property. The `Culture` property is used to format the decimal separator and group separator of the `DoubleTextBox` value based on the respective culture.

XML

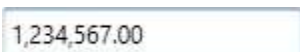
```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Height="25" Width="150"
Culture="bs-Latn" Value="1234567"/>
```

C#

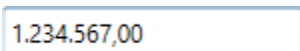
```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 150;
doubleTextBox.Height = 25;
doubleTextBox.Value = 1234567;
//Setting Latin culture for double textbox.
doubleTextBox.Culture = new System.Globalization.CultureInfo("bs-Latn");
```

By default the US culture uses “,” as the `NumberGroupSeparator` and “.” as the `NumberDecimalSeparator` where as the Latin culture uses “.” as the `NumberGroupSeparator` and “,” as the `NumberDecimalSeparator`.

Default Culture



Latin Culture



NumberFormatInfo based formatting

The number formatting of `DoubleTextBox` can be customized by setting `NumberFormat` property.

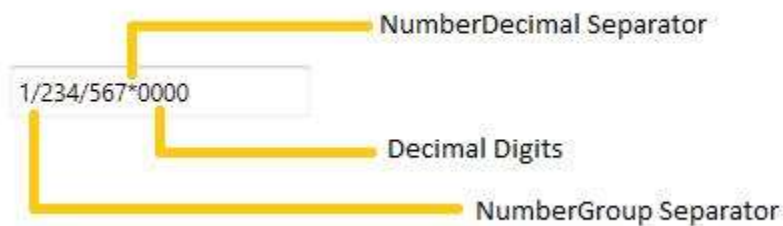
XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Height="25"
Width="200" Value="1234567"
GroupSeparatorEnabled = "True">
<syncfusion:DoubleTextBox.NumberFormat>
<numberformat:NumberFormatInfo NumberGroupSeparator="/"
NumberDecimalDigits="4"
```

```
NumberDecimalSeparator="*"/>
</syncfusion:DoubleTextBox.NumberFormat>
</syncfusion:DoubleTextBox>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 200;
doubleTextBox.Height = 25;
doubleTextBox.Value = 1234567;
doubleTextBox.GroupSeparatorEnabled = true;
doubleTextBox.NumberFormat = new NumberFormatInfo()
{
    NumberGroupSeparator = "/",
    NumberDecimalDigits = 4,
    NumberDecimalSeparator = "*"
};
```



The following code illustrate how to set number group size by using the **NumberFormat** property.

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 200;
doubleTextBox.Height = 25;
doubleTextBox.Value = 123456789;
doubleTextBox.NumberFormat = new System.Globalization.NumberFormatInfo()
{
    NumberDecimalDigits = 4,
    NumberGroupSeparator = "/",
    NumberDecimalSeparator = "*",
    // Adding the Number group size via NumberFormat property.
    NumberGroupSizes = new int[] { 2, 3, 4 }
};
```

1234/567/89*0000

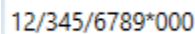
Formatting with dedicated properties

The number formatting of **DoubleTextBox** can also be customized by setting [NumberGroupSeparator](#), [NumberGroupSizes](#), [NumberDecimalDigits](#), and [NumberDecimalSeparator](#) properties. You can show the group separator by enable the [GroupSeparatorEnabled](#) property to **true**.

The following code illustrate how to format using the `NumberDecimalSeparator`, `NumberDecimalDigits`, `NumberGroupSeparator`, `NumberGroupSizes` property of the `DoubleTextBox`.

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 150;
doubleTextBox.Height = 25;
doubleTextBox.Value = 123456789;
doubleTextBox.NumberGroupSeparator = "/";
doubleTextBox.NumberDecimalSeparator = "*";
doubleTextBox.NumberDecimalDigits = 3;
// Adding the Number group size via NumberGroupSizes property.
doubleTextBox.NumberGroupSizes = new Int32Collection() { 4, 3, 2};
```



Note: When you use both the `NumberFormat` and the dedicated properties (`NumberGroupSeparator`, `NumberGroupSizes`, `NumberDecimalDigits`, and `NumberDecimalSeparator`) to format the value of `DoubleTextbox`, the `NumberGroupSeparator`, `NumberGroupSizes`, `NumberDecimalDigits`, and `NumberDecimalSeparator` properties have higher priority.

Note: When you use both `NumberFormat` and `Culture`, the `NumberFormat` will have a higher priority.

Appearance in WPF Double TextBox

This section deals with the appearance of `DoubleTextBox` control and contains the following topics.

Setting the Foreground

The `DoubleTextBox` control `Foreground` can be modified based on the value of the control. The following are the foreground for `DoubleTextBox` control.

Foreground for Positive Value

We can change a positive color for the value of `DoubleTextBox` by setting the `PositiveForeground` property and it will be applied when the `Value` is positive. The default color of `PositiveForeground` is Black.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Value="10" Width="100"
Height="25" PositiveForeground="Blue" />
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.PositiveForeground = Brushes.Blue;
```



Foreground for Negative Value

We can change a negative color for the value of DoubleTextBox 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:DoubleTextBox x:Name="doubleTextBox" Value="10" Width="100"
Height="25"
NegativeForeground="SpringGreen" ApplyNegativeForeground="True" />
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.ApplyNegativeForeground = true;
doubleTextBox.NegativeForeground = Brushes.SpringGreen;
```

*Foreground for Zero Value*

We can change a zero color for the value of DoubleTextBox 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:DoubleTextBox x:Name="doubleTextBox" Value="0" Width="100"
Height="25"
ApplyZeroColor="True" ZeroColor="DarkGoldenrod"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Value = 0;
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.ApplyZeroColor = true;
doubleTextBox.ZeroColor = Brushes.DarkGoldenrod;
```

*Setting the Background*

DoubleTextBox 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:DoubleTextBox x:Name="doubleTextBox" Width="100"
Height="25" Value="80" Background="Cyan"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.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 DoubleTextBox, use [CornerRadius](#) property. The default value of [CornerRadius](#) property is 1.

XML

```
<syncfusion:DoubleTextBox x:Name="DoubleTextBox1" Width="100" Height="25"
CornerRadius="5"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.CornerRadius = new CornerRadius(5);
```


Apply Background for Selection

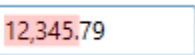
DoubleTextBox allows different brushes to highlight the selected text by setting the [SelectionBrush](#) and [SelectionOpacity](#) properties. The [SelectionOpacity](#) property specifies the opacity of the [SelectionBrush](#).

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="100" Height="25"
SelectionBrush="Red" SelectionOpacity="0.5"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.SelectionBrush = Brushes.Red;
doubleTextBox.SelectionOpacity = 0.3;
```



Align Value

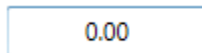
DoubleTextBox allows to display the value from right or center or left side by setting the [TextAlignment](#) property to **Right** or **Left** or **Center**. The Default value of **TextAlignment** is **Left**.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="100" Height="25"
TextAlignment="Center"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.TextAlignment = TextAlignment.Center;
```



Setting Tooltip

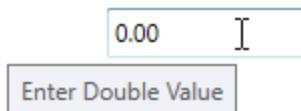
DoubleTextBox provides support for Tooltip to display certain information when the mouse hovers on the DoubleTextBox. You can customize the tooltip information by setting the [ToolTip](#) property.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" Width="100" Height="25"
ToolTip="Enter Double Value"/>
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.Width = 100;
doubleTextBox.Height = 25;
doubleTextBox.ToolTip = "Enter Double value";
```



Theme

DoubleTextBox supports various built-in themes. Refer to the below links to apply themes for the DoubleTextBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Range Adorner in WPF Double TextBox

Value of DoubleTextBox can be visually indicated like a progress bar using range-adorner feature, this feature is disabled by default. You can show the adorner over [DoubleTextBox](#) 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:DoubleTextBox x:Name="doubleTextBox" MinValue="0" MaxValue="100"
Value="63" EnableRangeAdorner="True" />
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.MinValue = 0;
doubleTextBox.MaxValue = 100;
doubleTextBox.Value = 63;
doubleTextBox.EnableRangeAdorner = true;
```



Changing background of range-adorner

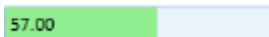
You can change the background color of the range adorner using [RangeAdornerBackground](#) property.

XML

```
<syncfusion:DoubleTextBox x:Name="doubleTextBox" MinValue="0" MaxValue="100"
Value="57" EnableRangeAdorner="True" RangeAdornerBackground="LightGreen" />
```

C#

```
DoubleTextBox doubleTextBox = new DoubleTextBox();
doubleTextBox.MinValue = 0;
doubleTextBox.MaxValue = 100;
doubleTextBox.Value = 57;
doubleTextBox.EnableRangeAdorner = true;
doubleTextBox.RangeAdornerBackground = Brushes.LightGreen;
```



DropDownButtonAdv

WPF Dropdown Button (DropDownButtonAdv) Overview

The WPF dropdown button (or DropDownButtonAdv) provides advanced menu-like appearance to the button UI. When the arrow is clicked, it displays a dropdown list with various options for selection.

Key features

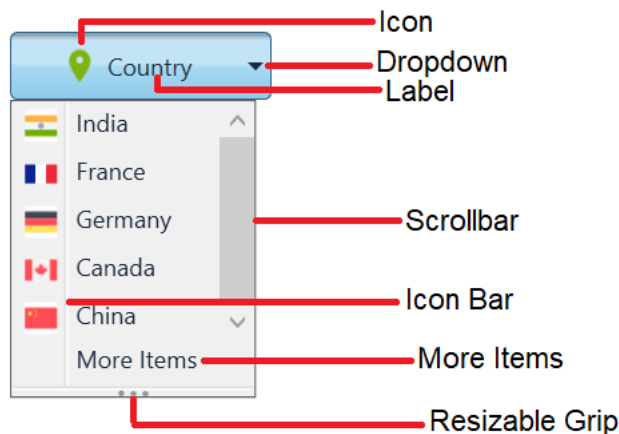
- **Data Binding** - Data binding allows the flow of data between UI elements and data object on user interface.

- **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.
- **Drop Direction** - The direction of the drop-down popup can be changed in a number of ways.
- **Resizing** - Use re-sizing gripper to increase or decrease pop-up height and width.
- **Multiline** - Provides support for displaying multiple lines of text in large 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.
- **Custom Items** - Provides support to add custom items to dropdown menu group.

Getting Started with WPF Dropdown Button (DropDownButtonAdv)

This section provides an overview of how to work with [Dropdown Button](#) control. It describes the control structure, the control initialization, the image setting for the control and adding items to the Dropdown Button control.

Control structure



Assembly deployment

Refer [DropDownButtonAdv](#) control dependencies section to get the list of assemblies or [NuGet package](#) needs to be added as reference to use the DropDownButtonAdv control in any application.

Creating simple application with Dropdown Button

In this walk through, will create WPF application that contains Dropdown Button control. By the following ways, one can add the controls:

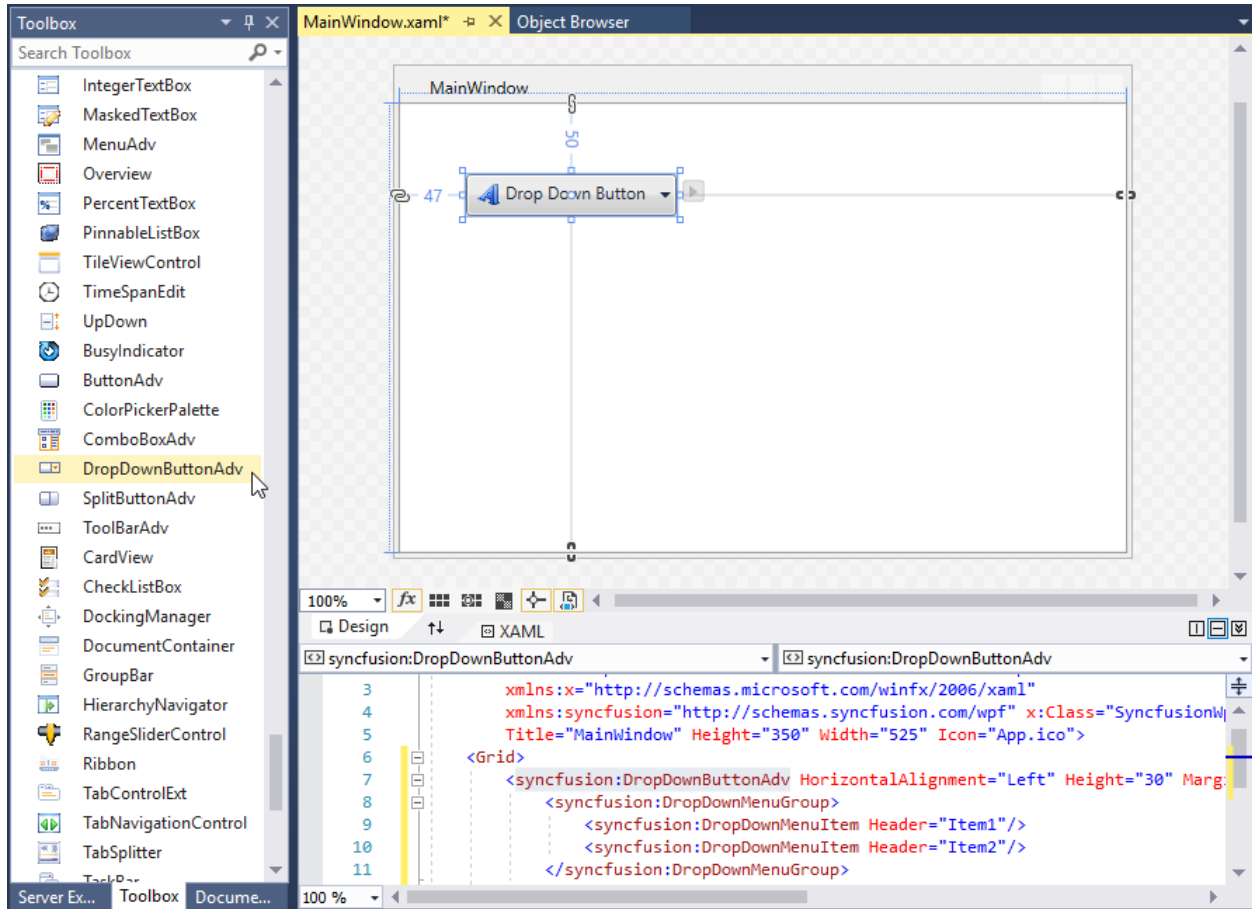
1. [Adding control via designer](#)
2. [Adding control manually in XAML](#)
3. [Adding control manually in C#](#)

Adding control via designer

Dropdown Button control can be added to the application by dragging **DropDownButtonAdv** from toolbox and dropping it in designer view. After dropping the controls in designer view, the assemblies such as **Syncfusion.Shared.WPF** gets added into the project automatically. The following code snippets will be added into the XAML.

XML

```
<syncfusion:DropDownButtonAdv x:Name="dropdownButtonAdv" Label="Drop Down Button"/>
```



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.
 - o 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 DropDownButtonAdv 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:DropDownButtonadv_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:DropDownButtonAdv Height="44" VerticalAlignment="Center"
HorizontalAlignment="Center" Width="162" />
</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.
 - o Syncfusion.Shared.WPF
2. Import the `Syncfusion.Windows.Tools.Controls` namespace.
3. Create DropDownButtonAdv 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:DropDownButtonadv_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 x:Name="Root">
</Grid>
</Window>

```

C#

```

using Syncfusion.Windows.Tools.Controls;
namespace DropDownButtonSample
{
public partial class MainWindow : Window
{
public MainWindow()
{
InitializeComponent();
DropDownButtonAdv dropdownButtonAdv = new DropDownButtonAdv();
dropdownButtonAdv.Height=44;
dropdownButtonAdv.Width=31;
Root.Children.Add(dropdownButtonAdv);
}
}
}

```

Setting label

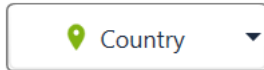
The label on the button is a text that explains its action to the end-user. Apply the text by using the [Label](#) property.

XML


```
<syncfusion:DropDownButtonAdv SmallIcon="Images\flagsmall.png"
Label="Country"/>
```

C#

```
DropDownButtonAdv button = new DropDownButtonAdv();
button.Label = "Country";
button.SmallIcon = new BitmapImage(new Uri("Images\flagsmall.png",
UriKind.RelativeOrAbsolute));
```

**Setting size mode**

Size mode is used to render Dropdown 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:DropDownButtonAdv SmallIcon="Images\flagsmall.png"
SizeMode="Small" Label="Country"/>
```

C#

```
DropDownButtonAdv button = new DropDownButtonAdv();
button.Label = "Country";
button.SizeMode = SizeMode.Small;
button.SmallIcon = new BitmapImage(new Uri("Images\flagsmall.png",
UriKind.RelativeOrAbsolute));
```

**Normal mode**

In a normal size button, a small image with the text on the side will be displayed.

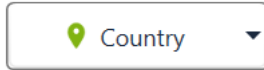
XML

```
<syncfusion:DropDownButtonAdv SizeMode="Normal"
SmallIcon="Images\flagsmall.png" Label="Country"/>
```

C#

```
DropDownButtonAdv button = new DropDownButtonAdv();
```

```
button.Label = "Country";
button.SizeMode = SizeMode.Normal;
button.SmallIcon = new BitmapImage(new Uri("Images\\flagsmall.png",
UriKind.RelativeOrAbsolute));
```



Large mode

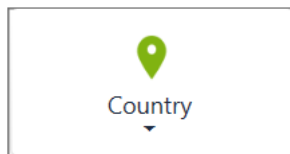
In a large size button, a large image along with the text at the bottom will be displayed.

XML

```
<syncfusion:DropDownButtonAdv LargeIcon="Images\\flaglarge.png"
SizeMode="Large" Label="Country"/>
```

C#

```
DropDownButtonAdv button = new DropDownButtonAdv();
button.Label = "Country";
button.SizeMode = SizeMode.Large;
button.LargeIcon = new BitmapImage(new Uri("Images\\flaglarge.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 DropDownButtonAdv. The icon will automatically resize the template content according to its size provided in the data template.

XML

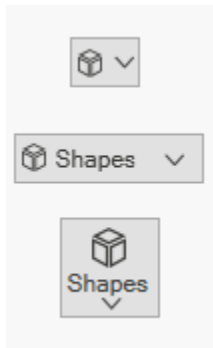
```
<Window x:Class="DropDownButtonAdv_IconTemplate.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:DropDownButtonAdv_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="M25.990023,11.689011L17.00602,15.955031 17.00602,28.605018
25.990023,23.618021z M2.0180035,11.688987L2.0180035,23.617996
11.003013,28.605 11.003013,15.95499z
M27.990021,8.526019L27.990021,24.796029 15.006021,32.002019
15.006021,14.690017z M0.018000603,8.5259848L13.003015,14.68999
```

```

13.003015,32.002002 0.018000603,24.795997z
M14.044014,2.2129984L4.6760044,6.6469963 14.043014,11.079994
23.348023,6.6469963z M14.047014,0L27.996028,6.6469963 14.047014,13.293993
0,6.6469963z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="normalIconTemplate">
<Grid Width="16" Height="16">
<Path
Data="M25.990023,11.689011L17.00602,15.955031 17.00602,28.605018
25.990023,23.618021z M2.0180035,11.688987L2.0180035,23.617996
11.003013,28.605 11.003013,15.95499z
M27.990021,8.526019L27.990021,24.796029 15.006021,32.002019
15.006021,14.690017z M0.018000603,8.5259848L13.003015,14.68999
13.003015,32.002002 0.018000603,24.795997z
M14.044014,2.2129984L4.6760044,6.6469963 14.043014,11.079994
23.348023,6.6469963z M14.047014,0L27.996028,6.6469963 14.047014,13.293993
0,6.6469963z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="largeIconTemplate">
<Grid Width="16" Height="16">
<Path
Width="16"
Height="16"
Data="M25.990023,11.689011L17.00602,15.955031 17.00602,28.605018
25.990023,23.618021z M2.0180035,11.688987L2.0180035,23.617996
11.003013,28.605 11.003013,15.95499z
M27.990021,8.526019L27.990021,24.796029 15.006021,32.002019
15.006021,14.690017z M0.018000603,8.5259848L13.003015,14.68999
13.003015,32.002002 0.018000603,24.795997z
M14.044014,2.2129984L4.6760044,6.6469963 14.043014,11.079994
23.348023,6.6469963z M14.047014,0L27.996028,6.6469963 14.047014,13.293993
0,6.6469963z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
</Window.Resources>
<StackPanel>
<syncfusion:DropDownButtonAdv x:Name="smallIcon" SizeMode="Small"
Label="Login" HorizontalAlignment="Center" VerticalAlignment="Center"
Margin="10" IconTemplate="{StaticResource smallIconTemplate}">
</syncfusion:DropDownButtonAdv>
<syncfusion:DropDownButtonAdv x:Name="normalIcon" SizeMode="Normal"
Label="Login" HorizontalAlignment="Center" VerticalAlignment="Center"
Margin="10" IconTemplate="{StaticResource normalIconTemplate}">
</syncfusion:DropDownButtonAdv>
<syncfusion:DropDownButtonAdv x:Name="largeIcon" SizeMode="Large"
Label="Login" HorizontalAlignment="Center" VerticalAlignment="Center"
Margin="10" IconTemplate="{StaticResource largeIconTemplate}">
</syncfusion:DropDownButtonAdv>
</StackPanel>

```

```
</Window>
```



Note: The [DropDownButtonAdv](#) the icon in the following priority order.

- [IconTemplate](#)
- [LargelIcon](#)
- [SmallIcon](#)

Setting icon template selector

The [IconTemplateSelector](#) property which allows you to specify a different data template based on the value given in the data templates.

XML

```
<Window x:Class="TemplateSelector_DropDownButtonAdv.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_DropDownButtonAdv"
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:DropDownButtonAdv 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 [DropDownButtonAdv](#) the icon in the following priority order.

- [IconTemplateSelector](#)
- [IconTemplate](#)

- [LargelIcon](#)
- [SmallIcon](#)

Setting image

The image option helps to provide pictorial representation of the button. Image can be added either using the [SmallIcon](#) or [LargelIcon](#) property.

- **SmallIcon** — This property will be used to set the image when size mode is **Normal** or **Small**.
- **LargelIcon** — This property will be used to set the image when size mode is **Large**.

The **SmallIcon** property can be set as follows:

XML

```
<syncfusion:DropDownButtonAdv SizeMode="Small" Label="Syncfusion" SmallIcon="Images\syncfusion.png"/>
```

C#

```
DropDownButtonAdv button = new DropDownButtonAdv();
button.Label = "Syncfusion";
button.SizeMode = SizeMode.Small;
button.SmallIcon = new BitmapImage(new Uri("Images\syncfusion.png",
UriKind.RelativeOrAbsolute));
```



The **SmallIcon** property can be set even when the sizeMode is **Normal**.

XML

```
<syncfusion:DropDownButtonAdv SizeMode="Normal" SmallIcon="image\syncfusion.png" Label="Syncfusion"/>
```

C#

```
DropDownButtonAdv button = new DropDownButtonAdv();
button.Label = "Syncfusion";
button.SizeMode = SizeMode.Normal;
button.SmallIcon = new BitmapImage(new Uri("Images\syncfusion.png",
UriKind.RelativeOrAbsolute));
```



The **LargelIcon** property can be set as follows:

XML

```
<syncfusion:DropDownButtonAdv SizeMode="Large" LargelIcon="Images\syncfusion.png" Label="Syncfusion"/>
```

C#

```
DropDownButtonAdv button = new DropDownButtonAdv();
button.Label = "Syncfusion";
button.SizeMode = SizeMode.Large;
button.LargeIcon = new BitmapImage(new Uri("Images\\syncfusion.png",
UriKind.RelativeOrAbsolute));
```



Setting icon width and height

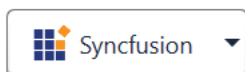
Icon width and icon height can be set using [IconWidth](#) and [IconHeight](#) properties respectively.

XML

```
<syncfusion:DropDownButtonAdv x:Name="button1" SizeMode="Normal"
IconHeight="20" IconWidth="20" Label="Syncfusion" SmallIcon
="Images\\syncfusion.png" />
```

C#

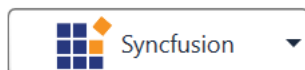
```
DropDownButtonAdv button1 = new DropDownButtonAdv();
button1.Label = "Syncfusion";
button1.IconWidth=20;
button1.IconHeight=20;
button1.SmallIcon = new BitmapImage(new Uri("Images\\syncfusion.png",
UriKind.RelativeOrAbsolute));
```

**XML**

```
<syncfusion:DropDownButtonAdv x:Name="button2" SizeMode="Normal"
IconHeight="30" IconWidth="30" Label="Syncfusion" SmallIcon
="Images\\syncfusion.png" />
```

C#

```
DropDownButtonAdv button2 = new DropDownButtonAdv();
button2.Label = "Syncfusion";
button2.IconWidth=30;
button2.IconHeight=30;
button2.SmallIcon = new BitmapImage(new Uri("Images\\syncfusion.png",
UriKind.RelativeOrAbsolute));
```



Note: View [sample](#) in GitHub. This sample showcases how to add dropdown button control and its basic features like image sizing options and size modes.

Adding items to Dropdown Button

The [DropDownMenuGroup](#) acts as a container for the Dropdown Button control. It provides options to add menu items and also options like header name, re-sizing and scrollbar.

Note: For more information on how to bind data with command actions for Dropdown Button please refer to the topics [Data Binding](#) and [Command Binding](#).

XML

```
<Window x:Class="DropDown_Button_Menuitem_Binding.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:DropDown_Button_Menuitem_Binding"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:CountryViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:DropDownButtonAdv Label="Country"
SmallIcon="Images/flagsmall.png" >
<syncfusion:DropDownMenuGroup ItemsSource="{Binding DropDownItems}">
<syncfusion:DropDownMenuGroup.ItemTemplate>
<DataTemplate>
<syncfusion:DropDownMenuItem Header="{Binding Name}">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="{Binding Flag}"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
</DataTemplate>
</syncfusion:DropDownMenuGroup.ItemTemplate>
</syncfusion:DropDownMenuGroup>
</syncfusion:DropDownButtonAdv>
</Grid>
</window>
```

C#

```
public class Country
{
private string name;
public string Name
{
get
{
return name;
}
set
{
name = value;
}
```



```
}  
}  
private BitmapImage flag;  
public BitmapImage Flag  
{  
    get  
    {  
        return flag;  
    }  
    set  
    {  
        flag = value;  
    }  
}  
  
public class CountryViewModel  
{  
    private List<Country> dropDownItems;  
    public List<Country> DropDownItems  
    {  
        get  
        {  
            return dropDownItems;  
        }  
        set  
        {  
            dropDownItems = value;  
        }  
    }  
    public CountryViewModel()  
    {  
        DropDownItems = new List<Country>();  
        DropDownItems.Add(new Country()  
        {  
            Name = "India",  
            Flag = new BitmapImage(new Uri("/Images/india.png",  
            UriKind.RelativeOrAbsolute))  
        });  
        DropDownItems.Add(new Country()  
        {  
            Name = "France",  
            Flag = new BitmapImage(new Uri("/Images/france.png",  
            UriKind.RelativeOrAbsolute))  
        });  
        DropDownItems.Add(new Country()  
        {  
            Name = "Germany",  
            Flag = new BitmapImage(new Uri("/Images/germany.png",  
            UriKind.RelativeOrAbsolute))  
        });  
    }  
}
```

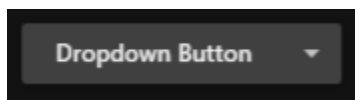


Note: View [sample](#) in GitHub.

Theme

Dropdown Button supports various built-in themes. Refer to the below links to apply themes for the Dropdown Button,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Data Binding in WPF Dropdown Button (DropDownButtonAdv)

Data binding provides an easier way to assign, visualize and interact with the collection of predefined data. The data binding can be achieved by populating the [DropDownMenuGroup.ItemsSource](#) property.

Creating model

Create a class that holds the model properties of the menu items. For example, `Country` class has been created with properties `Name` and `Flag`.

C#

```
public class Country
{
    private string name;
    public string Name
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
    private BitmapImage flag;
    public BitmapImage Flag
    {
        get
        {
            return flag;
        }
        set
        {
            flag = value;
        }
    }
}
```

```
}  
}
```

Creating view model

Create a class that populates the list of model object representing dropdown menu items. For example, `CountryViewModel` class has been created with property [DropDownItems](#) with return type `List<Country>`.

C#

```
public class CountryViewModel  
{  
    private List<Country> dropDownItems;  
    public List<Country> DropDownItems  
    {  
        get  
        {  
            return dropDownItems;  
        }  
        set  
        {  
            dropDownItems = value;  
        }  
    }  
    public CountryViewModel()  
    {  
        DropDownItems = new List<Country>();  
        DropDownItems.Add(new Country()  
        {  
            Name = "India",  
            Flag = new BitmapImage(new Uri("Images/india.png",  
            UriKind.RelativeOrAbsolute))  
        });  
        DropDownItems.Add(new Country()  
        {  
            Name = "France",  
            Flag = new BitmapImage(new Uri("Images/france.png",  
            UriKind.RelativeOrAbsolute))  
        });  
        DropDownItems.Add(new Country()  
        {  
            Name = "Germany",  
            Flag = new BitmapImage(new Uri("Images/germany.png",  
            UriKind.RelativeOrAbsolute))  
        });  
    }  
}
```

Bind data from view model

Bind the list of menu items to [DropDownMenuGroup.ItemsSource](#) property of [DropDownMenuGroup](#) and also set the `DataContext` with ViewModel instance. For example, `CountryViewModel` instance has been set as `DataContext`.

XML

```
<syncfusion:DropDownButtonAdv Label="Country"
    SmallIcon="Images\flagsmall.png" >
    <syncfusion:DropDownMenuGroup ItemsSource="{Binding DropDownItems}">
    <syncfusion:DropDownMenuGroup.ItemTemplate>
    <DataTemplate>
    <syncfusion:DropDownMenuItem Header="{Binding Name}">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="{Binding Flag}"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    </DataTemplate>
    </syncfusion:DropDownMenuGroup.ItemTemplate>
    </syncfusion:DropDownMenuGroup>
    </syncfusion:DropDownButtonAdv>
```

C#

```
public partial class MainWindow:Window
{
    public MainWindow()
    {
        InitializeComponent();
        this.DataContext = new CountryViewModel();
    }
}
```

Bind command from view model

Bind the command to [DropDownMenuItem.Command](#) property of [DropDownMenuItem](#). For example, `ClickCommand` has been bounded to `DropDownMenuItem`.

Note: For more information on Command Binding, please refer [Command Binding](#)

XML

```
<Window x:Class="Dropdown_Button_Data_Binding.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:Dropdown_Button_Data_Binding"
    xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
    mc:Ignorable="d"
    Title="MainWindow" Height="450" Width="800">
    <Window.DataContext>
    <local:CountryViewModel/>
    </Window.DataContext>
    <Grid VerticalAlignment="Center">
    <Grid.ColumnDefinitions>
    <ColumnDefinition Width="270"/>
    <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <CheckBox IsChecked="{Binding CanPerformAction}" Grid.Column="0"
        Content="Can perform action in dropdown menu items"/>
    <syncfusion:DropDownButtonAdv x:Name="dropdownButton" Label="Country"
        Grid.Column="1" SmallIcon="Images\flagsmall.png" >
```

```

<syncfusion:DropDownMenuGroup ItemsSource="{Binding DropDownItems}">
<syncfusion:DropDownMenuGroup.ItemTemplate>
<DataTemplate>
<syncfusion:DropDownMenuItem Header="{Binding Name}"
Command="{Binding DataContext.ClickCommand, Source={x:Reference
dropdownButton}}"
CommandParameter="{Binding .}">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="{Binding Flag}"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
</DataTemplate>
</syncfusion:DropDownMenuGroup.ItemTemplate>
</syncfusion:DropDownMenuGroup>
</syncfusion:DropDownButtonAdv>
</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)
        {

```

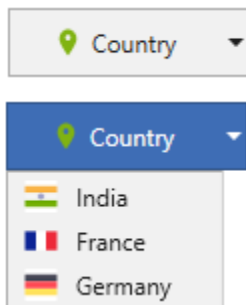
```

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 CountryViewModel: NotificationObject
{
    private List<Country> dropDownItems;
    private bool _canperformaction = true;
    public List<Country> DropDownItems
    {
        get
        {
            return dropDownItems;
        }
        set
        {
            dropDownItems = value;
        }
    }
    public bool CanPerformAction
    {
        get
        {
            return _canperformaction;
        }
    }
}

```

```
set
{
    _canperformaction = value;
    this.ClickCommand.RaiseCanExecuteChanged();
    this.RaisePropertyChanged("CanPerformAction");
}
}

public CountryViewModel()
{
    DropDownItems = new List<Country>();
    ClickCommand = new DelegateCommand<object>(ClickAction,
    CanPerformClickAction);
    DropDownItems.Add(new Country()
    {
        Name = "India",
        Flag = new BitmapImage(new Uri("Images/india.png",
        UriKind.RelativeOrAbsolute))
    });
    DropDownItems.Add(new Country()
    {
        Name = "France",
        Flag = new BitmapImage(new Uri("Images/france.png",
        UriKind.RelativeOrAbsolute))
    });
    DropDownItems.Add(new Country()
    {
        Name = "Germany",
        Flag = new BitmapImage(new Uri("Images/germany.png",
        UriKind.RelativeOrAbsolute))
    });
    private bool CanPerformClickAction(object parameter)
    {
        return CanPerformAction;
    }
    public DelegateCommand<object> ClickCommand { get; set; }
    private void ClickAction(object parameter)
    {
        Country country = (Country)parameter;
        MessageBox.Show(country.Name + " has been clicked");
    }
}
```



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

Command Binding in WPF Dropdown Button (DropDownButtonAdv)

The command and command parameter properties allow to execute any action on clicking the dropdown menu items.

- **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 x:Class="DropDown_Button_Command_Binding.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:Button_Sample"
xmlns:Syncfusion="http://schemas.microsoft.com/netfx/2009/xaml/presentation"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
xmlns:syncfusionskin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:DropDownViewModel />
</Window.DataContext>
<Grid VerticalAlignment="Center">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="270"/>
<ColumnDefinition Width="*/>
</Grid.ColumnDefinitions>
<CheckBox IsChecked="{Binding CanPerformAction}" Content="Can perform action
in dropdown menu item"/>
<syncfusion:DropDownButtonAdv Label="Country" SizeMode="Large"
LargeIcon="Images\flaglarge.png" HorizontalAlignment="Center"
VerticalAlignment="Center" Grid.Column="1" Height="58" Width="89">
<syncfusion:DropDownMenuGroup>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="India"
Command="{Binding ClickCommand}" CommandParameter="India">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/india.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="France"
Command="{Binding ClickCommand}" CommandParameter="France" >
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/france.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Germany"
Command="{Binding ClickCommand}" CommandParameter="Germany" >
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/germany.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
```



```

<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Canada"
Command="{Binding ClickCommand}" CommandParameter="Canada" >
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/Canada.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="China"
Command="{Binding ClickCommand}" CommandParameter="China" >
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/china.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
</syncfusion:DropDownMenuGroup >
</syncfusion:DropDownButtonAdv>
</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
}

class DropDownViewModel: NotificationObject
{
    private bool _canperformaction = true;
    public DropDownViewModel()
    {
        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() + " dropdown menu item has been
    clicked");
}
}

```

Note: View [sample](#) in GitHub. This sample showcases how to provide command binding for DropDownButtonAdv control.

Dropdown Menu Items in WPF Dropdown Button (DropDownButtonAdv)

Setting icon for dropdown menu items

The icon option helps to provide pictorial representation of the dropdown menu item. One can apply the icon by setting the [Icon](#) property value to an image source.

XML

```

<syncfusion:DropDownButtonAdv Label="Country" x:Name="dropdownbutton"
DropDirection="BottomRight" SizeMode="Normal"
SmallIcon="Images\country.png">
<syncfusion:DropDownMenuGroup>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="India">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images\india.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="France"/>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Germany"/>
</syncfusion:DropDownMenuGroup>
</syncfusion:DropDownButtonAdv>

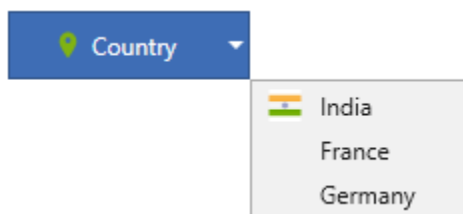
```

C#

```

DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
DropDownMenuGroup menu = new DropDownMenuGroup();
DropDownMenuItem Item1 = new DropDownMenuItem() { Header="India", Icon=new
BitmapImage(new Uri("Images\india.png", UriKind.RelativeOrAbsolute)),
HorizontalAlignment="Left"};
DropDownMenuItem Item2 = new DropDownMenuItem() { Header = "France",
HorizontalAlignment="Left" };
DropDownMenuItem Item3 = new DropDownMenuItem() { Header = "Germany",
HorizontalAlignment="Left" };
menu.Items.Add(Item1);
menu.Items.Add(Item2);
menu.Items.Add(Item3);
dropdownbutton.Content = menu;
dropdownbutton.Label = "Country";
dropdownbutton.DropDirection = DropDirection.BottomRight;
dropdownbutton.SizeMode = SizeMode.Normal;
dropdownbutton.SmallIcon = new BitmapImage(new Uri("Images\country.png",
UriKind.RelativeOrAbsolute));

```



Setting icon bar visibility

The icon bar option helps to enable/disable the vertical bar next to the Dropdown menu item icon. One can change the icon bar visibility by setting the [IconBarEnabled](#) property to **true** or **false**.

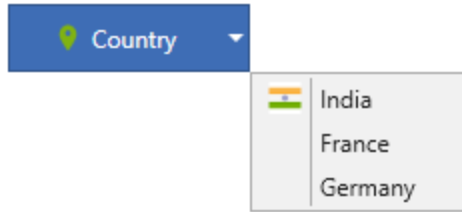
Note: The default value of [IconBarEnabled](#) is **false**.

XML

```
<syncfusion:DropDownButtonAdv Label="Country" x:Name="dropdownbutton"
DropDirection="BottomRight" SizeMode="Normal"
SmallIcon="Images\country.png">
  <syncfusion:DropDownMenuGroup IconBarEnabled="True">
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="India">
      <syncfusion:DropDownMenuItem.Icon>
        <Image Source="Images\india.png"/>
      </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="France"/>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Germany"/>
  </syncfusion:DropDownMenuGroup>
</syncfusion:DropDownButtonAdv>
```

C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
DropDownMenuGroup menu = new DropDownMenuGroup();
DropDownMenuItem Item1 = new DropDownMenuItem() { Header="India", Icon=new
BitmapImage(new Uri("Images\india.png", UriKind.RelativeOrAbsolute)),
HorizontalAlignment="Left"};
DropDownMenuItem Item2 = new DropDownMenuItem() { Header = "France",
HorizontalAlignment="Left" };
DropDownMenuItem Item3 = new DropDownMenuItem() { Header = "Germany",
HorizontalAlignment="Left" };
menu.Items.Add(Item1);
menu.Items.Add(Item2);
menu.Items.Add(Item3);
menu.IconBarEnabled =true;
dropdownbutton.Content = menu;
dropdownbutton.Label = "Country";
dropdownbutton.SizeMode = SizeMode.Normal;
dropdownbutton.DropDirection = DropDirection.BottomRight;
dropdownbutton.SmallIcon = new BitmapImage(new Uri("Images\country.png",
UriKind.RelativeOrAbsolute));
```



Setting scrollbar visibility

The dropdown menu group supports built-in scrollbar to show large number of menu items in a compact view. One can enable the visibility of scroll bar by setting the [ScrollBarVisibility](#) property to **Visible**.

XML

```
<syncfusion:DropDownButtonAdv Label="Country" DropDirection="BottomRight"
x:Name="dropdownbutton" SizeMode="Normal" SmallIcon="Images\country.png">
<syncfusion:DropDownMenuGroup MaxHeight="111" ScrollBarVisibility="Visible">
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="India">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/india.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="France">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Image\france.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Germany" >
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Image\germany.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Canada">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Image\canada.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="China">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Image\china.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="United
States"/>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Italy"/>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Japan"/>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Spain"/>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Pakistan"/>
</syncfusion:DropDownMenuGroup>
</syncfusion:DropDownButtonAdv>
```

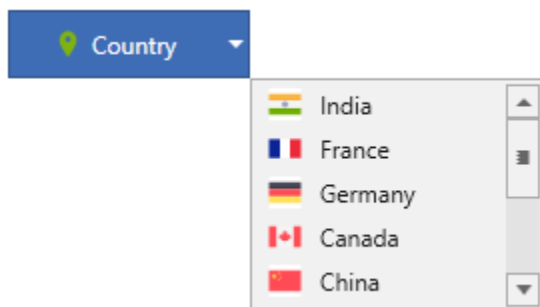
C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
DropDownMenuGroup menu = new DropDownMenuGroup();
```

```

DropDownMenuItem Item1 = new DropDownMenuItem() { Header="India",Icon=new
BitmapImage(new Uri("Images\india.png", UriKind.RelativeOrAbsolute)),
HorizontalAlignment="Left"};
DropDownMenuItem Item2 = new DropDownMenuItem() { Header ="France", Icon=new
BitmapImage(new Uri("Images\france.png", UriKind.RelativeOrAbsolute)),
HorizontalAlignment="Left"};
DropDownMenuItem Item3 = new DropDownMenuItem() { Header ="Germany",
Icon=new BitmapImage(new Uri("Images\germany.png",
UriKind.RelativeOrAbsolute)), HorizontalAlignment="Left"};
DropDownMenuItem Item4 = new DropDownMenuItem() { Header ="Canada", Icon=new
BitmapImage(new Uri("Images\canada.png", UriKind.RelativeOrAbsolute)),
HorizontalAlignment="Left"};
DropDownMenuItem Item5 = new DropDownMenuItem() { Header ="China", Icon=new
BitmapImage(new Uri("Images\china.png", UriKind.RelativeOrAbsolute)),
HorizontalAlignment="Left"};
DropDownMenuItem Item6 = new DropDownMenuItem() { Header ="United State",
HorizontalAlignment="Left"};
DropDownMenuItem Item7 = new DropDownMenuItem() { Header ="Italy",
HorizontalAlignment="Left"};
DropDownMenuItem Item8 = new DropDownMenuItem() { Header ="Japan",
HorizontalAlignment="Left"};
DropDownMenuItem Item9 = new DropDownMenuItem() { Header ="Spain",
HorizontalAlignment="Left"};
DropDownMenuItem Item10 = new DropDownMenuItem() { Header ="Pakistan",
HorizontalAlignment="Left"};
menu.Items.Add(Item1);
menu.Items.Add(Item2);
menu.Items.Add(Item3);
menu.Items.Add(Item4);
menu.Items.Add(Item5);
menu.Items.Add(Item6);
menu.Items.Add(Item7);
menu.Items.Add(Item8);
menu.Items.Add(Item9);
menu.Items.Add(Item10);
menu.MaxHeight=111;
menu.ScrollBarVisibility = ScrollBarVisibility.Visible;
dropdownbutton.Content = menu;
dropdownbutton.Label = "Country";
dropdownbutton.SizeMode = SizeMode.Normal;
dropdownbutton.DropDirection = DropDirection.BottomRight;
dropdownbutton.SmallIcon = new BitmapImage(new Uri("Images\country.png",
UriKind.RelativeOrAbsolute));

```



Resizing dropdown menu

The dropdown menu group height can be increased or decreased using the resizing gripper. One can enable the resizing behavior by setting the [IsResizable](#) property to **true**.

XML

```
<syncfusion:DropDownButtonAdv Label="Country" x:Name="dropdownbutton"
    SmallIcon="Images\country.png">
    <syncfusion:DropDownMenuGroup IsResizable="True">
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="India">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="Images\india.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="France">
    <syncfusion:DropDownMenuItem.Icon >
    <Image Source="Images\france.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Germany">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="Images\germany.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    </syncfusion:DropDownMenuGroup>
    </syncfusion:DropDownButtonAdv>
```

C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
DropDownMenuGroup menu = new DropDownMenuGroup();
DropDownMenuItem Item1 = new DropDownMenuItem() { Header = "India", Icon =new
BitmapImage(new Uri("Images\india.png", UriKind.RelativeOrAbsolute)),
HorizontalAlignment="Left"};
DropDownMenuItem Item2 = new DropDownMenuItem() { Header = "France", Icon
=new BitmapImage(new Uri("Images\france.png", UriKind.RelativeOrAbsolute)),
HorizontalAlignment="Left"};
DropDownMenuItem Item3 = new DropDownMenuItem() { Header = "Germany", Icon
=new BitmapImage(new Uri("Images\germany.png", UriKind.RelativeOrAbsolute)),
HorizontalAlignment="Left"};
menu.Items.Add(Item1);
menu.Items.Add(Item2);
menu.Items.Add(Item3);
menu.IsResizable = true;
dropdownbutton.Content = menu;
dropdownbutton.Label = "Country";
dropdownbutton.DropDirection = DropDirection.BottomRight;
dropdownbutton.SmallIcon = new BitmapImage(new Uri("Images\country.png",
UriKind.RelativeOrAbsolute));
```



Checkable dropdown menu items

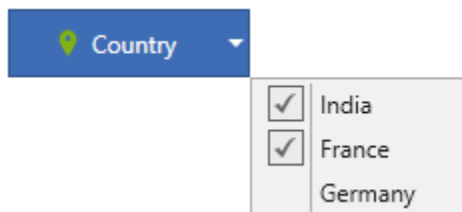
The checkable option helps to check/uncheck the dropdown menu item on selection by setting the [IsCheckable](#) property to **true**.

XML

```
<syncfusion:DropDownButtonAdv Label="Country" DropDirection="BottomRight"
x:Name="dropdownbutton" SizeMode="Normal" SmallIcon="Images\country.png">
  <syncfusion:DropDownMenuGroup>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="India"
IsChecked="True" IsCheckable="True"/>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="France"
IsChecked="True" IsCheckable="True"/>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Germany"/>
  </syncfusion:DropDownMenuGroup>
</syncfusion:DropDownButtonAdv>
```

C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
DropDownMenuGroup menu = new DropDownMenuGroup();
DropDownMenuItem Item1 = new DropDownMenuItem() { Header="India",
IsChecked=true, IsCheckable=true, HorizontalAlignment="Left"};
DropDownMenuItem Item2 = new DropDownMenuItem() { Header ="France",
IsChecked=true, IsCheckable=true, HorizontalAlignment="Left"};
DropDownMenuItem Item3 = new DropDownMenuItem() { Header ="Germany",
HorizontalAlignment="Left"};
menu.Items.Add(Item1);
menu.Items.Add(Item2);
menu.Items.Add(Item3);
dropdownbutton.Content = menu;
dropdownbutton.Label = "Country";
dropdownbutton.SizeMode = SizeMode.Normal;
dropdownbutton.DropDirection = DropDirection.BottomRight;
dropdownbutton.SmallIcon = new BitmapImage(new Uri("Images\country.png",
UriKind.RelativeOrAbsolute));
```



Note: View [sample](#) in GitHub. This sample showcases how to add custom dropdown menu items and handle visibility of custom items icon bar in dropdown button control.

Adding custom dropdown menu items

The dropdown menu group has option to load custom items apart from actual dropdown menu items. One can populate the custom items using the [MoreItems](#) property.

Note: The **MoreItems** property has return type `ObservableCollection<UIElement>`, so it can accept any `UIElement` as its child items.

XML

```
<Window x:Class="Dropdown_Button_Custom_Items.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:Dropdown_Button_Custom_Items"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:ColorViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:DropDownButtonAdv Label="Colors" x:Name="dropdownbutton"
SizeMode="Normal" SmallIcon="Images\colors.png">
<syncfusion:DropDownMenuGroup MoreItems="{Binding Items}"
IconBarEnabled="True" IsMoreItemsIconTrayEnabled="False">
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Black">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images\black.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Orange">
<syncfusion:DropDownMenuItem.Icon >
<Image Source="Images\orange.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Red">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images\red.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
</syncfusion:DropDownMenuGroup>
</syncfusion:DropDownButtonAdv>
</Grid>
</Window>
```

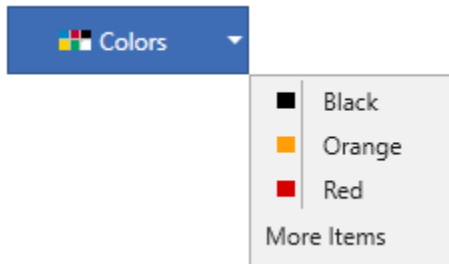
C#

```
using Syncfusion.Windows.Shared;
using System;
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Controls;
```

```

namespace Dropdown_Button_Custom_Items
{
    public class ColorViewModel : NotificationObject
    {
        public ObservableCollection<UIElement> items = new
        ObservableCollection<UIElement>();
        public ObservableCollection<UIElement> Items
        {
            get { return items; }
            set { items = value; RaisePropertyChanged("Items"); }
        }
        public ColorViewModel()
        {
            Items.Add(new Label() { Content = "More Items" });
        }
    }
}

```



Setting icon bar visibility for custom dropdown menu items

The custom dropdown menu items icon visibility can be enabled/disabled by setting the [IsMoreItemsIconTrayEnabled](#) property either to **true** or **false**.

XML

```

<Window x:Class="Dropdown_Button_Custom_Items.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:Dropdown_Button_Custom_Items"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
    <Window.DataContext>
        <local:ColorViewModel/>
    </Window.DataContext>
    <Grid>
        <syncfusion:DropDownButtonAdv Label="Colors" x:Name="dropdownbutton"
SizeMode="Normal" SmallIcon="Images\colors.png">
            <syncfusion:DropDownMenuItem MoreItems="{Binding Colors}"
IconBarEnabled="True" IsMoreItemsIconTrayEnabled="True">
                <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Black">
                    <syncfusion:DropDownMenuItem.Icon>
                        <Image Source="Images\black.png"/>
                    </syncfusion:DropDownMenuItem.Icon>
                </syncfusion:DropDownMenuItem>
            </syncfusion:DropDownMenuItem>
        </syncfusion:DropDownButtonAdv>
    </Grid>

```

```

<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Orange">
<syncfusion:DropDownMenuItem.Icon >
<Image Source="Images\orange.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Red">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images\red.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
</syncfusion:DropDownMenuGroup>
</syncfusion:DropDownButtonAdv>
</Grid>
</Window>

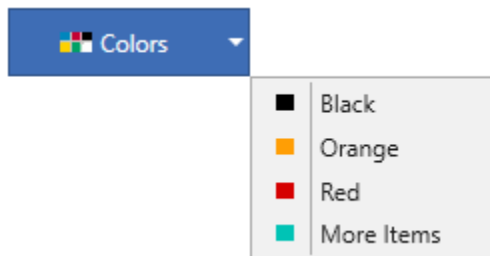
```

C#

```

using Syncfusion.Windows.Shared;
using Syncfusion.Windows.Tools.Controls;
using System;
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Media.Imaging;
public class ColorViewModel : NotificationObject
{
    public ObservableCollection<UIElement> color = new
    ObservableCollection<UIElement>();
    public ObservableCollection<UIElement> Colors
    {
        get { return color; }
        set { color = value; RaisePropertyChanged("Colors"); }
    }
    public ColorViewModel()
    {
        Colors.Add(new DropDownMenuItem() { Header = "More Items", Icon = new
        Image() { Source = new BitmapImage(new Uri("/Images/skyblue.png",
        UriKind.RelativeOrAbsolute)) } });
    }
}

```



Note: View [sample](#) in GitHub. This sample showcases how to add custom dropdown menu items and handle visibility of custom items icon bar in dropdown button control.

Dropdown Direction in WPF Dropdown Button (DropDownButtonAdv)

Dropdown direction is used to change the position of the popup being loaded while clicking the dropdown arrow. The direction can be changed using the [DropDirection](#) enumeration.

The [DropDirection](#) enumeration comprises of following values:

- Left
- Right
- BottomLeft
- BottomRight
- TopLeft
- TopRight

Note: The default value is **BottomLeft**.

XML

```
<syncfusion:DropDownButtonAdv DropDirection="BottomLeft"
    SmallIcon="images\country.png" Label="Country">
    <syncfusion:DropDownMenuGroup>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="India">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="images\india.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="France">
    <syncfusion:DropDownMenuItem.Icon >
    <Image Source="images\france.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Germany">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="images\germany.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Canada">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="images\canada.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="China">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="images\china.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    </syncfusion:DropDownMenuGroup>
    </syncfusion:DropDownButtonAdv>
```

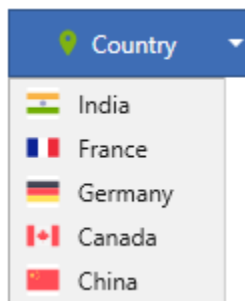
C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
dropdownbutton.Label = "Country";
dropdownbutton.DropDirection = DropDirection.BottomLeft;
dropdownbutton.SmallIcon = new BitmapImage(new Uri("images\country.png"));
DropDownMenuGroup menu = new DropDownMenuGroup();
```

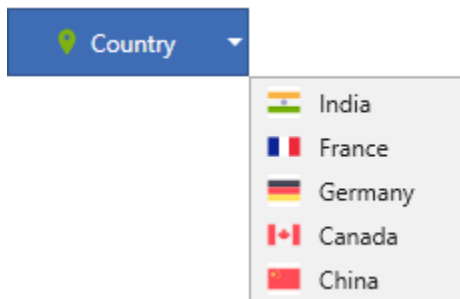
```

DropDownMenuItem Item1 = new DropDownMenuItem() { Header = "India", Icon =new
BitmapImage(new Uri("images\india.png")), HorizontalAlignment="Left"};
DropDownMenuItem Item2 = new DropDownMenuItem() { Header = "France", Icon
=new BitmapImage(new Uri("images\france.png")), HorizontalAlignment="Left"};
DropDownMenuItem Item3 = new DropDownMenuItem() { Header = "Germany", Icon
=new BitmapImage(new Uri("images\germany.png")),
HorizontalAlignment="Left"};
DropDownMenuItem Item4 = new DropDownMenuItem() { Header = "Canada", Icon
=new BitmapImage(new Uri("images\canada.png")), HorizontalAlignment="Left"};
DropDownMenuItem Item5 = new DropDownMenuItem() { Header = "China", Icon =new
BitmapImage(new Uri("images\china.png")), HorizontalAlignment="Left"};
menu.Items.Add(Item1);
menu.Items.Add(Item2);
menu.Items.Add(Item3);
menu.Items.Add(Item4);
menu.Items.Add(Item5);
dropdownbutton.Content = menu;

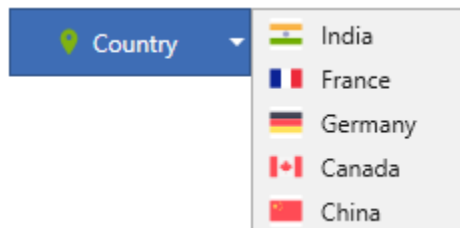
```



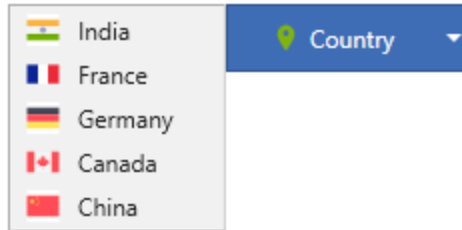
Drop Direction - BottomLeft



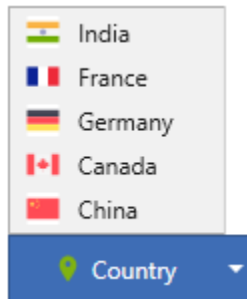
Drop Direction - BottomRight



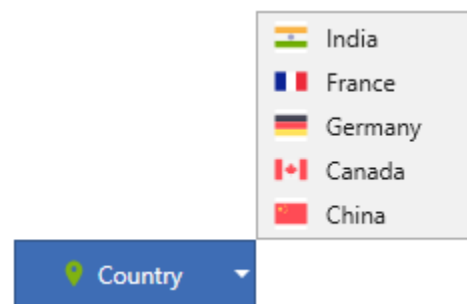
Drop Direction - Right



Drop Direction - Left



Drop Direction - TopLeft



Drop Direction - TopRight

Multiline Text in WPF Dropdown Button (DropDownButtonAdv)

Multiline text support is used to render text content of the Dropdown Button control in multiple lines for precise view. One can apply the multiline text by using the [IsMultiLine](#) property.

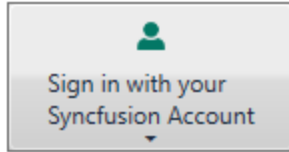
Note: This property is only applicable for large size mode of the Dropdown Button.

XML

```
<syncfusion:DropDownButtonAdv Label="Sign in with your Syncfusion Account"
    LargeIcon="image\employee.png" SizeMode="Large" IsMultiLine="True"/>
```

C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
dropdownbutton.SizeMode = SizeMode.Large;
dropdownbutton.Label = "Sign in with your Syncfusion Account";
dropdownbutton.IsMultiLine = true;
dropdownbutton.LargeIcon = new BitmapImage(new Uri("image\employee.png"));
```



Events in WPF Dropdown Button (DropDownButtonAdv)

The Dropdown Button control comprises of various pre-defined events to perform any required action that are illustrated below.

DropDownOpening

The event occurs before opening the dropdown menu popup and any action can be handled in the respective event handler.

XML

```
<syncfusion:DropDownButtonAdv x:Name="dropdownbutton"
    DropDownOpening="dropdownbutton_DropDownOpening"/>
```

C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
dropdownbutton.DropDownOpening +=new
CancelEventHandler(dropdownbutton_DropDownOpening);
private void dropdownbutton_DropDownOpening(object sender,
System.ComponentModel.CancelEventArgs e)
{
}
```

DropDownOpened

The event occurs after opening the dropdown menu popup and any action can be handled in respective event handler.

XML

```
<syncfusion:DropDownButtonAdv x:Name="dropdownbutton"
    DropDownOpened="dropdownbutton_DropDownOpened"/>
```

C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
dropdownbutton.DropDownOpened +=new
RoutedEventHandler(dropdownbutton_DropDownOpened);
private void dropdownbutton_DropDownOpened(object sender, RoutedEventArgs e)
{
}
```

DropDownClosing

The event occurs before closing the dropdown menu popup and any action can be handled in respective event handler.

XML

```
<syncfusion:DropDownButtonAdv x:Name="dropdownbutton"
DropDownClosing="dropdownbutton_DropDownClosing"/>
```

C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
dropdownbutton.DropDownClosing +=new
CancelEventHandler(dropdownbutton_DropDownClosing);
private void dropdownbutton_DropDownClosing(object sender,
System.ComponentModel.CancelEventArgs e)
{
}
```

DropDownClosed

The event occurs after closing the dropdown menu popup and any action can be handled in respective event handler.

XML

```
<syncfusion:DropDownButtonAdv x:Name="dropdownbutton"
DropDownClosed="dropdownbutton_DropDownClosed"/>
```

C#

```
DropDownButtonAdv dropdownbutton = new DropDownButtonAdv();
dropdownbutton.DropDownClosed +=new
RoutedEventHandler(dropdownbutton_DropDownClosed);
private void dropdownbutton_DropDownClosed(object sender, RoutedEventArgs e)
{
}
```

Events for dropdown menu items**Click**

The event occurs when the dropdown menu item is clicked and any action can be handled in respective event handler.

XML

```
<syncfusion:DropDownMenuItem x:Name="dropDownMenuItem"
Click="dropDownMenuItem_Click/>
```

C#

```
DropDownMenuItem dropDownMenuItem = new DropDownMenuItem();
dropDownMenuItem.Click +=new RoutedEventHandler(dropDownMenuItem_Click);
private void dropDownMenuItem_Click(object sender, RoutedEventArgs e)
{
}
```

IsCheckedChanged

The event occurs when the dropdown menu item is checked or unchecked, that is, only when [IsCheckable](#) property is set to **true**. Any action can be handled in the respective event handler.

XML

```
<syncfusion:DropDownMenuItem x:Name="dropDownMenuItem" IsCheckable="True"
IsCheckedChanged="DropDownMenuItem_IsCheckedChanged"/>
```

C#

```
DropDownMenuItem dropDownMenuItem = new DropDownMenuItem();
dropDownMenuItem.IsCheckable=true;
dropDownMenuItem.IsCheckedChanged +=new
RoutedEventHandler(dropDownMenuItem_IsCheckedChanged);
private void dropDownMenuItem_IsCheckedChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
}
```

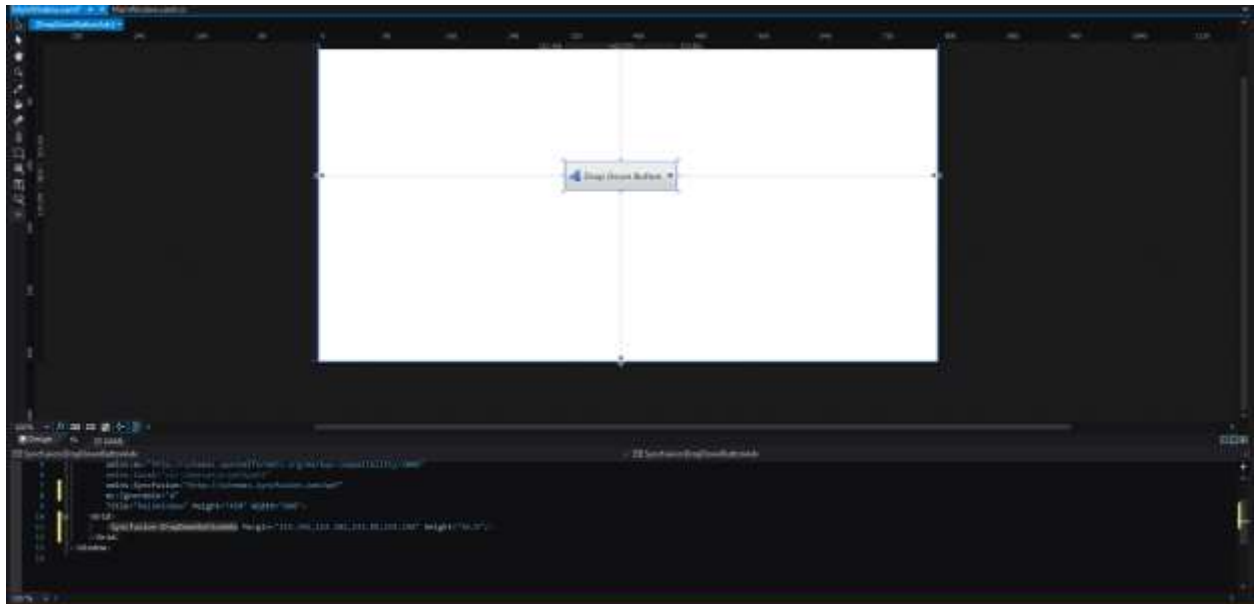
Styles and Templates in WPF Dropdown Button (DropDownButtonAdv)

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 Dropdown Button control. In addition, one can edit the structure of the Dropdown Button control using Blend and Visual Studio that helps to customize their appearances.

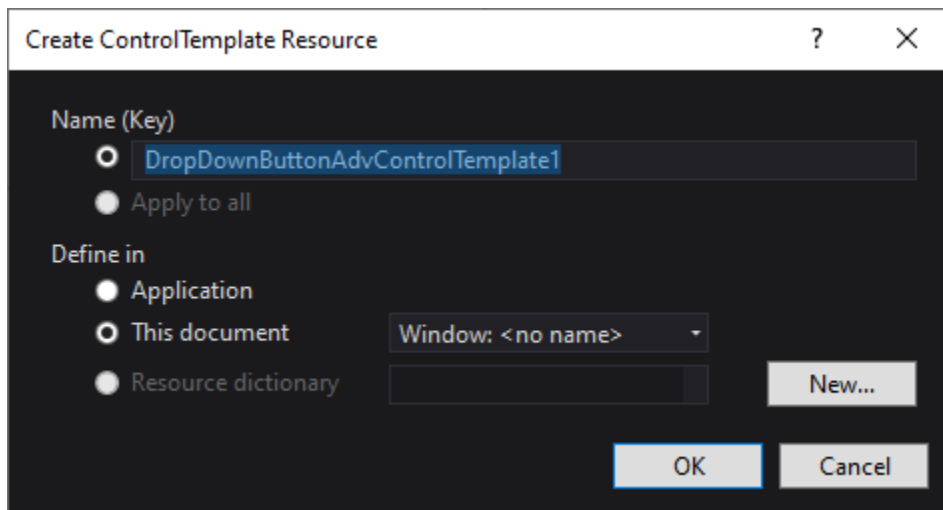
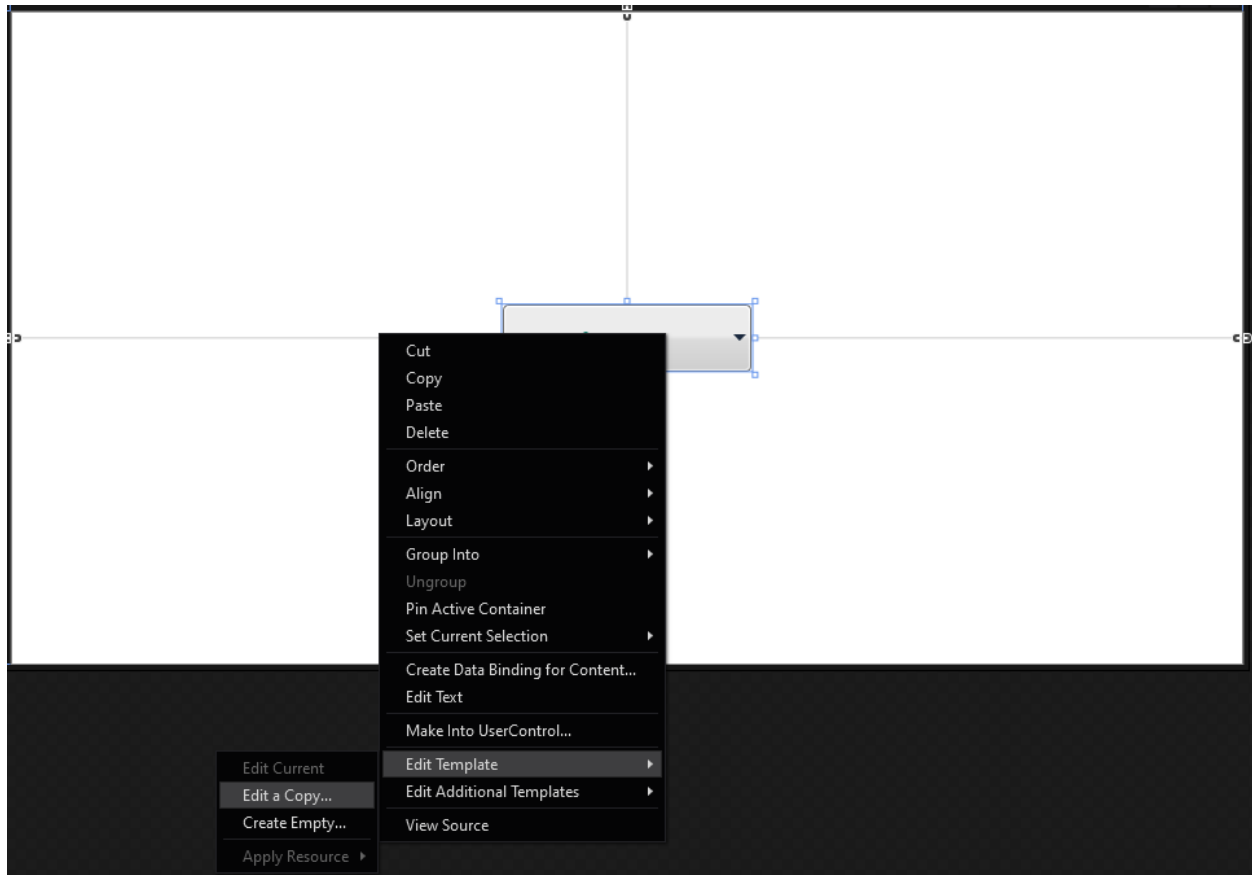
Edit appearance in Expression Blend

- Open the application in Expression Blend.
- Select the Dropdown Button control from the window.



- Right click on the Dropdown 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 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 Dropdown Button control template 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 Dropdown Button control template. 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.

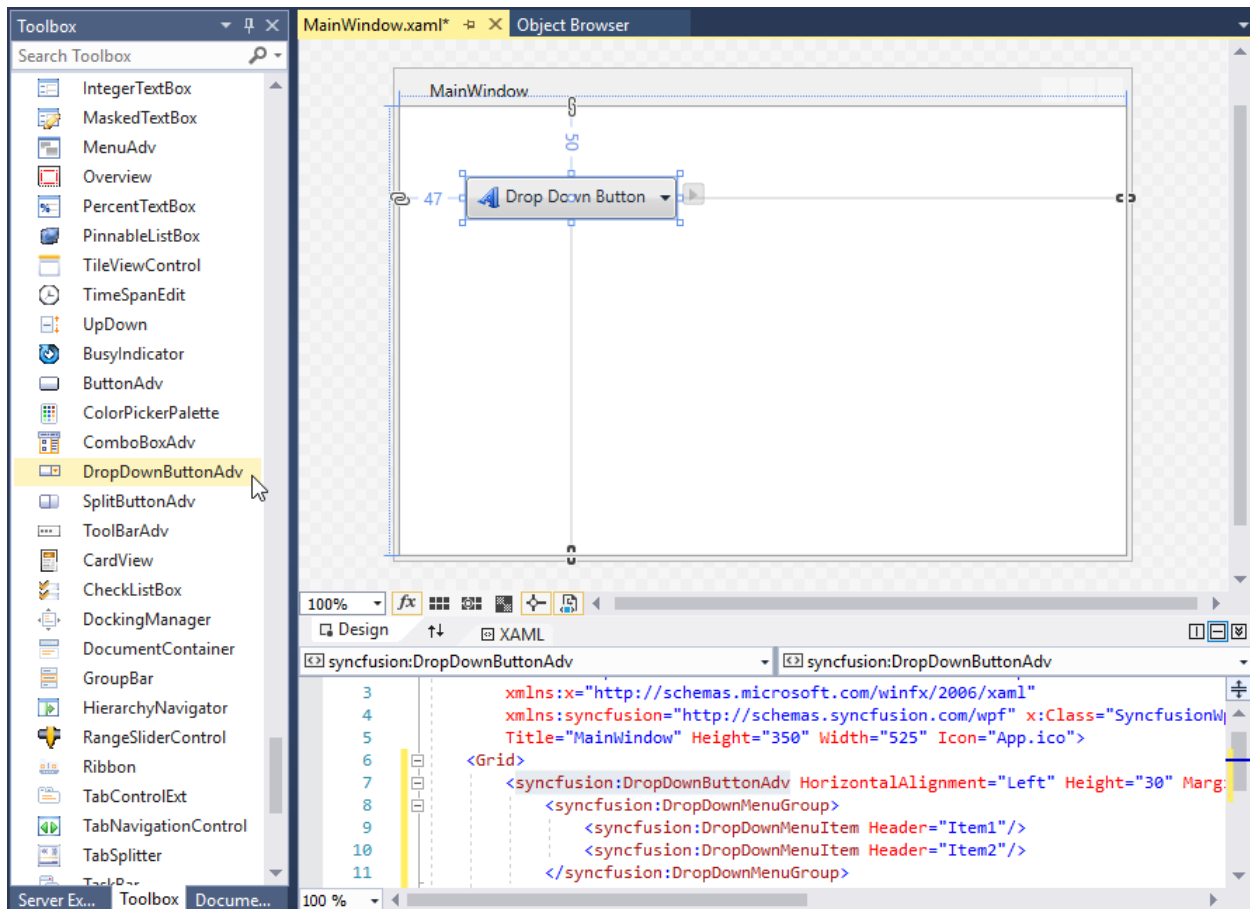


```
<Window.Resources>
    <ControlTemplate x:Key="DropDownButtonAdvControlTemplate1" TargetType="{x:Type syncfusion:DropDownButtonAdv}"...
</Window.Resources>
```

Dropdown Button control edited in Expression Blend

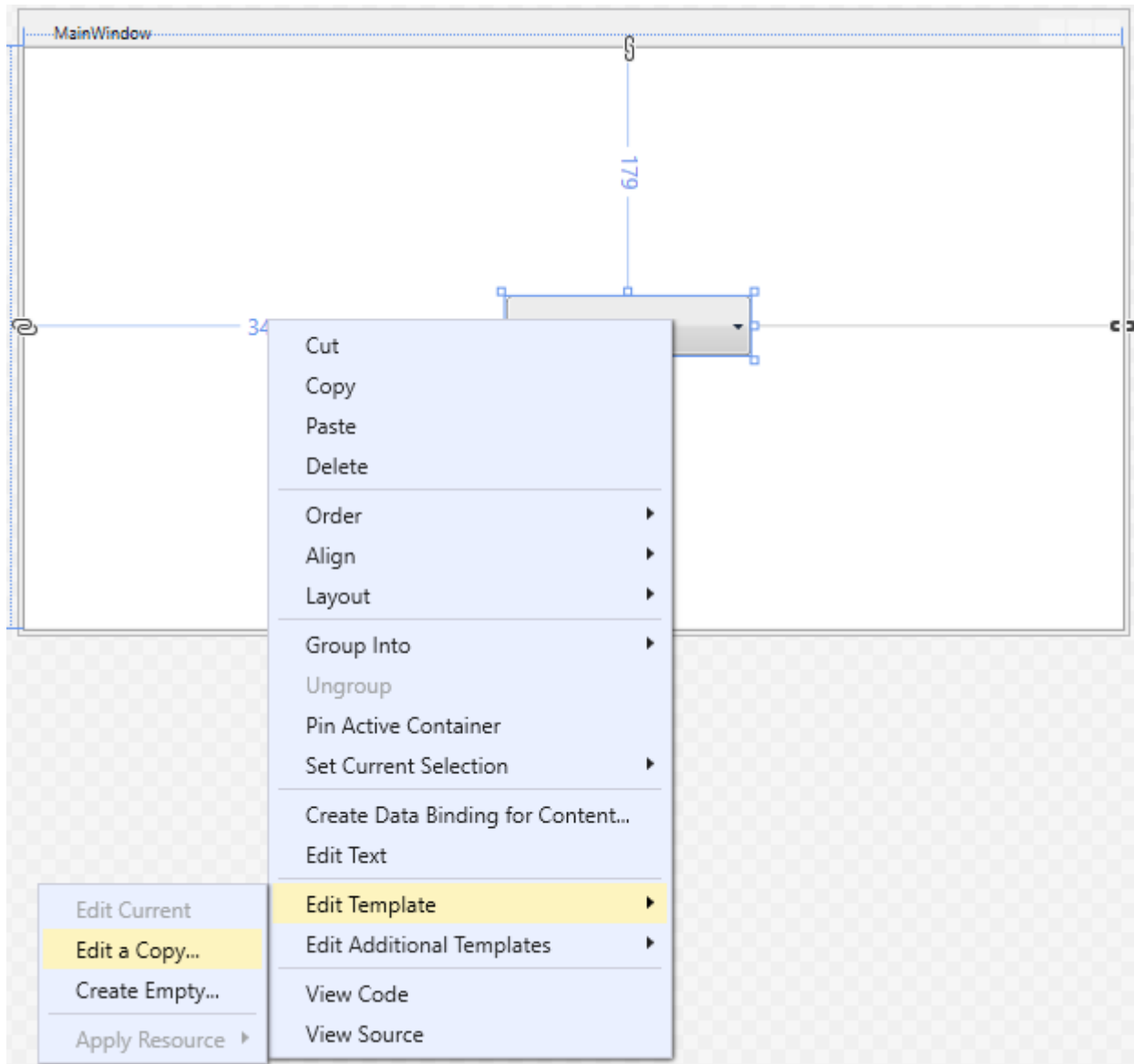
Edit appearance in Visual Studio

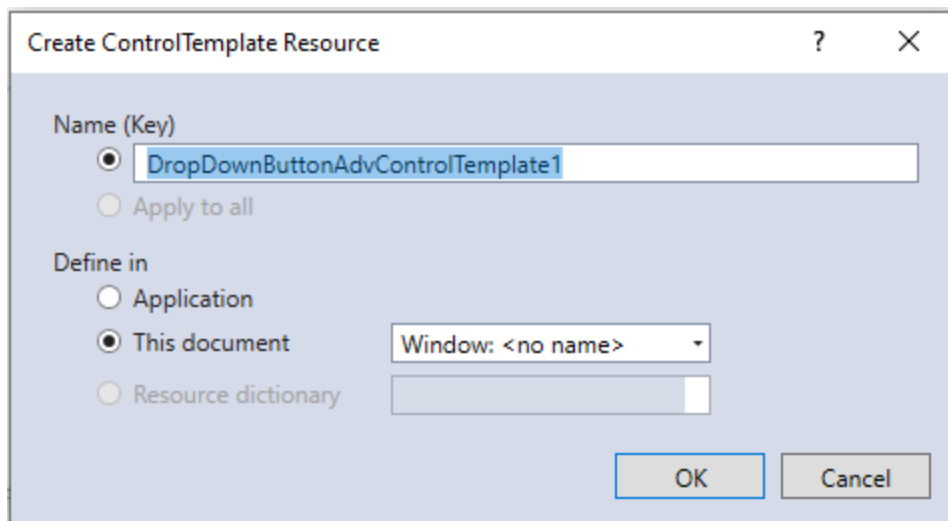
- Open the application in Visual Studio.
- Open design view and select the Dropdown Button control. Now right click on the Dropdown Button control and you can see some menu options showing up.



- 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.



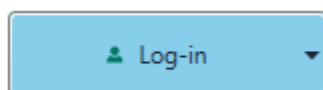


The **Create ControlTemplate Resource** dialog allows to enter the name or change the control template name, as well as choose the location for the template. When **OK** is pressed, the Dropdown Button control template is generated in the **Resource** section. The generated XAML can be edited in XAML view.

- **Create Empty...** - Creates an empty Dropdown 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="DropDownButtonAdvControlTemplate1" TargetType="{x:Type syncfusion:DropDownButtonAdv}"...>
</Window.Resources>
```

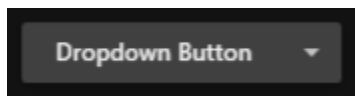


Dropdown Button control edited in Visual Studio

Themes in WPF Dropdown Button (DropDownButtonAdv)

Dropdown Button supports various built-in themes. Refer to the below links to apply themes for the Dropdown Button,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



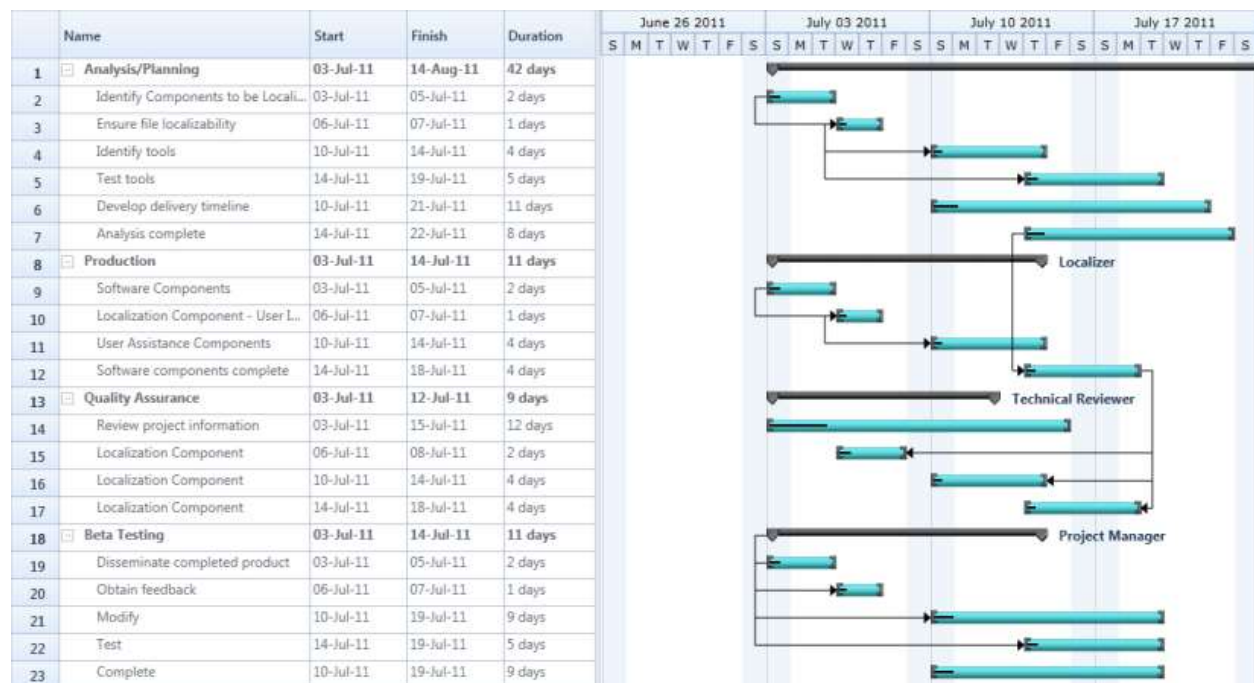
Gantt

WPF Gantt control Overview

Essential Gantt for WPF is a MS Project-like Project Viewer with built-in grid, schedule and resource assignment constraints. It is designed to assist project managers in developing plans, assigning resources to the task, tracking task progress and so on. Some of the key features of Gantt control are: drag support for increasing and decreasing the Start and Finish Date of the Task , Drag and Drop support for the task within a row and automatic data synchronization between Grid and Chart. This also supports to import and export the Task Details as XML file.

Real World Scenario

Research scholars, IT companies or any organization that following work breakdown structure can use Gantt control to schedule and track their tasks/ activities. This helps tracking the progress of an assignment. By tracking the progress one can change or reschedule the plan to achieve the goal.

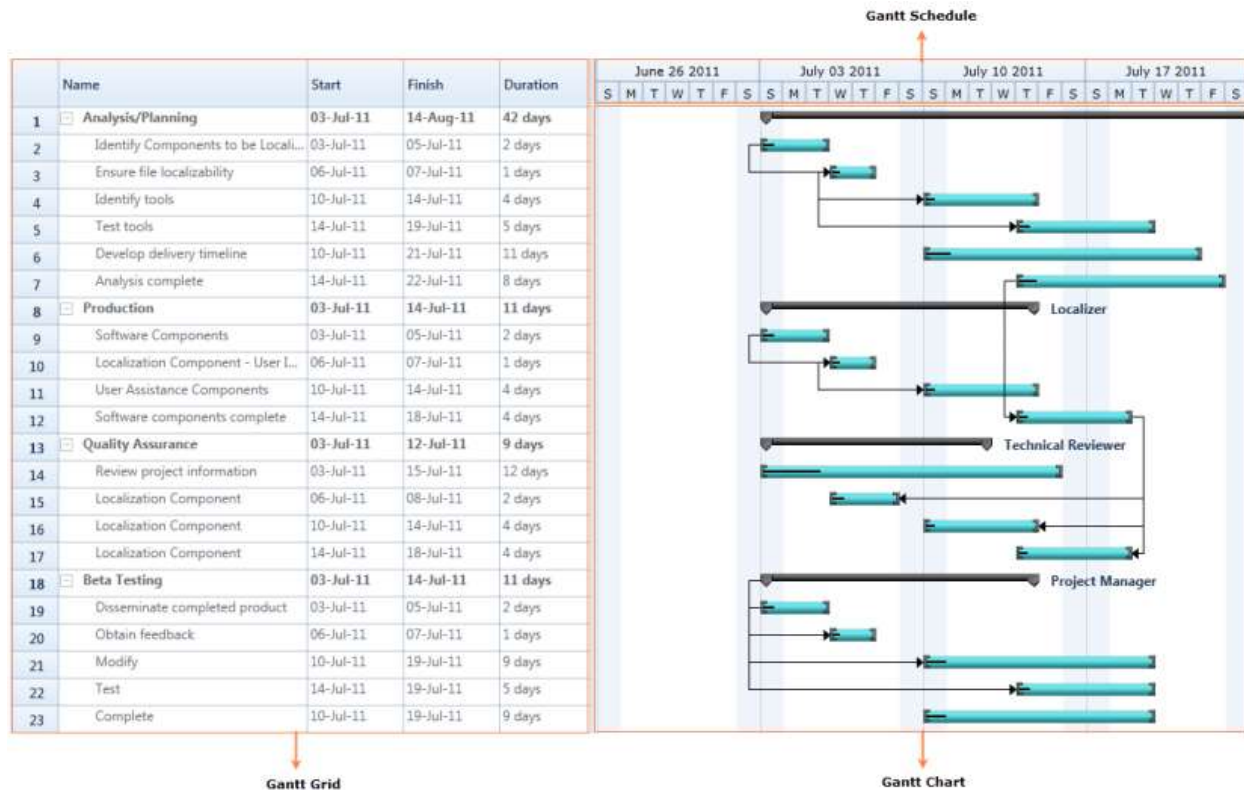


Getting Started with WPF Gantt

Appearance and structure of Gantt

Gantt control is composed of three controls. They are:

- GanttGrid
- ScheduleHeader
- GanttChartVisualControl



Gantt grid

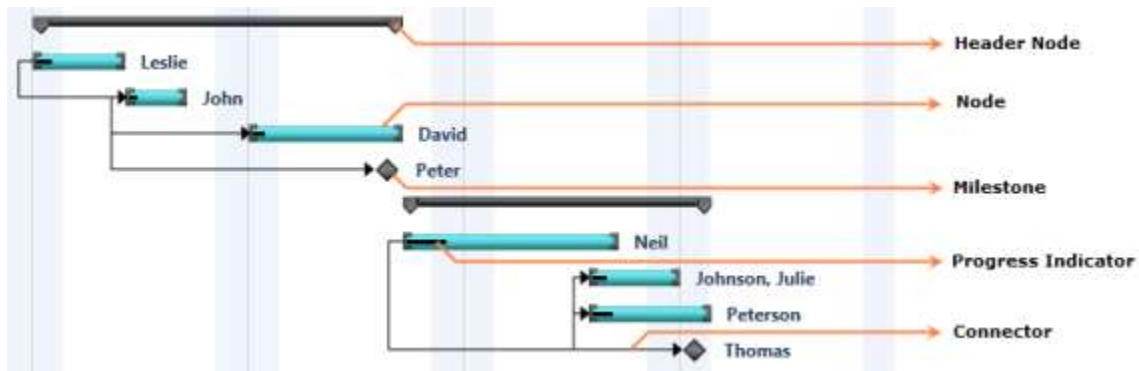
Gantt Grid is a table view control which displays the scheduled tasks/activities of the project with its hierarchy. You can edit the fields of the bounded tasks using this grid.

	Name	Start	Finish	Duration	Header
1	Analysis/Planning	03-Jul-11	14-Aug-11	42 days	Parent Task
2	Identify Components to be Locali...	03-Jul-11	05-Jul-11	2 days	Child Tasks
3	Ensure file localizability	06-Jul-11	07-Jul-11	1 days	
4	Identify tools	10-Jul-11	14-Jul-11	4 days	
5	Test tools	14-Jul-11	19-Jul-11	5 days	
6	Develop delivery timeline	10-Jul-11	21-Jul-11	11 days	
7	Analysis complete	14-Jul-11	22-Jul-11	8 days	Expand/Collapse Button
8	Production	03-Jul-11	14-Jul-11	11 days	
9	Software Components	03-Jul-11	05-Jul-11	2 days	
10	Localization Component - User I...	06-Jul-11	07-Jul-11	1 days	
11	User Assistance Components	10-Jul-11	14-Jul-11	4 days	
12	Software components complete	14-Jul-11	18-Jul-11	4 days	
13	Quality Assurance	03-Jul-11	12-Jul-11	9 days	
14	Review project information	03-Jul-11	15-Jul-11	12 days	
15	Localization Component	06-Jul-11	08-Jul-11	2 days	
16	Localization Component	10-Jul-11	14-Jul-11	4 days	
17	Localization Component	14-Jul-11	18-Jul-11	4 days	
18	Beta Testing	03-Jul-11	14-Jul-11	11 days	
19	Disseminate completed product	03-Jul-11	05-Jul-11	2 days	
20	Obtain feedback	06-Jul-11	07-Jul-11	1 days	
21	Modify	10-Jul-11	19-Jul-11	9 days	
22	Test	14-Jul-11	19-Jul-11	5 days	
23	Complete	10-Jul-11	19-Jul-11	9 days	

- **Header**— Header represents the table header which contains the field name of the task.
- **Parent Task**—Parent task represents the summary of the child tasks. This is an activity which will be further split into various child tasks.
- **Child Task**—Child task represents an individual task. This contains only the information about the specific task. The Child task is a part of parent task.
- **Expand/Collapse Button**—Expand/Collapse button allows you to expand or collapse the particular hierarchy.

Gantt chart

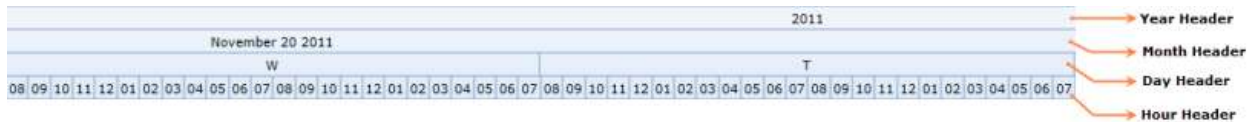
Gantt Chart is an items control which provides a graphically representation of the task/activity that are currently scheduled. Gantt Chart have different components to represent the type of Task, Progress of the Task and Relationship between Tasks.



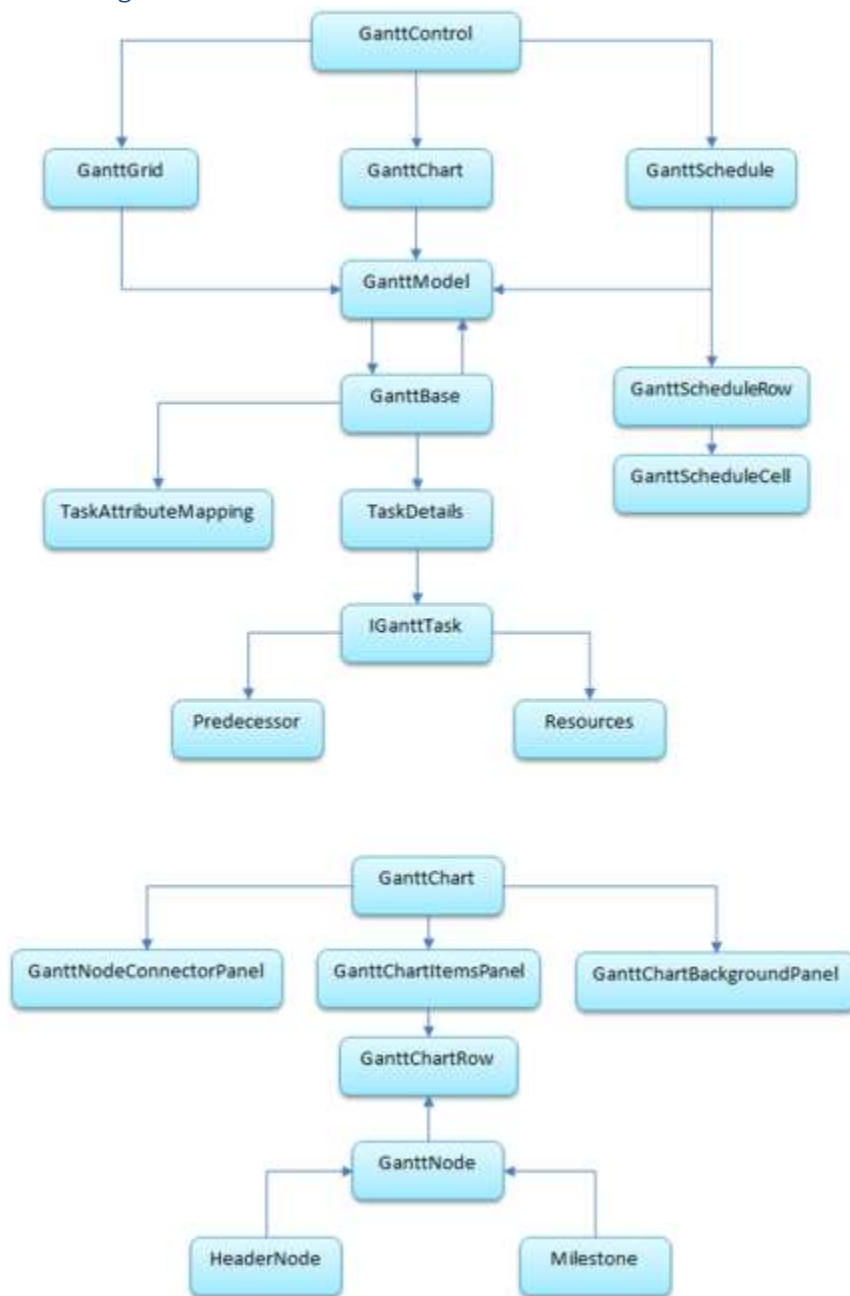
- Node— Node represents an individual or child task.
- Header Node—HeaderNode represents the parent or summary task of the projects.
- Milestone—Milestone represents the target to be completed in a day.
- Progress Indicator—Progress indicator represents the percentage of work completed for the task.
- Connector—Connector represents the dependency relationship between the tasks.

Gantt schedule

Gantt Schedule is a medium to measure the progress in the Gantt control. Using this you can track or measure the progress of the task or activity.



Class diagram



Feature summary

The following features are available in the Essential Gantt for WPF:

- Data Binding
- TaskDetails Binding
- External Property Binding
- Dependency relationship
- CustomToolTip
- Calendar customization
- Custom Node style

- VisualStyle
- XML Import/Export

Adding GanttControl to an application

You can create a project management application using Essential Gantt WPF.

You can create Gantt control in two methods. They are:

- Programmatically
- Through Designer

Programmatically creating GanttControl

The following are the steps to create GanttControl programmatically:

Adding GanttControl

You can add Gantt control to the application using the following code:

XML

```
<Sync:GanttControl x:Name="Gantt" />
```

C#

```
//Initializing Gantt
GanttControl Gantt = new GanttControl();
```

When the code runs, an empty Gantt with in-built TaskDetails collection will be displayed.

Binding data to GanttControl

Create a collection of tasks and bind it to the newly created GanttControl as given in the following code:

XML

```
<Sync:GanttControl ItemsSource="{Binding TaskCollection}">
<Sync:GanttControl.DataContext>
<local:ViewModel></local:ViewModel>
</Sync:GanttControl.DataContext>
</Sync:GanttControl>
```

C#

```
//Initializing Gantt
GanttControl Gantt = new GanttControl();
ViewModel model= new ViewModel();
this.Gantt.DataContext = model;
Gantt.ItemsSource = model.GanttItemSource;
```

C#

```
[C#]
public class ViewModel
{
public ObservableCollection<TaskDetails> TaskCollection { get; set; }
public ViewModel()
```

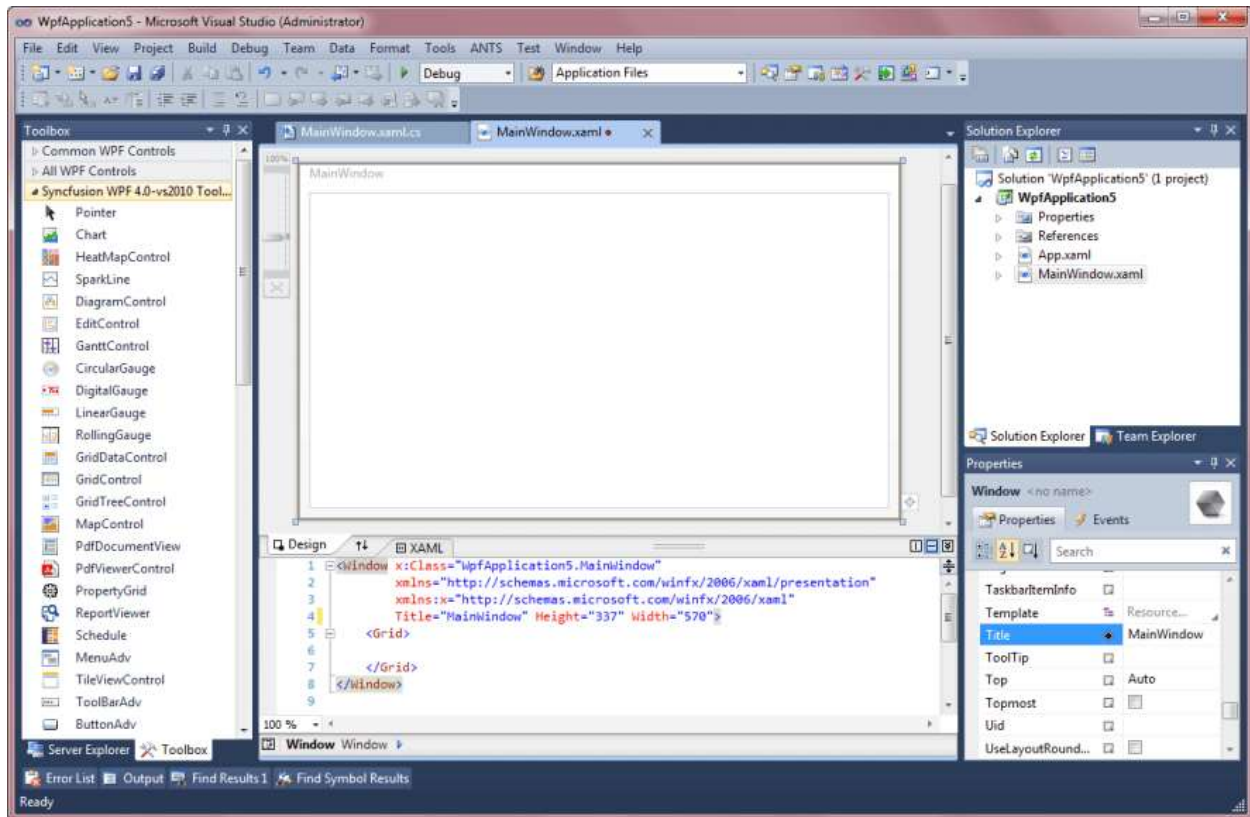
```
{
TaskCollection = this.GetDataSource();
}
private ObservableCollection<TaskDetails> GetDataSource()
{
ObservableCollection<TaskDetails> task = new
ObservableCollection<TaskDetails>();
task.Add(
new TaskDetails
{
TaskId = 1,
TaskName = "Scope",
StartDate = new DateTime(2011, 1, 3),
FinishDate = new DateTime(2011, 1, 14),
Progress = 40d
});
task[0].Child.Add(
new TaskDetails
{
TaskId = 2,
TaskName = "Determine project office scope",
StartDate = new DateTime(2011, 1, 3),
FinishDate = new DateTime(2011, 1, 5),
Progress = 20d
});
task[0].Child.Add(
new TaskDetails
{
TaskId = 3,
TaskName = "Justify project office via business model",
StartDate = new DateTime(2011, 1, 6),
FinishDate = new DateTime(2011, 1, 7),
Duration = new TimeSpan(1, 0, 0, 0),
Progress = 20d
});
task[0].Child.Add(
new TaskDetails
{
TaskId = 4,
TaskName = "Secure executive sponsorship",
StartDate = new DateTime(2011, 1, 10),
FinishDate = new DateTime(2011, 1, 14),
Duration = new TimeSpan(1, 0, 0, 0),
Progress = 20d
});
task[0].Child.Add(
new TaskDetails
{
TaskId = 5,
TaskName = "Secure complete",
StartDate = new DateTime(2011, 1, 14),
FinishDate = new DateTime(2011, 1, 14),
Duration = new TimeSpan(1, 0, 0, 0),
Progress = 20d
});
return task;
}
```

```
}
```

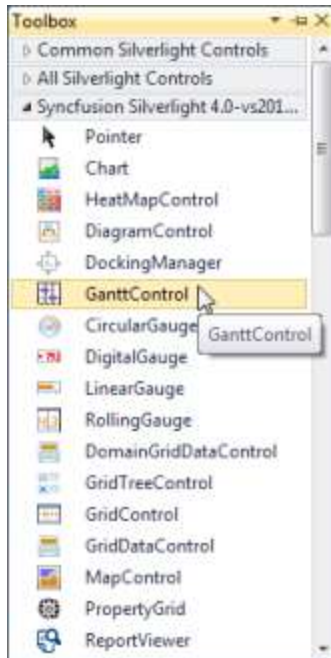
Adding GanttControl through designer

The following are the steps to create Gantt control through designer.

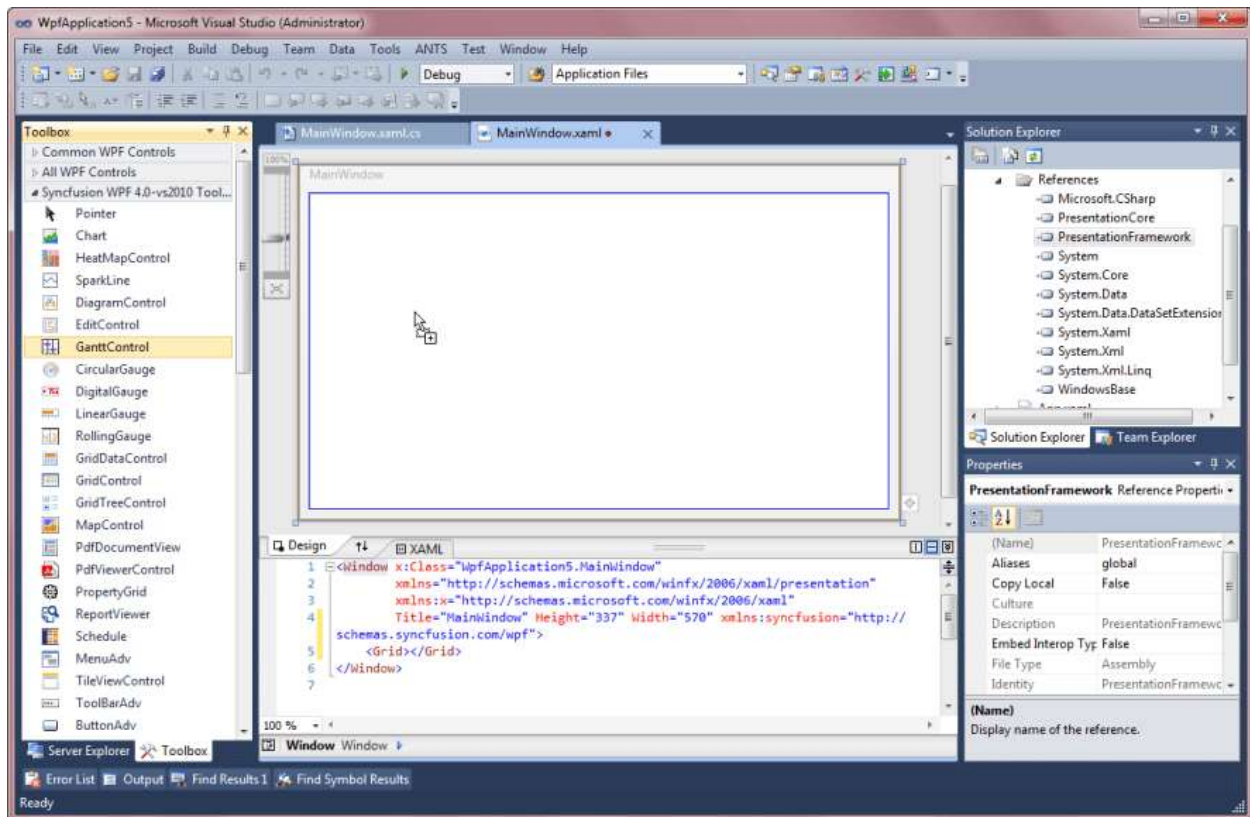
1. Open the XAML page of the application.



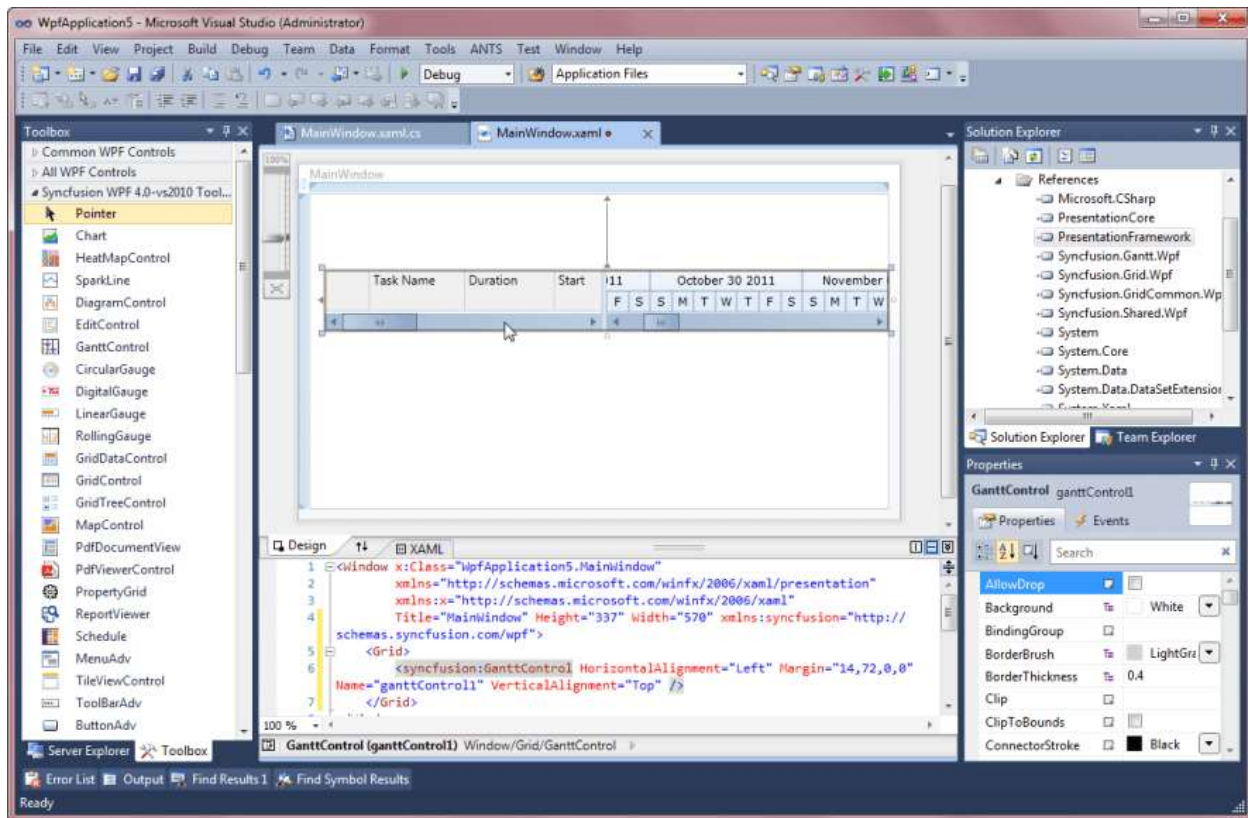
2. Select GanttControl from ToolBox.



3. Drag-and-drop the GanttControl to Designer View.



4. Gantt control is added to the window. Assembly reference will also be added to Project file.



5. Now you can customize the properties of Gantt control in the Properties Window.

Adjusting chart and grid size

The GanttControl allows users to set the width for GanttChart and GanttGrid using the [ChartWidth](#) and [GridWidth](#) properties. The following code sample demonstrates how to set width for chart and grid.

XML

```
<sync:GanttControl x:Name="control" GridWidth="200" ChartWidth="800" >
</Sync:GanttControl>
```

C#

```
//Initializing Gantt
GanttControl control = new GanttControl();
control.GridWidth = new GridLength(200);
control.ChartWidth = new GridLength(800);
```

Schedule padding

Gantt schedule view can be extended by using the [ScheduleRangePadding](#) property in GanttControl. This property extends the schedule with number of lower schedule units in starting position to improve the user experience.

XML

```
<Sync:GanttControl x:Name="control" ItemsSource="{Binding TaskCollection}"
ScheduleRangePadding="5">
```



```
</Sync:GanttControl>
```

C#

```
//Initializing Gantt
GanttControl control = new GanttControl();
control.ScheduleRangePadding = 5;
```

ScheduleType

By using the [ScheduleType](#) enum in the GanttControl, you can define the specific schedule range. The [ScheduleType](#) is an enum, which contains the following schedule types:

- HoursWithSeconds
- MinutesWithSeconds
- WeekWithDays
- DayWithHours
- DayWithMinutes
- MonthWithHours
- MonthWithDays
- YearWithDays
- YearWithMonths
- CustomDateTime
- CustomNumeric

The following code sample demonstrates how to set **ScheduleType** for GanttControl.

XML

```
<Sync:GanttControl x:Name="control" ItemsSource="{Binding TaskCollection}"
ScheduleType="YearWithMonths" >
</Sync:GanttControl>
```

C#

```
//Initializing Gantt
GanttControl control = new GanttControl();
control.DataContext = new ViewModel();
control.SetBinding(GanttControl.ItemsSourceProperty, "TaskCollection");
control.ScheduleType = ScheduleType.YearWithMonths;
```

Showing date with time in GanttGrid

To show the date with time in the GanttGrid, enable the ShowDateWithTime property as shown in the following code sample.

XML

```
<sync:GanttControl x:Name="control" ShowDateWithTime="True"
ItemsSource="{Binding TaskCollection}">
<sync:GanttControl.DataContext>
<local:ViewModel/>
</sync:GanttControl.DataContext>
```



```
</Sync:GanttControl>
```

C#

```
//Initializing Gantt
GanttControl control = new GanttControl();
control.DataContext = new ViewModel();
control.SetBinding(GanttControl.ItemsSourceProperty, "TaskCollection");
control.ShowDateWithTime = true;
```

Note: By default, GanttGrid will show the date alone.

see also

[How to show horizontal and vertical grid lines in WPF gantt control](#)

[How to enable horizontal lines for gantt chart rows](#)

[How to create gantt chart control in C# WPF](#)

[How to create a gantt chart by hiding the gantt grid](#)

[How to wrap WPF gantt in Windows Forms](#)

Data Binding in WPF Gantt

TaskDetails Binding

Essential Gantt for WPF includes an built-in class called TaskDetails, which is inherited from the IGanttTask interface. A collection of the TaskDetails can be bounded as an ItemsSource for the GanttControl.

Use Case Scenarios

You can easily create the task details collection using the TaskDetails class or by creating a new class by inheriting the IGantt interface.

Binding TaskDetails collection to Gantt Control

The following code illustrates how to bind the Task Details to the Gantt Control:

XML

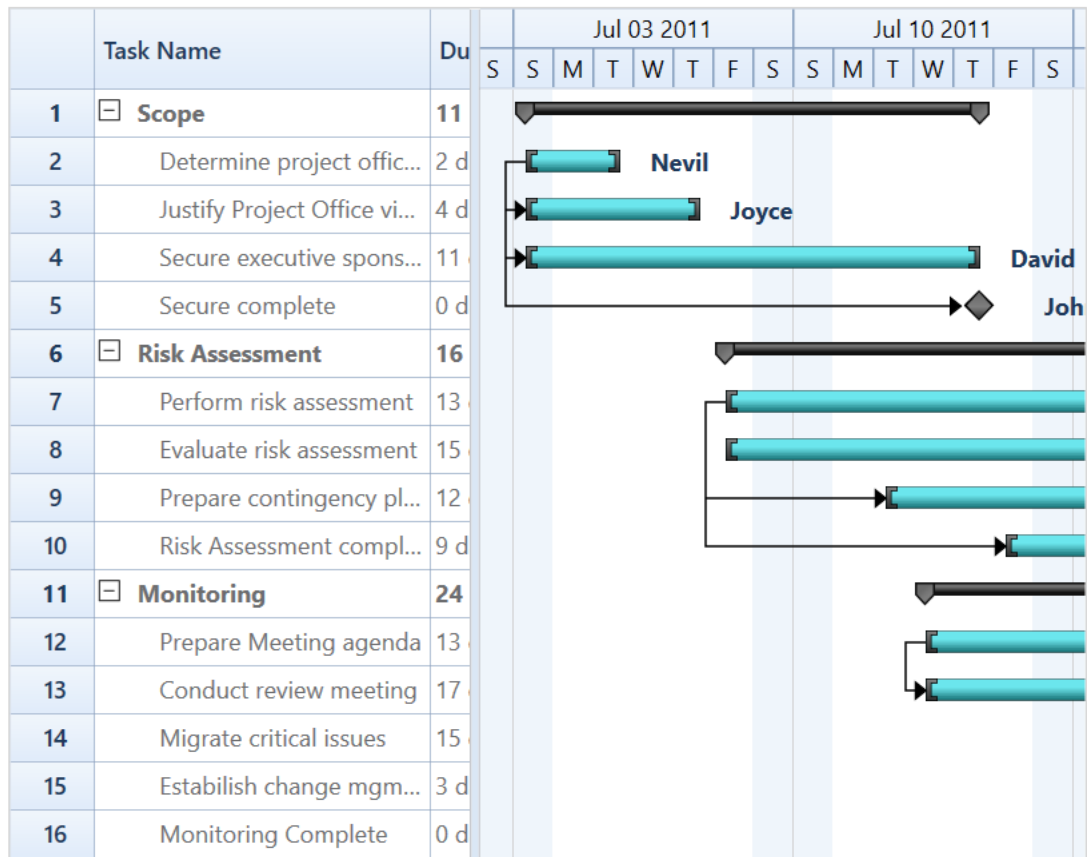
```
<Sync:GanttControl ItemsSource="{Binding TaskCollection}">
<Sync:GanttControl.DataContext>
<local:ViewModel></local:ViewModel>
</Sync:GanttControl.DataContext>
</Sync:GanttControl>
```

C#

```
GanttControl Gantt = new GanttControl();
ViewModel model = new ViewModel();
this.Gantt.DataContext = model;
Gantt.ItemsSource = model.GanttItemSource;
public class ViewModel
{
    public ObservableCollection<TaskDetails> TaskCollection { get; set; }
    public ViewModel()
    {
    }
```

```
TaskCollection = this.GetDataSource();
}
private ObservableCollection<TaskDetails> GetDataSource()
{
    ObservableCollection<TaskDetails> task = new
    ObservableCollection<TaskDetails>();
    task.Add(
        new TaskDetails
        {
            TaskId = 1,
            TaskName = "Scope",
            StartDate = new DateTime(2011, 1, 3),
            FinishDate = new DateTime(2011, 1, 14),
            Progress = 40d
        });
    task[0].Child.Add(
        new TaskDetails
        {
            TaskId = 2,
            TaskName = "Determine project office scope",
            StartDate = new DateTime(2011, 1, 3),
            FinishDate = new DateTime(2011, 1, 5),
            Progress = 20d
        });
    task[0].Child.Add(
        new TaskDetails
        {
            TaskId = 3,
            TaskName = "Justify project office via business model",
            StartDate = new DateTime(2011, 1, 6),
            FinishDate = new DateTime(2011, 1, 7),
            Duration = new TimeSpan(1, 0, 0, 0),
            Progress = 20d
        });
    task[0].Child.Add(
        new TaskDetails
        {
            TaskId = 4,
            TaskName = "Secure executive sponsorship",
            StartDate = new DateTime(2011, 1, 10),
            FinishDate = new DateTime(2011, 1, 14),
            Duration = new TimeSpan(1, 0, 0, 0),
            Progress = 20d
        });
    task[0].Child.Add(
        new TaskDetails
        {
            TaskId = 5,
            TaskName = "Secure complete",
            StartDate = new DateTime(2011, 1, 14),
            FinishDate = new DateTime(2011, 1, 14),
            Duration = new TimeSpan(1, 0, 0, 0),
            Progress = 20d
        });
    return task;
}
```

The following image shows the BindingTask Details:



[Samples Link](#)

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel.
4. Select Gantt.
5. Expand the DataBinding Features item in the Sample Browser.
6. Choose the Binding Task Details samples to launch.

External Property Binding

Essential Gantt for WPF allow you to bind any type of IEnumerable source to Gantt. You can bind any collection to Gantt using the TaskAttributeMapping class. This will get the mapping name of the required fields from the underlying source. With this mapping the Gantt will get the required information to render the Chart nodes.

The following code illustrate how to map the properties using the TaskAttributeMapping class:

XML

```
<Sync:TaskAttributeMapping
TaskIdMapping="Id"
TaskNameMapping="Name"
StartDateMapping="StartDate"
ChildMapping="ChildTask"
FinishDateMapping="EndDate"
DurationMapping="Duration"
ResourceInfoMapping="Resource"
ProgressMapping="Complete"
PredecessorMapping="Predecessor">
</Sync:TaskAttributeMapping>
```

C#

```
TaskAttributeMapping attributes = new TaskAttributeMapping();
attributes.TaskIdMapping = "Id";
attributes.TaskNameMapping = "Name";
attributes.StartDateMapping = "StartDate";
attributes.FinishDateMapping = "EndDate";
attributes.DurationMapping = "Duration";
attributes.ChildMapping = "ChildTask";
attributes.ResourceInfoMapping = "Resource";
attributes.ProgressMapping = "Predecessor";
this.Gantt.TaskAttributeMapping = attributes;
```

The following code illustrates how to bind the external source to Gantt control:

XML

```
<Sync:GanttControl ItemsSource="{Binding TaskCollection}">
<Sync:GanttControl.DataContext>
<local:ViewModel></local:ViewModel>
</Sync:GanttControl.DataContext>
<Sync:GanttControl.TaskAttributeMapping>
<Sync:TaskAttributeMapping
TaskIdMapping="ID"
TaskNameMapping="Name"
StartDateMapping="StartDate"
FinishDateMapping="EndDate"
ChildMapping="ChildCollection"
ProgressMapping="Progress"
DurationMapping="Duration">
</Sync:TaskAttributeMapping>
</Sync:GanttControl.TaskAttributeMapping>
</Sync:GanttControl>
```

C#

```
//Initializing Gantt
GanttControl Gantt = new GanttControl();
ViewModel model= new ViewModel();
TaskAttributeMapping attributes = new TaskAttributeMapping();
```

```
attributes.TaskIdMapping = "ID";
attributes.TaskNameMapping = "Name";
attributes.StartDateMapping = "StartDate";
attributes.FinishDateMapping = "EndDate";
attributes.DurationMapping = "Duration";
attributes.ChildMapping = "ChildCollection";
this.Gantt.DataContext = model;
Gantt.TaskAttributeMapping = attributes;
Gantt.ItemsSource = model.GanttItemSource;
public class Task : INotifyPropertyChanged
{
    private DateTime startDate, endDate;
    private TimeSpan duration;
    private double progress;
    private int id;
    private string name;
    private ObservableCollection<Task> childCollection;
    /// <summary>
    /// Property for Start Date.
    /// </summary>
    public DateTime StartDate
    {
        get
        {
            return this.startDate;
        }
        set
        {
            this.startDate = value;
            OnPropertyChanged("StartDate");
        }
    }
    /// <summary>
    /// Property for Finish Date.
    /// </summary>
    public DateTime EndDate
    {
        get
        {
            return this.endDate;
        }
        set
        {
            this.endDate = value;
            OnPropertyChanged("EndDate");
        }
    }
    /// <summary>
    /// Property for duration value.
    /// </summary>
    public TimeSpan Duration
    {
        get
        {
            return this.duration;
        }
        set
    }
}
```

```
{
    this.duration = value;
    OnPropertyChanged("Duration");
}
}
/// <summary>
/// Property for ID value.
/// </summary>
public int ID
{
    get
    {
        return this.id;
    }
    set
    {
        this.id = value;
        OnPropertyChanged("ID");
    }
}
/// <summary>
/// Property for Name.
/// </summary>
public string Name
{
    get
    {
        return this.name;
    }
    set
    {
        this.name = value;
        OnPropertyChanged("Name");
    }
}
/// <summary>
/// Property to define progress value.
/// </summary>
public double Progress
{
    get
    {
        return this.progress;
    }
    set
    {
        this.progress = value;
        OnPropertyChanged("Progress");
    }
}
public ObservableCollection<Task> ChildCollection
{
    get
    {
        return this.childCollection;
    }
    set
}
```

```

{
    this.childCollection = value;
    OnPropertyChanged("ChildCollection");
}
}
private void OnPropertyChanged(string propName)
{
    if (this.PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propName));
    }
}
public event PropertyChangedEventHandler PropertyChanged;
}
public class ViewModel
{
    public ObservableCollection<Task> TaskCollection { get; set; }
    public ViewModel()
    {
        TaskCollection = this.GetDataSource();
    }
    private ObservableCollection<Task> GetDataSource()
    {
        ObservableCollection<Task> task = new ObservableCollection<Task>();
        task.Add(
            new Task
            {
                ID = 1,
                Name = "Scope",
                StartDate = new DateTime(2011, 1, 3),
                EndDate = new DateTime(2011, 1, 14),
                Progress = 40d
            });
        task[0].ChildCollection = new ObservableCollection<Task>();
        task[0].ChildCollection.Add(
            new Task
            {
                ID = 2,
                Name = "Determine project office scope",
                StartDate = new DateTime(2011, 1, 3),
                EndDate = new DateTime(2011, 1, 5),
                Progress = 20d
            });
        task[0].ChildCollection.Add(
            new Task
            {
                ID = 3,
                Name = "Justify project office via business model",
                StartDate = new DateTime(2011, 1, 6),
                EndDate = new DateTime(2011, 1, 7),
                Duration = new TimeSpan(1, 0, 0, 0),
                Progress = 20d
            });
        task[0].ChildCollection.Add(
            new Task
            {
                ID = 4,

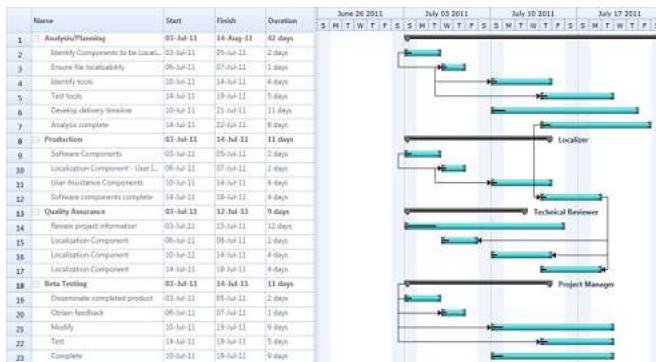
```

```

Name = "Secure executive sponsorship",
StartDate = new DateTime(2011, 1, 10),
EndDate = new DateTime(2011, 1, 14),
Duration = new TimeSpan(1, 0, 0, 0),
Progress = 200
});
task[0].ChildCollection.Add(
new Task
{
ID = 5,
Name = "Secure complete",
StartDate = new DateTime(2011, 1, 14),
EndDate = new DateTime(2011, 1, 14),
Duration = new TimeSpan(1, 0, 0, 0),
Progress = 200
});
return task;
}
}

```

The following image shows the External Property Binding:



[Samples Link](#)

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel.
4. Select Gantt.
5. Expand the DataBinding Features item in the Sample Browser.
6. Choose the External Property Binding sample to launch.

Dependency Relationship in WPF Gantt

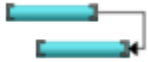
Dependency relationship is the relationship between two tasks. These relationship has been categorized into four types based on the start and finish date of the task. They are:

- FinishToStart
- FinishToFinish
- StartToStart
- StartToFinish

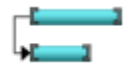
Finish-to-start—You cannot start a task until the other task is completed.



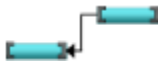
Finish-to-finish—You cannot finish a task until the other task is completed.



Start-to-start—You cannot start a task until the other task is also started.



Start-to-Finish—You cannot finish a task until another the other task is started.



Properties

Property	Description	Type	Data Type	Reference links
Predecessor	This enables you to set the relationship between the tasks.	Object	Object	NA
GanttTaskRelationship	This contains four relationships. They are: <i>StartToStart</i> <i>StartToFinish</i> <i>FinishToFinish</i> <i>FinishToStart</i> You can assign this to the <i>TaskDetails</i> to set the relationship between tasks.	Predecessor	Enum	NA

Specifying the Relationship between Tasks

The following code illustrates how to add the Dependency Relationship between tasks:

C#

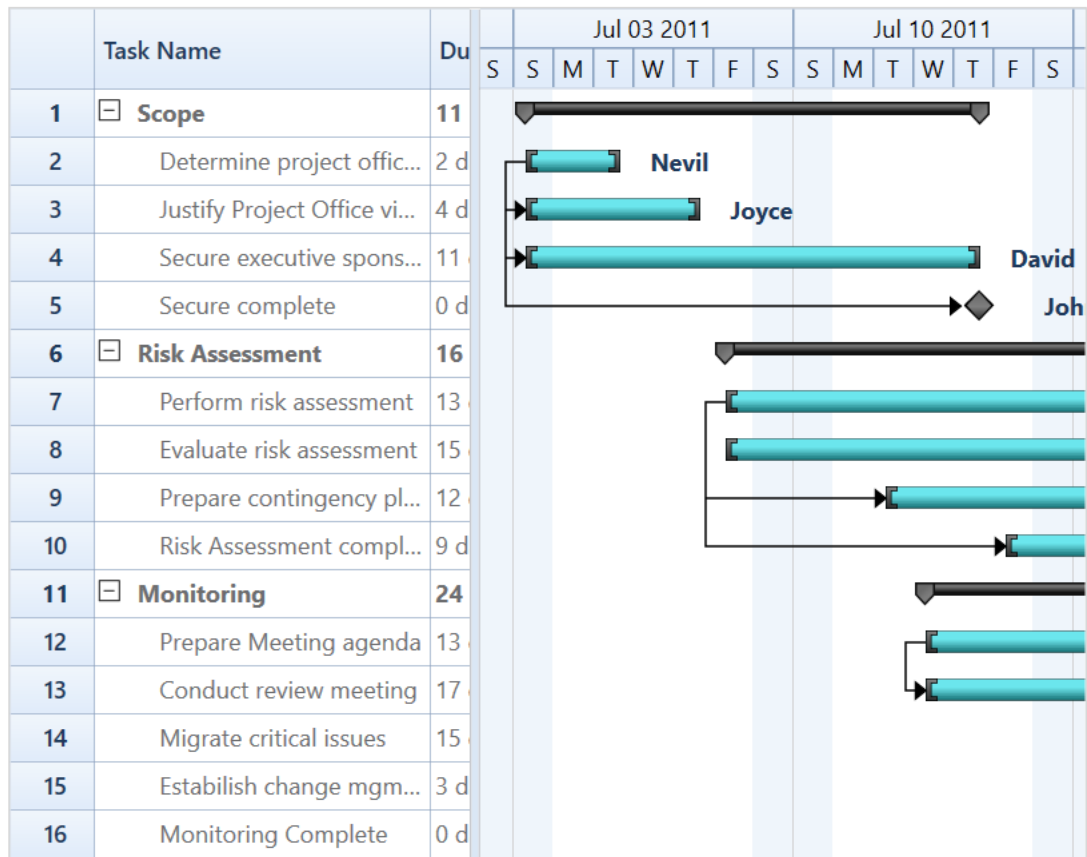
```
// Adding dependency relationship.
task[0].ChildCollection[1].Predecessors = new
ObservableCollection<Predecessor>(
task[0].ChildCollection[1].Predecessors.Add(new Predecessor()
{
GanttTaskIndex =
GanttTaskRelationship = GanttTaskRelationship.StartToStart
});
task[0].ChildCollection[2].Predecessors = new
ObservableCollection<Predecessor>();
task[0].ChildCollection[2].Predecessors.Add(new Predecessor()
{
GanttTaskIndex =
```

```

GanttTaskRelationship = GanttTaskRelationship.StartToFinish
});
task[0].ChildCollection[3].Predecessors = new
ObservableCollection<Predecessor>();
task[0].ChildCollection[3].Predecessors.Add(new Predecessor()
{
GanttTaskIndex =
GanttTaskRelationship = GanttTaskRelationship.FinishToFinish
});

```

The following image shows the Dependency Relationship:



[Samples Link](#)

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel .
4. Select Gantt.
5. Expand the Connectors Features item in the Sample Browser.

- Choose the Predecessor sample to launch.

Dynamic Predecessors and Resources

Essential Gantt provides support for dynamic editing of predecessors in the Grid area. Using this feature, you can dynamically add the predecessor on-demand basis to Gantt. You can add/remove the predecessor in the corresponding cell in the GanttGrid.

You can add/remove/update the predecessors and resources of tasks at run time. Initially, the Gantt is loaded with the Predecessor/Resource information in the underlying collection and you can update this information if required.

You can edit the predecessor information from the GanttGrid. For resource, you can edit in the underlying source, Gantt will listen to the change in the underlying source and reflect it in both GanttGrid and GanttChart.

Predecessor Validation

There are two predecessor validation modes in Gantt Control.

- Auto - Successor nodes will adjust its position based on its predecessor nodes automatically.
- Manual - Successor nodes need to adjust manually with respect to predecessor.

Properties

Property	Description	Type	Data Type	Reference links
ValidationMode	<p>This contains two modes.</p> <ul style="list-style-type: none"> Manual, Auto. <p>Default value is Manual.</p>	DependencyProperty	Enum	NA

The following image shows the Predecessor in Manual Mode:

![Predecessors ManualMode](Dependency-Relationship_images/Predecessors ManualMode.png)

The following image shows the Predecessor in Auto Mode:

![Predecessors AutoMode](Dependency-Relationship_images/Predecessors AutoMode.png)

Editing Predecessors

While creating a new predecessor in Grid, it should be in the following format:

C#

```
{Task ID}{GanttTaskRelationship}
```

Here the {GanttTaskRelationship} is any one of the following:

- SS
- FS
- SF
- FF

The following are what you should use for different relationships:

- Use “SS” to define the “StartToStart” relationship
- Use “FS” to define the “FinishToStart” relationship
- Use “SF” to define the “StartToFinish” relationship
- Use “FF” to define the “FinishToFinish” relationship

If any other data is added, the current editing relationship will be deleted and only the valid predecessors remain for the task.

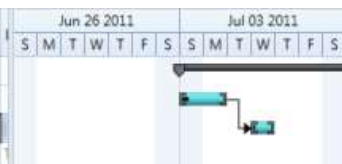
Editing Resources

As of now, resources cannot be edited in Grid. You can update the resource collection in the underlying source whenever you need. Gantt will listen to the changes in the collection and will update the GanttGrid and GanttChart accordingly.

Use Case Scenario

This helps to change the dependency relationships and resources of the tasks dynamically.

	Task Name	Start	Finish	Duration	Progress (%)	Predecessor	
1	Analysis/Planning	7/3/2011	7/26/2011	62 days	40		
2	Identify Components to be Localized	7/3/2011	7/5/2011	3 days	20		
3	Ensure file localizability	7/6/2011	7/7/2011	2 days	20	2FS	
4	Identify tools	7/10/2011	7/14/2011	5 days	10		



Adding Dynamic Predecessors and Resources to an Application

The dynamic editing of predecessor will be automatically included in the Gantt by default. There is no need to provide any additional data for that. The following codes illustrate this:

XML

```
<Sync:GanttControl ItemsSource="{Binding TaskCollection}" >
<Sync:GanttControl.TaskAttributeMapping>
<Sync:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"
DurationMapping="Duration"
StartDateMapping="StartDate"
FinishDateMapping="EndDate"
ChildMapping="ChildTask"
ProgressMapping="Complete"
PredecessorMapping="Predecessors"
ResourceInfoMapping="Resource">
</Sync:TaskAttributeMapping>
</Sync:GanttControl.TaskAttributeMapping>
</Sync:GanttControl>
```

The following code will illustrate how to dynamically add resource and predecessor in the underlying collection:

C#

```
// To Add the Dynamic Predecessors
this.viewModel.GanttItemSource[0].ChildCollection[2].Predecessors.Add(new
Predecessor()
{
GanttTaskIndex = 3,
```

```
GanttTaskRelationship = GanttTaskRelationship.StartToFinish
});
//To Add the Dynamic Resources
this.viewModel.GanttItemSource[0].ChildTask[2].Resource.Add(new Resource {
ID = 3, Name = "Resource3" });
```

Auto Update Hierarchy in WPF Gantt

Essential Gantt provides support for auto updating hierarchy, in which the Gantt control will listen to the change in child tasks/activities and automatically update them in the parent task/activity accordingly. There is no need to have any custom logics in business objects to update the hierarchy. You can enable or disable this functionality by using the UseAutoUpdateHierarchy property.

Use Case Scenario

1. When using this, no need to have custom logics in the business objects for updating the hierarchy so that codes in a business object class/data structure will be reduced.
2. This enables you to create the abstract class/data structure for the business objects.

Properties

Property	Description	Type	Data Type
UseAutoUpdateHierarchy	Enable or disable the auto update hierarchy support.	Dependency Property	bool

Using the Auto Update Hierarchy Support in an Application

By default, the UseAutoUpdateHierarchy property is set to true so that this feature will work in all applications by default. If you need to prevent this and want to update the hierarchy by your own logics, then you have to set this property as false.

Default Scenario

To use the auto updating hierarchy support in an application:

1. Create a simple class structure for business objects.

C#

```
public class Task : NotificationObject
{
    public Task()
    {
        ChildTask = new ObservableCollection<Task>();
    }
    //No need to do the calculation for end date when the duration is changed
    public TimeSpan Duration
    {
        get
        {
            return duration;
        }
        set
```

```
{
    duration = value;
    RaisePropertyChanged("Duration");
}
}
//No need to do the calculation for duration when the end date is changed.
public DateTime EndDate
{
    get
    {
        return endDate;
    }
    set
    {
        endDate = value;
        RaisePropertyChanged("EndDate");
    }
}
//No need to do the calculation for duration when the starting date is
changed
public DateTime StDate
{
    get
    {
        return stDate;
    }
    set
    {
        stDate = value;
        RaisePropertyChanged("StDate");
    }
}
//No need to hook the collection based on the changes made in child nodes to
listen and refresh the parent nodes.
public ObservableCollection<Task> ChildTask
{
    get
    {
        return childTask;
    }
    set
    {
        childTask = value;
        RaisePropertyChanged("ChildTask");
    }
}
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
        RaisePropertyChanged("Name");
    }
}
```

```

}
public int Id
{
    get
    {
        return id;
    }
    set
    {
        id = value;
        RaisePropertyChanged("Id");
    }
}
public double Complete
{
    get
    {
        return Math.Round(complete, 2);
    }
    set
    {
        complete = value;
        RaisePropertyChanged("Complete");
    }
}
}

```

2. Create a collection of business objects to bind them as ItemsSource for the Gantt control.

C#

```

TaskDetails = new ObservableCollection<Task>();
TaskDetails = GetData();
ObservableCollection<Task> GetData()
{
    ObservableCollection<Task> data = new ObservableCollection<Task>();
    data.Add(
        new Task()
        {
            Id = 1,
            Name = "Analysis/Planning",
            StDate = new DateTime(2012, 7, 3),
            EndDate = new DateTime(2012, 8, 14),
            Complete = 40d
        });
    data[0].ChildTask.Add(
        (new Task()
        {
            Id = 2,
            Name = "Identify Components to be Localized",
            StDate = new DateTime(2012, 7, 3),
            EndDate = new DateTime(2012, 7, 5),
            Complete = 20d
        }));
    data[0].ChildTask.Add(

```

```
(new Task()
{
    Id = 3,
    Name = "Ensure file localizability",
    StDate = new DateTime(2012, 7, 6),
    EndDate = new DateTime(2012, 7, 7),
    Complete = 20d
}));
data.Add(
new Task()
{
    Id = 8,
    Name = "Production",
    StDate = new DateTime(2012, 7, 3),
    EndDate = new DateTime(2012, 7, 14),
    Complete = 40d
});
data[1].ChildTask.Add(
(new Task()
{
    Id = 9,
    Name = "Software Components",
    StDate = new DateTime(2012, 7, 3),
    EndDate = new DateTime(2012, 7, 5),
    Complete = 20d,
}));
data[1].ChildTask.Add(
(new Task()
{
    Id = 10,
    Name = "Localization Component - User Interface",
    StDate = new DateTime(2012, 7, 6),
    EndDate = new DateTime(2012, 7, 7),
    Complete = 20d
}));
data.Add(
new Task()
{
    Id = 13,
    Name = "Quality Assurance",
    StDate = new DateTime(2012, 7, 3),
    EndDate = new DateTime(2012, 7, 12),
    Complete = 40d,
});
data[2].ChildTask.Add(
(new Task()
{
    Id = 14,
    Name = "Review project information",
    StDate = new DateTime(2012, 7, 3),
    EndDate = new DateTime(2012, 7, 15),
    Complete = 20d
}));
data[2].ChildTask.Add(
(new Task()
{
    Id = 15,
```



```

Name = "Localization Component",
StartDate = new DateTime(2012, 7, 6),
EndDate = new DateTime(2012, 7, 8),
Complete = 20d
}));
}

```

3. Set the collection as ItemsSource of the Gantt control.

XML

```

<gantt:GanttControl x:Name="Gantt"
Grid.Row="1"
UseAutoUpdateHierarchy="True"
ItemsSource="{Binding TaskDetails}">
<gantt:GanttControl.TaskAttributeMapping>
<gantt:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"
StartDateMapping="StartDate"
ChildMapping="ChildTask"
FinishDateMapping="EndDate"
DurationMapping="Duration"/>
</gantt:GanttControl.TaskAttributeMapping>
</gantt:GanttControl>

```

Using Custom Logics in Business Objects

To use your own logics in business objects:

1. Create a simple class structure for business objects. Add custom calculations on business objects to update the hierarchy when its state changed.

C#

```

public class Task : NotificationObject
{
    public Task()
    {
        ChildTask = new ObservableCollection<Task>();
    }
    public TimeSpan Duration
    {
        get
        {
            if (childTask != null && childTask.Count >= 1)
            {
                var sum = new TimeSpan(0, 0, 0, 0);
                sum = childTask.Aggregate(sum, (current, task) => current + task.Duration);
                return sum;
            }
            /// The difference between the end date and starting date is calculated exactly.
            duration = endDate.Subtract(startDate);
            return duration;
        }
    }
}

```

```

    }
    set
    {
        if (childTask != null && childTask.Count >= 1)
        {
            var sum = new TimeSpan(0, 0, 0, 0);
            sum = childTask.Aggregate(sum, (current, task) => current + task.Duration);
            duration = sum;
            return;
        }
        duration = value;
        /// The end date is calculated to make the change in end date based on
        duration. The duration is interlinked with the starting date and end date,
        so it will affect the both based on the changes.
        EndDate = stDate.AddDays(Double.Parse(duration.TotalDays.ToString()));
    }
}

public DateTime EndDate
{
    get { return endDate; }
    set
    {
        if (childTask != null && childTask.Count >= 1)
        {
            /// If this task is a parent task, then it should have the maximum end time
            to compare the date with maximum date of its child tasks.
            if (value >= childTask.Max(s => s.EndDate) && endDate != value)
                endDate = value;
        }
        else
            endDate = value;
        RaisePropertyChanged("EndDate");
        /// The changed duration is invoked to notify the change in duration based
        on the new end date.
        RaisePropertyChanged("Duration");
    }
}

public DateTime StDate
{
    get
    {
        return stDate;
    }
    set
    {
        /// If this task is a parent task, then it should have the minimum starting
        time to compare the date with minimum date of its child tasks.
        if (childTask != null && childTask.Count >= 1)
        {
            if (value <= childTask.Min(s => s.stDate) && stDate != value)
                stDate = value;
        }
        else
            stDate = value;
        RaisePropertyChanged("StDate");
        /// The changed duration is invoked to notify the change in duration based
        on the new start date.
    }
}

```

```

RaisePropertyChanged("Duration");
}
}
public string Name
{
    get { return name; }
    set
    {
        name = value;
        RaisePropertyChanged("Name");
    }
}
public int Id
{
    get { return id; }
    set
    {
        id = value;
        RaisePropertyChanged("Id");
    }
}
public ObservableCollection<Task> ChildTask
{
    get
    {
        if (childTask == null)
        {
            childTask = new ObservableCollection<Task>();
            /// The changed collection of child tasks is hooked to listen and refresh
the parent node based on the changes made in child des.
            childTask.CollectionChanged += ChildNodesCollectionChanged;
        }
        return childTask;
    }
    set
    {
        childTask = value;
        ///The changed collection of child tasks is hooked to listen and refresh the
parent node based on the changes made in child nodes.
        childTask.CollectionChanged += ChildNodesCollectionChanged;
        if (value.Count > 0)
        {
            childTask.ToList().ForEach(n =>
            {
                /// To listen the changes made in child tasks.
                n.PropertyChanged += ChildNodePropertyChanged;
            });
            UpdateData();
        }
        RaisePropertyChanged("ChildTask");
    }
}
void ChildNodePropertyChanged(object sender, PropertyChangedEventArgs e)
{
    if (e.PropertyName != null)
    if (e.PropertyName == "StartDate" || e.PropertyName == "EndDate" ||
        e.PropertyName == "Complete")

```

```

{
    UpdateData();
}
}
private void UpdateData()
{
    /// Update the starting date and end date based on the changes made in the
    date of child tasks.
    StDate = childTask.Select(c => c.StDate).Min();
    EndDate = childTask.Select(c => c.EndDate).Max();
    Complete = (childTask.Aggregate(0d, (cur, task) => cur + task.Complete)) /
    childTask.Count;
}
public void ChildNodesCollectionChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
    if (e.Action == NotifyCollectionChangedAction.Add)
    {
        foreach (Task node in e.NewItems)
        {
            node.PropertyChanged += ChildNodePropertyChanged;
        }
    }
    else
    {
        foreach (Task node in e.OldItems)
        {
            node.PropertyChanged -= ChildNodePropertyChanged;
        }
    }
    UpdateData();
}
}

```

2. Create a collection of business objects to bind it as ItemsSource of the Gantt control.

C#

```

TaskDetails = new ObservableCollection<Task>();
TaskDetails = GetData();
ObservableCollection<Task> GetData()
{
    ObservableCollection<Task> data = new ObservableCollection<Task>();
    data.Add(new Task()
    {
        Id = 1,
        Name = "Analysis/Planning",
        StDate = new DateTime(2012, 7, 3),
        EndDate = new DateTime(2012, 8, 14),
        Complete = 40d
    });
    data[0].ChildTask.Add((new Task()
    {
        Id = 2,
        Name = "Identify Components to be Localized",
        StDate = new DateTime(2012, 7, 3),
        EndDate = new DateTime(2012, 7, 5),
    }));
}

```

```

Complete = 20d
}});
data[0].ChildTask.Add((new Task()
{
    Id = 3,
    Name = "Ensure file localizability",
    StDate = new DateTime(2012, 7, 6),
    EndDate = new DateTime(2012, 7, 7),
    Complete = 20d
}}));
data.Add(new Task()
{
    Id = 8,
    Name = "Production",
    StDate = new DateTime(2012, 7, 3),
    EndDate = new DateTime(2012, 7, 14),
    Complete = 40d
});
data[1].ChildTask.Add((new Task()
{
    Id = 9,
    Name = "Software Components",
    StDate = new DateTime(2012, 7, 3),
    EndDate = new DateTime(2012, 7, 5),
    Complete = 20d,
}}));
data[1].ChildTask.Add((new Task()
{
    Id = 10,
    Name = "Localization Component - User Interface",
    StDate = new DateTime(2012, 7, 6),
    EndDate = new DateTime(2012, 7, 7),
    Complete = 20d
}}));
data.Add(new Task()
{
    Id = 13,
    Name = "Quality Assurance",
    StDate = new DateTime(2012, 7, 3),
    EndDate = new DateTime(2012, 7, 12),
    Complete = 40d,
});
data[2].ChildTask.Add((new Task()
{
    Id = 14,
    Name = "Review project information",
    StDate = new DateTime(2012, 7, 3),
    EndDate = new DateTime(2012, 7, 15),
    Complete = 20d
}}));
data[2].ChildTask.Add((new Task()
{
    Id = 15,
    Name = "Localization Component",
    StDate = new DateTime(2012, 7, 6),
    EndDate = new DateTime(2012, 7, 8),
    Complete = 20d
}

```

```

    } } } ;
}

```

- Set the UseAutoUpdateHierarchy property as false to handle the custom calculations on the business objects. Set the collection as ItemsSource of the Gantt control.

Note: If you use the TaskDetails class as your business object, then you should not set the UseAutoUpdateHierarchy property as false.

XML

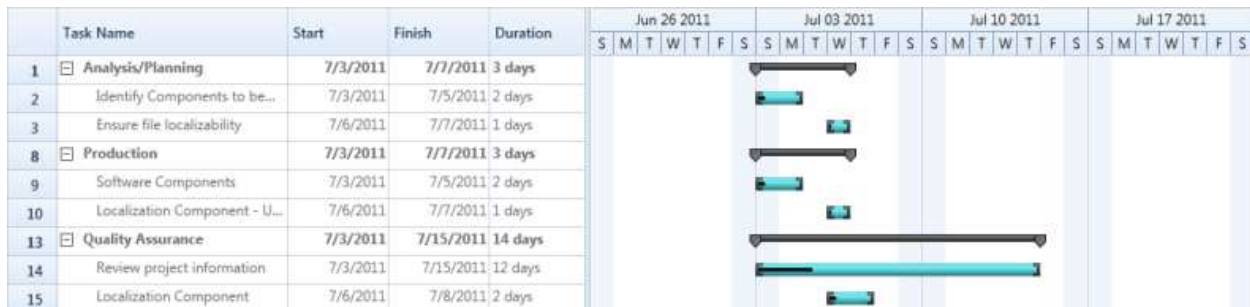
```

<gantt:GanttControl x:Name="Gantt"
Grid.Row="1"
UseAutoUpdateHierarchy="False"
ItemsSource="{Binding TaskDetails}">
<gantt:GanttControl.TaskAttributeMapping>
<gantt:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"
StartDateMapping="StartDate"
ChildMapping="ChildTask"
FinishDateMapping="EndDate"
DurationMapping="Duration"/>
</gantt:GanttControl.TaskAttributeMapping>
</gantt:GanttControl>

```

Output:

The corresponding output for the code is as follows.



Gantt Control with Auto Updated Hierarchy

[Sample Link](#)

To view samples:

- Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

- Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
- Click Run Samples for WPF under User Interface Edition panel .

4. Select Gantt.
5. Expand the Data Binding Features item in the Sample Browser.
6. Choose the External Property Binding Demo sample to launch.

Baseline Support in WPF Gantt

Baseline Table View

Baseline Table View will display the differences between the current progresses and the baseline of the project in the Gantt Grid.

While invoking the appropriate method, the variance will be calculated between the actual value and the mapped baseline value, and displayed as variance view in the Gantt Grid.

The variances will be calculated only for the mapped baseline fields. For example, if you have mapped only the Baseline Start to TaskMappingAttribute, then it will calculate the variance only between Start and Baseline Start and display it in the Grid.

You can revert to the default view of the Grid by invoking the corresponding method in Gantt.

Note: Variance view will have the read-only Grid. You cannot edit the cells in Gantt Grid, when it is in variance view.

Use Case Scenario

This will help the Project Leads to compare the current progress of the project to the baseline and modify or rework the plan of existing tasks in order to meet deadlines. Organizations can use this to compare the current progress of the project to the initial estimation, analyze the rework of the plan or the budget of the project in to meet deadlines and exact user requirements.

Methods

Method	Description	Parameters	Type	Return Type
LoadVarianceTableView	this method is used to load the Variance view of the Gantt.	LoadVarianceTableView()	N/A	void
LoadDefaultTableView	this method is used to load the Default (Editing) view of the Gantt.	LoadDefaultTableView()	N/A	void

Adding Baseline Table View to an Application

To add Baseline Table View to an application:

1. Declare the baseline properties in the view model.
2. Map the properties to Gantt in the corresponding TaskAttributeMappingProperty.
3. To load the Variance view, call the LoadVarianceTableView() method of the Gantt.
4. To load the Default (Editing) view, call the LoadDefaultTableView() method of the Gantt.

The following codes illustrate this:

XML

```
<gantt:GanttControl Grid.Row="1" x:Name="Gantt"
```

```

ItemsSource="{Binding GanttItemSource}"
ToolTipTemplate="{StaticResource toolTipTemplate}"
ShowBaseline="True">
<gantt:GanttControl.TaskAttributeMapping>
<gantt:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"
DurationMapping="Duration"
StartDateMapping="StartDate"
FinishDateMapping="EndDate"
ChildMapping="ChildTask"
ProgressMapping="Complete"
PredecessorMapping="Predecessor"
ResourceInfoMapping="Resource"
CostMapping="Cost"
BaselineCostMapping="BaselineCost"
BaselineFinishMapping="BaselineEnd"
BaselineStartMapping="BaselineStart" >
</gantt:TaskAttributeMapping>
</gantt:GanttControl.TaskAttributeMapping>
</gantt:GanttControl>

```

C#

```

// To load the Variance Table View
this.Gantt.LoadVarianceTableView();
// To load the Default [Editing] View
this.Gantt.LoadDefaultTableView();

```

The following image shows the BaseLine Table View:

	Task Name	Duration	Start	Finish	Cost	Baseline Start	Baseline Finish	Baseline Cost	Start Variance
1	<input type="checkbox"/> Residential Construction (2...	134 days	3/1/2012	7/17/2012	\$18,979.00	2/1/2012	8/28/2012	\$20,535.00	29 days
2	<input type="checkbox"/> General Considerations	24 days	3/1/2012	3/26/2012	\$1,621.00	2/1/2012	3/12/2012	\$2,010.00	29 days
3	<input type="checkbox"/> Finalize and Approve PL...	21 days	3/1/2012	3/22/2012	\$1,000.00	2/1/2012	2/17/2012	\$1,666.00	29 days
4	Review and Finalize Site...	19 days	3/1/2012	3/20/2012	\$500.00	2/1/2012	2/17/2012	\$833.00	29 days
5	Sign contract and Proce...	2 days	3/20/2012	3/22/2012	\$500.00	2/1/2012	2/17/2012	\$833.00	48 days
6	<input type="checkbox"/> Contracts and Agreeeme...	0 days	3/22/2012	3/22/2012	\$482.00	3/12/2012	3/12/2012	\$192.00	10 days

BaseLine Table View

Baseline Chart View

In Gantt chart, baseline start and end values are graphically visualized.

- Line shape represents the header node and node.
- Diamond shape represents the milestone.

Properties

Property	Description	Type	Data Type	Reference links
ShowBaseline	Controls the view of baseline in the Gantt Chart. Default value is false	DependencyProperty	bool	NA

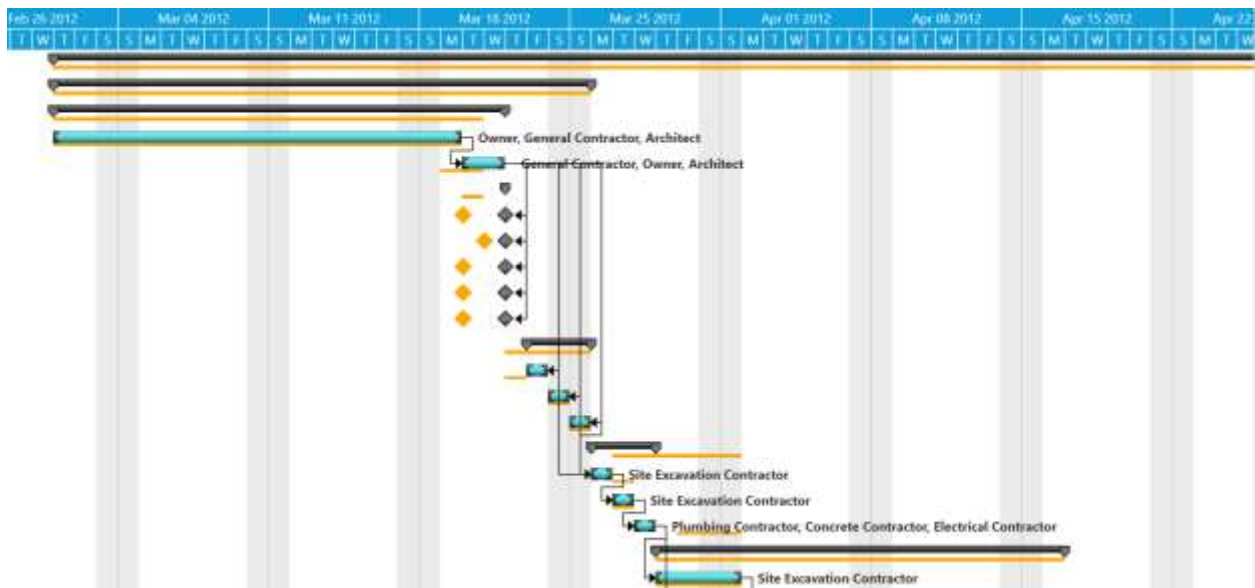
Baseline Customization

Able to customize the baseline color and thickness by using following properties.

Properties

Property	Description	Type	Data Type	Reference links
BaselineColor	Used to customize the baseline color. Default value is Orange	DependencyProperty	Brush	NA
BaselineStrokeThickness	Used to customize the baseline thickness. Default value is 3d	DependencyProperty	bool	NA

The following image shows the BaseLine Chart View:



Samples

To view samples:

1. Select Start -> Programs -> Syncfusion -> Essential Studio x.x.xx -> Dashboard.
2. Click Run Samples for WPF under User Interface Edition panel.
3. Select Gantt.
4. Expand the Baseline Support item in the Sample Browser.
5. Choose the Baseline TableView sample to launch.

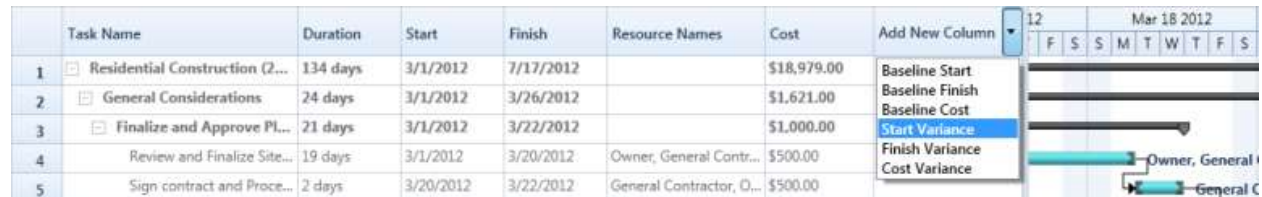
On-Demand Baseline Column Inclusion

On-Demand Baseline Column inclusion will dynamically add the baseline columns namely Baseline Start, Baseline End, Baseline Cost, Start Variance, Finish Variance, Cost Variance to Gantt Grid on demand basis. These columns will be listed in a drop down cell. By picking a column from that cell, it will get added to the current table.

Initially, Gantt will get loaded with a default set of columns. Then, you can add the baseline columns on demand basis to Gantt Grid, by selecting the columns from the drop down cell.

Use Case Scenario

This helps the Project Lead to store the estimation and will help to schedule the project in an efficient way by comparing the progress on planning itself. Organizations can have the default set of columns on Gantt Grid on loading, and when they need to compare a field with the estimate data, they can pick that column from the drop down.



	Task Name	Duration	Start	Finish	Resource Names	Cost
1	Residential Construction (2...	134 days	3/1/2012	7/17/2012		\$18,979.00
2	General Considerations	24 days	3/1/2012	3/26/2012		\$1,621.00
3	Finalize and Approve Pl...	21 days	3/1/2012	3/22/2012		\$1,000.00
4	Review and Finalize Site...	19 days	3/1/2012	3/20/2012	Owner, General Contr...	\$500.00
5	Sign contract and Proce...	2 days	3/20/2012	3/22/2012	General Contractor, O...	\$500.00

On-Demand Baseline Column Inclusion

Properties

Property	Description	Type	Data Type
ShowAddNewColumn	Gets/Sets the value that is used in displaying/hiding the add new column combo box in grid	DependencyProperty	bool

Adding On-Demand Baseline Column Inclusion to an Application

To add the On-Demand Baseline Column Inclusion to an application you need to enable the AddNewColumn of Gantt Grid, which will have a drop down cell on its head, which in turn will have the baseline columns that can be added dynamically. To enable this feature:

1. To show the Add New Column of Gantt Grid, set the ShowAddNewColumn Property to true in Gantt.
2. To include the Baseline columns in the Add New Columns drop down cell, provide the corresponding mapping name in TaskAttributeMapping.
3. Mapped Baseline columns will be listed in the drop down cell.
4. By selecting a column name from the drop down cell, the corresponding column will be dynamically added to the Gantt Grid.

The following codes illustrate Adding On-Demand Baseline Column Inclusion to an Application:

XML

```
<gantt:GanttControl Grid.Row="1" x:Name="Gantt"
ItemsSource="{Binding GanttItemSource}"
ToolTipTemplate="{StaticResource toolTipTemplate}"
ShowAddNewColumn="True">
<gantt:GanttControl.TaskAttributeMapping>
<gantt:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"
DurationMapping="Duration"
StartDateMapping="StDate"
FinishDateMapping="EndDate"
```

```

ChildMapping="ChildTask"
ProgressMapping="Complete"
PredecessorMapping="Predecessor"
ResourceInfoMapping="Resource"
CostMapping="Cost"
BaselineCostMapping="BaselineCost"
BaselineFinishMapping="BaselineEnd"
BaselineStartMapping="BaselineStart">
</gantt:TaskAttributeMapping>
</gantt:GanttControl.TaskAttributeMapping>
</gantt:GanttControl>

```

C#

```

// Displaying the Add New column drop down
this.Gantt.ShowAddNewColumn = true;

```

Samples Link

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel.
4. Select Gantt.
5. Expand the Baseline Support item in the Sample Browser.
6. Choose the Baseline Table View sample to launch.

Project Statistics

Project statistics will provide the current statistics of the project in an object form. It will have all the basic information about the project and will provide the variance between the initial plan and current progress.

This will provide enough information about the project. The entire project information will be available in the form of an object of type "ProjectInfo". You can set your own visual to display the statistics.

Use Case Scenario

This will be useful for Project Leads to take decisions based on the current status. An Organization can use this to present the current statuses of their projects to their clients. They can also use this for analysis before making important decisions on projects.

Methods

Method	Description	Parameters	Type	Return Type
GetProjectStatistics	this method is used to get the current statistics information about the project	GetProjectStatistics()	N/A	ProjectInfo

Adding Project Statistics to an Application

To add Project Statistics to an application:

1. Initialize new instance of 'ProjectInfo' class.
2. Assign the GetProjectStatistics() Methods' return value to that instance.
3. Use the instance for further process.

The following codes illustrate adding Project Statistics to an application:

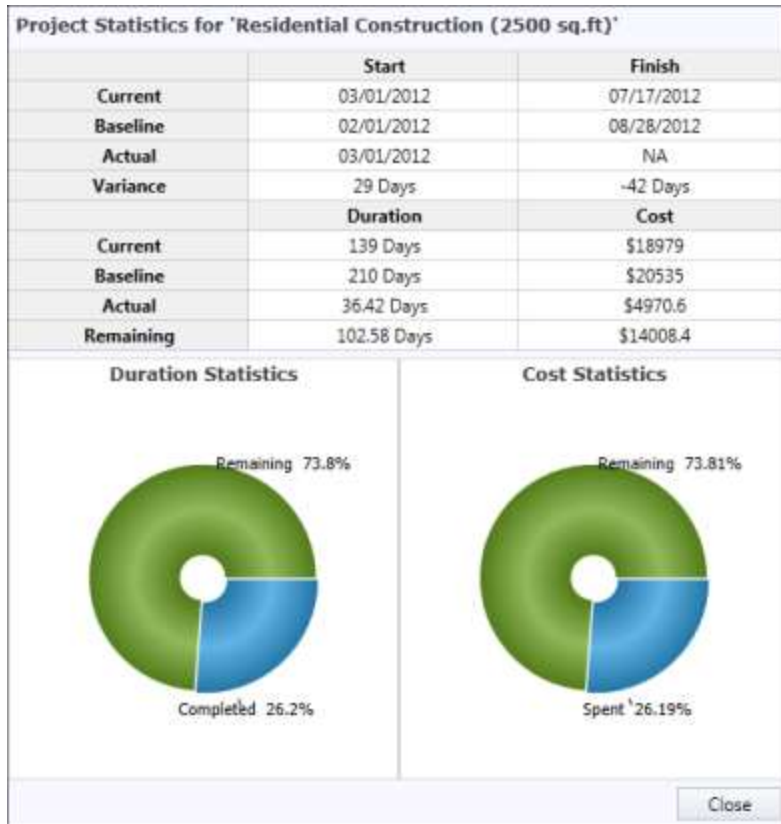
XML

```
<gantt:GanttControl Grid.Row="1" x:Name="Gantt"
ItemsSource="{Binding GanttItemSource}"
ToolTipTemplate="{StaticResource toolTipTemplate}">
<gantt:GanttControl.TaskAttributeMapping>
<gantt:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"
DurationMapping="Duration"
StartDateMapping="StartDate"
FinishDateMapping="EndDate"
ChildMapping="ChildTask"
ProgressMapping="Complete"
PredecessorMapping="Predecessor"
ResourceInfoMapping="Resource"
CostMapping="Cost"
BaselineCostMapping="BaselineCost"
BaselineFinishMapping="BaselineEnd"
BaselineStartMapping="BaselineStart">
</gantt:TaskAttributeMapping>
</gantt:GanttControl.TaskAttributeMapping>
</gantt:GanttControl>
```

C#

```
// To get the Project Statistics
ProjectInfo projInfo = this.Gantt.GetProjectStatistics();
(or)
ProjectInfo projInfo = new ProjectInfo();
projInfo = this.Gantt.GetProjectStatistics();
```

Sample Project Statistic Visual:



Project Statistics

[Samples](#)

To view samples:

1. Select Start -> Programs -> Syncfusion -> Essential Studio x.x.xx -> Dashboard.
2. Click Run Samples for WPF under User Interface Edition panel.
3. Select Gantt.
4. Expand the Baseline Support item in the Sample Browser.
5. Choose the Project Statistics sample to launch.

[CustomToolTip in WPF Gantt](#)

Essential Gantt provides tooltip support for the TaskBar. Tooltip is a small pop-up box, which is usually displayed on mouse hover. This is used to display additional information about the elements without increasing the window size. Essential Gantt provides support to add customizable tooltip. You can add text or image to your tooltip as needed.

[Properties](#)

Property	Description	Type	Data Type
ToolTipTemplate	Gets or set the TaskBarCollection Property of GanttControl	Dependency Property	DataTemplate

Adding CustomToolTip to Gantt

The following code illustrates how to add a custom tooltip to the Gantt control.

XML

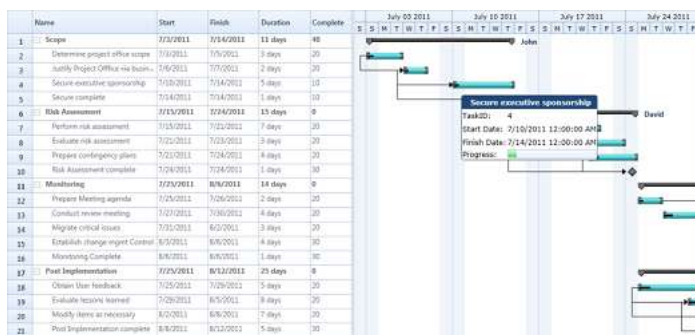
```
<DataTemplate x:Key="ToolTipTemp">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition/>
      <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="40"/>
      <RowDefinition Height="40"/>
      <RowDefinition Height="40"/>
      <RowDefinition Height="40"/>
      <RowDefinition Height="40"/>
    </Grid.RowDefinitions>
    <Border Grid.Column="0"
      Grid.Row="0"
      Margin="-2"
      CornerRadius="5"
      Grid.ColumnSpan="2"
      Background="#FF1F4877"
      BorderThickness="2">
      <TextBlock Text="{Binding TaskName}"
        Margin="5,0,0,0"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        FontWeight="Bold"
        FontFamily="Verdana"
        Foreground="WhiteSmoke"/>
    </Border>
    <TextBlock Margin="1"
      Text="TaskID:"
      Grid.Column="0"
      Grid.Row="1"
      VerticalAlignment="Center"
      FontFamily="Verdana"/>
    <TextBlock Margin="1" Text="{Binding TaskId}"
      Grid.Column="1"
      VerticalAlignment="Center"
      Grid.Row="1"
      FontFamily="Verdana"/>
    <TextBlock Margin="1" Text="Start Date:"
      Grid.Column="0"
      Grid.Row="2"
      VerticalAlignment="Center"
      FontFamily="Verdana"/>
    <TextBlock Margin="1" Text="{Binding StartDate}"
      Grid.Row="2"
      Grid.Column="1"
      VerticalAlignment="Center"
      FontFamily="Verdana"/>
    <TextBlock Margin="1" Text="Finish Date:"
      Grid.Column="0"
      Grid.Row="3">
```

```

VerticalAlignment="Center"
FontFamily="Verdana"/>
<TextBlock Margin="1" Text="{Binding FinishDate}"
Grid.Column="1"
Grid.Row="3"
VerticalAlignment="Center"
FontFamily="Verdana"/>
<TextBlock Margin="1" Text="Progress:"
Grid.Column="0"
Grid.Row="4"
VerticalAlignment="Center"
FontFamily="Verdana"/>
<ProgressBar Margin="1" Height="25"
Value="{Binding Progress}"
Grid.Column="1"
VerticalAlignment="Center"
Grid.Row="4"/>
</Grid>
</DataTemplate>
<Sync:GanttControl x:Name="Gantt"
ToolTipTemplate="{StaticResource ToolTipTemp}"/>

```

The following image shows Custom ToolTip:



Custom ToolTip Demo

Samples Link

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{{ site.releaseversion }}}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel .
4. Select Gantt.
5. Expand the Interactive Features item in the Sample Browser.
6. Choose the CustomToolTip samples to launch.

Custom Schedule in WPF Gantt

Essential Gantt provides the custom schedule support that allows you to define your own schedule for Gantt to track the progress of projects. You can define the schedule for any measurement unit or for different types of date time formats namely quarterly basis scale and so on.

This feature will get the information from you and will draw the Gantt schedule with that information. Custom schedule is categorized into two types namely:

- Custom Numeric
- Custom DateTime

These types are included in the existing ScheduleType enum.

Custom Numeric

Custom Numeric schedule is used to define your own schedule with any numeric measurement unit other than date time. With this schedule, you can track the progress based on your own measurement and there is no need to depend on Date Time. Two new API's are added to the Mapping attributes to support this schedule in GanttChart and GanttGrid.

Custom DateTime

Custom DateTime schedule is used to define your own date time schedule, which can match your current financial calendar. By using schedule type as custom schedule, you can define your schedule on quarterly basis.

In both the custom schedules, Gantt will get the information from the application to render the schedule. Gantt will accept the custom schedule information in the form of a collection of GanttScheduleRowInfo object, and process it to draw the schedule.

GanttScheduleRowInfo class will have following fields:

1. PixelsPerUnit—Gets the information of the pixel value equivalent to one unit in custom measurement.
2. CellsPerUnit—Gets the information of a cell size of preceding row in the schedule based on the immediate next row. In CustomDateTime Schedule, the CellsPerUnit will be used to customize the cell. For example, in quarterly basis month cell, You need to draw a schedule by consolidating three months. For this, you need to define the CellsPerUnit of that corresponding row as 3.
3. TimeUnit—Gets the information about the type of row, when the schedule type is CustomDateTime. The Time unit can be any one of the following:
 - Seconds—represents the corresponding row as second's row.
 - Minutes—represents the corresponding row as minute's row.
 - Hours—represents the corresponding row as hour's row.
 - Days—represents the corresponding row as day's row.
 - Weeks—represents the corresponding row as week's row.
 - Months—represents the corresponding row as month's row.
 - Years—represents the corresponding row as year's row.

Use Case Scenario

This will be useful when you like to define your own schedules with your own measurements/calendars.

Example 1: The Research organizations may follow different measurements to track their work progress and the measurement will depend on their products. In this case, they can use CustomNumeric schedule to define the schedule with their own measure.

Example 2: A very big construction project many have the time period of many years or months. They need some customized way of date time schedule to track their progress. In this scenario, they can use the CustomDateTime schedule to form the custom schedule like the schedule that has the time scale on quarterly basis to track their progress.

Properties

Property	Description	Type	Data Type
CustomScheduleSource	Gets/Sets the custom schedule items Source of the Gantt.	DependencyProperty	IList<GanttScheduleRowInfo>

Events

Event	Description	Arguments	Type
ScheduleCellCreated	Event will be triggered whenever a schedule cell is created. The handler of the event will have the newly created cell (GanttScheduleCell) in the argument. By handling this event, you can customize the appearance of the cell.	ScheduleCellCreated (object sender, ScheduleCellCreatedEventArgs args)	Routed Event

GanttScheduleCell Class

The properties of the GanttScheduleCell class are as follows:

Properties

Property	Description	Type	Data Type
CellDate	Gets/Sets the current schedule cell date in the datetime schedule.	Dependency Property	DateTime
CellToolTip	Gets/Sets the current schedule cell tool tip.	Dependency Property	Object
CellTimeUnit	Gets/Sets the current schedule row time unit (like weeks, months and so on).	Dependency Property	TimeUnit (Enum)
Content	Gets/Sets the current schedule cell content.	Dependency Property	Object

Adding Custom Schedule to an Application

To Add CustomNumeric Schedule to an application:

1. Define the Mapping for StartPointMapping and FinishPointMapping in TaskAttributeMapping.

2. Set the Gantt Schedule type as CustomNumeric.
3. Bind the GanttScheduleRowInfo collection to the CustomScheduleSource property of the Gantt.

The following code illustrates Adding Custom Schedule to an Application:

XML

```
<sync:GanttControl Grid.Row="1"
ScheduleType="CustomNumeric"
x:Name="Gantt"
VisualStyle="Office2010Black">
<sync:GanttControl.TaskAttributeMapping>
<sync:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"
StartPointMapping="Start"
FinishPointMapping="End"
ChildMapping="ChildTask"
ProgressMapping="Complete"
ResourceInfoMapping="Resource">
</sync:TaskAttributeMapping>
</sync:GanttControl.TaskAttributeMapping>
</sync:GanttControl>
```

C#

```
// Assigning the custom schedule Items Source
this.Gantt.CustomScheduleSource = this.GetInfo();
/// Gets the Numeric Schedule Items Info
private ObservableCollection<GanttScheduleRowInfo> GetInfo()
{
    // Creating a new collection
    ObservableCollection<GanttScheduleRowInfo> RowInfo = new
    ObservableCollection<GanttScheduleRowInfo>();
    // Defining the top most row of the schedule
    RowInfo.Add(new GanttScheduleRowInfo() { CellsPerUnit = 3 });
    // Defining the consecutive rows of the schedule
    RowInfo.Add(new GanttScheduleRowInfo() { CellsPerUnit = 2 });
    RowInfo.Add(new GanttScheduleRowInfo() { CellsPerUnit = 5 });
    // Defining the bottom most row of the schedule
    // Here we are setting the cell width in pixels
    RowInfo.Add(new GanttScheduleRowInfo() { PixelsPerUnit = 30d });
    return RowInfo;
}
```

The following image shows Custom Schedule:



Custom Schedule

[Samples Link](#)

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under the User Interface Edition panel.
4. Select Gantt.
5. Expand the Custom Schedule item in the Sample Browser.
6. Choose the Custom Numeric Schedule sample to launch.

[Adding CustomDateTime Schedule to an Application](#)

To Add CustomDateTime Schedule to an application:

1. Define the Gantt Schedule type as CustomDateTime.
2. Bind the GanttScheduleRowInfo collection to the CustomScheduleSource property of the Gantt.

The following code illustrates this:

XML

```
<sync:GanttControl Grid.Row="1" x:Name="Gantt"
ScheduleType="CustomDateTime"
VisualStyle="Office2010Black"
ItemsSource="{Binding GanttItemSource}"
ShowChartLines="False"
ShowNonWorkingHoursBackground="False"
ToolTipTemplate="{StaticResource tooltipTemplate}">
<sync:GanttControl.TaskAttributeMapping>
<sync:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"
StartDateMapping="StartDate"
ChildMapping="ChildTask"
FinishDateMapping="EndDate"
DurationMapping="Duration"
ProgressMapping="Complete"

```

```

ResourceInfoMapping="Resource"
PredecessorMapping="Predecessor"      >
</sync:TaskAttributeMapping>
</sync:GanttControl.TaskAttributeMapping>
</sync:GanttControl>

```

C#

```

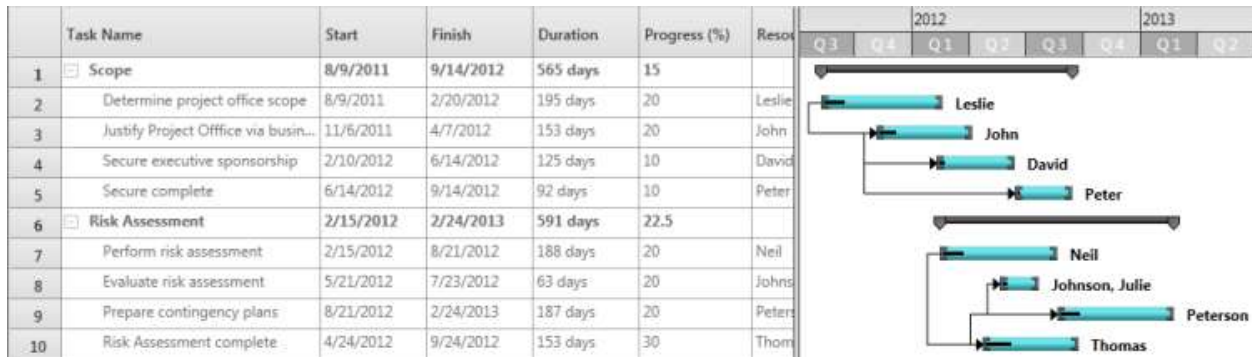
// Assigning the custom schedule Items Source.
this.Gantt.CustomScheduleSource = this.GetCustomScheduleSource();
// Hooks the schedule cell created event to customize the schedule cell
appearance.
this.Gantt.ScheduleCellCreated += new
GanttControl.ScheduleCellCreatedEventHandler(Gantt_ScheduleCellCreated);
// Gets the Custom Schedule Items Info
public IList<GanttScheduleRowInfo> GetCustomScheduleSource()
{
    List<GanttScheduleRowInfo> RowInfo = new List<GanttScheduleRowInfo>();
    // Defining the top most row of the schedule
    // Here we need the Year Schedule in this row. So we are defining the
    TimeUnit as years
    RowInfo.Add(new GanttScheduleRowInfo()
    {
        TimeUnit = TimeUnit.Years,
        CellsPerUnit = 1,
        HorizontalAlignment = HorizontalAlignment.Left
    });
    // Defining the bottom most row of the schedule
    // Here we need to display the three months in a cell so we are defining
    TimeUnit in months, and cells per Unit as 3
    // Bottom Most row should consist information about the pixels per Unit, so
    we define the pixels per unit as 15 (here this is a one month width).
    RowInfo.Add(new GanttScheduleRowInfo()
    {
        TimeUnit = TimeUnit.Months,
        CellsPerUnit = 3,
        PixelsPerUnit = 15
    });
    return RowInfo;
}

/// Handles the Schedule cell Created Event of the Gantt
void Gantt_ScheduleCellCreated(object sender, ScheduleCellCreatedEventArgs
args)
{
    DateTime currentDate = args.CurrentCell.CellDate;
    if (args.CurrentCell.CellTimeUnit == TimeUnit.Months)
    {
        args.CurrentCell.Foreground = new SolidColorBrush(Colors.White);
        // Quarter 1 dates contain months below 3 as we are checking the cell date
        and changing the content of the cell.
        if (currentDate.Month <= 3)
        {
            args.CurrentCell.Content = "Q 1";
            args.CurrentCell.CellToolTip = "Quarter 1";
            args.CurrentCell.Background = new SolidColorBrush(Colors.DarkGray);
        }
    }
}

```

```
// Quarter 2 dates contain months between 4 - 6 as we are checking the cell
// dates and changing the content of the cell.
else if (currentDate.Month > 3 && currentDate.Month <= 6)
{
    args.CurrentCell.Content = "Q 2";
    args.CurrentCell.CellToolTip = "Quarter 2";
    args.CurrentCell.Background = new SolidColorBrush(Colors.LightGray);
}
// Quarter 3 dates contains months between 6 - 9 as we are checking the cell
// date and changing the Content of the cell.
else if (currentDate.Month > 6 && currentDate.Month <= 9)
{
    args.CurrentCell.Content = "Q 3";
    args.CurrentCell.CellToolTip = "Quarter 3";
    args.CurrentCell.Background = new SolidColorBrush(Colors.DarkGray);
}
// Quarter 4 dates contains months between 9 - 12. So we are checking the
// cell date and changing the content of the cell.
else if (currentDate.Month > 9 && currentDate.Month <= 12)
{
    args.CurrentCell.Content = "Q 4";
    args.CurrentCell.CellToolTip = "Quarter 4";
    args.CurrentCell.Background = new SolidColorBrush(Colors.LightGray);
}
}
```

The following image shows Custom DateTime Schedule:



Custom DateTime Schedule

[Samples Link](#)

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel.

4. Select Gantt.
5. Expand the Custom Schedule item in the Sample Browser.
6. Choose the Customized Schedule Appearance sample to launch.

ScheduleCellCreatedEventArgs Class

The ScheduleCellCreatedEventArgs consists of the current schedule cell in the name of CurrentCell. It is the GanttScheduleCell type.

Calendar Customization in WPF Gantt

This feature allows you to set your own schedule for the entire project. Using this feature you can customize the calendar as per the organization's requirement.

Use Case Scenarios

You can use this feature when you want to change the schedule as needed. For example, if April to March is your financial year, you can set this as your fiscal year and schedule the tasks accordingly.

You can also use this to schedule the works that have different week cycle. For example if your organization follows the week cycle from Wednesday to Tuesday, you can achieve this using calendar Customization feature.

Properties

Property	Description	Type	Data Type	Reference links
WeekBeginsOn	Gets or sets the starting day of a week in the project schedule. By default it is set to Sunday.	DependencyProperty	Enum	N/A
FiscalYearBeginsOn	Gets or sets the starting month of a fiscal year. By default it is set to January	DependencyProperty	Enum	N/A
IsFYNumberingEnabled	Gets or sets the Fiscal Year Numbering. If this property is changed, it will be reflected in the schedule. By default FY Numbering is set to false.	Dependency Property	bool	N/A
DefaultStartTime	Gets or sets the task starting time in a day. This is based on the <i>GanttTime</i> class of the Gantt control. By	Dependency Property	GanttTime	N/A

	default this is set to 9.00 AM			
DefaultEndTime	Gets or sets the task ending time in a day. This is based on the <i>GanttTime</i> class of the Gantt control. By default this is set to 6.00 PM	Dependency Property	GanttTime	N/A
Weekends	Gets or sets the weekend days. Default value is Saturday, Sunday.	Dependency Property	Days	N/A
ShowWeekends	Enables or disables the weekend days in schedule. Default value is true.	Dependency Property	bool	N/A
ExcludeWeekends	Excludes or includes the weekend days in duration calculation. Default value is false.	Dependency Property	bool	N/A
ShowNonWorkingHoursBackground	Enables or disables the background for weekend days. Default value is true.	Dependency Property	bool	N/A

Note: Currently DefaultStartTime and DefaultEndTime will reflect only in the Chart Background Panel.

Adding Calendar Customization to an Application

Define the value to weekdays, months, FY Numbering, default start time and default end time as required and assign it to the appropriate APIs in the Gantt.

The following code illustrates adding Calendar Customization to an Application:

XML

```
<!--Gantt Control Calendar Customization and Weekends Customization Info-->
<sync:GanttControl x:Name="Gantt" Grid.Row="1" WeekBeginsOn="Friday"
Weekends="Wednesday,Thursday"
ItemsSource="{Binding TaskCollection}"
FiscalYearBeginsOn="June"
IsFYNumberingEnabled="True" DefaultEndTime="16:00:00"
ShowWeekends="True" ExcludeWeekends="True"
DefaultStartTime="10:00:00"
ShowNonWorkingHoursBackground="True">
<sync:GanttControl.TaskAttributeMapping>
<sync:TaskAttributeMapping TaskIdMapping="Id"
```

```

TaskNameMapping="Name"
StartDateMapping="StartDate"
ChildMapping="ChildTask"
FinishDateMapping="EndDate"
DurationMapping="Duration"
ProgressMapping="Complete"
PredecessorMapping="Predecessor">
</sync:TaskAttributeMapping>
</sync:GanttControl.TaskAttributeMapping>
</sync:GanttControl>

```

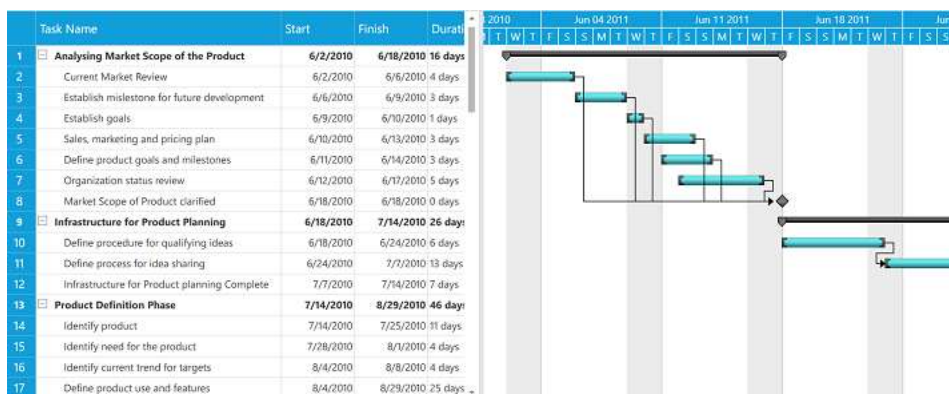
C#

```

//To set WeekBeginsOn
Gantt.WeekBeginsOn = DayOfWeek.Friday;
//To set FY Numbering
Gantt.IsFYNumberingEnabled = true;
//To set FiscalYearBeginsOn
Gantt.FiscalYearBeginsOn = Month.June;
//To set DefaultStartTime
Gantt.DefaultStartTime = new TimeSpan(10,0,0);
//To set DefaultEndTime
Gantt.DefaultEndTime = new TimeSpan(16, 0, 0);
//To set Weekends
Gantt.Weekends = Days.Wednesday | Days.Thursday;
//To set ShowWeekends
Gantt.ShowWeekends = true;
//To set ExcludeWeekends
Gantt.ExcludeWeekends = true;
//To set ShowNonWorkingHoursBackground
Gantt.ShowNonWorkingHoursBackground = true;
// To Set FY Numbering
Gantt.IsFYNumberingEnabled = true;

```

The following image shows Customized Calendar and Weekends:



Customized Calendar

[Samples Link](#)

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel.
4. Select Gantt.
5. Expand the Interactive Features item in the Sample Browser.
6. Choose the Calendar Customization sample to launch.

Holidays Customization in WPF Gantt

The GanttControl allows customers to differentiate the dates of holidays. This support is used to highlight the holidays in the Gantt chart.

The holidays can be specified using the **Holidays** property in the SfGantt, which holds a collection of GanttHoliday.

The following properties in the GanttHoliday are used to define the holidays:

- **Day**: Specifies the holiday date.
- **Background**: Specifies the brush to highlight the holiday.

Note: If the background is not defined, then the Non-WorkingDays background will be applied.

The following properties in the GanttControl are used to customize the holiday feature:

- **ShowHolidays**: Indicates whether to enable or disable the holidays in the Gantt chart.
- **ExcludeHolidays**: Indicates whether to include or exclude the holidays in duration calculation.

The following code sample demonstrates how to define the holidays.

XML

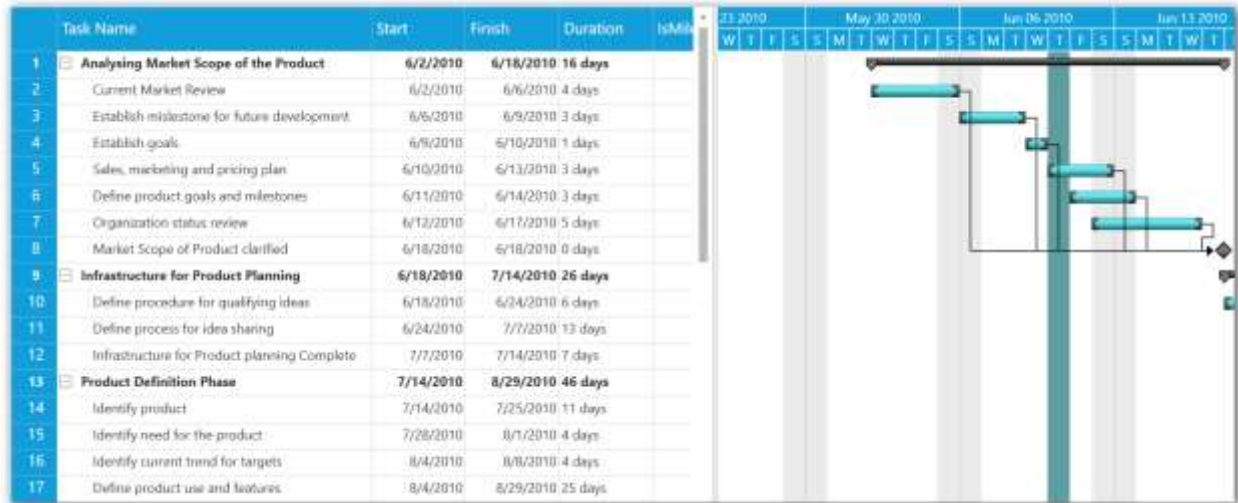
```
<!-- GanttControl Information -->
<sync:GanttControl x:Name="Gantt"
ItemsSource="{Binding TaskCollection}"
VisualStyle="Metro"
ShowHolidays="True"
ExcludeHolidays="False">
  <sync:GanttControl.Holidays>
    <sync:GanttHolidayCollection>
      <sync:GanttHoliday Day="6/10/2010" Background="CadetBlue"/>
      <sync:GanttHoliday Day="7/8/2010" Background="CadetBlue"/>
      <sync:GanttHoliday Day="8/3/2010" Background="CadetBlue"/>
      <sync:GanttHoliday Day="9/20/2010" Background="CadetBlue"/>
    </sync:GanttHolidayCollection>
  </sync:GanttControl.Holidays>
  <sync:GanttControl.TaskAttributeMapping>
    <sync:TaskAttributeMapping TaskIdMapping="TaskId"
TaskNameMapping="TaskName"
StartDateMapping="StartDate"
ChildMapping="Child">
```

```
FinishDateMapping="FinishDate"
DurationMapping="Duration"
MileStoneMapping="IsMileStone"
ProgressMapping="Progress"
PredecessorMapping="Predecessor"/>
</sync:GanttControl.TaskAttributeMapping>
</sync:GanttControl>
```

C#

```
public partial class MainWindow : Window
{
    ViewModel viewModel;
    /// <summary>
    /// Initializes a new instance of the <see cref="MainWindow"/> class.
    /// </summary>
    public MainWindow()
    {
        InitializeComponent();
        viewModel = new ViewModel();
        GanttControl gantt = new GanttControl();
        gantt.ItemsSource = this.viewModel.TaskCollection;
        gantt.VisualStudio = VisualStyle.Metro;
        gantt.ShowHolidays = true;
        gantt.ExcludeHolidays = false;
        gantt.Holidays = new GanttHolidayCollection
        {
            new GanttHoliday { Day = new DateTime(2010, 6, 10), Background =
                Brushes.CadetBlue },
            new GanttHoliday { Day = new DateTime(2010, 7, 8), Background =
                Brushes.CadetBlue },
            new GanttHoliday { Day = new DateTime(2010, 8, 3), Background =
                Brushes.CadetBlue },
            new GanttHoliday { Day = new DateTime(2010, 9, 20), Background =
                Brushes.CadetBlue }
        };
        this.AddChild(gantt);
    }
}
```

The following screenshot illustrates the customized holidays sample.



You can download the holiday customization sample from the following link:

[Holiday customization sample.](#)

Zooming in WPF Gantt

Zooming allows you to zoom in and zoom out of the schedule and chart of the Gantt control between year and minute time units. On zooming, the schedule's time unit and schedule cell size will be dynamically changed based on the zoom factor. You can increase/decrease the width of the tasks in the chart based on the schedule's time unit. Zooming can be categorized into two types:

- Built-in zooming
- Custom zooming

Built-in Zooming

Built-in zooming allows you to zoom in and zoom out of the schedule rows. The built-in zoom options are handled by the Gantt control. The zoom operations take place dynamically based on the following zoom factors:

- Zoom-in: Zoom-in increases the cell size dynamically. When the cell size exceeds the specified range, it will split the schedule cells. When a cell cannot be split further, a new schedule row with the next time unit is added beneath the last row of the current schedule. The cell split-up and adding a new row inclusion is based on the zoom factor.
- Zoom-out: Zoom-out decreases the cell size dynamically. When the cell size is within the specified range, it will merge the schedule cells. When the cells cannot be merged further, the last row of the current schedule is removed. Merging and removing a row is based on the zoom factor.

Adding Built-in Zooming to an Application

To add the built-in zooming:

1. Define the Gantt with initial values.
2. Set the UseOnDemandSchedule API value as *true*. If need set BaseCellMinLength and BaseCellMaxLength, the default value of these APIs are 20 and 40 respectively.

- Use a slider or any control to provide the zoom factor dynamically. Bind the Gantt's zoom factor to that control value.

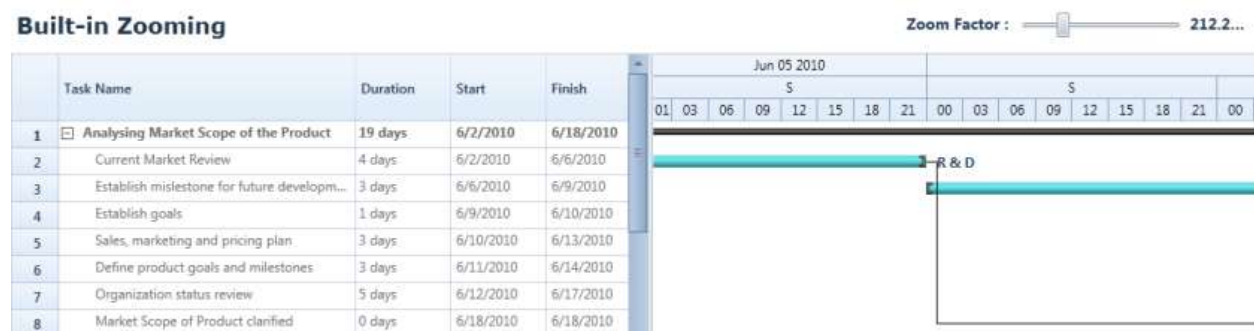
XML

```
<Slider Minimum="80" Maximum="600" Value="100" x:Name="ZoomSlider" Width="150"/>
<sync:GanttControl x:Name="Gantt"
ItemsSource="{Binding GanttItemSource}"
UseOnDemandSchedule="True"
ZoomFactor="{Binding ElementName=ZoomSlider, Path=Value}"/>
```

C#

```
/// Defining the Slider
Slider slider = new Slider();
slider.Minimum = 1;
slider.Maximum = 600;
//Hooking the value changed event of the slider
slider.ValueChanged += slider_ValueChanged;
//Defining the Gantt
GanttControl Gantt = new GanttControl();
Gantt.ItemsSource = view.GanttItemSource;
Gantt.UseOnDemandSchedule = true;
/// <summary>
/// Handles the ValueChanged event of the slider control.
/// </summary>
void slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    //Changing the value of zoom factor
    this.Gantt.ZoomFactor = (sender as Slider).Value;
}
```

The following image shows Built-in Zooming in Gantt:



Samples Link

To view samples:

- Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under the User Interface Edition panel.
4. Select Gantt.
5. Expand the Interactive Features item in the Sample Browser.
6. Choose the Built-in Zooming to launch the sample.

Custom Zooming

In custom zooming, everything is handled at the application level. You can change the schedule row information and cell size dynamically by handling the provided event handler. While the zooming is handled at the application level, built-in zooming will not work. You cannot use built-in zooming and custom zooming together.

Use Case Scenarios

When you like to view tasks that are scheduled on a month unit in the day/hours unit, you have to restart the application by changing the schedule type to day/hours unit. Zooming allows you to easily zoom in on the day/hours unit by providing the zoom factor without restarting the application.

Adding Custom Zooming to an Application

To add custom zooming:

1. Define the Gantt with initial values.
2. Set the UseOnDemandSchedule API value as true.
3. If needed, set BaseCellMinLength and BaseCellMaxLength. The default value of these APIs are 20 and 40 respectively.
4. Use any control to provide the zoom factor dynamically. Bind the Gantt's zoom factor to that control value.
5. Handle the ZoomChanged event handler in code behind and change the schedule row information in that event handler as illustrated in the following code example:

XML

```
<ComboBox x:Name="ZoomBox"
Width="75"
ItemsSource="{Binding ZoomFactors}"
SelectedItem="{Binding ZoomFactor}"/>
<sync:GanttControl x:Name="Gantt"
ItemsSource="{Binding GanttItemSource}"
UseOnDemandSchedule="True"
ZoomFactor="{Binding ZoomFactor}"
ZoomChanged="Gantt_ZoomChanged"/>
```

C#

```
/// APIs in View Model.
private List<double> _zoomFactors = new List<double> { 100d, 200d, 300d, 400d, 600d, 800d, 1000d };
/// <summary>
```

```

/// Gets or sets the zoom factors.
/// </summary>
/// <value>The zoom factors.</value>
public List<double> ZoomFactors
{
    get
    {
        return _zoomFactors;
    }
    set
    {
        _zoomFactors = value;
        this.OnPropertyChanged("ZoomFactors");
    }
}

private double _zoomFactor;
/// <summary>
/// Gets or sets the zoom factor.
/// </summary>
/// <value>The zoom factor.</value>
public double ZoomFactor
{
    get
    {
        return _zoomFactor;
    }
    set
    {
        _zoomFactor = value;
        this.OnPropertyChanged("ZoomFactor");
    }
}

/// Event Handler in code behind
/// <summary>
/// Handles the ZoomChanged event of the Gantt control.
/// </summary>
/// <param name="sender">The source of the event.</param>
/// <param name="args">The <see cref="Syncfusion.Windows.Controls.Gantt.Zoom
ChangedEventArgs"/> instance containing the event data.</param>
private void Gantt_ZoomChanged(object sender, ZoomChangedEventArgs args)
{
    if (args.ZoomFactor == 100)
    {
        args.ScheduleHeaderInfo = new List<GanttScheduleRowInfo>
        {
            new GanttScheduleRowInfo{ CellsPerUnit=1, TimeUnit= TimeUnit.Weeks },
            new GanttScheduleRowInfo{CellsPerUnit=1,TimeUnit = TimeUnit.Days, PixelsPerUnit= 20}
        };
    }
    else if (args.ZoomFactor == 200)
    {
        args.ScheduleHeaderInfo = new List<GanttScheduleRowInfo>
        {
            new GanttScheduleRowInfo{ CellsPerUnit=1, TimeUnit= TimeUnit.Days, CellTextFormat = "ddd d MMM" },

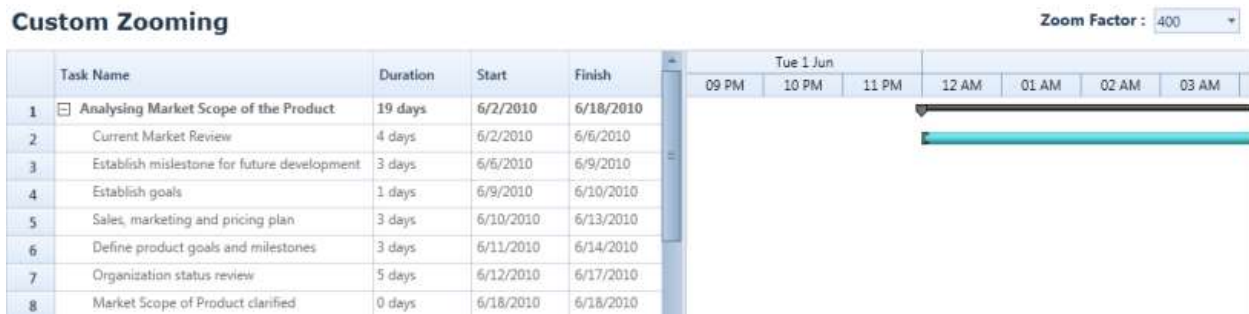
```

```

new GanttScheduleRowInfo{CellsPerUnit=12,TimeUnit = TimeUnit.Hours, PixelsPerUnit= 4, CellTextFormat="hh tt"}
};
}
else if (args.ZoomFactor == 300)
{
args.ScheduleHeaderInfo = new List<GanttScheduleRowInfo>
{
new GanttScheduleRowInfo{ CellsPerUnit=1, TimeUnit= TimeUnit.Days, CellTextFormat ="ddd d MMM" },
new GanttScheduleRowInfo{CellsPerUnit=6,TimeUnit = TimeUnit.Hours, PixelsPerUnit= 8, CellTextFormat="hh tt"}
};
}
else if (args.ZoomFactor == 400)
{
args.ScheduleHeaderInfo = new List<GanttScheduleRowInfo>
{
new GanttScheduleRowInfo{ CellsPerUnit=1, TimeUnit= TimeUnit.Days, CellTextFormat ="ddd d MMM" },
new GanttScheduleRowInfo{CellsPerUnit=1,TimeUnit = TimeUnit.Hours, PixelsPerUnit= 69, CellTextFormat="hh tt"}
};
}
else if (args.ZoomFactor == 600)
{
args.ScheduleHeaderInfo = new List<GanttScheduleRowInfo>
{
new GanttScheduleRowInfo{ CellsPerUnit=1, TimeUnit= TimeUnit.Hours, CellTextFormat="hh:mm tt" },
new GanttScheduleRowInfo{CellsPerUnit=15,TimeUnit = TimeUnit.Minutes, PixelsPerUnit= 2}
};
}
else if (args.ZoomFactor == 800)
{
args.ScheduleHeaderInfo = new List<GanttScheduleRowInfo>
{
new GanttScheduleRowInfo{ CellsPerUnit=1, TimeUnit= TimeUnit.Hours, CellTextFormat="hh:mm tt" },
new GanttScheduleRowInfo{CellsPerUnit=5,TimeUnit = TimeUnit.Minutes, PixelsPerUnit= 4}
};
}
else if (args.ZoomFactor == 1000)
{
args.ScheduleHeaderInfo = new List<GanttScheduleRowInfo>
{
new GanttScheduleRowInfo{ CellsPerUnit=1, TimeUnit= TimeUnit.Hours, CellTextFormat="hh:mm tt" },
new GanttScheduleRowInfo{CellsPerUnit=1,TimeUnit = TimeUnit.Minutes, PixelsPerUnit= 20}
};
}
args.Handled = true;
}

```

The following image shows Custom Zooming in Gantt:



[Samples Link](#)

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under the User Interface Edition panel.
4. Select Gantt.
5. Expand the Interactive Features item in the Sample Browser.
6. Choose the Custom Zooming to launch the sample.

Tables for Zooming Properties and Events

[Properties](#)

Property	Description	Type	Data Type
UseOnDemandSchedule	To get the expected performance you need to use dynamic schedule rendering.	Dependency	boolean
BaseCellMaxLength	Splits the schedule cells by comparing the cell size with the API value on zooming-in	Dependency	double
BaseCellMinLength	Merges schedule cells by comparing the cell size with the API value on zooming-out	Dependency	double
ZoomFactor	Zooming takes place based on the ZoomFactor. The ZoomFactor should be greater than zero.	Dependency	double

[Events](#)

Event	Description	Arguments	Type
-------	-------------	-----------	------

ZoomChanged	Event triggers when the zoom factor is changed. This can be handled in application level.	ZoomChangedEventArgs	Simple event
-------------	---	----------------------	--------------

Highlighting Tasks in WPF Gantt

Highlighting tasks allows you to highlight a specific set of tasks in the Gantt chart region. This feature will get the set of tasks as input and highlight those tasks with the specified color in the Gantt chart region. It will accept a linear set of tasks of type `IList` as input. You can also specify the highlighting brush through the provided API.

Use Case Scenario

- When you like to view some specific set of tasks with a different color, you can easily achieve that by providing that set of tasks as input.
- This feature also helps the user to highlight tasks that are in a critical path of a project.

Properties

Property	Description	Type	Data Type
HighlightedItems	The items that are passed through this API will be highlighted in Gantt chart region. The items should be in linear form.	Dependency	<code>IList</code>
HighlightItemBrush	Specifies the color in which items should be highlight	Dependency	Brush

Adding Highlighting Tasks to an Application

To highlight a set of tasks in Gantt chart region:

1. Define the Gantt with initial setup.
2. Bind the tasks that need to be highlighted with Gantt's `HighlightedItems` API.
3. If required, change the value of the Gantt's `HighlightItemBrush` API to change the item highlight color. The default color is red.

XML

```
<sync:GanttControl x:Name="Gantt"
ItemsSource="{Binding GanttItemSource}"
HighlightedItems="{Binding HighlightedTasks}">
</sync:GanttControl>
```

C#

```
/// Codes in View Model
/// <summary>
/// Initializes a new instance of the <see cref="ViewModel"/> class.
/// </summary>
public ViewModel()
{
```

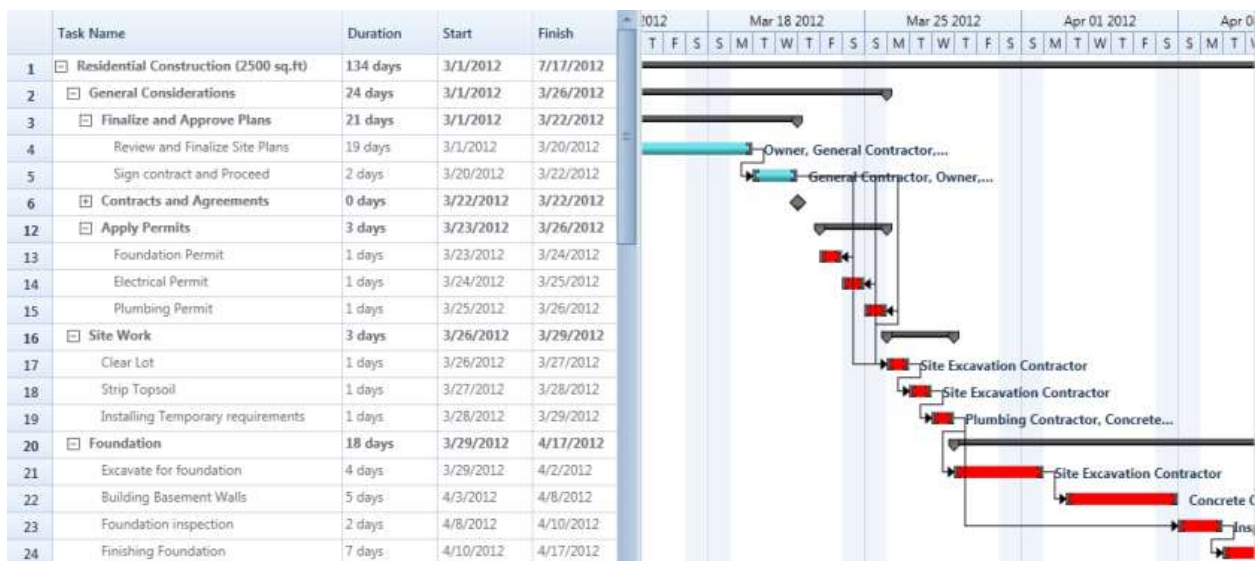
```

this.AddHighlightedTasks();
}
IList _highlightedTasks = new List<Task>();
/// <summary>
/// Gets or sets the highlighted tasks.
/// </summary>
/// <value>The highlighted tasks.</value>
public IList HighlightedTasks
{
    get
    {
        return _highlightedTasks;
    }
    set
    {
        _highlightedTasks = value;
        this.OnPropertyChanged("HighlightedTasks");
    }
}
/// <summary>
/// Adds the highlighted tasks.
/// </summary>
internal void AddHighlightedTasks()
{
    List<Task> tasks = new List<Task>();
    /// Adding the list of tasks
    tasks.AddRange(this.GanttItemSource[0].ChildTask[0].ChildTask[2].ChildTask);
    tasks.AddRange(this.GanttItemSource[0].ChildTask[1].ChildTask);
    this.HighlightedTasks = tasks;
}
/// Adding Highlighted items code behind
this.Gantt.HighlightedItems = this.view.HighlightedTasks;

```

{% endtabs %}

The following image shows the Gantt with Highlighted Tasks:



Gantt with Highlighted Tasks

[Samples Link](#)

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under the User Interface Edition panel
4. Select Gantt
5. Expand the Interactive Features item in the Sample Browser
6. Choose Highlighting Tasks to launch the sample.

see also

[How to view the gantt control with the desired date tasks at load time in WPF](#)

Custom Node Style in WPF Gantt

Custom node style enables you to design your own style to the nodes that will be displayed in the Gantt. You can also customize the progress bar of the Task Node. Currently Gantt Control supports three types of node. They are:

- Header Node
- Task Node
- Milestone

You can apply custom styles for all the three nodes. The basic functionalities of the Gantt nodes like resizing, drag and drop and tooltip are available only when the custom node style has the built-in node style's such as drag and drop the thumb and resizing the thumbs. Otherwise the custom node will work properly, but you cannot access these features of Gantt.

Use Case Scenarios

You can customize the node to give a similar look and feel of your product. For example if you are from a steel company, then you can add a style that gives steel rod like look and feel to the node.

You can also design the node based on your organization's logo.

Adding Custom Node Style to an Application

The following are the steps to add custom node style to an application:

1. Define a style as needed with the target type for each node type as given in the following table:

Custom Node Types

Node Type	Target Type
Header Node	HeaderNode

Task Node	GanttNode
Milestone	MileStone

The following code illustrates how to define style:

XML

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:gantt="http://schemas.syncfusion.com/wpf"
xmlns:chart="clr-namespace:Syncfusion.Windows.Controls.Gantt.Chart;assembly=Syncfusion.Gantt.Wpf" >
<!-- Header Node style-->
<Style TargetType="chart:HeaderNode">
<Setter Property="MaxHeight" Value="24"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="chart:HeaderNode">
<Border Background="{TemplateBinding Background}" Name="PART_HeaderBorder"
BorderBrush="{TemplateBinding BorderBrush}" BorderThickness="0"/>
<Grid Width="{TemplateBinding NodeWidth}" VerticalAlignment="Center">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="10" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="10" />
</Grid.ColumnDefinitions>
<Rectangle HorizontalAlignment="Left" Grid.Column="1" Height="6.4"
VerticalAlignment="Top"
Width="{TemplateBinding NodeWidth}"
Stroke="#FF111111">
<Rectangle.Fill>
<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
<GradientStop Color="#FF414141" Offset="0" />
<GradientStop Color="#FF6D6D6D" Offset="0.087" />
<GradientStop Color="#FF767676" Offset="0.304" />
<GradientStop Color="Black" Offset="0.957" />
<GradientStop Color="#FF343434" Offset="1" />
</LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<Path Data="M0.3,0.3 L9.834909,0.30036073 9.8351226,5.9832297 5.0695471,10.7
34966 0.32096295,5.9863821 z"
HorizontalAlignment="Left"
Grid.Column="0"
Height="11.435"
Stretch="Fill"
Stroke="#FF111111"
VerticalAlignment="Top"
Width="10.135">
<Path.Fill>
<LinearGradientBrush EndPoint="1.009,0.985" StartPoint="0.339,0.286">
<GradientStop Color="DimGray" Offset="0.008" />
<GradientStop Color="#FF7F7F7F" Offset="0.511" />
</LinearGradientBrush>
</Path.Fill>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
```

```

<GradientStop Color="#FF2A2A2A" Offset="1" />
</LinearGradientBrush>
</Path.Fill>
</Path>
<Path Data="M0.3,0.3 L9.834909,0.30036073 9.8351226,5.9832297 5.0695471,10.7
34966 0.32096295,5.9863821 z"
HorizontalAlignment="Left"
Grid.Column="2"
Height="11.435"
Stretch="Fill"
Stroke="#FF111111"
VerticalAlignment="Top"
Width="10.135">
<Path.Fill>
<LinearGradientBrush EndPoint="1.009,0.985" StartPoint="0.339,0.286">
<GradientStop Color="DimGray" Offset="0.008" />
<GradientStop Color="#FF7F7F7F" Offset="0.511" />
<GradientStop Color="#FF2A2A2A" Offset="1" />
</LinearGradientBrush>
</Path.Fill>
</Path>
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<!-- Task Node style-->
<Style x:Key="TaskNode" TargetType="{x:Type chart:GanttNode}">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type chart:GanttNode}">
<Border Name="PART_Border" Height="18" VerticalAlignment="Center" BorderThic
kness="0.5" BorderBrush="#FF4D4D4D">
<Border.Background>
<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
<GradientStop Color="#FFF6F6F6" Offset="0"/>
<GradientStop Color="#FFC5C5C5" Offset="1"/>
</LinearGradientBrush>
</Border.Background>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<!--
- To access the drag and drop feature, have the thumb with specified name --
>
<Thumb Cursor="SizeAll" x:Name="PART_DragDropThumb" Grid.Column="0" Grid.Co
lumnSpan="3">
<Thumb.Template>
<ControlTemplate>
<Border Background="Transparent"/>
</ControlTemplate>
</Thumb.Template>
</Thumb>

```

```

<Border HorizontalAlignment="Left"
VerticalAlignment="Center"
Width="{TemplateBinding ProgressWidth}"
Grid.Column="0"
Grid.ColumnSpan="3">
<Grid HorizontalAlignment="Stretch" Width="{TemplateBinding ProgressWidth}">
<Rectangle HorizontalAlignment="Stretch" Height="12"
Stroke="#FFD26202"
VerticalAlignment="Center">
<Rectangle.Fill>
<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
<GradientStop Color="#FFFEB300" Offset="0"/>
<GradientStop Color="#FFE7600" Offset="1"/>
</LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<!--
- To access the progress resizing feature, have the thumb with specified nam
e -->
<Thumb Cursor="SizeWE" x:Name="PART_ProgressThumb" HorizontalAlignment="Righ
t">
<Thumb.Template>
<ControlTemplate>
<Border Background="Transparent" BorderBrush="Transparent">
<Rectangle Height="2" Width="5" Fill="Transparent" />
</Border>
</ControlTemplate>
</Thumb.Template>
</Thumb>
</Grid>
</Border>
<!-- To access the resizing feature, have the thumb with the
specified name -->
<Thumb Cursor="Scrolle" Grid.Column="2" x:Name="PART_RightThumb" HorizontalAl
ignment="Right">
<Thumb.Template>
<ControlTemplate>
<Rectangle HorizontalAlignment="Right"
Height="20"
VerticalAlignment="Center"
Width="6"
Stroke="#FFD26202">
<Rectangle.Fill>
<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
<GradientStop Color="#FFFEB300" Offset="0"/>
<GradientStop Color="#FFE7600" Offset="1"/>
</LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
</ControlTemplate>
</Thumb.Template>
</Thumb>
<!-- To access the resizing feature, have the thumb with the
specified name -->
<Thumb Cursor="SizeWE" Grid.Column="0" x:Name="PART_LeftThumb" HorizontalAli
gnment="Left">
<Thumb.Template>

```

```

<ControlTemplate>
<Border Background="Transparent"
BorderBrush="Transparent"
BorderThickness="0"
Width="4" Height="20"/>
</ControlTemplate>
</Thumb.Template>
</Thumb>
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<!--Milestone style-->
<Style x:Key="MileStone" TargetType="chart:MileStone">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="chart:MileStone">
<Grid>
<Path Stretch="Fill"
Data="F1 M 551.156,416.878L 552.734,419.766L 555.621,421.344L 552.734,422.92
2L 551.156,425.81L 549.577,422.922L 546.69,421.344L 549.577,419.766L 551.156
,416.878 Z"
HorizontalAlignment="Left"
Height="19" Width="19"
VerticalAlignment="Center"
Stroke="#FFD26202">
<Path.Fill>
<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
<GradientStop Color="#FFFE300" Offset="0"/>
<GradientStop Color="#FFFE7600" Offset="1"/>
</LinearGradientBrush>
</Path.Fill>
</Path>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>

```

1. Add the style as a resource to the Gantt control in your application.

The following code illustrates how to add the styles to the application:

XML

```

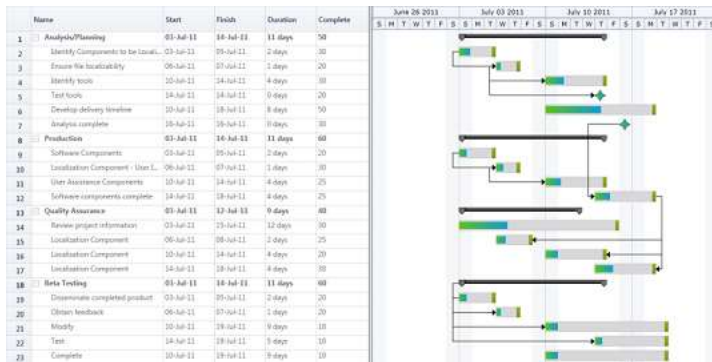
<sync:GanttControl Grid.Row="1" x:Name="Gantt"
ItemsSource="{Binding GanttItemSource}"
ToolTipTemplate="{StaticResource toolTipTemplate}"
VisualStyle="Office2010Silver">
<sync:GanttControl.TaskAttributeMapping>
<sync:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"

```

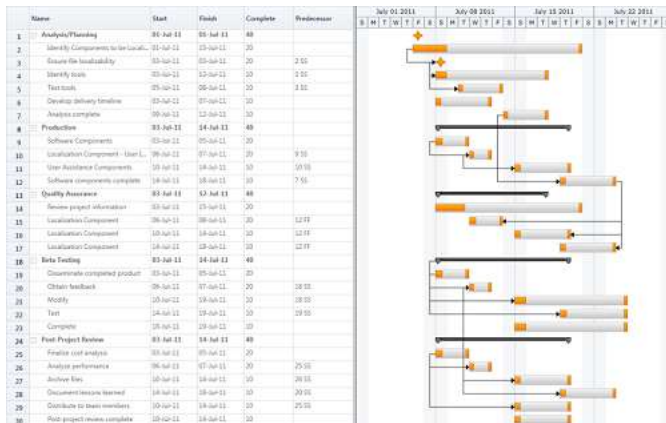
```

StartDateMapping="StartDate"
ChildMapping="ChildTask"
FinishDateMapping="EndDate"
DurationMapping="Duration"
ProgressMapping="Complete"
PredecessorMapping="Predecessor">
</sync:TaskAttributeMapping>
</sync:GanttControl.TaskAttributeMapping>
<sync:GanttControl.Resources>
<Style TargetType="chart:GanttNode" BasedOn="{StaticResource TaskNode}"/>
<Style TargetType="chart:MileStone" BasedOn="{StaticResource MileStone}"/>
</sync:GanttControl.Resources>
</sync:GanttControl>

```



Custom Node Style



Custom Node Style

Samples Link

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel.
4. Select Gantt.
5. Expand the Styles item in the Sample Browser.
6. Choose the Custom Node Style sample to launch.

Visual Style in WPF Gantt

Essential Gantt enables you to customize the appearance of the control. This supports the following themes:

- Office2010Blue
- Office2010Black
- Office2010Silver
- Metro

Properties

Property	Description	Type	Data Type	Reference links
VisualStyle	Gets or set the VisualStyle Property of Gantt control.	Dependency Property	EnumVisualStyle.Office2010BlueVisualStyle.Office2010BlackVisualStyle.Office2010SilverVisualStyle.Metro	NA

Adding VisualStyle to Gantt Control

You can customize the theme using the VisualStyle property.

The following code illustrates how to set the VisualStyle of Gantt control:

XML

```
<Sync:GanttControl x:Name="Gantt" VisualStyle="Office2010Blue"/>
```

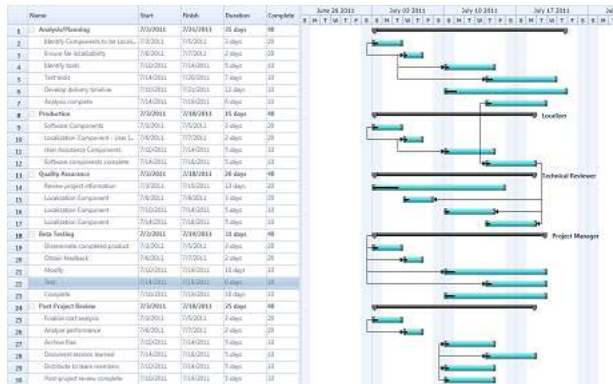
C#

```
Gantt.VisualStyle = VisualStyle.Office2010Blue;
```

The following shows Office 2010 Blue:

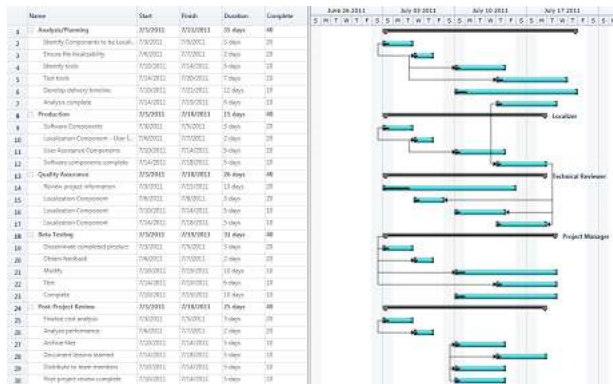
Gantt

Visual Style in WPF Gantt



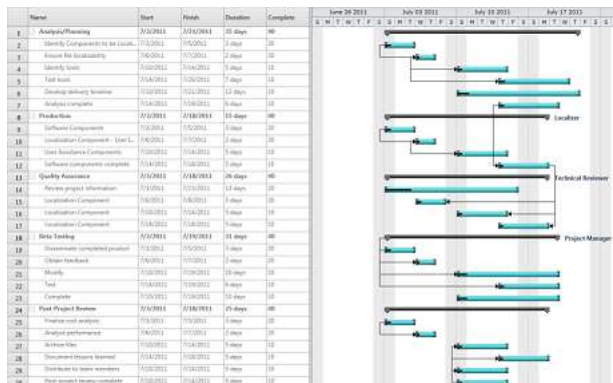
Office 2010 Blue

The following image shows Office 2010 silver:



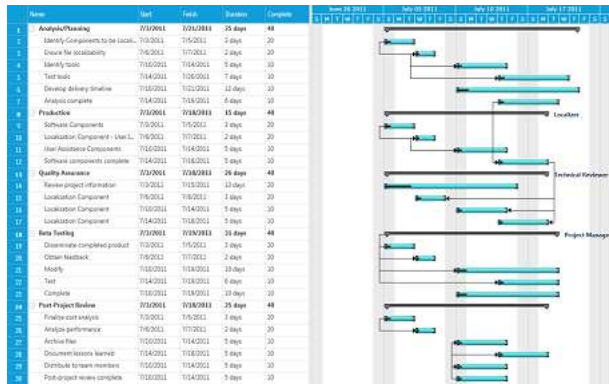
Office 2010Silver

The following image shows Office 2010Silver:



Office2010Black

The following image shows Office 2010Metro:



Metro

Samples Link

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel.
4. Select Gantt.
5. Expand the Styles item in the Sample Browser.
6. Choose the Gantt Visual Style sample to launch.

DateTime Indicator Customization in WPF Gantt

Essential Gantt provides support for indicating the current date or time in the Gantt chart region. You can choose the following positions in the Gantt chart region for the DateTime indicator:

- Today
- Loaded Time
- Dynamic Time
- Absolute

These are all included in the `CurrentDateLinePositions` enum.

Essential Gantt allows you to customize the appearance of the DateTime indicator by using the `CurrentDateLine` property.

- Today: DateTime indicator will be positioned at the current date.
- LoadedTime: DateTime indicator will be positioned at the Gantt control loaded time.
- DynamicTime: DateTime indicator will be positioned at the Gantt control loaded time. The position of the DateTime indicator will change based on the system time change.
- Absolute: The DateTime indicator will be positioned at a user-defined position.

Use Case Scenarios

1. The DateTime indicator enables you to find the current date or time in a scheduled timeline.
2. You can customize the DateTime indicator to give a similar look and feel to your product.
3. You can change the style of DateTime indicator to differentiate the DateTime indicator from other vertical lines.

Properties

Property	Description	Type	Data Type
CurrentDateLine	Get the user-defined line for the DateTime indicator.	Dependency Property	Line
StickCurrentDateLineTo	Get/sets the StickCurrentDateLineTo property of the Gantt control. By default this is set to Today.	Dependency Property	EnumNoneTodayDynamicTimeLoaded TimeAbsolute

Adding the DateTime Indicator to an Application

Adding the DateTime indicator customization will change the appearance and position of the DateTime indicator. By default, the DateTime indicator will appear in the current date. The following steps explain how to add a customized DateTime indicator:

1. Set the StickCurrentDateLineTo value for positioning the DateTime indicator. By default this is set as Today.
2. Customize the appearance of the DateTime indicator using the CurrentDateLine API in the Gantt control.

The following code samples illustrate how to customize the DateTime indicator.

XML

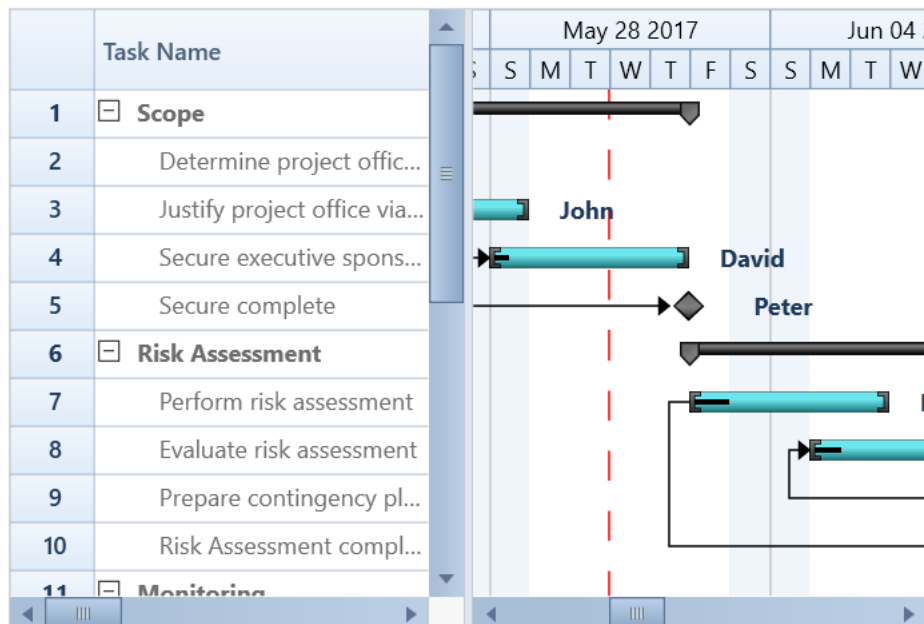
```
<gantt:GanttControl x:Name="Gantt"
Grid.Row="1"
StickCurrentDateLineTo="Today"
ItemsSource="{Binding TaskDetails}">
  <gantt:GanttControl.CurrentDateLine>
    <Line Stroke="Green"
StrokeDashArray="5 1 5"
StrokeThickness="2" />
  </gantt:GanttControl.CurrentDateLine>
  <gantt:GanttControl.TaskAttributeMapping>
    <gantt:TaskAttributeMapping TaskIdMapping="Id"
TaskNameMapping="Name"
StartDateMapping="StartDate"
ChildMapping="ChildTask"
FinishDateMapping="EndDate"
DurationMapping="Duration" />
  </gantt:GanttControl.TaskAttributeMapping>
</gantt:GanttControl>
```

C#

```
//StrokeDashArray will change the style of Line
DoubleCollection strokeArray = new DoubleCollection();
strokeArray.Add(5);
strokeArray.Add(1);
strokeArray.Add(5);
this.Gantt.CurrentDateLine.Stroke = Brushes.Green;
this.Gantt.CurrentDateLine.StrokeDashArray = strokeArray;
this.Gantt.CurrentDateLine.StrokeThickness = 1;
```

Output

The following image shows the resultant output:

**Customized DateTime Indicator***Sample Link*

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under the User Interface Edition panel.
4. Select Gantt.
5. Expand the Styles category in the Sample Browser.
6. Choose the Gantt Style Properties sample.

7. The sample contains different styles of the DateTime indicator.

Resource View Gantt Inline Items in WPF Gantt

By default, the Gantt chart will display a single node in a row. This helps you to manage the project. When you want to manage the resources in a project, you need multiple nodes in a single row. A Resource view Gantt enables you to manage the resources involved in a project.

In a normal Gantt, a node represents a task or activity of the project. In a resource view Gantt, the node represents task assigned to a resource. Multiple tasks assigned to a resource can be displayed in a single row. You can achieve this by using the mapping attribute of the `InLineTaskMapping`.

Essential Gantt will listen to the dynamic inclusion of inline items and refresh the chart region.

The following code illustrates how to add inline items dynamically to the underlying collection.

C#

```
//To add dynamic inline items
Item item = new Item()
{
    StartDate = new DateTime(2012, 01, 19),
    FinishDate = new DateTime(2012, 01, 25)
};
viewModel.GanttItemSource[0].SubItems[0].InLineItems.Add(item);
```

Use Case Scenarios

A resource view is very useful when you need to manage the resources in a project.

Example: A very big development project that contains multiple phases to complete the project. The management needs to track the status of the project. In this scenario, they can manage the project with a Resource view Gantt easily. The Resource view Gantt will show all the tasks in a single row that are assigned to a particular resource.

Adding Inline Items to an Application

You can populate a resource view Gantt by populating the collection of tasks in a single row by mapping the corresponding field in the underlying source to the `InLineTaskMapping`.

You can populate a Resource view Gantt for both date-time schedules and numeric schedules.

Resource View Gantt in a Custom Numeric Schedule

To populate a Resource view Gantt in a Custom Numeric Schedule:

1. Define the Gantt with a custom numeric schedule source. For more information about custom numeric schedules, visit the following link:
2. You can populate the collection of tasks in a single row by mapping the corresponding field in the underlying source to the `InLineTaskMapping`.

The following code illustrates this.

XML

```
<gantt:GanttControl x:Name="Gantt"
    Grid.Row="1"
    ScheduleType="CustomNumeric"
```

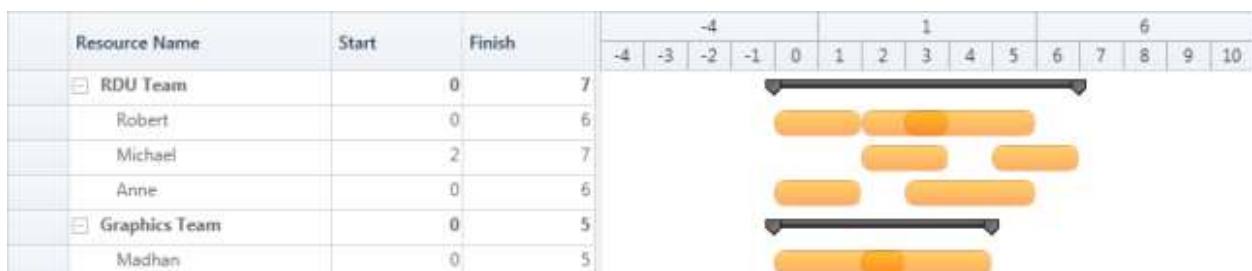
```
CustomScheduleSource="{Binding CustomScheduleInfo}"
ItemsSource="{Binding TeamDetails}">
<gantt:GanttControl.TaskAttributeMapping>
<gantt:TaskAttributeMapping TaskNameMapping="Name"
ChildMapping="SubItems"
StartPointMapping="Start"
FinishPointMapping="End"
InLineTaskMapping="InLineItems"/>
</gantt:GanttControl.TaskAttributeMapping>
</gantt:GanttControl>
```

The following code is the sample data source of a resource view Gantt in a custom numeric schedule.

C#

```
ObservableCollection<Item> teams = new ObservableCollection<Item>();
teams.Add(new Item() { Name = "RDU Team" });
Item Person = new Item() { Name = "Robert" };
Person.InLineItems.Add(new Item() { Start = 0, End=2, Name = "Market
Analysis", Progress = 50d });
Person.InLineItems.Add(new Item() { Start = 2, End=4, Name = "Competitor
Analysis", Progress = 20d });
Person.InLineItems.Add(new Item() { Start = 3, End = 6, Name = "Design Spec"
});
teams[0].SubItems.Add(Person);
Person = new Item() { Name = "Michael" };
Person.InLineItems.Add(new Item() { Start = 2, End = 4, Name = "Basic
Requirement Analysis", Progress = 40 });
Person.InLineItems.Add(new Item() { Start = 5, End = 7, Name = "Requirement
Spec" });
teams[0].SubItems.Add(Person);
Person = new Item() { Name = "Anne" };
Person.InLineItems.Add(new Item() { Start = 0, End = 2, Name = "Estimation",
Progress = 30 });
Person.InLineItems.Add(new Item() { Start = 3, End = 6, Name = "Budget &
Plan Spec" });
teams[0].SubItems.Add(Person);
teams.Add(new Item() { Name = "Graphics Team" });
Person = new Item() { Name = "Madhan" };
Person.InLineItems.Add(new Item() { Start = 0, End = 3, Name = "Identifying
UI modules", Progress = 40 });
Person.InLineItems.Add(new Item() { Start = 2, End = 5, Name = "Defining UI
Design" });
teams[1].SubItems.Add(Person);
```

The following shows the Resultant output:



Resource View Gantt with a Date-Time Schedule

To populate the Resource view Gantt with a date-time schedule:

1. Define the Gantt with DateTime values.
2. Populate the collection of tasks in a single row by mapping the corresponding field in the underlying source to the InLineTaskMapping.

The following code sample illustrates this:

XML

```
<gantt:GanttControl Grid.Row="1" x:Name="Gantt">
  <gantt:GanttControl.TaskAttributeMapping>
    <gantt:TaskAttributeMapping
      TaskNameMapping="Name"
      StartDateMapping="StartDate"
      ChildMapping="SubItems"
      FinishDateMapping="FinishDate"
      InLineTaskMapping="InLineItems">
    </gantt:TaskAttributeMapping>
  </gantt:GanttControl.TaskAttributeMapping>
</gantt:GanttControl>
```

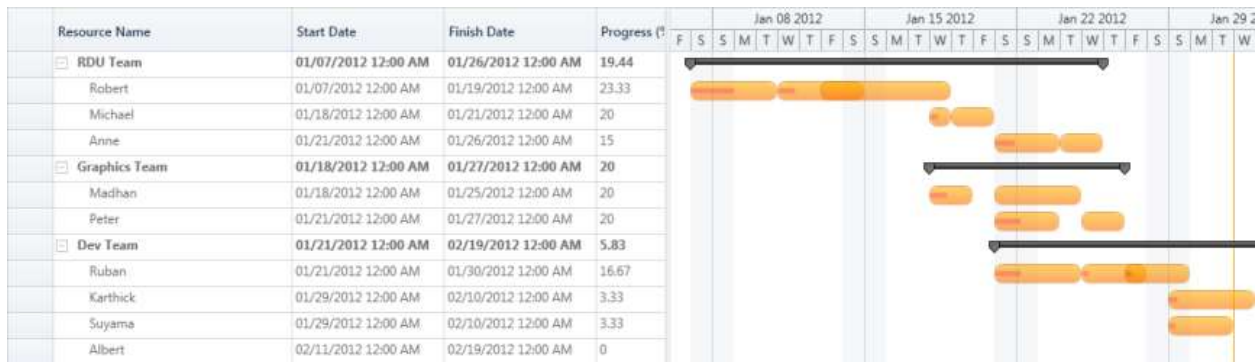
The following is the sample data source for the resource view Gantt:

C#

```
ObservableCollection<Item> teams = new ObservableCollection<Item>();
teams.Add(new Item() { Name = "RDU Team" });
Item Person = new Item() { Name = "Robert" };
Person.InLineItems.Add(new Item() { StartDate = new DateTime(2012, 01, 07),
FinishDate = new DateTime(2012, 01, 11), Name = "Market Analysis", Progress
= 50d });
Person.InLineItems.Add(new Item() { StartDate = new DateTime(2012, 01, 11),
FinishDate = new DateTime(2012, 01, 15), Name = "Competitor Analysis",
Progress = 20d });
Person.InLineItems.Add(new Item() { StartDate = new DateTime(2012, 01, 13),
FinishDate = new DateTime(2012, 01, 19), Name = "Design Spec" });
teams[0].SubItems.Add(Person);
Person = new Item() { Name = "Michael" };
Person.InLineItems.Add(new Item() { StartDate = new DateTime(2012, 01, 18),
FinishDate = new DateTime(2012, 01, 19), Name = "Basic Requirement
Analysis", Progress = 40 });
Person.InLineItems.Add(new Item() { StartDate = new DateTime(2012, 01, 19),
FinishDate = new DateTime(2012, 01, 21), Name = "Requirement Spec" });
teams[0].SubItems.Add(Person);
Person = new Item() { Name = "Anne" };
Person.InLineItems.Add(new Item() { StartDate = new DateTime(2012, 01, 21),
FinishDate = new DateTime(2012, 01, 24), Name = "Estimation", Progress = 30
});
Person.InLineItems.Add(new Item() { StartDate = new DateTime(2012, 01, 24),
FinishDate = new DateTime(2012, 01, 26), Name = "Budget & Plan Spec" });
teams[0].SubItems.Add(Person);
```

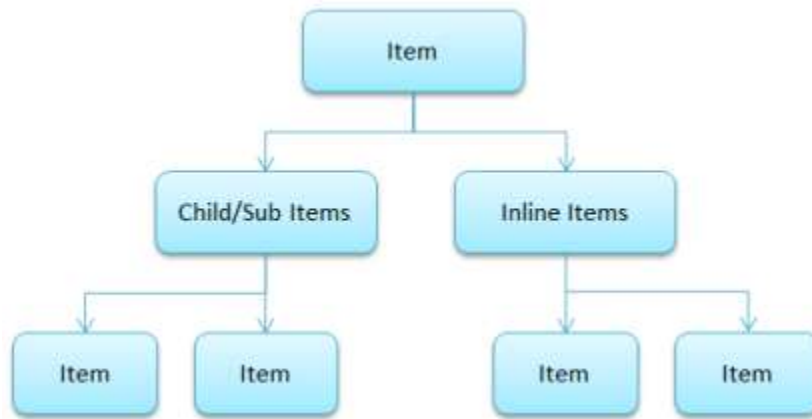
Output

The following image shows the resultant output:



Data Structure

The following is the data structure used to build a Resource view Gantt:



- team holds information about the team.
- SubItems of Team will hold the list of Resources in that particular team.
- InLineItems of each Resource will hold the tasks assigned to the particular resource.

Information Displayed in Gantt

Grid Region: The grid will display only the information about the team and its resources (SubItems). It will not display the information about assigned tasks (InLineItems).

Chart Region: The chart will display only the information about the team and the tasks assigned to each resource in the team (InLineItems). It will not display the information about resources (SubItems).

Sample Link

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel.
4. Select Gantt.
5. Expand the Data Binding item in the Sample Browser.
6. Choose the Resource View Gantt sample to launch.

Strip Lines in WPF Gantt

The Gantt provides support to add strip lines in the Gantt chart region that denotes an important event in a sequential timeline. By using this feature, you can add strip lines to highlight the important days in your project. You can add a collection of strip lines using the provided API.

Strip lines in Essential Gantt support the following features:

Strip lines can be repeatable in the Gantt chart region based on repeat behavior and repeat interval.

- You can modify the content or appearance of the strip lines at run time by changing the values of the underlying collection source.
- The visibility of strip lines can be toggled using the [ShowStripLines](#) property in the Gantt control.

The Gantt control will get the information from the application to draw the strip lines. Gantt will accept the strip line information in the form of a collection of `StripLineInfo` objects and process it to draw the strip lines.

Repeat behavior

The available repeat behaviors are as follows:

- Year
- Month
- Week
- Day
- Hour
- Minute

Style selector

It used to pass the style of the strip lines dynamically. Based on constraints.

Template selector

It used to pass the content template of the strip lines dynamically based on constraints.

Types of strip lines

There are two types of strip lines available in Essential Gantt. They are:

- Regular
- Absolute—Absolute type will place the strip line at any user-defined point.

Properties

Property	Description	Type	Data Type
Background	Gets/sets background color of strip line.	CLR	Brush

Content	Gets/sets the content of the strip line.	CLR	Object
ContentTemplate	Gets/sets the content template of the strip line.	CLR	DataTemplate
ContentTemplateSelector	Gets/sets the TemplateSelector of the strip line.	CLR	DataTemplateSelector
StartDate	Gets/sets the start date of the strip line.	CLR	DateTime
EndDate	Gets/sets the end date of the strip line.	CLR	DateTime
RepeatBehavior	Gets/sets the repeat behavior of the strip line.	CLR	Repeat (Enum)
RepeatFor	Gets/sets the intervals between the repeating strip lines.	CLR	Integer
RepeatUpto	Gets/sets DateTime value. The strip line will be repeated up to this value.	CLR	DateTime
Style	Gets/sets the style for the strip line.	CLR	Style
StyleSelector	Gets/sets the style selector of the strip line.	CLR	StyleSelector
VerticalContentAlignment	Gets/sets the vertical alignment of the content present in the strip line.	CLR	VerticalAlignment
HorizontalContentAlignment	Gets/sets the horizontal alignment of the content present in the strip line.	CLR	Horizontal Alignment
Type	Gets/sets the type of the strip line.	CLR	StriplineType(Enum)
Position	Gets/sets the absolute position of the strip line for Absolute strip line type.	CLR	Point
Height	Gets/sets the absolute height of the strip line for Absolute strip line type.	CLR	Double
Width	Get/sets the absolute width of the strip line for Absolute strip line type.	CLR	Double

Use Case Scenarios

- You can mark the important dates and meetings in the scheduled time line.
- Strip lines help you to avoid missing important events.

Properties

Property	Description	Type	Data Type
ShowStripLines	Get the user option to show the strip lines.	Dependency Property	Bool

StripLines	Get/sets the collection of StripLineInfo from the user.	Dependency Property	IEnumerable
------------	---	---------------------	-------------

Enums

Property	Description
Repeat	This property contains the following values:Year: Repeating the strip line on a yearly basis depends on the RepeatFor value in StripLineInfo.Month: Repeating the strip line on a monthly basis depends on the RepeatFor value in StripLineInfo.Week: Repeating the strip line on a weekly basis depends on the RepeatFor value in StripLineInfo.Day: Repeating the strip line on a daily basis depends on the RepeatFor value in StripLineInfo.Hour: Repeating the strip line on an hourly basis depends on the RepeatFor value in StripLineInfo.Minute: Repeating the strip line on per-minute basis depends on the RepeatFor value in StripLineInfo.
StriplineType	This property contains the following values:Regular: This denotes the normal strip line.Absolute: This denotes the absolute strip line. You can customize the position, size, and appearance of the strip line in this type.

Events

By handling its event, you can customize the strip lines dynamically.

Event	Description	Arguments	Type
StripLineCreated	Whenever a strip line is created, this event will be triggered. the handler of the event will have the newly created strip line (StripLineInfo) in the argument.By handling this event, you can customize the appearance of the strip line.	StripLineCreated(object sender, StriplineCreatedEventArgs args)	Event

Adding strip lines to application

Regular strip lines

The following code sample demonstrates how to define a collection of regular strip lines.

C#

```
StripCollection = new List<StripLineInfo>();
//Getting the collection of StripLineInfo
StripCollection = GetStripCollection();
//Method will return the collection StripLineInfo
private List<StripLineInfo> GetStripCollection()
{
    List<StripLineInfo> stripCollection = new List<StripLineInfo>();
    stripCollection.Add(new StripLineInfo()
    {
        Content = "Weekly Team Meeting",
        StartDate = new DateTime(2012, 6, 4),
```

```

EndDate = new DateTime(2012, 6, 4),
HorizontalContentAlignment = HorizontalAlignment.Center,
VerticalContentAlignment = VerticalAlignment.Center,
Background = Brushes.Gold, RepeatBehavior = Repeat.Week, RepeatFor = 1,
RepeatUpto = new DateTime(2012, 12, 10),
});
return stripCollection;
}

```

The following code sample demonstrates how to bind the regular strip line collection to strip lines.

XML

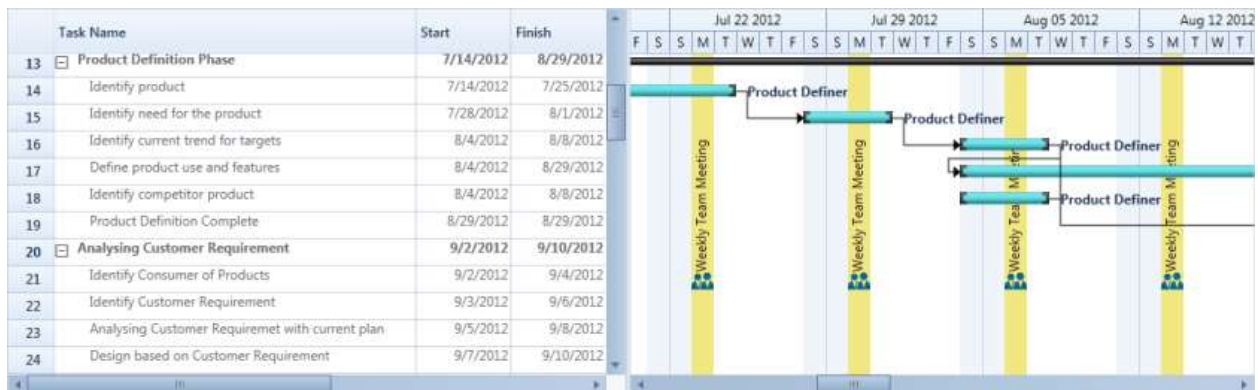
```

<sync:GanttControl x:Name="Gantt"
Grid.Row="1"
ShowStripLines="True"
StripLines="{Binding StripCollection}">
<sync:GanttControl.TaskAttributeMapping>
<sync:TaskAttributeMapping TaskIdMapping="TaskId"
TaskNameMapping="TaskName"
StartDateMapping="StartDate"
FinishDateMapping="FinishDate"
ChildMapping="Child"
DurationMapping="Duration"
ProgressMapping="Progress"
PredecessorMapping="Predecessor"
ResourceInfoMapping="Resources"/>
</sync:GanttControl.TaskAttributeMapping>
</sync:GanttControl>

```

Output

The following screenshot illustrates how to render the regular strip lines.



Strip lines in the Gantt chart

Absolute Strip lines

The following code sample demonstrates how to define a collection of absolute strip lines.

C#

```

StripCollection = new List<StripLineInfo>();
//Getting the collection of StripLineInfo
StripCollection = GetStripCollection();

```

```
//Method will return the collection StripLineInfo
private List<StripLineInfo> GetStripCollection()
{
    List<StripLineInfo> stripCollection = new List<StripLineInfo>();
    stripCollection.Add(new StripLineInfo()
    {
        Type = StriplineType.Absolute,
        Height = 1500,
        Width = 200,
        Position = new System.Windows.Point(300, 5),
        Background = new SolidColorBrush(Colors.LightGray)
    });
    return stripCollection;
}
```

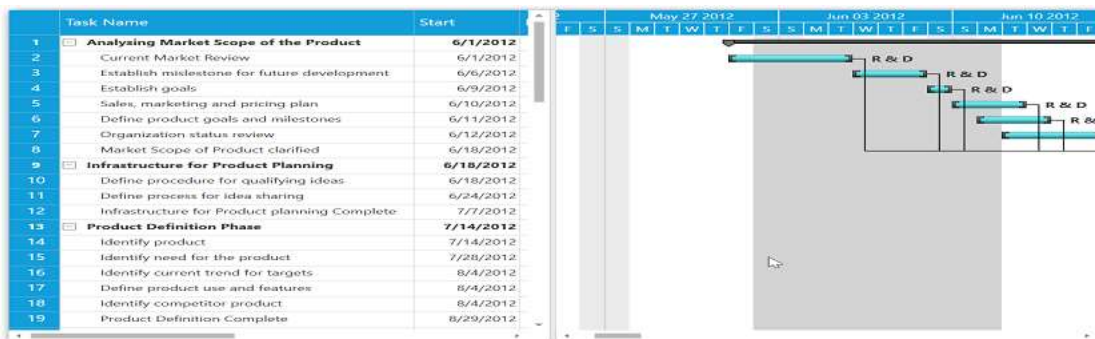
The following code sample demonstrates how to bind the absolute strip line collection to strip lines.

XML

```
<sync:GanttControl x:Name="Gantt"
    Grid.Row="1"
    ShowStripLines="True"
    StripLines="{Binding StripCollection}">
    <sync:GanttControl.TaskAttributeMapping>
    <sync:TaskAttributeMapping TaskIdMapping="TaskId"
        TaskNameMapping="TaskName"
        StartDateMapping="StartDate"
        FinishDateMapping="FinishDate"
        ChildMapping="Child"
        DurationMapping="Duration"
        ProgressMapping="Progress"
        PredecessorMapping="Predecessor"
        ResourceInfoMapping="Resources"/>
    </sync:GanttControl.TaskAttributeMapping>
</sync:GanttControl>
```

Output

The following screenshot illustrates how to render the absolute strip lines.



Strip lines in the Gantt chart

Sample Link

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under the User Interface Edition panel.
4. Select Gantt.
5. Expand the Interactive Features item in the Sample Browser.
6. Choose the Strip Lines sample to launch.

see also

[How to enable horizontal lines for gantt chart rows](#)

Import and Export Support in WPF Gantt

Essential Gantt allows you to export and import the task details. You can export the task details as XML files and import them again when needed. You can open the exported XML files in MS Project too. The XML file, exported from MS Project can also be opened in Gantt control. You can import and export the details using the provided APIs.

Properties

Property	Description	Type	Data Type
ImportFromXMLCommand	Command binding used to import the XML file generated from MS Project to populate dataâ€™s in Gantt control.	Command	DelegateCommand
ExportToXMLCommand	Command binding used to export the XML file generated from Gantt control to populate dataâ€™s in MS Project.	Command	DelegateCommand

Methods

Method	Description	Parameters	Type	Return Type
ExportToXML()	Responsible for exporting the GanttControl to MSProject XML File.	-	-	bool
ImportFromXML()	Responsible for importing the data from MS Project XML file to GanttControl.	-	-	bool

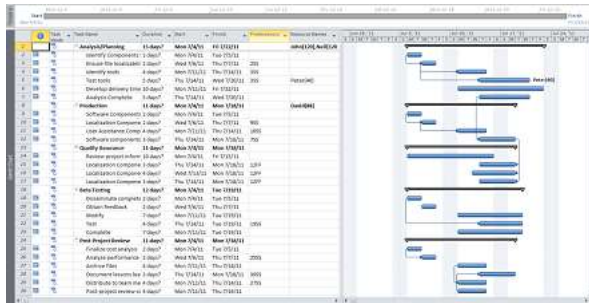
Import/Export Task Details from/to XML

The following code illustrates how to Import and Export Task Details from or to XML.

```
{% tabs %}
```

XML

```
<Sync:GanttControl x:Name="Gantt" />
```

Exported document opened in MS Project

[Samples Link](#)

To view samples:

1. Go to the Syncfusion Essential Studio installed location.

Location: Installed Location\Syncfusion\Essential Studio\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion Control Panel

2. Open the Syncfusion Control Panel in the above location (or) Double click on the Syncfusion Control Panel desktop shortcut menu.
3. Click Run Samples for WPF under User Interface Edition panel .
4. Select Gantt.
5. Expand the Import Export Features item in the Sample Browser.
6. Choose the Import Export Demo sample to launch.

Flow Direction in WPF Gantt

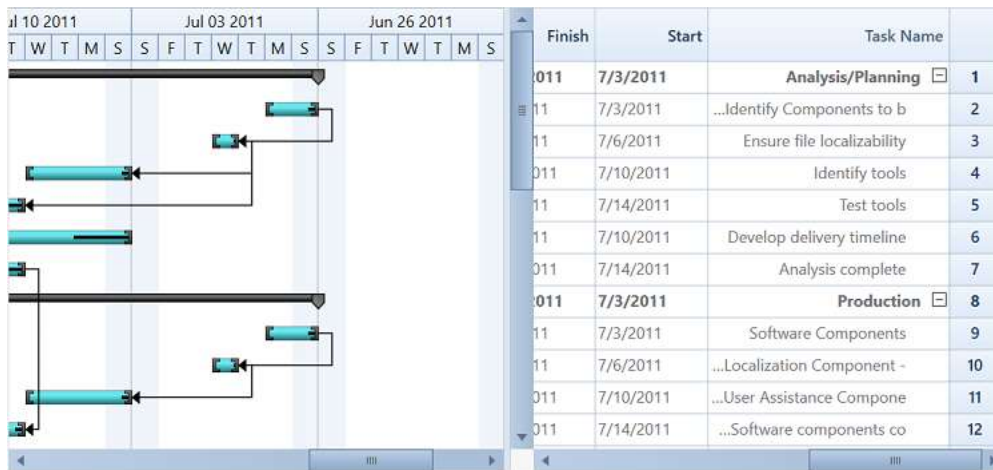
Gantt provides support to display the contents from right-to-left or left-to-right direction. It can be achieved by setting the `FlowDirection` property value as “RightToLeft” or “LeftToRight” in the Gantt control. The following code sample explains how to set this property.

XML

```
<Sync:GanttControl x:Name="Gantt" ItemsSource="{Binding GanttItemSource}"
FlowDirection="RightToLeft"/>
```

C#

```
this.Gantt.FlowDirection = System.Windows.FlowDirection.RightToLeft;
```



Localization in WPF Gantt

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the GanttControl by [adding resource file](#). Application culture can be changed by setting `CurrentUICulture` and `CurrentCulture` before `InitializeComponent()` method.

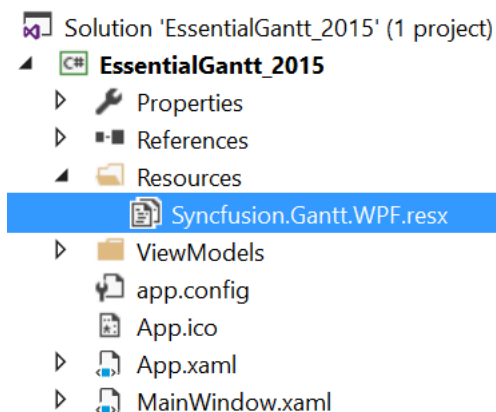
In the below application, culture is configured to French language.

C#

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

To localize the GanttControl based on `CurrentUICulture` using resource files, follow the below steps.

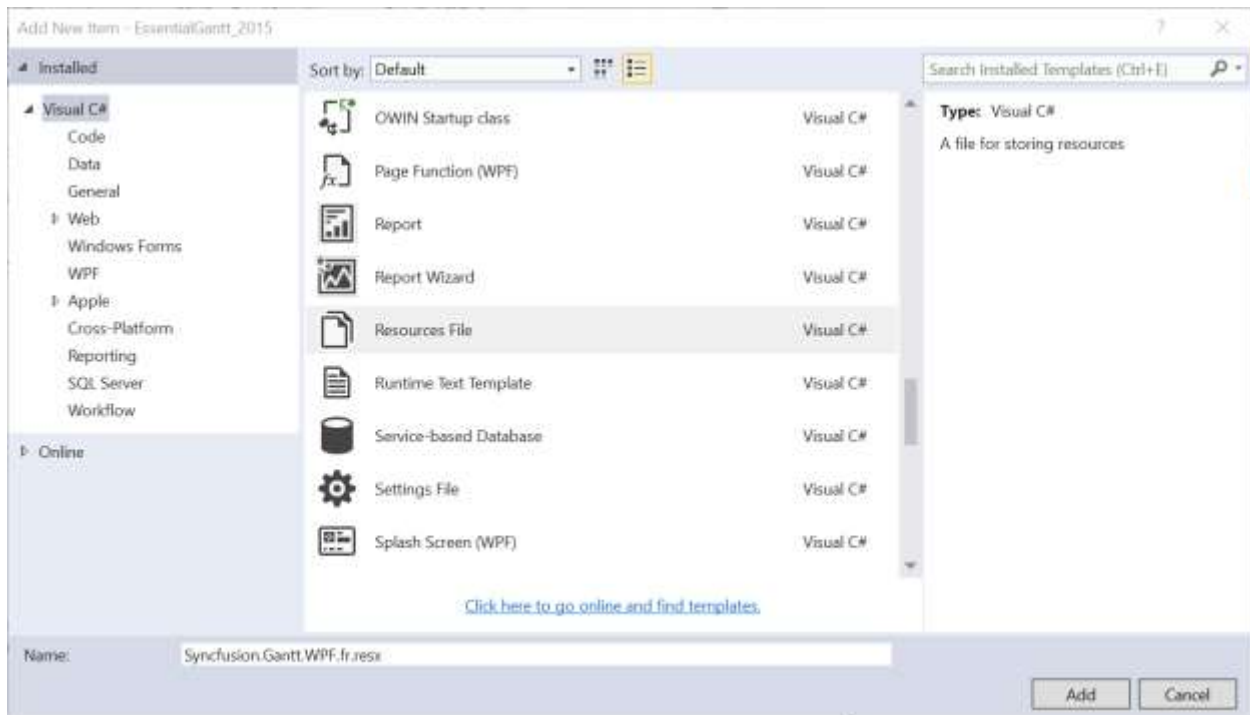
1. Create new folder and named as **Resources** in your application. 2. Add the default resource file of GanttControl into **Resources** folder. You can download the Syncfusion.Gantt.WPF.resx [here](#).



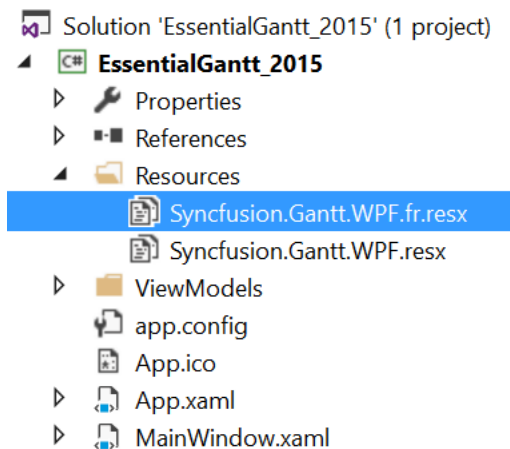
3. Right-click on the Resources folder, select **Add** and then **NewItem**. 4. In **Add New Item** wizard, select the **Resource File** option and name the filename as **Syncfusion.Gantt.WPF.<culture name>.resx**.

For example, you have to give name as **Syncfusion.Gantt.WPF.fr.resx** for French culture.

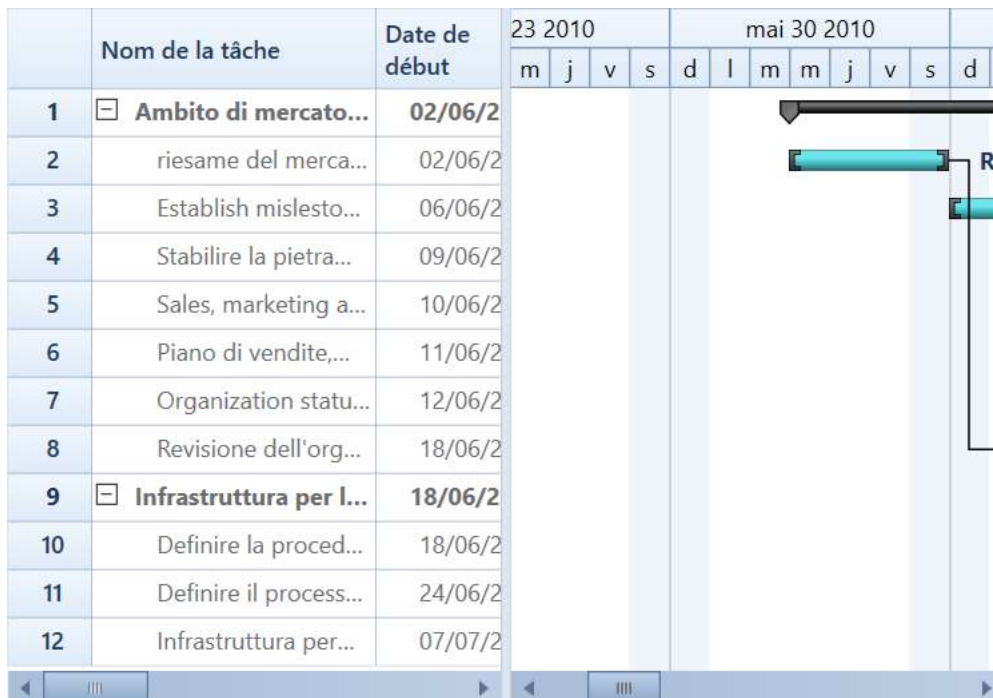
5.The culture name that indicates the name of language and country.



6.Now, select **Add** option to add the resource file in **Resources** folder.



7.Add the Name/Value pair in Resource Designer of **Syncfusion.Gantt.WPF.fr.resx** file and change its corresponding value to corresponding culture.



You can download the sample for localization of Gantt from [here](#)

How to

Restrict SelectionMode as Single

To restrict the selection mode to single, set the `ListBoxSelectionMode` options of `GanttGrid`'s `Model` in the `GanttControl` loaded event.

The following codes illustrate this

XML

```
...
<gantt:GanttControl x:Name="Gantt"
ItemsSource="{Binding TaskDetails}"
VisualStyle="Metro"
Loaded="Gantt_Loaded">
</gantt:GanttControl>
...
```

C#

```
...
private void Gantt_Loaded(object sender, RoutedEventArgs e)
{
    this.Gantt.GanttGrid.Model.Options.ListBoxSelectionMode =
    Syncfusion.Windows.Controls.Grid.GridSelectionMode.One;
}
...
```

GridControl

WPF GridControl Overview

The grid at its core functions as a very efficient display engine for tabular data that can be customized down to the cell level. It does not make any assumptions on the structure of the data (many grid controls implemented as straight data-bound controls make such explicit assumptions). This leads to a very flexible design that can be easily adapted to a variety of tasks including the display of completely unstructured data and the display of structured data from a database.

The display system also hosts a powerful and complete styles architecture. Settings can be specified at the cell level or higher levels using parent styles that are referred to as base styles. Base styles can affect groups of cells. Cell level settings override any higher-level settings and enable easy customization right down to the cell level.

With this version, our core focus has been on the underlying architecture for displaying cells with virtualized cell editors in a manner that enables good performance characteristics. The core display system also supports several building-block features such as nested grids, virtual modes, and support for a virtually unlimited number of rows and columns.

Key Features

The Excel like WPF grid control (virtual grid) is a cell-oriented control for displaying tabular data. It does not make any assumptions regarding the structure of the data.

Below are key features of WPF Grid Control:

- Easy APIs to add, delete, or move rows and columns – You can easily add, delete, or move rows and columns throughout the Grid control using its well-defined APIs.
- Clipboard Support – Essential Grid provides excellent clipboard support that allows users to copy and paste grid cell content to text or any format.
- Frozen Rows and Columns – Essential Grid allows users to freeze grid columns to the left or right or freeze rows to the top or bottom of the grid.
- Resize Rows and Columns – Essential Grid provides options for resizing rows and columns.
- Hide Rows and Columns – Essential Grid provides support for hiding or displaying a range of rows and columns.
- Keyboard Interface – Essential Grid provides extensive support for keyboard handling. The following list contains some supported keys and actions:
 - Arrow keys – To move cell focus.
 - PageUp/PageDown – To scroll a grid by page.
 - F2 – To activate/deactivate a current cell.
 - F4+ALT – To open/close the pop-up of a drop-down cell.
 - CTRL + Arrow – To move to the first or last row or column.
 - SHIFT + Arrow keys – To select cells.
 - DELETE – To delete an entire row in the GridData control.
 - CTRL+X, CTRL+V, CTRL+C – For common clipboard operations.
 - All keyboard operations can be customized.
- Selection Modes - Essential Grid offers different kinds of selection modes such as row only, column only, and cell only for selecting a particular row, column, or cell, respectively.
- Drag-Drop Support - Essential Grid lets you drag any column and drop it at any position in the grid. This allows columns to be repositioned as required.

- Virtual Mode - Essential Grid for WPF supports a virtual mode, which lets you dynamically provide data to the grid from an external data source through an event. This means the grid does not store any data in its internal data structure.

Feature Summary

This section provides basic information, such as definitions and usage, regarding important features of Essential Grid.

Grid Control Features

The Grid control is a cell-based, data-representation control. It can load millions of rows very quickly and is very easy to customize. The following features are just a sampling of what you can expect from Essential Grid.

Data Population and Virtual Mode

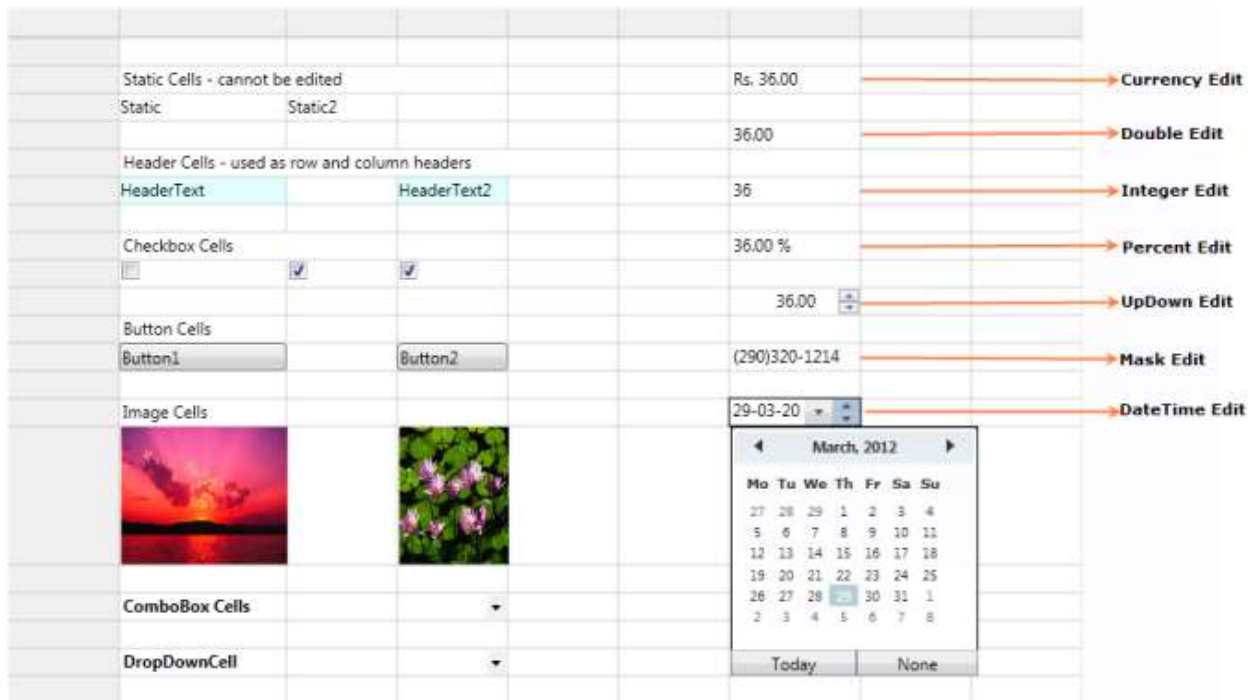
Because the Grid control is cell based, it doesn't have any internal data structures, so it cannot store data internally. Data can be populated by either looping through the cells or by a virtual mode, which dynamically provides data to the grid by handling an event. In virtual mode, the Grid control can display millions of rows easily.

	999990	999991	999992	999993	999994
999982	999982/999...	999982/999...	999982/999...	999982/999...	999982/999...
999983	999983/999...	999983/999...	999983/999...	999983/999...	999983/999...
999984	999984/999...	999984/999...	999984/999...	999984/999...	999984/999...
999985	999985/999...	999985/999...	999985/999...	999985/999...	999985/999...
999986	999986/999...	999986/999...	999986/999...	999986/999...	999986/999...
999987	999987/999...	999987/999...	999987/999...	999987/999...	999987/999...
999988	999988/999...	999988/999...	999988/999...	999988/999...	999988/999...
999989	999989/999...	999989/999...	999989/999...	999989/999...	999989/999...
999990	999990/999...	999990/999...	999990/999...	999990/999...	999990/999...
999991	999991/999...	999991/999...	999991/999...	999991/999...	999991/999...
999992	999992/999...	999992/999...	999992/999...	999992/999...	999992/999...
999993	999993/999...	999993/999...	999993/999...	999993/999...	999993/999...
999994	999994/999...	999994/999...	999994/999...	999994/999...	999994/999...
999995	999995/999...	999995/999...	999995/999...	999995/999...	999995/999...
999996	999996/999...	999996/999...	999996/999...	999996/999...	999996/999...

Virtual Grid

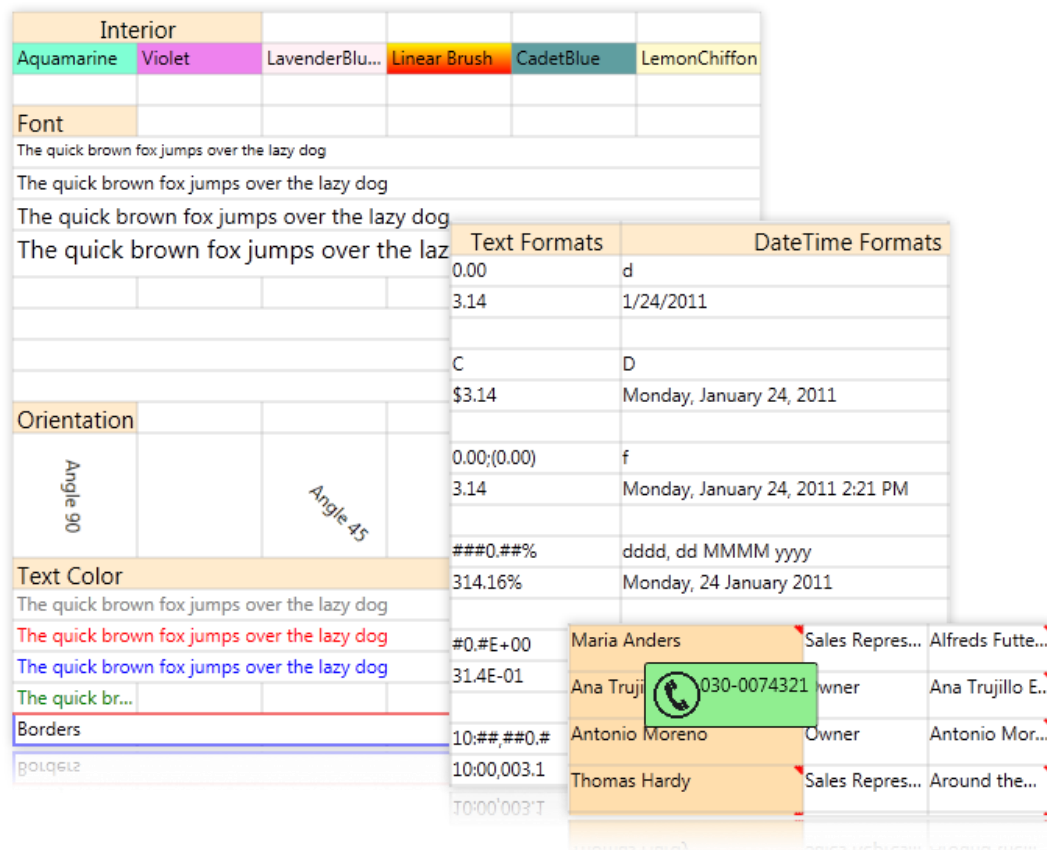
Data Presentation and Formatting

The Grid control supports many cell types to present data. In addition to built-in cell types, any WPF control or any custom control can be placed inside a cell by using a data-template cell.



Appearance

The Grid control's appearance can be customized very easily. Common style properties include options to change background or text color, font, alignment, orientation, border, and wrap text. Various cell formats include date-time, currency, numeric, and text formats, and Excel-like comments are supported.



Customization Possibilities of the Grid Control

- **Excel-Like Behaviors:** The Grid control supports most Excel behaviors, including the following:
- **Various Cell Support:** Cells in the Grid control are very similar to Excel cells. Unique cells such as covered cells and floating cells are supported.
- **Formula Support:** The Grid control supports Excel-like formulas in each cell. The control comes with an extensive formula function library that supports more than 150 built-in formulas.
- **Printing:** The Grid control provides built-in support for printing grid content.
- **Excel-Like Interactions:** The Grid control supports Excel-like user interaction features such as cell range drag-and-drop, auto-fill selections, and visual indicators to show hidden rows and columns.
- **Excel-Like Rows and Columns:** The Grid control provides various row and column options such as interactive column rearranging with drag-and-drop operations; hiding or displaying a range of rows and columns; resizing row heights and column widths to fit cell content; and options to fix rows and columns to the left, right, top, or bottom of the grid, similar to freezing panes in Excel.
- **Excel-Like key navigation:**
 - Arrow keys – To move current cell focus.
 - PageUp/PageDown – To scroll a grid by page.
 - F2 – To activate/deactivate a current cell.
 - F4+ALT – To open/close the pop-up of a drop-down cell.
 - CTRL + Arrow keys – To move to the first or last row or column.

- SHIFT + Arrow keys – To select cells.
- CTRL+X, CTRL+V, CTRL+C, INSERT & DELETE – For common clipboard operations.
- Cell Selection – The Grid control also supports Excel-like cell selection.

Product	Period	Sales
Product 1	Jan	\$5,000.00
Product 1	Feb	\$6,000.00
Product 1	Mar	\$7,000.00
Product 1	Apr	\$8,000.00
Product 1	May	\$9,000.00
Product 1	Jun	\$10,000.00
Product 1	Jul	\$11,000.00
Product 1	Aug	\$12,000.00
Product 1	Sep	\$13,000.00
Product 1	Oct	\$14,000.00
Product 1	Nov	\$15,000.00
Product 1	Dec	\$16,000.00
Product 1	Total	\$100,000.00
Product 2	Jan	\$1,000.00
Product 2	Feb	\$1,200.00
Product 2	Mar	\$1,400.00
Product 2	Apr	\$1,600.00
Product 2	May	\$1,800.00
Product 2	Jun	\$2,000.00
Product 2	Jul	\$2,200.00
Product 2	Aug	\$2,400.00
Product 2	Sep	\$2,600.00
Product 2	Oct	\$2,800.00
Product 2	Nov	\$3,000.00
Product 2	Dec	\$3,200.00
Product 2	Total	\$20,000.00
Product 3	Jan	\$500.00
Product 3	Feb	\$600.00
Product 3	Mar	\$700.00
Product 3	Apr	\$800.00
Product 3	May	\$900.00
Product 3	Jun	\$1,000.00
Product 3	Jul	\$1,100.00
Product 3	Aug	\$1,200.00
Product 3	Sep	\$1,300.00
Product 3	Oct	\$1,400.00
Product 3	Nov	\$1,500.00
Product 3	Dec	\$1,600.00
Product 3	Total	\$10,000.00
Product 4	Jan	\$200.00
Product 4	Feb	\$250.00
Product 4	Mar	\$300.00
Product 4	Apr	\$350.00
Product 4	May	\$400.00
Product 4	Jun	\$450.00
Product 4	Jul	\$500.00
Product 4	Aug	\$550.00
Product 4	Sep	\$600.00
Product 4	Oct	\$650.00
Product 4	Nov	\$700.00
Product 4	Dec	\$750.00
Product 4	Total	\$4,000.00
Total Sales		\$134,000.00

Excel Compatibility

- Import Options: The import feature allows an Excel workbook to be imported into a grid while preserving the workbook's look and feel. The following items can be imported: entire spreadsheets, formulas, styles, conditional formats, frozen panes, backgrounds, foregrounds, and comments.
- Export Options: The Grid control provides inherent support for exporting content to Excel files (.xls and .xlsx format) and to .csv files.

Getting Started with WPF GridControl

This section is designed to help you understand and quickly get started using Essential Grid in your WPF application.

Syncfusion WPF suite comes up with a package of powerful grid controls that provides cell-oriented features and acts as an efficient display engine for tabular data that can be customized down to the cell level. It also offers excellent performance characteristics, such as a virtual mode and high-frequency updates, which makes the grid suitable for real-time applications.

The Essential Studio for WPF is comprised of following three types of grid controls:

- [Grid Control](#)
- [SfDataGrid](#) and GridDataControl (classic)
- [SfTreeGrid](#) and GridTreeControl (classic)

Note: Refer [Choose between different Grid's](#) to take closer look at the characteristics of each of these controls.

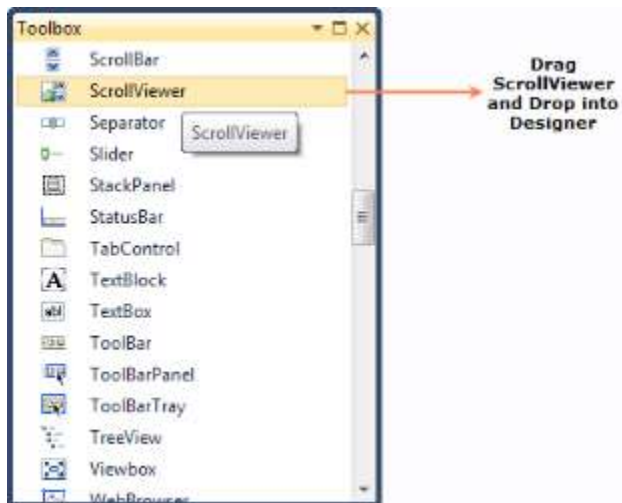
Adding the Grid Control to a WPF Application

In this section, we will see how to add the Grid control to a WPF application and load random data. The Grid control can be added to an application through one of the following methods: through a designer or programmatically.

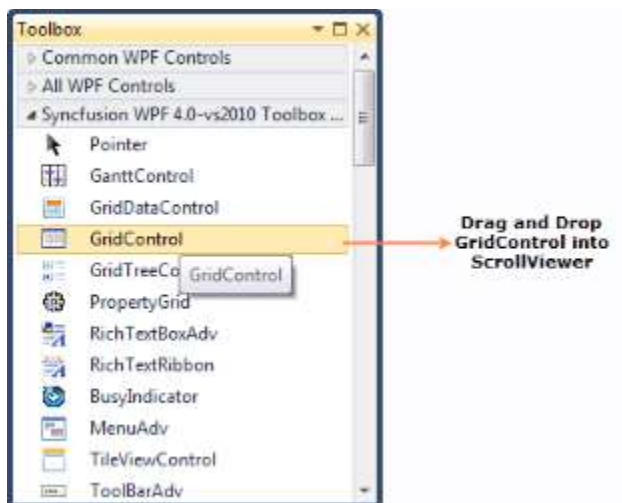
Adding the Grid Control through a Designer

Please follow the steps below to add the Grid control through a designer.

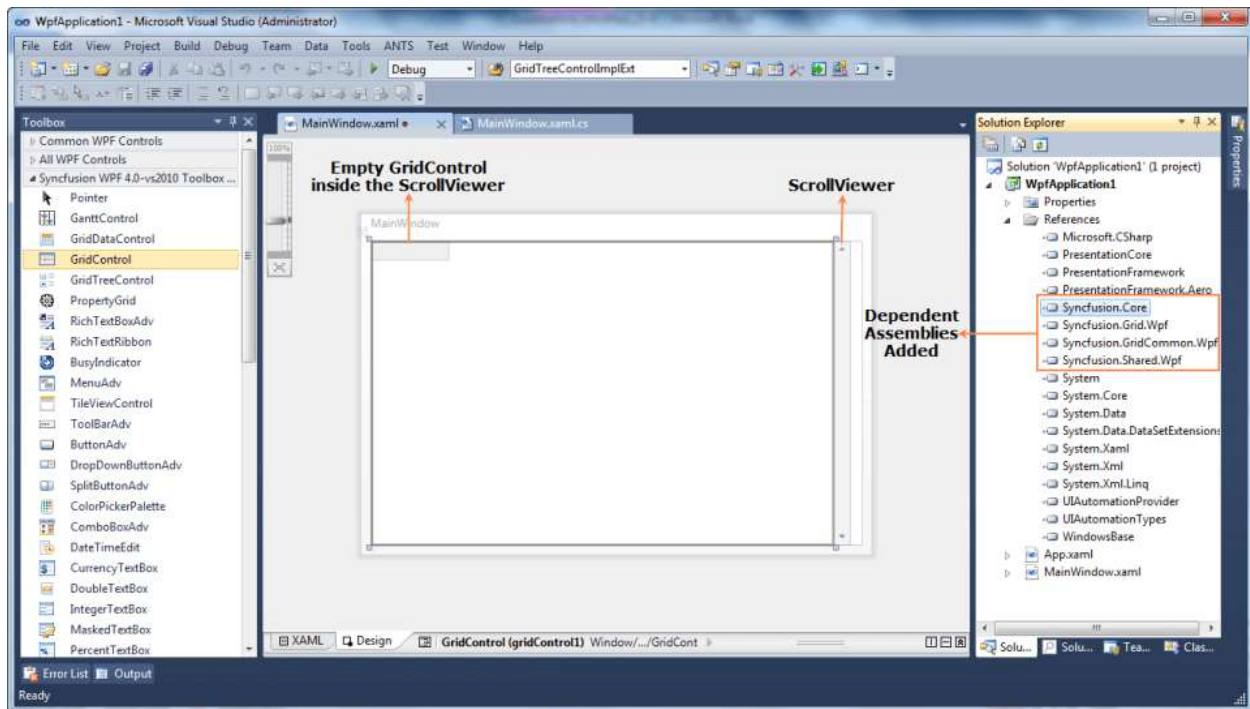
1. Create new WPF application.
2. Open the Designer window.
3. Drag ScrollViewer from the Toolbox and drop it in the Designer window (Since the Grid control doesn't have a built-in ScrollViewer, to make the grid flow based on data, the grid should be placed inside the ScrollViewer control).



4. Drag GridControl from the Toolbox and drop it inside the ScrollViewer.



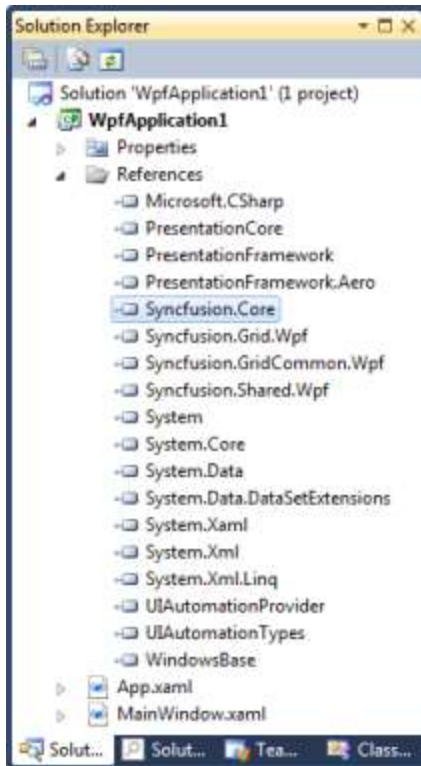
5. Once you drag GridControl and drop it in ScrollViewer, the grid control will be added to the designer and its dependent assemblies will be added to the project.



Programmatically Adding the Grid Control

Instead of adding it through a designer such as Visual Studio, you can add the Grid control programmatically.

1. Create a new WPF application.
2. Add the following Syncfusion assemblies to the project.
 - Syncfusion.Core.dll
 - Syncfusion.Grid.Wpf.dll
 - Syncfusion.GridCommon.Wpf.dll
 - Syncfusion.Shared.Wpf.dll



3. Name the root Grid as layoutRoot in the application's 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 Name="layoutRoot" />
</Window>
```

4. Create ScrollViewer and GridControl in code.
5. To add the grid to the view, add GridControl as content of ScrollViewer and then add the ScrollViewer as a child of layoutRoot (Grid).

C#

```
//ScrollViewer defined here
ScrollViewer ScrollViewer = new ScrollViewer();
//GridControl defined here
GridControl gridControl = new GridControl();
//GridControl set as the content of the ScrollViewer
ScrollViewer.Content = gridControl;
//To bring the Grid control to the view, ScrollViewer should be set as a
child of LayoutRoot
this.layoutRoot.Children.Add(ScrollViewer);
```

Populating the Grid control with Data

The Grid control is a cell-based control, so to populate it, RowCount and ColumnCount are mandatory. Once ColumnCount and RowCount are specified, data can be populated by using one of the following methods.

1.You can populate data by looping through the cells in the Grid control. The following code explains this scenario.

C#

```
//Specifying row and column count
gridControl.Model.RowCount = 100;
gridControl.Model.ColumnCount = 20;
//Looping through the cells and assigning the values based on row and column index
for (int i = 0; i < 100; i++)
{
    for (int j = 0; j < 20; j++)
    {
        gridControl.Model[i, j].CellValue = string.Format("{0}/{1}", i, j);
    }
}
```

2.You can populate data by handling the QueryCellInfo event of gridControl. This will load the data in and on-demand basis, ensuring optimized performance.

C#

```
//Specifying row and column count
gridControl.Model.RowCount = 100;
gridControl.Model.ColumnCount = 20;
this.gridControl.QueryCellInfo += new Syncfusion.Windows.Controls.Grid.GridQueryCellInfoEventHandler(gridControl_QueryCellInfo);
//Assigning values by handling the QueryCellInfo event
void gridControl_QueryCellInfo(object sender, Syncfusion.Windows.Controls.Grid.GridQueryCellInfoEventArgs e)
{
    e.Style.CellValue=string.Format("{0}/{1}", e.Cell.RowIndex, e.Cell.ColumnIndex);
}
```

3.Now, run the application. The grid will appear as follows.

0/0	0/1	0/2	0/3	0/4
1/0	1/1	1/2	1/3	1/4
2/0	2/1	2/2	2/3	2/4
3/0	3/1	3/2	3/3	3/4
4/0	4/1	4/2	4/3	4/4
5/0	5/1	5/2	5/3	5/4
6/0	6/1	6/2	6/3	6/4
7/0	7/1	7/2	7/3	7/4
8/0	8/1	8/2	8/3	8/4
9/0	9/1	9/2	9/3	9/4
10/0	10/1	10/2	10/3	10/4
11/0	11/1	11/2	11/3	11/4
12/0	12/1	12/2	12/3	12/4
13/0	13/1	13/2	13/3	13/4

Appearance in WPF GridControl

You can customize the appearance of the grid to the cell-level using our Grid properties.

Cell Styles

The EssentialGrid's cell style architecture plays an integral role in almost every aspect of Essential Grid. The display system hosts a powerful and complete Styles architecture. Settings can be specified at the cell level or at higher levels using parent styles that are referred to as Base Styles. Base Styles can affect a groups of cells. Cell level settings override any higher-level settings and enable easy customization to cell level. With this initial version, Syncfusion's core focus has been on the underlying architecture for displaying cells with virtualized cell editors to enable good performance characteristics.

Following are the two cell styles available:

Volatile Cell Styles

QueryCellInfo will be raised the first time you access the contents of a cell with a call to Grid.Model[rowIndex, columnIndex] or when the grid calls this indexer internally before painting cells. The indexer returns an object of type GridStyleInfo. After querying the cell contents they remain cached in a volatile cache that holds weak references to the cell styles. This ensures that this data is available for reuse when needed. At the same time it does not stand in the way of the garbage collector if memory needs to be freed. Once a style gets garbage collected it will be removed from the volatile cache. You can manually force QueryCellInfo to be called again when you call GridControl.InvalidateCell(cell) or GridModel.VolatileCellStyles.Clear(cell).

Render Cell Styles

Prior to display of a cell, the PrepareRenderCell event is raised. This event is not raised from the Model. Instead it is raised from the GridControlBase directly. This has the advantage that if you want multiple grids to display the same Model, individual grids can override the cell contents individually.

C#

```
// view specific cell color
grid.PrepareRenderCell += new
GridPrepareRenderCellEventHandler(grid_PrepareRenderCell);
void grid_PrepareRenderCell(object sender, GridPrepareRenderCellEventArgs e)
{
    if (e.Cell.RowIndex > 0 && e.Cell.ColumnIndex > 0)
    {
        if (e.Cell.RowIndex % 2 == 0)
            e.Style.Background = Brushes.LightSkyBlue;
    }
}
```

```
}
```

PrepareRenderCell is used to initialize the so called RenderStyles. RenderStyles are of type GridRenderStyleInfo and derive from GridStyleInfo. GridRenderStyleInfo are tied to a GridControlBase instance. The render style provides additional properties to obtain access to the associated GridControl, CellRenderer and the underlying ModelStyle instance from the volatile cells cache described earlier.

To access the render style for a cell you can call GridControlBase.GetRenderStyleInfo.

Render Styles are created only for the cells that are visible and will be discarded the moment a cell is scrolled out of view (with the exception being if the current cell is scrolled out of view; they are retained in such case alone).

In contrast, Volatile Cell Styles from the GridModel are often also created for cells outside the viewable area and can stay in the cache even when a cell is scrolled out of view.

A basic understanding of this layered cell style architecture will help you understand and learn grid behaviors. This is particularly important when you are trying to modify or extend some existing functionality.

GridStyleInfo class Overview

An EssentialGrid can be thought of as a rectangular table of grid cells. Each cell will contain distinct information and that can be displayed independently of other cells. Essential Grid uses GridStyleInfo object to store state information about the appearance of a grid cell. So attributes like the type of font, background color, cell value and cell type are all reflected in a single GridStyleInfo object.

Every cell in a grid may have such an object associated with it, giving the individual cell its unique appearance. It is not necessary that all cells should require fully populated GridStyleInfo objects stored in memory to function. And, for a given GridStyleInfo object, not all possible properties need to be populated in the object. So for example, a particular cell may or may not have a stored GridStyleInfo object, and if it does, this GridStyleInfo object may, or may not, contain a particular property such as font.

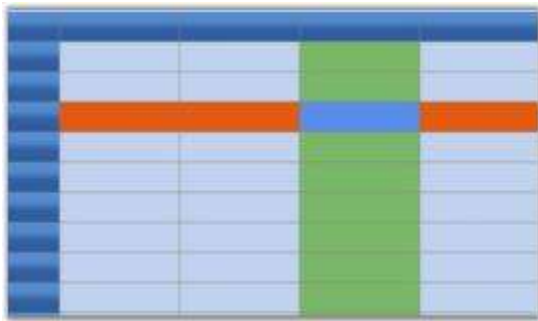
In general, when Essential Grid needs a cell's state information, usually to draw the cell, it uses an inheritance process to generate a GridStyleInfo from several parent styles. These parent styles are GridStyleInfo objects associated with particular grid entities:

- TableStyle is a single GridStyleInfo object associated with the entire grid.
- RowStyles are GridStyleInfo objects associated with each row.
- ColumnStyles are GridStyleInfo objects associated with each column.

These three GridStyleInfo objects may not be fully populated, meaning that some properties may not have been set. However, there is a fourth parent style referred to as the StandardStyle, which is a fully populated style object comprising settings for every property.

Hence, when Essential Grid needs to generate a composite GridStyleInfo object for a particular cell, it first looks at any property that may be specifically set in the GridStyleInfo (if one exists) for the current cell. If properties are not set in the cell-specific GridStyleInfo object, Essential Grid will then pick up the RowStyles for the cell and will populate any property that was explicitly set in the RowStyle and not set in the cell-specific GridStyleInfo object. After adding unset properties to the composite GridStyleInfo from the RowStyle, it does the same for the ColumnStyle, the TableStyle and finally the StandardStyle. In this manner, Essential Grid comes up with a fully populated composite GridStyleInfo object to use.

The following graphic illustrates the effect of using the GridStyleInfo inheritance to come up with the appearance of a cell. Even though the BackColor property is set in each of the table style, RowStyle and ColumnStyle objects, it is the cell specific style that determines the background color of cell.



Style Properties

Property settings for individual/groups of cells are stored in a GridStyleInfo property. The style allows you to set properties such as background, cell value, and cell type for a particular cell.

Essential Grid for WPF holds two different style caches that depend upon how the cell style is being used:

Volatile style-cache-Is maintained for styles populated through calls made to the QueryCellInfo (the virtual grid event that provides the cell values to the grid on demand) event. This volatile style-cache uses weak references to interact with the .NET Framework's Garbage collection to ensure optimal memory use. These styles remain cached as long as they are not garbage collected by the Framework.

Render style-cache-Is maintained for styles needed to draw the grid and are disposed of as soon as the cell scrolls out of view.

The combination of these two caches makes Essential Grid for WPF highly efficient. This section elaborates on important style properties.

Base Styles

Base Styles, otherwise named as Parent Styles, define the style information for individual cell groups such that all the cells belonging to a group will share the same Base style. On changing the common base style, the dependent cells styles also get updated automatically.

Following code snippet illustrates the effect of various Base styles.

C#

```
//Defines the base styles named PinkStyle and GreenStyle.
GridBaseStyle baseStyle1 = new GridBaseStyle();
baseStyle1.Name = "PinkStyle";
baseStyle1.StyleInfo.Background = Brushes.LightPink;
baseStyle1.StyleInfo.Foreground = Brushes.Maroon;
GridBaseStyle baseStyle2 = new GridBaseStyle();
baseStyle2.Name = "GreenStyle";
baseStyle2.StyleInfo.Background = Brushes.PaleGreen;
baseStyle2.StyleInfo.Foreground = Brushes.Olive;
//Add the above styles to the grid base styles collection.
grid.Model.BaseStylesMap.Add(baseStyle1);
grid.Model.BaseStylesMap.Add(baseStyle2);
//Applying base styles.
for (int i = 1; i <= grid.Model.RowCount; i++)
{
```



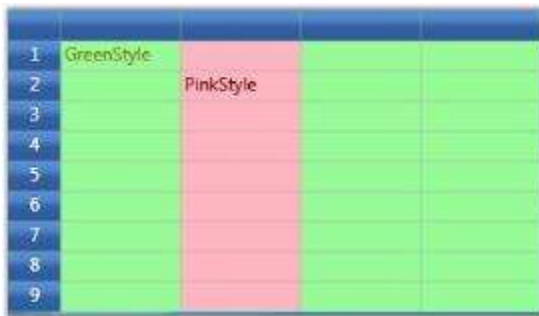
```

for (int j = 1; j <= grid.Model.ColumnCount; j++)
{
    if(j ==2)
    grid.Model[i, j].BaseStyle = "PinkStyle";
    else
    grid.Model[i, j].BaseStyle = "GreenStyle";
}
}

```

Output

The code above displays the following output:



Background

The Background property specifies a background brush for the grid cell. The cell's background can be painted with either a solid brush or a gradient brush.

Note: Gradient-A gradient brush uses two colors. These colors merge to create a transition or fading effect. * Solid-A solid brush is equipped with only one color.

Setting Background Brush Type

C#

```

this.grid.Model[2, 1].Background = Brushes.Aquamarine;
this.grid.Model[2, 2].Background = Brushes.Violet ;
this.grid.Model[2, 3].Background = Brushes.LawnGreen;
this.grid.Model[2, 4].Background = Brushes.LavenderBlush ;
this.grid.Model[2, 5].Background = Brushes.CadetBlue ;
this.grid.Model[2, 6].Background = Brushes.LemonChiffon;
this.grid.Model[3, 1].Background = GetLinerBrush();
this.grid.Model[3, 2].Background = new LinearGradientBrush(Colors.Turquoise,
Colors.White, 90.0);
this.grid.Model[3, 3].Background = new LinearGradientBrush(Colors.Firebrick,
Colors.Orange, 90.0);
this.grid.Model[3, 4].Background = new
LinearGradientBrush(Colors.CornflowerBlue, Colors.White, 0.0);
this.grid.Model[3, 5].Background = new LinearGradientBrush(Colors.Olive,
Colors.PaleGreen, 0.0);
this.grid.Model[3, 6].Background = new LinearGradientBrush(Colors.Gold,
Colors.Yellow, 90.0);

```

Output

The following output is generated using the code above.



Visual properties

The visual aspects of the cell text can be controlled by the following properties.

Property Name	Description
Text	Holds the text to be displayed in the cell
Foreground	Specifies text color
Font	Controls the font properties for the text in the cell
Orientation	Determines the angle of rotation of the text

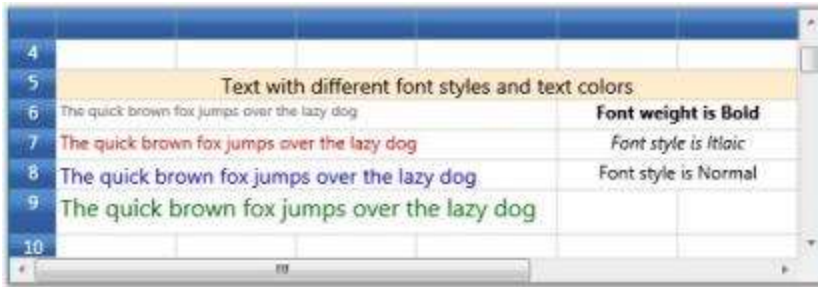
1.Setting Text, Foreground and Font Properties

C#

```
//Setting text, foreground and fonts
this.grid.Model[6, 1].Font.FontSize = 10;
this.grid.Model[6, 1].Text = "The quick brown fox jumps over the lazy dog";
this.grid.Model[6, 1].Foreground = Brushes.Gray;
this.grid.Model[7, 1].Font.FontSize = 12;
this.grid.Model[7, 1].Text = "The quick brown fox jumps over the lazy dog";
this.grid.Model[7, 1].Foreground = Brushes.Red;
this.grid.Model[8, 1].Font.FontSize = 14;
this.grid.Model[8, 1].Text = "The quick brown fox jumps over the lazy dog";
this.grid.Model[8, 1].Foreground = Brushes.Blue;
this.grid.Model[9, 1].Font.FontSize = 16;
this.grid.RowHeights[9] = 30d;
this.grid.Model[9, 1].Text = "The quick brown fox jumps over the lazy dog";
this.grid.Model[9, 1].Foreground = Brushes.Green;
//Setting font weights
this.grid.Model[6, 5].Font.FontWeight = FontWeights.Bold;
this.grid.Model[6, 5].HorizontalAlignment = HorizontalAlignment.Center;
this.grid.Model[6, 5].CellValue = "Font weight is Bold";
this.grid.Model[7, 5].Font.FontStyle = FontStyles.Italic;
this.grid.Model[7, 5].HorizontalAlignment = HorizontalAlignment.Center;
this.grid.Model[7, 5].CellValue = "Font style is Italic";
this.grid.Model[8, 5].Font.FontStyle = FontStyles.Normal;
this.grid.Model[8, 5].HorizontalAlignment = HorizontalAlignment.Center;
this.grid.Model[8, 5].CellValue = "Font style is Normal";
```

Output

The following output is generated using the code above.



2.Setting cell orientation

C#

```
this.grid.Model[12, 1].Font.Orientation = 45;
this.grid.Model[12, 1].CellValue = "Angle 45";
this.grid.Model[12, 1].HorizontalAlignment = HorizontalAlignment.Center;
this.grid.Model[12, 1].VerticalAlignment = VerticalAlignment.Center;
this.grid.Model[12, 2].Font.Orientation = 90;
this.grid.Model[12, 2].CellValue = "Angle 90";
this.grid.Model[12, 2].HorizontalAlignment = HorizontalAlignment.Center;
this.grid.Model[12, 2].VerticalAlignment = VerticalAlignment.Center;
this.grid.Model[12, 3].Font.Orientation = 180;
this.grid.Model[12, 3].CellValue = "Angle 180";
this.grid.Model[12, 3].HorizontalAlignment = HorizontalAlignment.Center;
this.grid.Model[12, 3].VerticalAlignment = VerticalAlignment.Center;
this.grid.Model[12, 4].Font.Orientation = 270;
this.grid.Model[12, 4].CellValue = "Angle 270";
this.grid.Model[12, 4].HorizontalAlignment = HorizontalAlignment.Center;
this.grid.Model[12, 4].VerticalAlignment = VerticalAlignment.Center;
this.grid.Model[12, 5].Font.Orientation = 320;
this.grid.Model[12, 5].CellValue = "Angle 320";
this.grid.Model[12, 5].HorizontalAlignment = HorizontalAlignment.Center;
this.grid.Model[12, 5].VerticalAlignment = VerticalAlignment.Center;
this.grid.Model.RowHeights[12] = 50;
```

Output

The following output is generated using the code above.



Borders

Cell borders can be customized to have different color, thickness and style. It is possible to have different border styles for top, bottom, left and right borders for the same cell.

Setting Borders

C#

```
this.grid.Model[15, 1].Borders.Bottom = new Pen(Brushes.Blue, 1);
this.grid.Model[15, 1].Borders.Top = new Pen(Brushes.Red, 1);
```

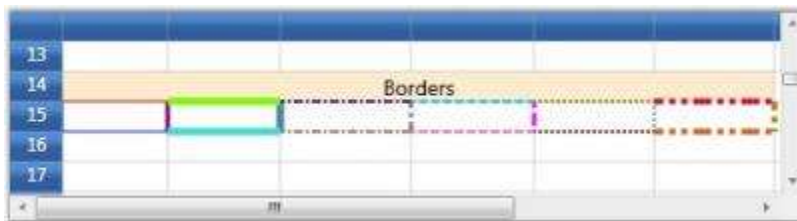
```

this.grid.Model[15, 1].Borders.Right = new Pen(Brushes.Purple, 3);
this.grid.Model[15, 1].Borders.Left = new Pen(Brushes.RoyalBlue, 1);
this.grid.Model[15, 2].Borders.Bottom = new Pen(Brushes.Turquoise, 1);
this.grid.Model[15, 2].Borders.Bottom.Thickness = 5;
this.grid.Model[15, 2].Borders.Top = new Pen(Brushes.LawnGreen, 1);
this.grid.Model[15, 2].Borders.Top.DashCap = PenLineCap.Round;
this.grid.Model[15, 2].Borders.Top.Thickness = 5;
this.grid.Model[15, 2].Borders.Right = new Pen(Brushes.SteelBlue, 3);
this.grid.Model[15, 2].Borders.Right.Thickness = 5;
this.grid.Model[15, 3].Borders.Bottom = new Pen(Brushes.IndianRed, 2);
this.grid.Model[15, 3].Borders.Bottom.DashStyle = DashStyles.DashDot;
this.grid.Model[15, 3].Borders.Top = new Pen(Brushes.Indigo, 2);
this.grid.Model[15, 3].Borders.Top.DashStyle = DashStyles.DashDot;
this.grid.Model[15, 3].Borders.Right = new Pen(Brushes.Gray, 3);
this.grid.Model[15, 3].Borders.Right.DashStyle = DashStyles.DashDot;
this.grid.Model[15, 4].Borders.Bottom = new Pen(Brushes.HotPink, 2);
this.grid.Model[15, 4].Borders.Top = new Pen(Brushes.DeepSkyBlue, 2);
this.grid.Model[15, 4].Borders.Right = new Pen(Brushes.Magenta, 3);
this.grid.Model[15, 4].Borders.Bottom.DashStyle = DashStyles.Dash;
this.grid.Model[15, 4].Borders.Top.DashStyle = DashStyles.Dash;
this.grid.Model[15, 4].Borders.Right.DashStyle = DashStyles.Dash;
this.grid.Model[15, 5].Borders.Bottom = new Pen(Brushes.Maroon, 2);
this.grid.Model[15, 5].Borders.Top = new Pen(Brushes.Olive, 2);
this.grid.Model[15, 5].Borders.Right = new Pen(Brushes.CadetBlue, 2);
this.grid.Model[15, 5].Borders.Bottom.DashStyle = DashStyles.Dot;
this.grid.Model[15, 5].Borders.Top.DashStyle = DashStyles.Dot;
this.grid.Model[15, 5].Borders.Right.DashStyle = DashStyles.Dot;
this.grid.Model[15, 6].Borders.Bottom = new Pen(Brushes.Chocolate, 4);
this.grid.Model[15, 6].Borders.Top = new Pen(Brushes.Crimson, 4);
this.grid.Model[15, 6].Borders.Right = new Pen(Brushes.DarkGoldenrod, 4);
this.grid.Model[15, 6].Borders.Bottom.DashStyle = DashStyles.DashDotDot;
this.grid.Model[15, 6].Borders.Top.DashStyle = DashStyles.DashDotDot;
this.grid.Model[15, 6].Borders.Right.DashStyle = DashStyles.DashDotDot;

```

Output

The following output is generated using the code above.



Data Formats

Essential Grid allows the user to specify the format string for Text and DateTime cell values. The following table lists the various format strings supported.

Text Formats	Example with Cell Value = Math.PI
0.00	3.14
C	\$3.14

0.00;(0.00)	3.14
###0.##%	314.16%
#0.##E+00	3L4E-01
10:##,##0.##	10.00,003.1

DateTime	Example with Cell Value = DateTime.Now
d	8/10/2009
D	Monday, August 10, 2009
f	Monday, August 10, 2009 7.00 AM
dddd	Monday, 10 August 2009
t	7.00 AM
s	2009-08-10T07:00:15

1.Setting text format

C#

```
//Setting Text formats
int rowIndex = 3;
int colIndex = 1;
GridModel model = this.grid.Model;
foreach (string format in new string[]
{
    "0.00",
    "C",
    "0.00; (0.00)",
    "###0.##%",
    "#0.##E+00",
    "10:##,##0.##"
})
{
    model[rowIndex - 1, colIndex].Text = format;
    model[rowIndex, colIndex].Format = format;
    model[rowIndex, colIndex].CellValue = Math.PI;
    model[rowIndex, colIndex].CellValueType = typeof(double);
    rowIndex += 3;
}
```

2.Setting DateTime format

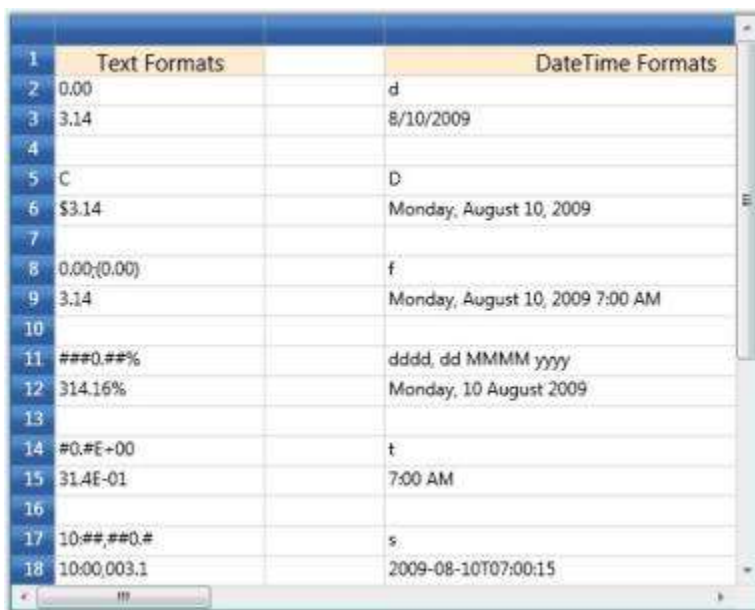
C#

```
//Setting DateTime formats
rowIndex = 2;
colIndex = 3;
```

```
foreach (string format in new string[]
{
    "d",
    "D",
    "f",
    "dddd, dd MMMM yyyy",
    "t",
    "s"
})
{
    model[rowIndex - 1, colIndex].Text = format;
    grid.Model.ColumnWidths[colIndex] = 150d;
    model[rowIndex, colIndex].Format = format;
    model[rowIndex, colIndex].CellValue = DateTime.Now;
    model[rowIndex, colIndex].CellValueType = typeof(DateTime);
    rowIndex += 3;
}
```

Output

The following output is generated using the code above.



	Text Formats	DateTime Formats
1		
2	0.00	d
3	3.14	8/10/2009
4		
5	C	D
6	\$3.14	Monday, August 10, 2009
7		
8	0.00;(0.00)	f
9	3.14	Monday, August 10, 2009 7:00 AM
10		
11	###0.##%	dddd, dd MMMM yyyy
12	314.16%	Monday, 10 August 2009
13		
14	#0.##E+00	t
15	31.4E-01	7:00 AM
16		
17	10.##,##0.##	s
18	10:00,003.1	2009-08-10T07:00:15

FormatProvider

The FormatProvider in GridStyleInfo is used to format the display text in the cells. The FormatProvider property supplies an object that provides formatting information for formatting operations. The formatting operations convert the value of a type to its string representation. The FormatProvider property allows the user to control the formatting of a particular cell.

Format using FormatProvider Property

In the Grid control, you can set the FormatProvider property as shown in the following code snippet.

C#

```
this.grid.Model[row, col].CellValue = Math.PI;
this.grid.Model[row, col].Format = "{0:ES}";
this.grid.Model[row, col].FormatProvider = new CustomNumberFormat();
```

```

this.grid.Model[row, col].CellValue = Math.PI;
this.grid.Model[row, col].Format = "{0:US}";
this.grid.Model[row, col].FormatProvider = new CustomNumberFormat();

```

VB.NET

```

Me.grid.Model(row, col).CellValue = Math.PI
Me.grid.Model(row, col).Format = "{0:ES}"
Me.grid.Model(row, col).FormatProvider = new CustomNumberFormat()
Me.grid.Model(row, col).CellValue = Math.PI
Me.grid.Model(row, col).Format = "{0:US}"
Me.grid.Model(row, col).FormatProvider = new CustomNumberFormat()

```

The following example illustrates the use of a class that implements the IFormatProvider interface.

C#

```

public class CustomNumberFormat : IFormatProvider, ICustomFormatter
{
    private const int ACCT_LENGTH = 6;
    public object GetFormat(Type formatType)
    {
        if (formatType == typeof(ICustomFormatter))
            return this;
        else
            return null;
    }
    public string Format(string fmt, object arg, IFormatProvider formatProvider)
    {
        // Provide default formatting, if arg is not double.
        if (arg.GetType() != typeof(double))
            return HandleOtherFormats(fmt, arg);
        // Provide default formatting for unsupported format strings.
        string ufmt = fmt.ToUpper(CultureInfo.InvariantCulture);
        if (!(ufmt == "US" || ufmt == "ES"))
            return HandleOtherFormats(fmt, arg);
        // Convert argument to a string.
        string result = arg.ToString();
        if (ufmt == "ES")
        {
            CultureInfo esESCulture = CultureInfo.GetCultureInfo("es-ES");
            result = Convert.ToString(arg, esESCulture);
        }
        else
        {
            CultureInfo esUSCulture = CultureInfo.GetCultureInfo("es-US");
            result = Convert.ToString(arg, esUSCulture);
        }
        // If number is less than 6 characters, pad with leading zeroes.
        if (result.Length < ACCT_LENGTH)
            result = result.PadRight(ACCT_LENGTH, '0');
        // If number is more than 6 characters, truncate to 6 characters.
        if (result.Length > ACCT_LENGTH)
            result = result.Substring(0, ACCT_LENGTH);
        return result;
    }
}

```

```
private string HandleOtherFormats(string format, object arg)
{
    if (arg is IFormattable)
        return ((IFormattable)arg).ToString(format, CultureInfo.CurrentCulture);
    else if (arg != null)
        return arg.ToString();
    else
        return String.Empty;
}
```

The CustomNumberFormat class converts the double value to string based on the culture specified in the format property.

The output is shown in the following screenshot:

Using the normal Format property		Using the FormatProvider(IFormatProvider) property	
Format	#,##00	Format	{0:ES}
Example	3.1416	Example	3.1415
		Format	{0:US}
		Example	3.1415

Cell Types in WPF GridControl

Essential Grid allows the inclusion of some special controls in the grid cells. This greatly improves the usability and appearance of the grid control. This attribute of a grid cell is referred to as its Cell Type. This section lists out various cell controls that can be placed inside the grid cells. Essential Grid currently supports 20+ cell types. It also provides support to create and use custom cell types.

Following is the list of built-in cell types:

Cell Type	Cell Type String	Usage
Header	"Header"	Used as row and column headers
Static	"Static"	Cannot be edited
Check Box	"CheckBox"	Used for toggling options
Button	"Button"	Provides Click event, which can be triggered to perform required action
Image	"ImageCell"	Used to display pictures
ComboBox	"ComboBox"	Used to choose the value
DropDownList	"DropDownList"	Used to display the multi column drop-down
CurrencyEdit	"CurrencyEdit"	Used to display the currency value
DateTimeEdit	"DateTimeEdit"	Used to display the date and time

DoubleEdit	"DoubleEdit"	Used to display the group separators and decimal digits
IntegerEdit	"IntegerEdit"	Used to display the group separators and decimal digits
MaskEdit	"MaskEdit"	Used to edit the cells
PercentEdit	"PercentEdit"	Used to display the percent values
RichTextBox	"RichText"	Used to format the cells
UpDownEdit	"UpDownEdit"	Used to increase or decrease the cell values
DataTemplate	"DataBoundTemplate"	used for cell cannot be changed or edited
Hyperlink	"Hyperlink"	Used to navigate to another cell
NestedGrid	"ScrollGrid"	Used to display the multiple scroller in row and column

Setting Cell Type

This section elaborates you on how to employ basic controls like Check Box, Radio Button and more in a grid cell.

To set up desired cell type, the [Style.CellType](#) property must be assigned with the corresponding format string. The list of cell types and their usages are described below.

Header cell type

A header cell type is used as row and column headers in the grid cell.

C#

```
var style = gridcontrol.Model[1, 2];
style.CellType = "Header";
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
e.Style.GridModel[1, 2].CellType = "Header";
}
```

Static cell type

The cell is specified as 'Static', that cell can not be changed or edited.

C#

```
var style = gridcontrol.Model[1, 2];
style.CellType = "Static";
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
e.Style.GridModel[1, 2].CellType = "Static";
}
```

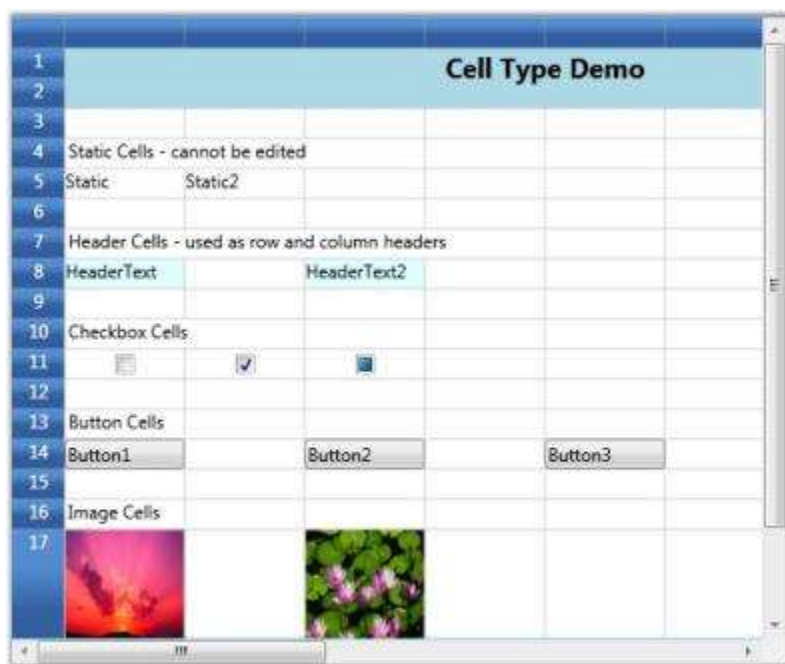
CheckBox cell type

The Checkbox cell type is used for toggling options. For instance, if you want to display a Checkbox control in the cell (2, 2), then you have to use the code below.

C#

```
var style = gridControl1.Model[1, 2];
style.CellType = "CheckBox";
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    e.Style.GridModel[1, 2].CellType = "CheckBox";
}
```

Likewise, you can also add other controls from the table above. A sample output is displayed below.



Button cell type

The cell type is used to the cell is changed as a button. It provides click event that can be triggered to perform the required option.

C#

```
var style = gridControl1.Model[1, 2];
style.CellType = "Button";
gridcontrol.CellButtonClick += Gridcontrol_CellButtonClick;
private void Gridcontrol_CellButtonClick(object sender,
GridCellButtonClickEventArgs e)
{
    MessageBox.Show("Clicked");
}
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
```

```
{
    e.Style.GridModel[1, 2].CellType = "Button";
}
```

Image cell type

The cell type is used to display the image in the specified cell.

C#

```
var style = gridControl1.Model[1, 2];
style.CellType = "Image";
style.GridModel.RowHeights[1] = 80;
Uri fileuri = new Uri(@"..\..\images\Avatar.jpeg", UriKind.Relative);
Image image = new System.Windows.Controls.Image();
image.Source = new BitmapImage(fileuri);
style.ImageList = new
System.Collections.ObjectModel.ObservableCollection<Image>() { image };
style.GridModel.ColumnWidths[5] = 120;
style.ImageIndex = 0;
```

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

ComboBox cell type

A combo box is a component with a drop-down arrow that users click to display an associated list of choices. The user displays the list by clicking or dragging the drop-down arrow.

This cell type allows you to choose the cell value from a drop-down list. You can customize this list in many ways by setting the appropriate GridStyleInfo property. Some interesting options are Autocomplete, associate a string collection, associate LINQ source etc. You can also use this drop-down like a foreign key– for example, displaying one column in the drop-down while saving the cell value from another column in the data source.

Note: A foreign key is a field in a relational table that matches the primary key column of another table. The foreign key can be used to cross-reference tables.

The table below lists various properties that can affect combo box cells.

GridStyleInfo Property	Description
CellType	Set to “ComboBox” for a Combo box control
ChoiceList	String collection for the drop-down
DropDownStyle	Determines the drop-down cell behavior. Editable Autocomplete Exclusive
ItemsSource	Specifies the binding source for the Combo box.
Display Member	String that names the public property from the data source object to be displayed in the cell.
Value Member	String that names the public property from the data source object to be used as the value for this cell.

Before we proceed further the following note provides more information on the drop-down styles:

Note: *Editable-Editable* combo boxes combine an editable text field and provide users the additional option of typing an item that might or might not be on the list. The item to be typed in the text field need not be case-sensitive. *_Autocomplete-Autocomplete* combo boxes predict a word or phrase that the user wants to type in the associated text box without the user actually typing it completely. * *Exclusive-* This is a non-editable combo box where user is allowed to select only the options available from the drop-down list.

Combo-boxes can be added to the Grid in two different ways as follows:

1. Using ChoiceList
2. Using ItemsSource

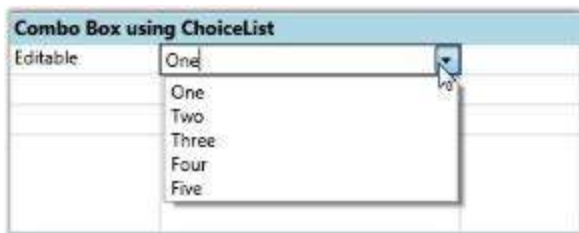
Using ChoiceList

Let us see how to build different kinds of combo boxes using ChoiceList. This allows you to customize the options to be displayed in a drop-down.

Setting Up an Editable ComboBox

C#

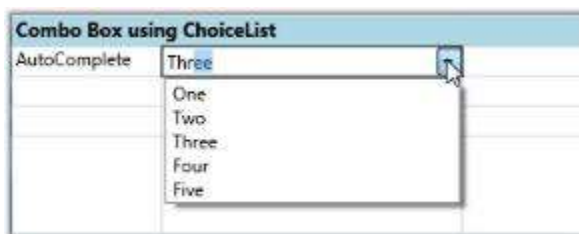
```
StringCollection list = new StringCollection();
list.Add("One");
list.Add("Two");
list.Add("Three");
list.Add("Four");
list.Add("Five");
//Editable Combo
var combol = this.grid.Model[1, 2];
combol.CellType = "ComboBox";
combol.ChoiceList = list;
combol.DropDownStyle = GridDropDownStyle.Editable;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
StringCollection collection = new StringCollection();
collection.Add("list1");
collection.Add("list2");
collection.Add("list3");
//Editable ComboBox
var combo = e.Style.GridModel[1, 2];
combo.CellType = "ComboBox";
combo.ChoiceList = collection;
combo.DropDownStyle = GridDropDownStyle.Editable;
}
```



Setting Up Autocomplete ComboBox

C#

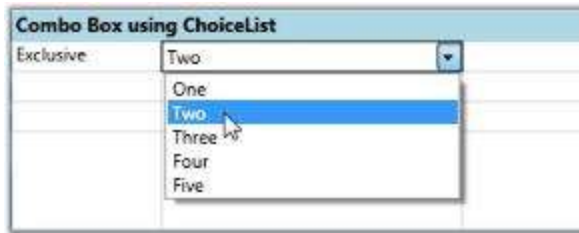
```
//Autocomplete combo
var combo2 = this.grid.Model[1, 2];
combo2.CellType = "ComboBox";
combo2.ChoiceList = list;
combo2.DropDownStyle = GridDropDownStyle.AutoComplete;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    //AutoComplete ComboBox
    var combo = e.Style.GridModel[1, 2];
    combo.CellType = "ComboBox";
    combo.ChoiceList = collection;
    combo.DropDownStyle = GridDropDownStyle.AutoComplete;
}
```



Setting Up Exclusive ComboBox

C#

```
//Exclusive Combo
var combo3 = this.grid.Model[1, 2];
combo3.CellType = "ComboBox";
combo3.ChoiceList = list;
combo3.DropDownStyle = GridDropDownStyle.Exclusive;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    //Exclusive ComboBox
    var combo = e.Style.GridModel[1, 2];
    combo.CellType = "ComboBox";
    combo.ChoiceList = collection;
    combo.DropDownStyle = GridDropDownStyle.Exclusive;
}
```



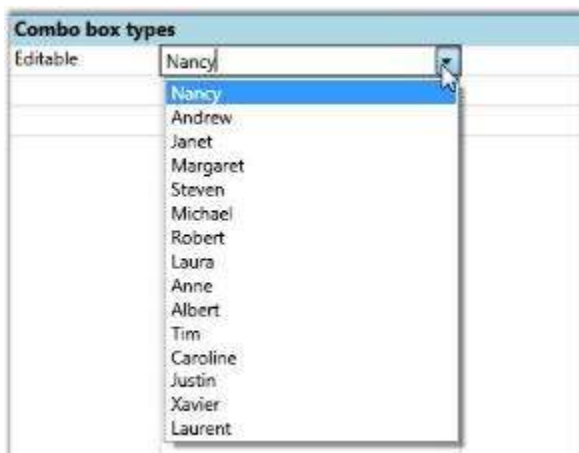
Using ItemsSource

The combo boxes created using ItemsSource class ensure that the options available in the drop-down list are populated from the data source the combo box is bound to. The user cannot customize the list unlike combo boxes created using ChoiceList class. The combo boxes in the following examples are bound to Northwind Employee table. The values of the FirstName column form the ItemsSource. The FirstName column is used as the display member of the combo box whose value member is EmployeeID.

Setting up editable combo box

C#

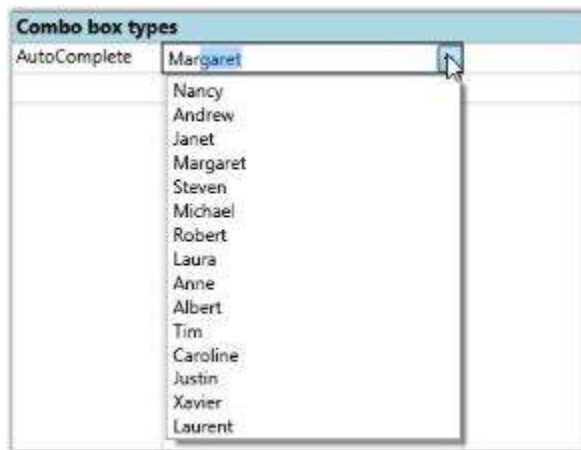
```
//Editable Combo bound to the "FirstName" column of Northwind Employee Table.
var combo1 = this.grid.Model[4, 2];
combo1.CellType = "ComboBox";
combo1.ItemsSource = northWind.Employees.Select(emp =>
emp.FirstName).ToList();
combo1.DropDownStyle = GridDropDownStyle.Editable;
```



Setting Up an Autocomplete Combo Box

C#

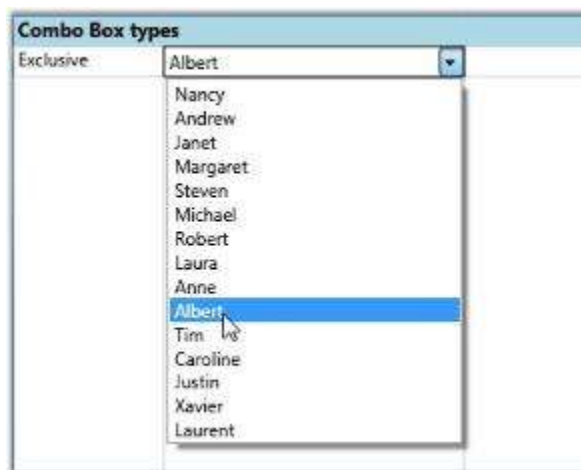
```
//Autocomplete Combo bound to the "FirstName" column of Northwind Employee Table.
var combo2 = this.grid.Model[5, 2];
combo2.CellType = "ComboBox";
combo2.ItemsSource = northWind.Employees.Select(emp =>
emp.FirstName).ToList();
combo2.DropDownStyle = GridDropDownStyle.AutoComplete;
combo2.DisplayMember = "FirstName";
combo2.ValueMember = "EmployeeID";
```



Setting Up an Exclusive Combo Box

C#

```
//Exclusive Combo bound to the "FirstName" column of Northwind Employee
//Table.
var combo3 = this.grid.Model[6, 2];
combo3.CellType = "ComboBox";
combo3.ItemsSource = northWind.Employees.Select(emp =>
emp.FirstName).ToList();
combo3.DropDownStyle = GridDropDownStyle.Exclusive;
```



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

DropDownList cell type

This cell type serves the same purpose as combo box control. The difference is that it will associate a multicolumn drop-down to the owner cell. The other common features like DropDownStyle, ItemsSource, DisplayMember and ValueMember are applicable to this cell too.

The code snippets below allow the user to construct different List Control Cells and their output. To set up drop-down List cell, set its CellType to "DropDownList".

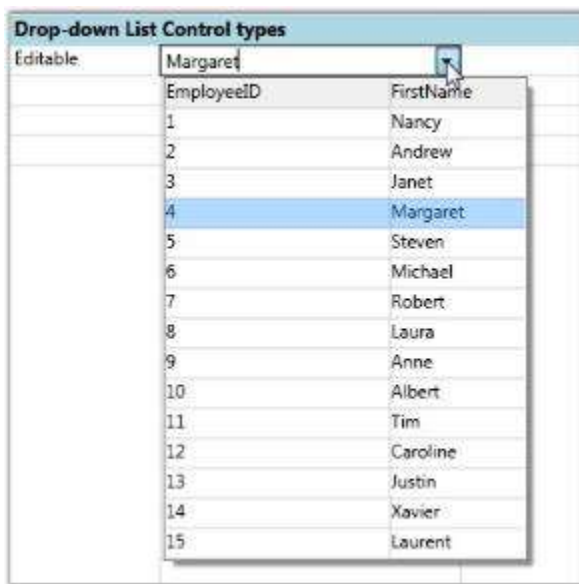
Creating Editable Drop-down List Bound to LINQ Source with 'FirstName' as its Display Member.

C#

```

var dropdown1 = this.grid.Model[7, 2];
dropdown1.CellType = "DropDownList";
dropdown1.ItemsSource = northWind.Employees.Select(emp =>
new
{
    EmployeeID = emp.EmployeeID,
    FirstName = emp.FirstName,
    LastName = emp.LastName,
    Phone = emp.HomePhone
}).ToList();
dropdown1.DisplayMember = "FirstName";
dropdown1.DropDownStyle = GridDropDownStyle.Editable;

```



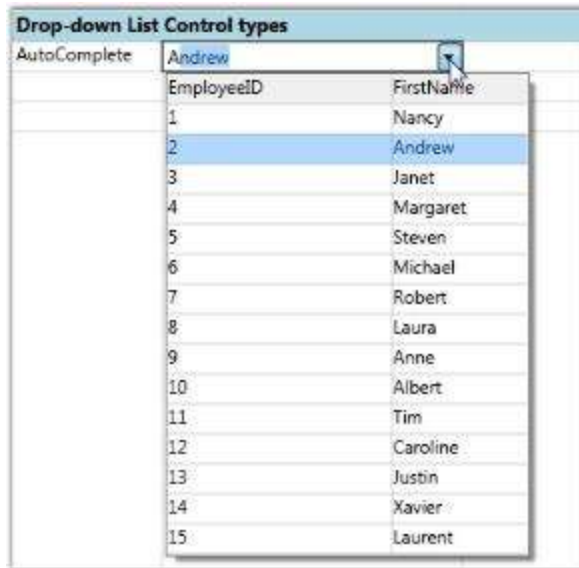
Autocomplete Drop-down List Bound to LINQ source with 'FirstName' as its Display Member and 'EmployeeID' as its ValueMember.

C#

```

var dropdown2 = this.grid.Model[8, 2];
dropdown2.CellType = "DropDownList";
dropdown2.ItemsSource = northWind.Employees.Select(emp =>
new
{
    EmployeeID = emp.EmployeeID,
    FirstName = emp.FirstName,
    LastName = emp.LastName,
    Phone = emp.HomePhone
}).ToList();
dropdown2.DisplayMember = "FirstName";
dropdown2.ValueMember = "EmployeeID";
dropdown2.DropDownStyle = GridDropDownStyle.AutoComplete;

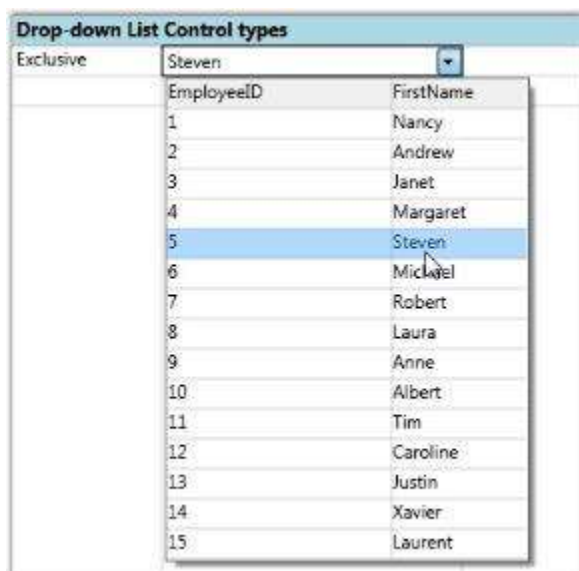
```

Exclusive Drop-down List Bound to LINQ Source with FirstName as its DisplayMember.

C#

```
var dropdown3 = this.grid.Model[9, 2];
dropdown3.CellType = "DropDownList";
dropdown3.ItemsSource = northWind.Employees.Select(emp =>
new
{
    EmployeeID = emp.EmployeeID,
    FirstName = emp.FirstName,
    LastName = emp.LastName,
    Phone = emp.HomePhone
}).ToList();
dropdown3.DisplayMember = "FirstName";
dropdown3.DropDownStyle = GridDropDownStyle.Exclusive;
```



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

[DateTimeEdit](#) cell type

The Date Time cells incorporate DateTimeEdit controls in grid cells that will help you to interactively set a date and time value. The style properties below are applicable to this cell type.

GridStyleInfo Property	Description
CellType	Set to "DateTimeEdit"
DateTimePattern	Sets the date-time pattern. The table below lists the available patterns with examples.
MaxDateTime, MinDateTime	Sets the maximum and minimum values for a DateTime cell.
IsCalendarEnabled	When true, enables the calendar popup
IsWatchEnabled	When true, enables the watch popup
NoneDateText	Specifies the text to be displayed when no date is set

Date and Time Pattern	Example
Short Date	8/6/2009
Long Date	Thursday, August 06, 2009
Long Time	7:01:33 AM
Short Time	7:01 AM
Full Date Time	Thursday, August 06, 2009 7:01:33 AM
MonthDay	August 06
RFC1123	Thu, 06 Aug 2009 07:01:33 GMT
Sortable Date Time	2009-08-06T07:01:33
Universal Sortable Date Time	2009-08-06 07:01:33Z
Year Month	(August, 2009 is correct)August, 2009

Setting Date and Time Cells with Different Date Time Patterns.

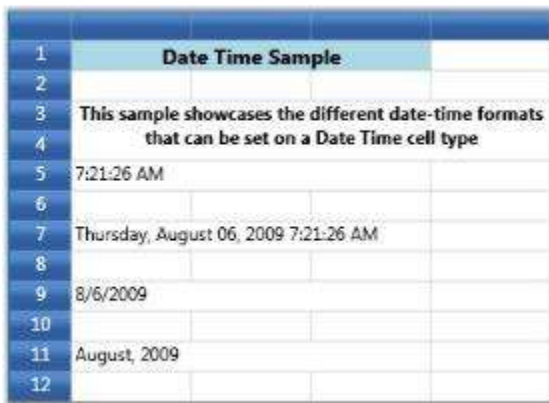
C#

```
grid.Model[5, 1].CellType = "DateTimeEdit";
grid.Model[5, 1].DateTimeEdit.DateTimePattern = DateTimePattern.LongTime;
grid.Model[5, 1].CellValue = DateTime.Now;
grid.Model[7, 1].CellType = "DateTimeEdit";
grid.Model[7, 1].DateTimeEdit.DateTimePattern =
DateTimePattern.FullDateTime;
grid.Model[7, 1].CellValue = DateTime.Now;
grid.Model[9, 1].CellType = "DateTimeEdit";
```

```

grid.Model[9, 1].DateTimeEdit.DateTimePattern = DateTimePattern.ShortDate;
grid.Model[9, 1].CellValue = DateTime.Now;
grid.Model[11, 1].CellType = "DateTimeEdit";
grid.Model[11, 1].DateTimeEdit.DateTimePattern = DateTimePattern.YearMonth;
grid.Model[11, 1].CellValue = DateTime.Now;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
e.Style.GridModel[5, 1].CellType = "DateTimeEdit";
e.Style.GridModel[5, 1].DateTimeEdit.DateTimePattern =
DateTimePattern.LongTime;
e.Style.GridModel[5, 1].CellValue = DateTime.Now;
e.Style.GridModel[7, 1].CellType = "DateTimeEdit";
e.Style.GridModel[7, 1].DateTimeEdit.DateTimePattern =
DateTimePattern.FullDateTime;
e.Style.GridModel[7, 1].CellValue = DateTime.Now;
e.Style.GridModel[9, 1].CellType = "DateTimeEdit";
e.Style.GridModel[9, 1].DateTimeEdit.DateTimePattern =
DateTimePattern.ShortDate;
e.Style.GridModel[9, 1].CellValue = DateTime.Now;
e.Style.GridModel[11, 1].CellType = "DateTimeEdit";
e.Style.GridModel[11, 1].DateTimeEdit.DateTimePattern =
DateTimePattern.YearMonth;
e.Style.GridModel[11, 1].CellValue = DateTime.Now;
}

```



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

IntegerEdit cell type

IntegerEdit is a specialized cell type that restricts the data entry to integers. The table below lists the style properties specific to this cell type.

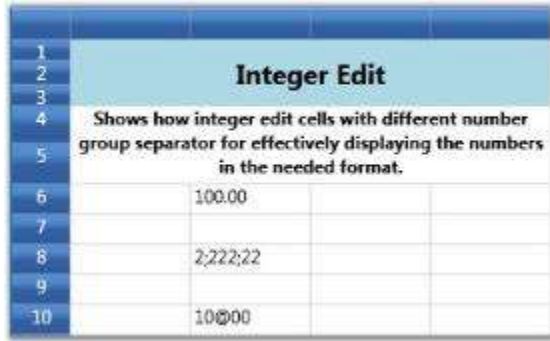
GridStyleInfo Property	Description
Cell Type	Set to "IntegerEdit"
NumberGroupSeparator	String that separates groups of digits
NumberGroupSizes	Number of digits in each group

UseNullOption	By default, the value of IntegerEdit cells is "0". Set UseNullOption as "True" to hide or delete the default value in IntegerEdit cells.
---------------	--

For example, setting up Three Different Integer Edit Cells.

C#

```
int[] sizes = { 2, 3, 4 };
grid.Model[12, 2].CellType = "IntegerEdit";
grid.Model[12, 2].IsEditable = true;
grid.Model[12, 2].NumberFormat = new NumberFormatInfo { NumberGroupSeparator = "," };
grid.Model[12, 2].NumberFormat.NumberGroupSizes = sizes;
grid.Model[12, 2].CellValue = 1;
grid.Model[8, 2].CellType = "IntegerEdit";
grid.Model[8, 2].IsEditable = true;
grid.Model[8, 2].NumberFormat = new NumberFormatInfo { NumberGroupSeparator = ";" };
grid.Model[8, 2].NumberFormat.NumberGroupSizes = sizes;
grid.Model[8, 2].CellValue = 222222;
grid.Model[10, 2].CellType = "IntegerEdit";
grid.Model[10, 2].IsEditable = true;
grid.Model[10, 2].NumberFormat = new NumberFormatInfo { NumberGroupSeparator = "@" };
grid.Model[10, 2].NumberFormat.NumberGroupSizes = sizes;
grid.Model[10, 2].CellValue = 1000;
grid.Model.ColStyles[3].CellType = "IntegerEdit";
grid.Model.ColStyles[3].IntegerEdit.UseNullOption = true;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    int[] sizes = { 2, 3, 4 };
    e.Style.GridModel[12, 2].CellType = "IntegerEdit";
    e.Style.GridModel[12, 2].IsEditable = true;
    e.Style.GridModel[12, 2].NumberFormat = new NumberFormatInfo {
        NumberGroupSeparator = "," };
    e.Style.GridModel[12, 2].NumberFormat.NumberGroupSizes = sizes;
    e.Style.GridModel[12, 2].CellValue = 1;
    e.Style.GridModel[8, 2].CellType = "IntegerEdit";
    e.Style.GridModel[8, 2].IsEditable = true;
    e.Style.GridModel[8, 2].NumberFormat = new NumberFormatInfo {
        NumberGroupSeparator = ";" };
    e.Style.GridModel[8, 2].NumberFormat.NumberGroupSizes = sizes;
    e.Style.GridModel[8, 2].CellValue = 222222;
    e.Style.GridModel[10, 2].CellType = "IntegerEdit";
    e.Style.GridModel[10, 2].IsEditable = true;
    e.Style.GridModel[10, 2].NumberFormat = new NumberFormatInfo {
        NumberGroupSeparator = "@" };
    e.Style.GridModel[10, 2].NumberFormat.NumberGroupSizes = sizes;
    e.Style.GridModel[10, 2].CellValue = 1000;
    e.Style.GridModel.ColStyles[3].CellType = "IntegerEdit";
    e.Style.GridModel.ColStyles[3].IntegerEdit.UseNullOption = true;
}
```



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

DoubleEdit cell type

Using DoubleEdit cell type will restrict the user to enter only double (value type) values into the cell. Thus it can be used to display System.Double type values. Below are the style properties that affect this cell.

GridStyleInfo Property	Description
Cell Type	Set to "DoubleEdit"
NumberGroupSeparator	String that separates groups of digits to the left of the decimal
NumberDecimalSeparator	String to use as decimal separator
NumberDecimalDigits	Number of decimal places

For example, setting up four Double Edit cells using different group separators and decimal digits.

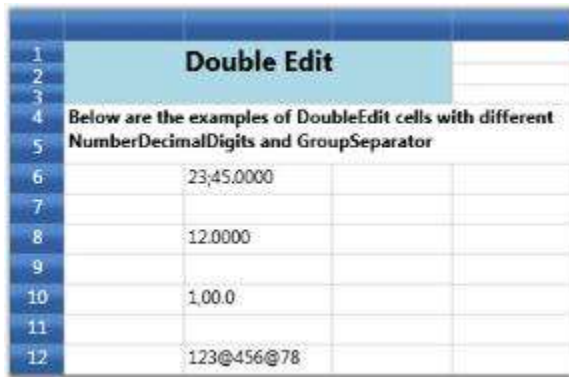
C#

```
int[] sizes = { 2, 3, 4 };
grid.Model[6, 2].CellType = "DoubleEdit";
grid.Model[6, 2].NumberFormat = new NumberFormatInfo
{
    NumberGroupSeparator = ";",
    NumberDecimalSeparator = ".",
    NumberDecimalDigits = 4
};
grid.Model[6, 2].NumberFormat.NumberGroupSizes = sizes;
grid.Model[6, 2].CellValue = 2345.00;
grid.Model[8, 2].CellType = "DoubleEdit";
grid.Model[8, 2].NumberFormat = new NumberFormatInfo
{
    NumberGroupSeparator = ",",
    NumberDecimalSeparator = ".",
    NumberDecimalDigits = 4
};
grid.Model[8, 2].NumberFormat.NumberGroupSizes = sizes;
grid.Model[8, 2].CellValue = 12;
grid.Model[10, 2].CellType = "DoubleEdit";
grid.Model[10, 2].NumberFormat = new NumberFormatInfo
{
    NumberGroupSeparator = ",",
```

```

NumberDecimalSeparator = ".",
NumberDecimalDigits = 1
};
grid.Model[10, 2].NumberFormat.NumberGroupSizes = sizes;
grid.Model[10, 2].CellValue = 100;
grid.Model[12, 2].CellType = "DoubleEdit";
grid.Model[12, 2].NumberFormat = new NumberFormatInfo
{
    NumberGroupSeparator = "@",
    NumberDecimalSeparator = ".",
    NumberDecimalDigits = 0
};
grid.Model[12, 2].NumberFormat.NumberGroupSizes = sizes;
grid.Model[12, 2].CellValue = 12345678.00;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    int[] sizes = { 2, 3, 4 };
    e.Style.GridModel[6, 2].CellType = "DoubleEdit";
    e.Style.GridModel[6, 2].NumberFormat.NumberGroupSizes = sizes;
    e.Style.GridModel[6, 2].CellValue = 2345.00;
    e.Style.GridModel[6, 2].NumberFormat = new NumberFormatInfo
    {
        NumberGroupSeparator = ";",
        NumberDecimalSeparator = ".",
        NumberDecimalDigits = 4
    };
    e.Style.GridModel[8, 2].CellType = "DoubleEdit";
    e.Style.GridModel[8, 2].NumberFormat.NumberGroupSizes = sizes;
    e.Style.GridModel[8, 2].CellValue = 12;
    e.Style.GridModel[8, 2].NumberFormat = new NumberFormatInfo
    {
        NumberGroupSeparator = ",",
        NumberDecimalSeparator = ".",
        NumberDecimalDigits = 4
    };
    e.Style.GridModel[10, 2].CellType = "DoubleEdit";
    e.Style.GridModel[10, 2].NumberFormat.NumberGroupSizes = sizes;
    e.Style.GridModel[10, 2].CellValue = 100;
    e.Style.GridModel[10, 2].NumberFormat = new NumberFormatInfo
    {
        NumberGroupSeparator = ",",
        NumberDecimalSeparator = ".",
        NumberDecimalDigits = 1
    };
    e.Style.GridModel[12, 2].CellType = "DoubleEdit";
    e.Style.GridModel[12, 2].NumberFormat = new NumberFormatInfo
    {
        NumberGroupSeparator = "@",
        NumberDecimalSeparator = ".",
        NumberDecimalDigits = 0
    };
    e.Style.GridModel[12, 2].NumberFormat.NumberGroupSizes = sizes;
    e.Style.GridModel[12, 2].CellValue = 12345678.00;
}

```



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

CurrencyEdit cell type

This cell type can be used to represent monetary values to achieve accuracy in the calculations. It will stripe the currency sign in the cell and attempt to parse only the number from the input. Use the GridStyleInfo properties below to customize these cells.

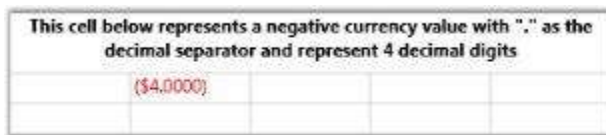
GridStyleInfo Property	Description
Cell Type	Set to "CurrencyEdit".
CurrencyDecimalDigits	Number of decimal places in currency value.
CurrencyDecimalSeparator	String to use as decimal separator.
CurrencyNegativePattern	Format pattern for negative currency values.
CurrencyPositivePattern	Format pattern for positive currency values.
CurrencySymbol	String to use as currency symbol.
CurrencyGroupSizes	Number of digits in each group to the left of the decimal.

Creating a Currency Cell with a Negative Currency Value with '.' as the Decimal Separator.

C#

```
int[] sizes = { 2, 3, 4 };
grid.Model[6, 2].CellType = "CurrencyEdit";
grid.Model[6, 2].IsEditable = true;
grid.Model[6, 2].NumberFormat = new NumberFormatInfo
{
    CurrencyDecimalDigits = 4,
    CurrencyDecimalSeparator = ".",
    CurrencyNegativePattern = 0,
    CurrencyPositivePattern = 0,
    CurrencySymbol = "$"
};
grid.Model[6, 2].NumberFormat.CurrencyGroupSizes = sizes;
grid.Model[6, 2].CellValue = -4.0;
//Using QueryCellInfo event
```

```
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    int[] sizes = { 2, 3, 4 };
    e.Style.GridModel[6, 2].CellType = "CurrencyEdit";
    e.Style.GridModel[6, 2].IsEditable = true;
    e.Style.GridModel[6, 2].NumberFormat = new NumberFormatInfo
    {
        CurrencyDecimalDigits = 4,
        CurrencyDecimalSeparator = ".",
        CurrencyNegativePattern = 0,
        CurrencyPositivePattern = 0,
        CurrencySymbol = "$"
    };
    e.Style.GridModel[6, 2].NumberFormat.CurrencyGroupSizes = sizes;
    e.Style.GridModel[6, 2].CellValue = -4.0;
}
```



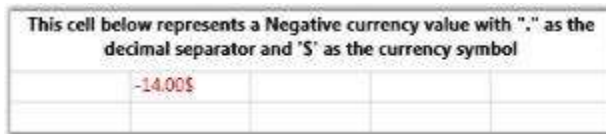
Currency Cell with a Negative Currency Value and a Different Negative Pattern

C#

```
int[] sizes = { 2, 3, 4 };
grid.Model[10, 2].CellType = "CurrencyEdit";
grid.Model[10, 2].IsEditable = true;
grid.Model[10, 2].NumberFormat = new NumberFormatInfo
{
    CurrencyDecimalDigits = 2,
    CurrencyDecimalSeparator = ".",
    CurrencyNegativePattern = 5,
    CurrencyPositivePattern = 1,
    CurrencySymbol = "$"
};
grid.Model[10, 2].NumberFormat.CurrencyGroupSizes = sizes;
grid.Model[10, 2].CellValue = -14.0;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    int[] sizes = { 2, 3, 4 };
    e.Style.GridModel[10, 2].CellType = "CurrencyEdit";
    e.Style.GridModel[10, 2].IsEditable = true;
    e.Style.GridModel[10, 2].NumberFormat = new NumberFormatInfo
    {
        CurrencyDecimalDigits = 2,
        CurrencyDecimalSeparator = ".",
        CurrencyNegativePattern = 5,
        CurrencyPositivePattern = 1,
        CurrencySymbol = "$"
    };
    e.Style.GridModel[10, 2].NumberFormat.CurrencyGroupSizes = sizes;
```



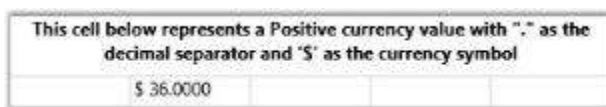
```
e.Style.GridModel[10, 2].CellValue = -14.0;
}
```



Currency Cell with a Positive Currency Value with '.' as the Decimal Separator and '\$' as Currency Symbol

C#

```
int[] sizes = { 2, 3, 4 };
grid.Model[14, 2].CellType = "CurrencyEdit";
grid.Model[14, 2].IsEditable = true;
grid.Model[14, 2].NumberFormat = new NumberFormatInfo
{
    CurrencyDecimalDigits = 4,
    CurrencyDecimalSeparator = ".",
    CurrencyNegativePattern = 11,
    CurrencyPositivePattern = 2,
    CurrencySymbol = "$"
};
grid.Model[14, 2].NumberFormat.CurrencyGroupSizes = sizes;
grid.Model[14, 2].CellValue = 36.0;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    int[] sizes = { 2, 3, 4 };
    e.Style.GridModel[14, 2].CellType = "CurrencyEdit";
    e.Style.GridModel[14, 2].IsEditable = true;
    e.Style.GridModel[14, 2].NumberFormat = new NumberFormatInfo
    {
        CurrencyDecimalDigits = 4,
        CurrencyDecimalSeparator = ".",
        CurrencyNegativePattern = 11,
        CurrencyPositivePattern = 2,
        CurrencySymbol = "$"
    };
    e.Style.GridModel[6, 2].NumberFormat.CurrencyGroupSizes = sizes;
    e.Style.GridModel[6, 2].CellValue = 36.0;
}
```



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

PercentEdit cell type

The PercentEdit cell type restricts the data entry to percentage values only. The following are the style properties used with this cell type.

GridStyleInfo Property	Description
------------------------	-------------

Cell Type	Set to "PercentEdit".
PercentEditMode	Indicates the way of editing the text in percent edit cells. Possible values – PercentMode and DoubleMode
PercentSymbol	String to use as the percent symbol.
PercentGroupSizes	Number of digits in each group to the left of the decimal.
PercentGroupSeparator	String that separates group of digits to the left of the decimal.
PercentDecimalDigits	Number of digits that appear after the decimal.

For example, setting up two Percent Edit cells with different group sizes and decimal digits.

The first cell operates in Percent mode of editing while the second cell follows Double mode.

Double mode displays the values in System.Double format and Percent mode adds a percent sign next to the numbers.

C#

```

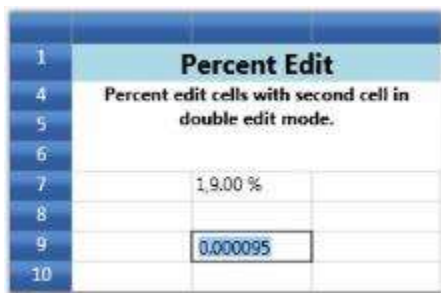
var percentStyleInfo = this.grid.Model[7, 2];
percentStyleInfo.CellType = "PercentEdit";
percentStyleInfo.NumberFormat = new NumberFormatInfo()
{
    PercentSymbol = "%",
    PercentGroupSizes = new int[] { 1, 2, 3 },
    PercentDecimalDigits = 2,
    PercentGroupSeparator = ",",
};
percentStyleInfo.PercentEditMode = PercentEditMode.PercentMode;
percentStyleInfo.CellValue = 19;
var percentStyleInfo2 = this.grid.Model[9, 2];
percentStyleInfo2.CellType = "PercentEdit";
percentStyleInfo2.NumberFormat = new NumberFormatInfo()
{
    PercentSymbol = "%",
    PercentGroupSizes = new int[] { 3 },
    PercentDecimalDigits = 4,
    PercentGroupSeparator = ",",
};
percentStyleInfo2.PercentEditMode = PercentEditMode.DoubleMode;
percentStyleInfo2.CellValue = 91;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    var percentStyleInfo = e.Style.GridModel[7, 2];
    percentStyleInfo.CellType = "PercentEdit";
    percentStyleInfo.NumberFormat = new NumberFormatInfo()
    {
        PercentSymbol = "%",
        PercentGroupSizes = new int[] { 1, 2, 3 },
        PercentDecimalDigits = 2,
        PercentGroupSeparator = ",",
    }
}

```

```

};
percentStyleInfo.PercentEditMode = PercentEditMode.PercentMode;
percentStyleInfo.CellValue = 19;
var percentStyleInfo2 = e.Style.GridModel[9, 2];
percentStyleInfo2.CellType = "PercentEdit";
percentStyleInfo2.NumberFormat = new NumberFormatInfo()
{
    PercentSymbol = "%",
    PercentGroupSizes = new int[] { 3 },
    PercentDecimalDigits = 4,
    PercentGroupSeparator = ",",
};
percentStyleInfo2.PercentEditMode = PercentEditMode.DoubleMode;
percentStyleInfo2.CellValue = 91;
}

```



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

MaskEdit cell type

MaskEdit cell type allows you to create specially formatted text cells that confirm to an edit mask that you specify. The `Style.MaskEdit.Mask` property holds the mask string, which will control the format of the input text being entered. The Mask Edit cells are useful when the user wants to display some formatted text such as Social Security Number (SSN), telephone number etc.

For example, setting up Mask Edit cells with different mask string.

C#

```

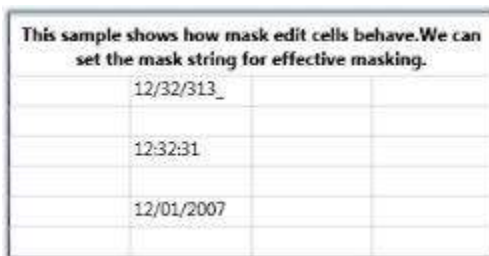
var maskStyleInfo = this.grid.Model[6, 2];
maskStyleInfo.CellType = "MaskEdit";
maskStyleInfo.MaskEdit = GridMaskEditInfo.Default;
maskStyleInfo.MaskEdit.Mask = "00/00/0000";
maskStyleInfo.CellValue = 1232313;
var maskStyleInfo1 = this.grid.Model[8, 2];
maskStyleInfo1.CellType = "MaskEdit";
maskStyleInfo1.MaskEdit = GridMaskEditInfo.Default;
maskStyleInfo1.MaskEdit.Mask = "00:00:00";
maskStyleInfo1.CellValue = 1232313;
var maskStyleInfo2 = this.grid.Model[10, 2];
maskStyleInfo2.CellType = "MaskEdit";
maskStyleInfo2.MaskEdit = GridMaskEditInfo.Default;
maskStyleInfo2.MaskEdit.Mask = "00/00/0000";
maskStyleInfo2.CellValue = "12012007";
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)

```

```

{
    var maskStyleInfo = e.Style.GridModel[6, 2];
    maskStyleInfo.CellType = "MaskEdit";
    maskStyleInfo.MaskEdit = GridMaskEditInfo.Default;
    maskStyleInfo.MaskEdit.Mask = "00/00/0000";
    maskStyleInfo.CellValue = 1232313;
    var maskStyleInfo1 = e.Style.GridModel[8, 2];
    maskStyleInfo1.CellType = "MaskEdit";
    maskStyleInfo1.MaskEdit = GridMaskEditInfo.Default;
    maskStyleInfo1.MaskEdit.Mask = "00:00:00";
    maskStyleInfo1.CellValue = 1232313;
    var maskStyleInfo2 = e.Style.GridModel[10, 2];
    maskStyleInfo2.CellType = "MaskEdit";
    maskStyleInfo2.MaskEdit = GridMaskEditInfo.Default;
    maskStyleInfo2.MaskEdit.Mask = "00/00/0000";
    maskStyleInfo2.CellValue = "12012007";
}

```



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

UpDownEdit cell type

UpDownEdit cell type makes the grid cell to host an Up and Down edit control which contains a pair of arrow buttons that increase or decrease the cell value. The style properties applicable to this cell type are provided below.

GridStyleInfo Property	Description
Cell Type	Set to "UpDownEdit"
NumberGroupSeparator	String that separates group of digits to the left of the decimal
NumberDecimalDigits	Number of digits that appear after the decimal
MaxValue	Upper limit in the range of applicable values
MinValue	Lower limit in the range of applicable values
Step	Unit value that is to be increased or decreased when the spin buttons are clicked
FocusedBorderBrush	Border brush; applied only when the cell is in focus
FocusedForeground	Foreground brush; applied only when the cell is in focus
FocusedBackground	Background brush; applied only when the cell is in focus

For example, the code below sets up two different Up and Down controls in grid cells.

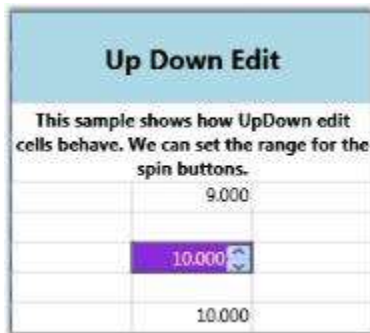
C#

```
var updownStyleInfo = this.grid.Model[6, 2];
updownStyleInfo.CellType = "UpDownEdit";
updownStyleInfo.NumberFormat = new NumberFormatInfo { NumberGroupSeparator =
" ", NumberDecimalDigits = 3 };
updownStyleInfo.UpDownEdit.FocusedBackground = Brushes.Tan;
updownStyleInfo.UpDownEdit.FocusedBorderBrush = Brushes.Red;
updownStyleInfo.UpDownEdit.FocusedForeground = Brushes.Yellow;
updownStyleInfo.UpDownEdit.MaxValue = 10.00;
updownStyleInfo.UpDownEdit.MinValue = 0;
updownStyleInfo.CellValue = 10.000;
var updownStyleInfo1 = this.grid.Model[8, 2];
updownStyleInfo1.CellType = "UpDownEdit";
updownStyleInfo1.NumberFormat = new NumberFormatInfo { NumberGroupSeparator =
" ", NumberDecimalDigits = 3 };
updownStyleInfo1.UpDownEdit.FocusedBackground = Brushes.BlueViolet ;
updownStyleInfo1.UpDownEdit.FocusedBorderBrush = Brushes.Red;
updownStyleInfo1.UpDownEdit.FocusedForeground = Brushes.Bisque ;
updownStyleInfo1.UpDownEdit.MaxValue = 100.00;
updownStyleInfo1.UpDownEdit.MinValue = 0;
updownStyleInfo1.CellValue = 10.000;
var updownStyleInfo2 = this.grid.Model[10, 2];
updownStyleInfo2.CellType = "UpDownEdit";
updownStyleInfo2.NumberFormat = new NumberFormatInfo { NumberGroupSeparator =
" ", NumberDecimalDigits = 3 };
updownStyleInfo2.UpDownEdit.FocusedBackground = Brushes.BurlyWood;
updownStyleInfo2.UpDownEdit.FocusedBorderBrush = Brushes.Red;
updownStyleInfo2.UpDownEdit.FocusedForeground = Brushes.Yellow;
updownStyleInfo2.UpDownEdit.MaxValue = 10.00;
updownStyleInfo2.UpDownEdit.MinValue = 0;
updownStyleInfo2.CellValue = 10.000;
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
var updownStyleInfo = e.Style.GridModel[6, 2];
updownStyleInfo.CellType = "UpDownEdit";
updownStyleInfo.NumberFormat = new NumberFormatInfo { NumberGroupSeparator =
" ", NumberDecimalDigits = 3 };
updownStyleInfo.UpDownEdit.FocusedBackground = Brushes.Tan;
updownStyleInfo.UpDownEdit.FocusedBorderBrush = Brushes.Red;
updownStyleInfo.UpDownEdit.FocusedForeground = Brushes.Yellow;
updownStyleInfo.UpDownEdit.MaxValue = 10.00;
updownStyleInfo.UpDownEdit.MinValue = 0;
updownStyleInfo.CellValue = 10.000;
var updownStyleInfo1 = e.Style.GridModel[8, 2];
updownStyleInfo1.CellType = "UpDownEdit";
updownStyleInfo1.NumberFormat = new NumberFormatInfo { NumberGroupSeparator =
" ", NumberDecimalDigits = 3 };
updownStyleInfo1.UpDownEdit.FocusedBackground = Brushes.BlueViolet;
updownStyleInfo1.UpDownEdit.FocusedBorderBrush = Brushes.Red;
updownStyleInfo1.UpDownEdit.FocusedForeground = Brushes.Bisque;
updownStyleInfo1.UpDownEdit.MaxValue = 100.00;
updownStyleInfo1.UpDownEdit.MinValue = 0;
```

```

updownStyleInfo1.CellValue = 10.000;
var updownStyleInfo2 = e.Style.GridModel[10, 2];
updownStyleInfo2.CellType = "UpDownEdit";
updownStyleInfo2.NumberFormat = new NumberFormatInfo { NumberGroupSeparator
= " ", NumberDecimalDigits = 3 };
updownStyleInfo2.UpDownEdit.FocusedBackground = Brushes.BurlyWood;
updownStyleInfo2.UpDownEdit.FocusedBorderBrush = Brushes.Red;
updownStyleInfo2.UpDownEdit.FocusedForeground = Brushes.Yellow;
updownStyleInfo2.UpDownEdit.MaxValue = 10.00;
updownStyleInfo2.UpDownEdit.MinValue = 0;
updownStyleInfo2.CellValue = 10.000;
}

```



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

RichText cell type

RichTextBox CellType is used to format the cells, where each character, word or a line can be given different formats. RichTextBox cell type also supports Printing, Importing and Exporting.

RichTextBox CellType can be defined in the Grid using the following code snippet:

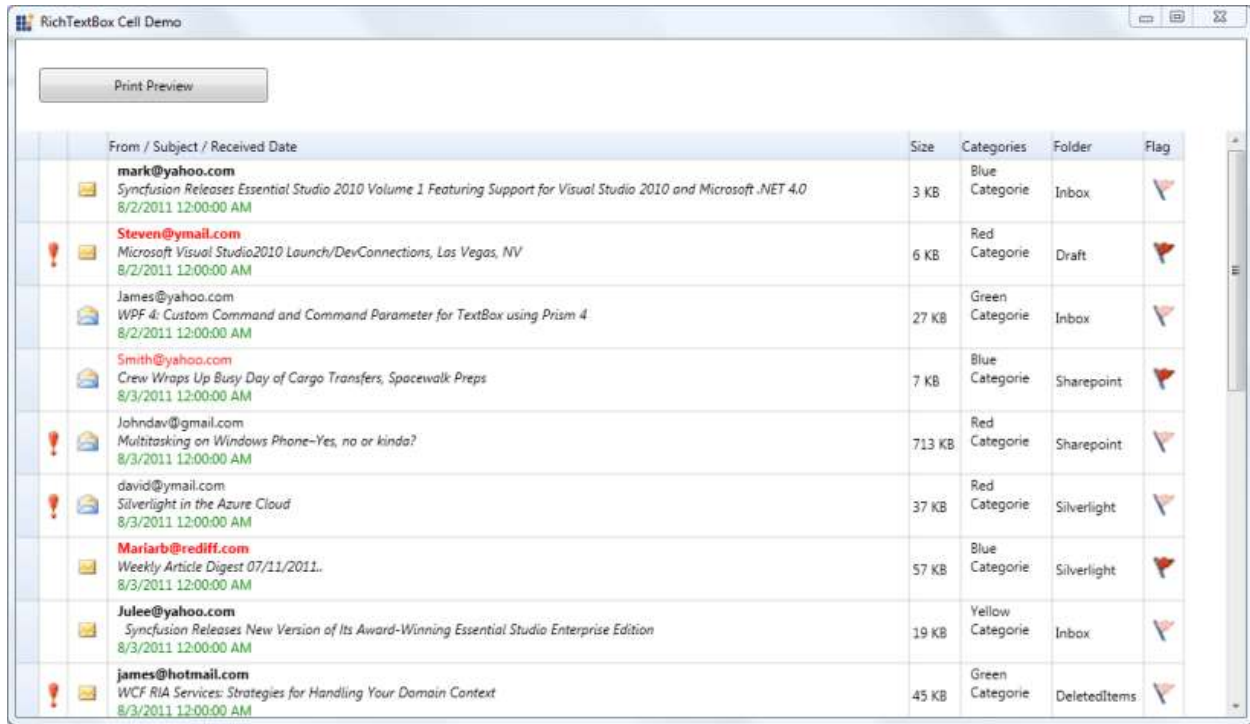
C#

```

//Cell type as RichText and Cell Value as FlowDocument
this.grid.Model[rowIndex, colIndex].CellType = "RichText";
this.grid.Model[rowIndex, colIndex].CellValue = _flowDocument;
//The Cell Value for RichTextBox must be in FlowDocument as shown below.
// Flow document type is supported for Rich Text Cell Type.
FlowDocument _flowDocument = new FlowDocument();
Paragraph _paragraph = new Paragraph();
Run _run1 = new Run();
_run1.Text = "This is RichText box Cell Type";
_run1.TextDecorations = TextDecorations.Underline;
Run _run2 = new Run();
_run2.Text = "Various formatting can be done in Single Cell.";
_run1.FontWeight = FontWeights.Bold;
_run2.Foreground = Brushes.Green;
Run _run3 = new Run();
_run3.Text = "Rich Text cell type also supports Images";
_run3.FontSize = 16;
_run3.FontStyle = FontStyles.Italic;
_paragraph.Inlines.Add(_run1);
_paragraph.Inlines.Add(_run2);
_paragraph.Inlines.Add(_run3);
_flowDocument.Blocks.Add(_paragraph);

```

```
//Cell type as RichText and Cell Value as FlowDocument
this.grid.Model[rowIndex, colIndex].CellType = "RichText";
this.grid.Model[rowIndex, colIndex].CellValue = _flowDocument;
```



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

DataTemplate cell type

The cell is specified as "DataBoundTemplate", that cell can not be changed or edited. You can show the cell information or text by setting the `DataTemplate` key to `GridStyleInfo.CellItemTemplateKey` property.

XML

```
<Window.Resources>
<DataTemplate x:Key="TextTemplate">
<Border BorderBrush="LightSlateGray"
BorderThickness="2"
CornerRadius="2">
<TextBlock Text="{Binding CellBoundValue}" ToolTip="{Binding
CellBoundValue}" />
</Border>
</DataTemplate>
</Window.Resources>
```

C#

```
gridcontrol.Model[2, 2].CellType = "DataBoundTemplate";
gridcontrol.Model[2, 2].CellItemTemplateKey = "TextTemplate";
gridcontrol.Model[2, 2].CellValue = gridcontrol.Model[2, 2].RowIndex;
```

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

Nested grid cell type

The Nested grid's can be added inside a row, column or covered range. When you nest a grid inside a covered range you can specify whether the rows or columns derive their state from the parent control. You have multiple independent options for both rows and columns.

The Nested grid cell type of WPF [GridControl](#) allows to host the one grid into another grid control. When you scroll within the parent grid, the nested grid also scrolls based on the parent grid scroller.

You can add the nested grids within the other grid control by the following ways.

- Nested grid inside a row or column
- Multiple nested grid inside a covered range with its rows using row layout
- Multiple nested grid inside a covered range with its columns using column layout
- Nested grid inside a covered with its rows and columns independent of parent grid

Note: The nested grid will have no separate scrollbars and shared with the parent grid.

Nested grid inside a row or column

You can add the nested grid into the parent grid row by using [grid.Model.CellModels.Add\(\)](#) method and need to specify the cell type as **ScrollGrid** in the first parameter and set the nested scroll grid model ([GridCellNestedScrollGridModel](#)) in the second parameter of this collection method. Before that, you must add a row and column to the nested grid and set the cell range for adding the nested grid to the row of parent grid.

Refer the below code for your reference.

C#

```
// Add Nested Scroll Grid cell model and set the cell type.
GridCellNestedScrollGridModel scrollGridModel = new
GridCellNestedScrollGridModel();
gridControl.Model.CellModels.Add("ScrollGrid", scrollGridModel);
gridControl.Model[2, 2].CellType = "ScrollGrid";
// Create a nested grid.
GridModel nestedGrid = new GridModel();
nestedGrid.Options.AllowSelection = GridSelectionFlags.Cell;
nestedGrid.RowHeights.DefaultLineSize = 20;
nestedGrid.RowCount = 20;
nestedGrid.ColumnWidths.DefaultLineSize = 50;
nestedGrid.ColumnCount = 15;
Brush headerBrush = ColorHelper.CreateFrozenSolidColorBrush(128,
Colors.DarkGray);
nestedGrid.BaseStylesMap["Header"].StyleInfo.Background = headerBrush;
for (int i = 0; i < nestedGrid.RowCount; i++)
{
    for (int j = 0; j < nestedGrid.ColumnCount; j++)
    {
        GridStyleInfo style = new GridStyleInfo();
        style.CellType = "TextBox";
        style.CellValue = String.Format("{0}:{1}", i, j);
        nestedGrid.Data[i, j] = style.Store;
    }
}
```



```
//Set the range and cell value for nested grid.
gridControl.Model[2, 2].CellValue = nestedGrid;
gridControl.CoveredCells.Add(new
Syncfusion.Windows.Controls.Cells.CoveredCellInfo(2, 2, 9, 5));
```

R0, C0	R0, C1	R0, C2	R0, C3	R0, C4	R0, C5	R0, C6
R1, C0						
R2, C0		0:0	0:1	0:2	0:3	0:4
R3, C0		1:0	1:1	1:2	1:3	1:4
R4, C0		2:0	2:1	2:2	2:3	2:4
R5, C0		3:0	3:1	3:2	3:3	3:4
R6, C0		4:0	4:1	4:2	4:3	4:4
R7, C0		5:0	5:1	5:2	5:3	5:4
R8, C0		6:0	6:1	6:2	6:3	6:4
R9, C0		<				>
R10, C0						

Note: Similarly, you can also create a nested grid inside a whole column.

Nested grid layout of row and column

[GridCellNestedGridModel](#) is the class to be used as model class for this cell type. Its constructor accepts two objects of type [GridNestedAxisLayout](#) enum, where the first parameter corresponds to row and second parameter corresponds to grid column. This enum value determines whether to share the row layout or column layout or the rows and columns are independent of parent grid.

Multiple nested grid inside a covered range with its rows using row layout

You can add the multiple number of nested grid into the parent grid row using row layout. You can achieve this requirement by the following way.

First, You can add the nested grid into the parent grid row by using [grid.Model.CellModels.Add\(\)](#) method and need to specify the cell type as **ScrollGrid** in the first parameter and set the nested cell grid model ([GridCellNestedGridModel](#)) in the second parameter of this collection method. Before that, you must add a row and column to the nested grid and set the cell range for adding the nested grid to the row of parent grid.

Note: You need to specify the row layout, use [GridNestedAxisLayout.Shared](#) enum in the first parameter and the [GridNestedAxisLayout.Normal](#) enum in the second parameter of [GridCellNestedGridModel](#).

Next, you can add another nested grid inside the first nested grid by setting the specific range of first nested grid and again do this for adding another nested grid.

Note: The nested grid will have its own unique column widths. But the row heights are shared with the parent grid.

For example, the below codes show a grid whose cell contains a nested grid, which again contains a nested grid in its cell, and this second nested grid again contains a nested grid in its cell.

C#

```
// Add Nested Grid cell model.
```

```

GridCellNestedGridModel shareRow = new
GridCellNestedGridModel(GridNestedAxisLayout.Shared,
GridNestedAxisLayout.Normal);
gridControl.Model.CellModels.Add("ScrollGrid", shareRow);
// Setup nested grid with shared row layout
gridControl.Model.RowCount = 200;
gridControl.Model.ColumnCount = 12;
gridControl.Model[2, 2].CellType = "ScrollGrid";
gridControl.Model[2, 2].Background = SystemColors.InactiveCaptionBrush;
GridModel nestedGridWithSharedRowsModel =
GetNestedGridWithSharedRowsModel();
gridControl.Model[2, 2].CellValue = nestedGridWithSharedRowsModel;
gridControl.CoveredCells.Add(new CoveredCellInfo(2, 2, 2 +
nestedGridWithSharedRowsModel.RowCount - 1, 9));
// Setup the top level(parent) nested grid
private GridModel GetNestedGridWithSharedRowsModel()
{
    GridModel model = new GridModel();
    Pen gridLinePen = new Pen(Brushes.DarkGray, 1);
    gridLinePen.Freeze();
    model.Options.AllowSelection = GridSelectionFlags.Cell;
    model.ColumnWidths.DefaultLineSize = 50;
    model.ColumnWidths.HeaderLineCount = 1;
    model.ColumnCount = 13;
    model.RowHeights.HeaderLineCount = 1;
    model.RowHeights.FooterLineCount = 1;
    model.RowCount = 151;
    Color clr = Color.FromArgb(128, 0, 0, 0);
    Brush headerBrush = new SolidColorBrush(clr);
    headerBrush.Freeze();
    Color clr2 = Color.FromArgb(128, 128, 0, 0);
    Brush footerBrush = new SolidColorBrush(clr2);
    footerBrush.Freeze();
    for (int i = 0; i < model.RowCount; i++)
    {
        for (int j = 0; j < model.ColumnCount; j++)
        {
            GridStyleInfo style = new GridStyleInfo();
            style.CellType = "TextBox";
            style.CellValue = String.Format("{0}:{1}", i, j);
            style.BorderMargins.Top = gridLinePen.Thickness;
            style.BorderMargins.Left = gridLinePen.Thickness;
            style.BorderMargins.Right = gridLinePen.Thickness / 2;
            style.BorderMargins.Bottom = gridLinePen.Thickness / 2;
            style.Borders.Right = gridLinePen;
            style.Background = null;
            style.Borders.Bottom = gridLinePen;
            model.Data[i, j] = style.Store;
            if (j == 0 || i == 0)
            {
                style.CellType = "Static";
                style.Background = headerBrush;
            }
            if (i == model.RowCount - 1)
            {
                style.CellType = "Static";
                style.Background = footerBrush;
            }
        }
    }
}

```

```

    }
    }
}
model[5, 2].CellType = "ScrollGrid";
model[5, 2].BorderMargins.Top = 0;
model[5, 2].BorderMargins.Left = 0;
model[5, 2].BorderMargins.Right = 0;
model[5, 2].BorderMargins.Bottom = 0;
model[5, 2].Background = SystemColors.InactiveCaptionBrush;
model.SelectedCells = GridRangeInfo.Empty;
// Creates a nested grid for second level.
GridModel nestedGridWithSharedRowsModel =
GetSecondNestedGridWithSharedRowsModel();
model[5, 2].CellValue = nestedGridWithSharedRowsModel;
model.CoveredCells.Add(new CoveredCellInfo(5, 2, 5 +
nestedGridWithSharedRowsModel.RowCount - 1, 7));
return model;
}
// Setup the second level nested grid
private GridModel GetSecondNestedGridWithSharedRowsModel()
{
    GridModel model = new GridModel();
    Pen gridLinePen = new Pen(Brushes.DarkGray, 1);
    gridLinePen.Freeze();
    model.Options.AllowSelection = GridSelectionFlags.Cell;
    model.ColumnWidths.DefaultLineSize = 40;
    model.ColumnWidths.HeaderLineCount = 1;
    model.ColumnCount = 8;
    model.RowHeights.HeaderLineCount = 1;
    model.RowHeights.FooterLineCount = 1;
    model.RowCount = 121;
    Color clr = Color.FromArgb(128, 0, 0, 128);
    Brush headerBrush = new SolidColorBrush(clr);
    headerBrush.Freeze();
    Color clr2 = Color.FromArgb(128, 0, 128, 0);
    Brush footerBrush = new SolidColorBrush(clr2);
    footerBrush.Freeze();
    for (int i = 0; i < model.RowCount; i++)
    {
        for (int j = 0; j < model.ColumnCount; j++)
        {
            GridStyleInfo style = new GridStyleInfo();
            style.CellType = "TextBox";
            style.CellValue = String.Format("{0}:{1}", i, j);
            style.BorderMargins.Top = gridLinePen.Thickness;
            style.BorderMargins.Left = gridLinePen.Thickness;
            style.BorderMargins.Right = gridLinePen.Thickness / 2;
            style.BorderMargins.Bottom = gridLinePen.Thickness / 2;
            style.Borders.Right = gridLinePen;
            style.Background = null; // Brushes.White;
            style.Borders.Bottom = gridLinePen;
            model.Data[i, j] = style.Store;
            if (i == 0 || j == 0)
            {
                style.CellType = "Static";
                style.Background = headerBrush;
            }
        }
    }
}

```

```

if (i == model.RowCount - 1)
{
    style.CellType = "Static";
    style.Background = footerBrush;
}
}
}
model[4, 2].CellType = "ScrollGrid";
model[4, 2].BorderMargins.Top = 0;
model[4, 2].BorderMargins.Left = 0;
model[4, 2].BorderMargins.Right = 0;
model[4, 2].BorderMargins.Bottom = 0;
model[4, 2].Background = Brushes.Wheat;
model.SelectedCells = GridRangeInfo.Empty;
//Creates a nested grid for third level.
GridModel nestedGridWithSharedRowsModel =
GetThirdNestedGridWithSharedRowsModel();
model[4, 2].CellValue = nestedGridWithSharedRowsModel;
model.CoveredCells.Add(new CoveredCellInfo(4, 2, 4 +
nestedGridWithSharedRowsModel.RowCount - 1, 5));
return model;
}
// Setup the third level nested grid
private GridModel GetThirdNestedGridWithSharedRowsModel()
{
    GridModel model = new GridModel();
    Pen gridLinePen = new Pen(Brushes.DarkGray, 1);
    gridLinePen.Freeze();
    model.Options.AllowSelection = GridSelectionFlags.Cell;
    model.ColumnWidths.DefaultLineSize = 35;
    model.ColumnWidths.HeaderLineCount = 1;
    model.ColumnCount = 5;
    model.RowHeights.HeaderLineCount = 1;
    model.RowHeights.FooterLineCount = 1;
    model.RowCount = 31;
    Color clr = Color.FromArgb(128, 0, 128, 128);
    Brush headerBrush = new SolidColorBrush(clr);
    headerBrush.Freeze();
    Color clr2 = Color.FromArgb(128, 128, 128, 0);
    Brush footerBrush = new SolidColorBrush(clr2);
    footerBrush.Freeze();
    for (int i = 0; i < model.RowCount; i++)
    {
        for (int j = 0; j < model.ColumnCount; j++)
        {
            GridStyleInfo style = new GridStyleInfo();
            style.CellType = "TextBox";
            style.CellValue = String.Format("{0}:{1}", i, j);
            style.BorderMargins.Top = gridLinePen.Thickness;
            style.BorderMargins.Left = gridLinePen.Thickness;
            style.BorderMargins.Right = gridLinePen.Thickness / 2;
            style.BorderMargins.Bottom = gridLinePen.Thickness / 2;
            style.Borders.Right = gridLinePen;
            style.Background = null;
            style.Borders.Bottom = gridLinePen;
            model.Data[i, j] = style.Store;
            if (i == 0 || j == 0)

```

```

{
    style.CellType = "Static";
    style.Background = headerBrush;
}
if (i == model.RowCount - 1)
{
    style.CellType = "Static";
    style.Background = footerBrush;
}
}
}
model.SelectedCells = GridRangeInfo.Empty;
return model;
}

```

0:0	0:1	0:2	0:3	0:4	0:5	0:6	0:7	0:8	0:9	0:10	0:11	0:12	0:13	0:14	0:15
1:0	1:1	1:2	1:3	1:4	1:5	1:6	1:7	1:8	1:9	1:10	1:11	1:12	1:13	1:14	1:15
2:0	2:1	2:2	2:3	2:4	2:5	2:6	2:7	2:8	2:9	2:10	2:11	2:12	2:13	2:14	2:15
3:0	3:1	3:2	3:3	3:4	3:5	3:6	3:7	3:8	3:9	3:10	3:11	3:12	3:13	3:14	3:15
4:0	4:1	4:2	4:3	4:4	4:5	4:6	4:7	4:8	4:9	4:10	4:11	4:12	4:13	4:14	4:15
5:0	5:1	5:2	5:3	5:4	5:5	5:6	5:7	5:8	5:9	5:10	5:11	5:12	5:13	5:14	5:15
6:0	6:1	6:2	6:3	6:4	6:5	6:6	6:7	6:8	6:9	6:10	6:11	6:12	6:13	6:14	6:15
7:0	7:1	7:2	7:3	7:4	7:5	7:6	7:7	7:8	7:9	7:10	7:11	7:12	7:13	7:14	7:15
8:0	8:1	8:2	8:3	8:4	8:5	8:6	8:7	8:8	8:9	8:10	8:11	8:12	8:13	8:14	8:15
9:0	9:1	9:2	9:3	9:4	9:5	9:6	9:7	9:8	9:9	9:10	9:11	9:12	9:13	9:14	9:15
10:0	10:1	10:2	10:3	10:4	10:5	10:6	10:7	10:8	10:9	10:10	10:11	10:12	10:13	10:14	10:15
11:0	11:1	11:2	11:3	11:4	11:5	11:6	11:7	11:8	11:9	11:10	11:11	11:12	11:13	11:14	11:15
12:0	12:1	12:2	12:3	12:4	12:5	12:6	12:7	12:8	12:9	12:10	12:11	12:12	12:13	12:14	12:15
13:0	13:1	13:2	13:3	13:4	13:5	13:6	13:7	13:8	13:9	13:10	13:11	13:12	13:13	13:14	13:15
14:0	14:1	14:2	14:3	14:4	14:5	14:6	14:7	14:8	14:9	14:10	14:11	14:12	14:13	14:14	14:15
15:0	15:1	15:2	15:3	15:4	15:5	15:6	15:7	15:8	15:9	15:10	15:11	15:12	15:13	15:14	15:15
16:0	16:1	16:2	16:3	16:4	16:5	16:6	16:7	16:8	16:9	16:10	16:11	16:12	16:13	16:14	16:15
17:0	17:1	17:2	17:3	17:4	17:5	17:6	17:7	17:8	17:9	17:10	17:11	17:12	17:13	17:14	17:15
18:0	18:1	18:2	18:3	18:4	18:5	18:6	18:7	18:8	18:9	18:10	18:11	18:12	18:13	18:14	18:15

Multiple nested grid inside a covered range with its columns using column layout

You can add the multiple number of nested grid into the parent grid column using column layout. You can achieve this requirement by the following way.

First, You can add the nested grid into the parent grid column by using [grid.Model.CellModels.Add\(\)](#) method and need to specify the cell type as **ScrollGrid** in the first parameter and set the nested cell grid model ([GridCellNestedGridModel](#)) in the second parameter of this collection method. Before that, you must add a row and column to the nested grid and set the cell range for adding the nested grid to the column of parent grid.

Note: You need to specify the column layout, use [GridNestedAxisLayout.Shared](#) enum in the second parameter and [GridNestedAxisLayout.Normal](#) enum in the first parameter of [GridCellNestedGridModel](#).

Next, you can add another nested grid inside the first nested grid by setting the specific range of first nested grid and again do this for adding another nested grid.

Note: The nested grid will have its own unique row height. But the column widths are shared with the parent grid.

For example, the below codes show a grid whose cell contains a nested grid, which again contains a nested grid in its cell.

C#

```
// Add the appropriate nested grid cell model.
GridCellNestedGridModel shareColumnLayoutGridModel = new
GridCellNestedGridModel(GridNestedAxisLayout.Normal,
GridNestedAxisLayout.Shared);
gridControl.Model.CellModels.Add("ScrollGrid", shareColumnLayoutGridModel);
gridControl.Model[2, 2].CellType = "ScrollGrid";
gridControl.Model[2, 2].BorderMargins.Top = 0;
gridControl.Model[2, 2].BorderMargins.Left = 0;
gridControl.Model[2, 2].BorderMargins.Right = 0;
gridControl.Model[2, 2].BorderMargins.Bottom = 0;
gridControl.Model[2, 2].Background = SystemColors.InactiveCaptionBrush;
GridModel nestedGridWithSharedColumnsModel =
GetNestedGridWithSharedColumnsModel();
// Creates a nested grid with shared column layout.
gridControl.Model[2, 2].CellValue = nestedGridWithSharedColumnsModel;
gridControl.CoveredCells.Add(new CoveredCellInfo(2, 2, 20, 1 +
nestedGridWithSharedColumnsModel.ColumnCount - 1));
// Sets up a nested grid with column layout shared
private GridModel GetNestedGridWithSharedColumnsModel()
{
GridModel model = new GridModel();
Pen gridLinePen = new Pen(Brushes.DarkGray, 1);
gridLinePen.Freeze();
model.Options.AllowSelection = GridSelectionFlags.Cell;
model.ColumnWidths.HeaderLineCount = 1;
model.ColumnCount = 10;
model.RowHeights.HeaderLineCount = 1;
model.RowHeights.FooterLineCount = 1;
model.RowCount = 13;
model.RowHeights.DefaultLineSize = 30;
Color clr = Color.FromArgb(128, 0, 0, 0);
Brush headerBrush = new SolidColorBrush(clr);
headerBrush.Freeze();
Color clr2 = Color.FromArgb(128, 128, 0, 0);
Brush footerBrush = new SolidColorBrush(clr2);
footerBrush.Freeze();
for (int i = 0; i < model.RowCount; i++)
{
for (int j = 0; j < model.ColumnCount; j++)
{
GridStyleInfo style = new GridStyleInfo();
style.CellType = "TextBox";
style.CellValue = String.Format("{0}:{1}", i, j);
style.BorderMargins.Top = gridLinePen.Thickness;
style.BorderMargins.Left = gridLinePen.Thickness;
style.BorderMargins.Right = gridLinePen.Thickness / 2;
style.BorderMargins.Bottom = gridLinePen.Thickness / 2;
style.Borders.Right = gridLinePen;
style.Background = null;
style.Borders.Bottom = gridLinePen;
model.Data[i, j] = style.Store;
if (j == 0 || i == 0)
```

```

{
    style.CellType = "Static";
    style.Background = headerBrush;
}
if (j == 3 || i == 3)
{
    style.CellType = "CheckBox";
    style.CellValue = false;
}
if (j == 4 || i == 4)
{
    style.CellType = "Static";
    style.CellValue = "Static";
}
if (i == model.RowCount - 1)
{
    style.CellType = "Static";
    style.Background = footerBrush;
}
}
}
model[4, 2].CellType = "ScrollGrid";
model[4, 2].BorderMargins.Top = 0;
model[4, 2].BorderMargins.Left = 0;
model[4, 2].BorderMargins.Right = 0;
model[4, 2].BorderMargins.Bottom = 0;
model[4, 2].Background = SystemColors.InactiveCaptionBrush;
model.SelectedCells = GridRangeInfo.Empty;
// Creates a nested grid for second level.
GridModel nestedGridWithSharedColumnsModel =
GetSecondNestedGridWithSharedColumnssModel();
model[4, 2].CellValue = nestedGridWithSharedColumnsModel;
model.CoveredCells.Add(new CoveredCellInfo(4, 2, 10, 1 +
nestedGridWithSharedColumnsModel.ColumnCount - 1));
return model;
}
private GridModel GetSecondNestedGridWithSharedColumnssModel()
{
    GridModel model = new GridModel();
    Pen gridLinePen = new Pen(Brushes.DarkGray, 1);
    gridLinePen.Freeze();
    model.Options.AllowSelection = GridSelectionFlags.Cell;
    model.ColumnWidths.HeaderLineCount = 1;
    model.ColumnCount = 10;
    model.RowHeights.HeaderLineCount = 1;
    model.RowHeights.FooterLineCount = 1;
    model.RowCount = 13;
    model.RowHeights.DefaultLineSize = 30;
    Color clr = Color.FromArgb(128, 0, 128, 128);
    Brush headerBrush = new SolidColorBrush(clr);
    headerBrush.Freeze();
    Color clr2 = Color.FromArgb(128, 128, 128, 0);
    Brush footerBrush = new SolidColorBrush(clr2);
    footerBrush.Freeze();
    for (int i = 0; i < model.RowCount; i++)
    {
        for (int j = 0; j < model.ColumnCount; j++)

```

```

{
    GridStyleInfo style = new GridStyleInfo();
    style.CellType = "TextBox";
    style.CellValue = String.Format("{0}:{1}", i, j);
    style.BorderMargins.Top = gridLinePen.Thickness;
    style.BorderMargins.Left = gridLinePen.Thickness;
    style.BorderMargins.Right = gridLinePen.Thickness / 2;
    style.BorderMargins.Bottom = gridLinePen.Thickness / 2;
    style.Borders.Right = gridLinePen;
    style.Background = null;
    style.Borders.Bottom = gridLinePen;
    model.Data[i, j] = style.Store;
    if (j == 0 || i == 0)
    {
        style.CellType = "Static";
        style.Background = headerBrush;
    }
    if (i == model.RowCount - 1)
    {
        style.CellType = "Static";
        style.Background = footerBrush;
    }
}
}
model.SelectedCells = GridRangeInfo.Empty;
return model;
}

```

0:0	0:1	0:2	0:3	0:4	0:5	0:6	0:7	0:8	0:9	0:10	0:11
1:0	1:1	1:2	1:3	1:4	1:5	1:6	1:7	1:8	1:9	1:10	1:11
2:0	2:1	0:0	0:1	0:2		Static	0:5	0:6	0:7	0:8	2:11
3:0	3:1	1:0	1:1	1:2		Static	1:5	1:6	1:7	1:8	3:11
4:0	4:1	2:0	2:1	2:2		Static	2:5	2:6	2:7	2:8	4:11
5:0	5:1					Static					5:11
6:0	6:1					Static					6:11
7:0	7:1					Static					7:11
8:0	8:1	Static	Static	0:0	0:1	0:2	0:3	0:4	0:5	0:6	8:11
9:0	9:1	5:0	5:1	1:0	1:1	1:2	1:3	1:4	1:5	1:6	9:11
10:0	10:1	6:0	6:1	2:0	2:1	2:2	2:3	2:4	2:5	2:6	10:11
11:0	11:1	7:0	7:1	3:0	3:1	3:2	3:3	3:4	3:5	3:6	11:11
12:0	12:1	8:0	8:1	4:0	4:1	4:2	4:3	4:4	4:5	4:6	12:11
13:0	13:1	9:0	9:1	5:0	5:1	5:2	5:3	5:4	5:5	5:6	13:11
14:0	14:1	10:0	10:1	6:0	6:1	6:2	6:3	6:4	6:5	6:6	14:11
15:0	15:1	11:0	11:1	7:0	7:1	7:2	7:3	7:4	7:5	7:6	15:11
16:0	16:1	12:0	12:1	8:0	8:1	8:2	8:3	8:4	8:5	8:6	16:11
17:0	17:1	13:0	13:1	9:0	9:1	9:2	9:3	9:4	9:5	9:6	17:11
18:0	18:1	14:0	14:1	10:0	10:1	10:2	10:3	10:4	10:5	10:6	18:11
19:0	19:1	15:0	15:1	11:0	11:1	11:2	11:3	11:4	11:5	11:6	19:11
20:0	20:1	16:0	16:1	12:0	12:1	12:2	12:3	12:4	12:5	12:6	20:11
21:0	21:1	17:0	17:1	13:0	13:1	13:2	13:3	13:4	13:5	13:6	21:11

Nested grid inside a covered with its rows and columns independent of parent grid

You can add the nested grid into the parent grid row and column by using [grid.Model.CellModels.Add\(\)](#) method and need to specify the cell type as **ScrollGrid** in the first parameter and set the nested cell grid model ([GridCellNestedGridModel](#)) in the second parameter of this collection method. Before that, you must add a row and column to the nested grid and set the cell range for adding the nested grid to the row of parent grid.

Note: You need to specify the row and column layout, use `Normal` option of [GridNestedAxisLayout](#) enum in both parameter of `GridCellNestedGridModel`.

In this case, The nested grid maintains its own row heights and column widths. You can scroll through this grid without scrolling the parent grid. You can resize the rows and columns in the nested grid without affect the parent grid.

For example, the below code is how to make rows and columns independent of parent grid.

C#

```
// Add Nested Grid cell model.
GridCellNestedGridModel gridModel = new
GridCellNestedGridModel(GridNestedAxisLayout.Normal,
GridNestedAxisLayout.Normal);
gridControl.Model.CellModels.Add("ScrollGrid", gridModel);
gridControl.Model[3, 2].CellType = "ScrollGrid";
// Create a simple nested grid.
GridModel model = new GridModel();
model.Options.AllowSelection = GridSelectionFlags.Cell;
model.RowHeights.DefaultLineSize = 20;
model.RowCount = 20;
model.ColumnWidths.DefaultLineSize = 50;
model.ColumnCount = 8;
model.HeaderRows = 0;
model.FrozenRows = 0;
model.HeaderColumns = 1;
model.FrozenColumns = 1;
for (int i = 0; i < model.RowCount; i++)
{
    for (int j = 0; j < model.ColumnCount; j++)
    {
        GridStyleInfo style = new GridStyleInfo();
        style.CellType = "TextBox";
        style.CellValue = String.Format("{0}:{1}", i, j);
        model.Data[i, j] = style.Store;
    }
}
gridControl.Model[3, 2].CellValue = model;
gridControl.CoveredCells.Add(new CoveredCellInfo(3, 2, 5, 4));
```

0:0	0:1	0:2	0:3	0:4	0:5	0:6	0:7
1:0	1:1	1:2	1:3	1:4	1:5	1:6	1:7
2:0	2:1	2:2	2:3	2:4	2:5	2:6	2:7
3:0	3:1	0:0	0:1	0:2	0:3	0:4	3:5
4:0	4:1	1:0	1:1	1:2	1:3	1:4	4:5
5:0	5:1	2:0	2:1	2:2	2:3	2:4	5:5
6:0	6:1	6:2	6:3	6:4	6:5	6:6	6:7
7:0	7:1	7:2	7:3	7:4	7:5	7:6	7:7
8:0	8:1	8:2	8:3	8:4	8:5	8:6	8:7
9:0	9:1	9:2	9:3	9:4	9:5	9:6	9:7

Note: [View sample in GitHub](#)

Creating custom cell type

Essential Grid allows you to create custom derived controls to use additional cell types. This requires a cell model class and a cell renderer class. The cell model class creates the actual cell control while the cell renderer class handles the UI requirements of the cell control. The custom cell type can be created by registering the cell model to the corresponding grid by naming this cell type. It can be enabled by assigning its name to the style.CellType property.

In general, the built-in cell types are also constructed only in this way. Every such cell type has its own cell model and renderer classes in the code base. These cell model and renderer classes originate from GridCellModelBase and GridCellRendererBase classes. These two classes define the basic functionality for a cell type.

Examples of custom cell types are discussed in later sections.

Custom Drop-down Cells

This cell displays customized drop-downs in grid cells. To attach a drop-down to a grid cell, you need to derive from GridCellDropDownCellModel and GridCellDropDownCellRenderer classes.

For example, let us create a custom drop-down which lists an image alongside text in each entry and sets the text of the current drop-down selection as the cell value. The cell model class just creates the cell type by calling the cell renderer. The cell renderer then loads the cell with ImageTextListBoxItem (a custom control having two properties, Image and Text) to show image alongside text. The renderer then overrides the ArrangeUIElement method in order to bind the drop down to the data source, which is a collection of ImageTextListBoxItem and sets its current selection based on current cell value. It triggers the ComboBoxSelectionChanged event to set the new cell value based on the current drop-down selection.

CellModel class

C#

```
public class CustomDropDownCellModel :
    GridCellDropDownCellModel<CustomDropDownRenderer>
{
}
```

CellRenderer Class

C#

```
public class CustomDropDownRenderer :
    GridCellDropDownCellRenderer<CustomDropDown>
{
    private CustomDropDownCellModel CustomDropDownModel
    {
        get
        {
            return this.CellModel as CustomDropDownCellModel;
        }
    }
    public override void OnInitializeContent(CustomDropDown dropDownControl,
        GridRenderStyleInfo style)
    {
        if (dropDownControl.ListBoxPart != null)
        {

```

```

dropDownControl.ListBoxPart.SelectionChanged -=
    this.OnComboBoxSelectionChanged;
}
base.OnInitializeContent(dropDownControl, style);
}
protected override void
ArrangeUIElement(Syncfusion.Windows.Controls.Cells.ArrangeCellArgs aca,
CustomDropDropDown uiElement, GridRenderStyleInfo style)
{
base.ArrangeUIElement(aca, uiElement, style);
var dropDownControl = uiElement;
if (style.ItemsSource != null)
{
dropDownControl.ListBoxPart.ItemsSource =
    this.CustomDropDropDownModel.GetDataSource(style);
dropDownControl.ListBoxPart.DisplayMemberPath = style.HasDisplayMember ?
    style.DisplayMember : string.Empty;
dropDownControl.ListBoxPart.SelectedValue = this.GetControlValue(style);
if (style.HasValueMember)
{
dropDownControl.ListBoxPart.SelectedValuePath = style.ValueMember;
}
}
uiElement.ListBoxPart.SelectionChanged += this.OnComboBoxSelectionChanged;
}
protected override void SetSelectedIndex(int index)
{
if (index != this.CurrentCellUIElement.ListBoxPart.SelectedIndex)
{
this.CurrentCellUIElement.ListBoxPart.SelectedIndex = index;
}
}
private void OnComboBoxSelectionChanged(object sender,
SelectionChangedEventArgs e)
{
if (e.AddedItems.Count > 0)
{
var item = e.AddedItems[0].ToString();
this.CustomDropDropDownModel.ListModel.CurrentIndex =
    this.CustomDropDropDownModel.FindValue(this.CurrentStyle, item);
if (!this.AlreadyTextChanged)
{
this.CurrentCellUIElement.TextBoxPart.Text = item;
}
}
}
}
}

```

Custom Drop-down control

C#

```

public class CustomDropDropDown : GridCellDropDownControlBase
{
public ImageTextListBox ListBoxPart
{
get

```

```

{
    if (this.PopupContent != null)
    {
        return this.PopupContent.Content as ImageTextListBox;
    }
    return null;
}

public override void OnApplyTemplate()
{
    base.OnApplyTemplate();
}

protected override void OnContentLoaded(ContentControl popupContent)
{
    ImageTextListBox l = new ImageTextListBox(this);
    l.Height = 200;
    popupContent.Content = l;
    // this will wire the events in the base implementation
    base.OnContentLoaded(popupContent);
}
}

```

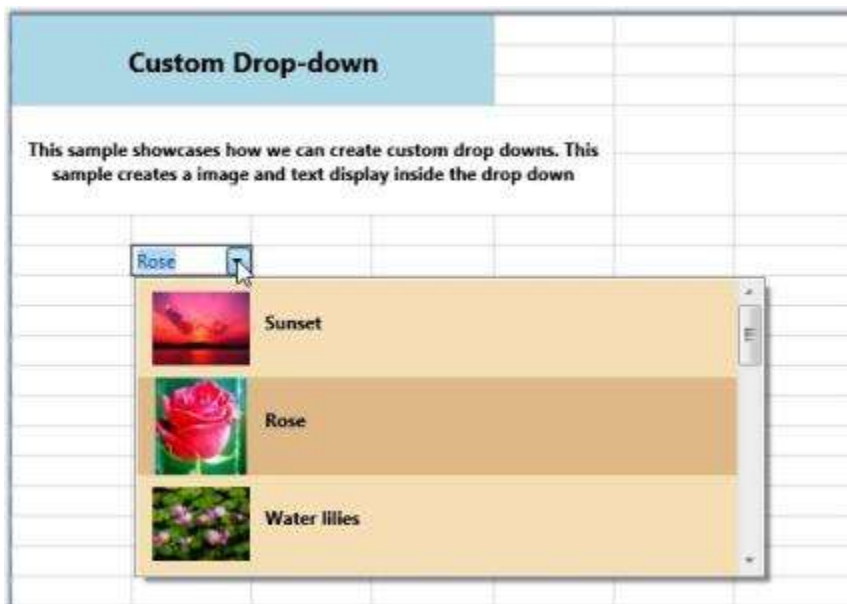
Associate this Cell Type to the Grid

C#

```

// Registering the cell model
this.grid.Model.CellModels.Add("CustomDropDown", new
CustomDropDownCellModel());
// Binding the celltype
var dropdown1 = this.grid.Model[7, 2];
dropdown1.CellType = "CustomDropDown";
dropdown1.ItemsSource = GenerateListBoxContent();
dropdown1.DisplayMember = "Text";
dropdown1.DropDownStyle = GridDropDownStyle.Editable;

```



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

Data Template Cells

This cell builds a custom data template that can be used to set enriched styles for associated cells. The `DataTemplateCellModel` creates the cell type with a Content Control (a WPF control) by calling the `renderer`.

Note: To create a cell type that hosts a WPF control, you should derive it from `GridVirtualizingCellRenderer`. The most important method to override is the `OnInitializeContent` method. It will be called for every UI element created for cells displaying this renderer. You can get access to the cell style from this method.

The `DataTemplateCellRenderer` is derived from `GridVirtualizingCellRenderer` and overrides `OnInitializeContent` and sets the Content Control template to `Style.CellItemTemplate` value.

CellModel Class

C#

```
public class DataTemplateCellModel : GridCellModel<DataTemplateCellRenderer>
{
}
```

CellRenderer Class

C#

```
public class DataTemplateCellRenderer :
GridVirtualizingCellRenderer<ContentControl>
{
    public DataTemplateCellRenderer()
    {
        IsFocusable = true;
        AllowRecycle = true;
    }
    public override void OnInitializeContent(ContentControl uiElement,
GridRenderStyleInfo style)
    {
        base.OnInitializeContent(uiElement, style);
        bool found = false;
        if (style.CellItemTemplateKey != null)
        {
            DataTemplate dt =
            (DataTemplate)style.GridControl.TryFindResource(style.CellItemTemplateKey);
            found = dt != null;
            if (found)
            {
                uiElement.ContentTemplate = dt;
            }
            if (!found)
            {
                uiElement.ContentTemplate = style.CellItemTemplate;
                uiElement.Content = style.CellValue;
            }
        }
        public override void CreateRendererElement(ContentControl uiElement,
GridRenderStyleInfo style)
        {
            bool found = false;
            if (style.CellItemTemplateKey != null)
            {

```

```

DataTemplate dt =
(DataTemplate)style.GridControl.TryFindResource(style.CellItemTemplateKey);
found = dt != null;
if (found)
uiElement.ContentTemplate = dt;
}
if (!found)
uiElement.ContentTemplate = style.CellItemTemplate;
uiElement.Content = style.CellValue;
base.CreateRenderElement(uiElement, style);
}
protected override string GetControlTextFromEditorCore(ContentControl
uiElement)
{
return uiElement.Content.ToString();
}
}

```

Data Template Definition

XML

```

<DataTemplate x:Key="editableEmployee">
<StackPanel Margin="8,0" Orientation="Horizontal">
<TextBlock FontWeight="Bold"
syncfusion:VisualContainer.WantsMouseInput="False" Text="{Binding
Path=Name}" Width="70" />
<TextBox Text="{Binding Path=Title}" BorderThickness="0" Padding="0"
Margin="0" Width="130" x:Name="tb"/>
</StackPanel>
</DataTemplate>

```

Setting up the Data Template Cell and assigning the Cell Template

C#

```

grid.Model.CellModels.Add("DataTemplate", new DataTemplateCellModel());
grid.Model.QueryCellInfo += new
Syncfusion.Windows.Controls.Grid.GridQueryCellInfoEventHandler(Model_QueryCe
llInfo);
void Model_QueryCellInfo(object sender,
Syncfusion.Windows.Controls.Grid.GridQueryCellInfoEventArgs e)
{
if (e.Cell.RowIndex > 1 && e.Cell.ColumnIndex == 2)
{
e.Style.CellType = "DataTemplate";
e.Style.CellItemTemplateKey = "editableEmployee";
e.Style.CellValue = employeesSource.Employees[e.Cell.RowIndex %
employeesSource.Employees.Count];
e.Style.Background = Brushes.Linen;
}
}
}

```



1			
2	Mark	Programming Writer	
3	Mary	Test Lead	
4	Karen	Developer	
5	George	Programming Writer	
6	Peter	Program Manager	
7	Matt	Program Manager	
8	Joan	Developer	
9	Mark	Programming Writer	
10	Mary	Test Lead	
11	Karen	Developer	
12	George	Programming Writer	

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

Rich Text Box Cells

The Rich Text control will allow you to display and edit rich text in grid cells. The control will allow you to modify the rich text through in-place editing.

For example, It can be built by hosting the Rich Text Box control in grid cells. To host this control, the cell renderer must be derived from `GridVirtualizingCellRenderer`, whose `OnInitializeContent` should be overridden to provide the content (as Flow Document) for the rich text box.

CellModel class

C#

```
public class RichTextBoxCellModel : GridCellModel<RichTextBoxCellRenderer>
{
}
```

CellRenderer class

C#

```
public class RichTextBoxCellRenderer :
    GridVirtualizingCellRenderer<RichTextBox>
{
    public RichTextBoxCellRenderer()
    {
        IsControlTextShown = false;
        IsFocusable = true;
    }

    public override void OnInitializeContent(RichTextBox textBox,
        GridRenderStyleInfo style)
    {
        textBox.Padding = new Thickness(0);
        FlowDocument document = GetControlValue(style) as FlowDocument;
        if (document == null)
            textBox.Document = new FlowDocument();
        if (document.Parent != null)
        {
            var parentTextBox = document.Parent as RichTextBox;
            parentTextBox.Document = new FlowDocument();
        }
        textBox.Document = document;
    }
}
```

```

VirtualizingCellsControl.SetWantsMouseInput(textBox, true);
}
protected override void OnUnwireUIElement(RichTextBox uiElement)
{
    uiElement.Document = new FlowDocument();
    base.OnUnwireUIElement(uiElement);
}
protected override object GetControlValueFromEditorCore(RichTextBox
uiElement)
{
    return uiElement.Document;
}
protected override void OnGridPreviewTextInput(TextCompositionEventArgs e)
{
    CurrentCell.ScrollInView();
    CurrentCell.BeginEdit(true);
}
protected override bool ShouldGridTryToHandlePreviewKeyDown(KeyEventArgs e)
{
    if (CurrentCellUIElement.IsFocused && e.Key != Key.Escape)
        return false;
    return true;
}
}

```

Setting up Rich Text Box Cell

C#

```

grid.Model.CellModels.Add("RichText", new RichTextBoxCellModel());
grid.Model.CellModels.Add("FlowDocumentReader", new
FlowDocumentReaderCellModel());
{
    // Create a FlowDocument to contain content for the RichTextBox.
    FlowDocument myFlowDoc = new FlowDocument();
    // Add paragraphs to the FlowDocument.
    myFlowDoc.Blocks.Add(new Paragraph(new Run("Paragraph 1")));
    myFlowDoc.Blocks.Add(new Paragraph(new Run("Paragraph 2")));
    myFlowDoc.Blocks.Add(new Paragraph(new Run("Paragraph 3")));
    grid.Model[2, 2].CellType = "RichText";
    grid.Model[2, 2].CellValue = myFlowDoc;
    grid.Model.CoveredCells.Add(new CoveredCellInfo(2, 2, 8, 8));
}
{
    Paragraph myParagraph = new Paragraph();
    // Add some Bold text to the paragraph
    myParagraph.Inlines.Add(new Bold(new Run("Some bold text in the
paragraph.")));
    // Add some plain text to the paragraph
    myParagraph.Inlines.Add(new Run(" Some text that is not bold."));
    // Create a List and populate with three list items.
    List myList = new List();
    // First create paragraphs to go into the list item.
    Paragraph paragraphListItem1 = new Paragraph(new Run("ListItem 1"));
    Paragraph paragraphListItem2 = new Paragraph(new Run("ListItem 2"));
    Paragraph paragraphListItem3 = new Paragraph(new Run("ListItem 3"));
    // Add ListItems with paragraphs in them.
}

```



```

myList.ListItems.Add(new ListItem(paragraphListItem1));
myList.ListItems.Add(new ListItem(paragraphListItem2));
myList.ListItems.Add(new ListItem(paragraphListItem3));
// Create a FlowDocument with the paragraph and list.
FlowDocument myFlowDocument = new FlowDocument();
myFlowDocument.Blocks.Add(myParagraph);
myFlowDocument.Blocks.Add(myList);
grid.Model[10, 2].CellType = "RichText";
grid.Model[10, 2].CellValue = myFlowDocument;
}

```

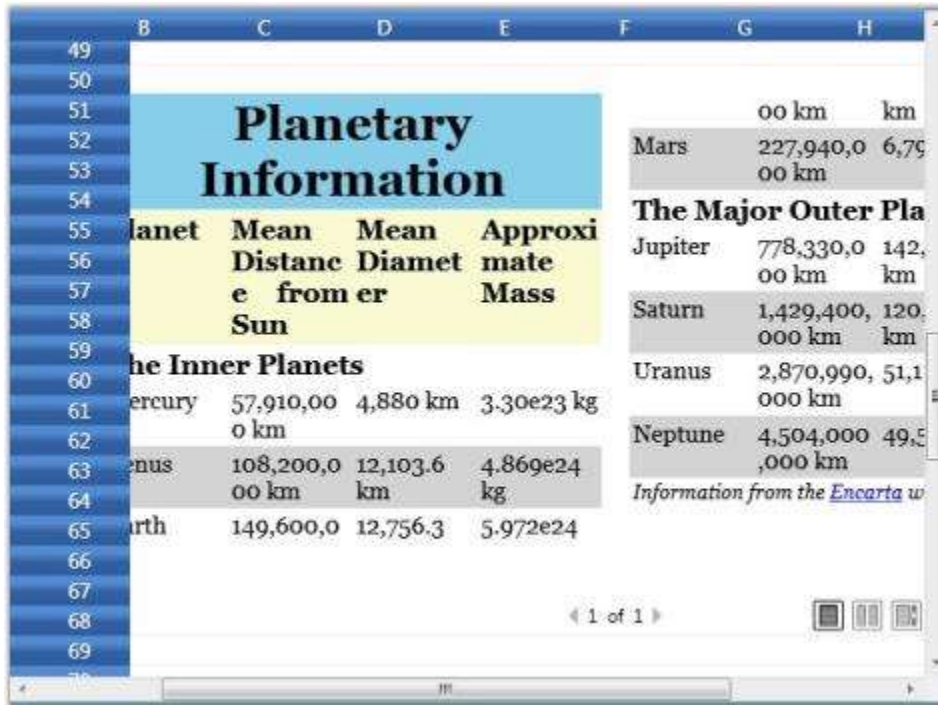


Chart Cells

Grid provides inherent support to host chart controls. This is achieved by using Data Template cells.

1. Define the Data Template that creates a chart. The template below illustrates the creation of a chart control with a single series and defines its attributes.

XML

```

<local:MyDataCollection x:Key="SeriesData1"/>
<DataTemplate x:Key="DataChart">
  <!--Hosting Chart control in second Row of the Grid-->
  <syncfusion:Chart x:Name="Chart1" Grid.Row="1" Margin="15">
    <!--Chart Legend declaration-->
    <syncfusion:Chart.Legends>
      <syncfusion:ChartLegend />
    </syncfusion:Chart.Legends>
    <!--Chart area to present chart segments-->
    <syncfusion:ChartArea IsContextMenuEnabled="True" >
      <!--Primary Axis (X)-->
      <syncfusion:ChartArea.PrimaryAxis>
        <!--X axis declaration with required property settings-->

```

```

<syncfusion:ChartAxis Header="Year" Interval="2" >
</syncfusion:ChartAxis>
</syncfusion:ChartArea.PrimaryAxis>
<!--Secondary Axis(Y)-->
<syncfusion:ChartArea.SecondaryAxis>
<!--Y axis declaration with required property settings-->
<syncfusion:ChartAxis Header="Profit" SmallTicksPerInterval="0"
LabelFormat="0.00">
</syncfusion:ChartAxis>
</syncfusion:ChartArea.SecondaryAxis>
<!--Chart 1st series declaration-->
<syncfusion:ChartSeries Name="series1" Label="Profit in $" Type="Spline"
StrokeThickness=" 3" Interior="Green" DataSource="{StaticResource
SeriesData1}"
BindingPathX="Year" BindingPathsY="Y1" IsIndexed="False">
</syncfusion:ChartSeries>
</syncfusion:ChartArea>
</syncfusion:Chart>
</DataTemplate >

```

Here is the data source definition that is used to define the chart series.

C#

```

public class MyData
{
    public int Year { get; set; }
    public double Y1 { get; set; }
    public double Y2 { get; set; }
    public double Y3 { get; set; }
    public double Y4 { get; set; }
}

public class MyDataCollection : ObservableCollection<MyData>
{
    public MyDataCollection()
    {
        Random rand = new Random(DateTime.Now.Millisecond);
        DateTime cdate = DateTime.Today.AddYears(-6);
        for (int i = 0; i < 5; i++)
        {
            this.Add(new MyData()
            {
                Year = cdate.AddYears(i).Year,
                Y1 = rand.Next(700, 1200),
            });
        }
    }
}

```

2.Bind the above template to the grid cell to form a chart cell.

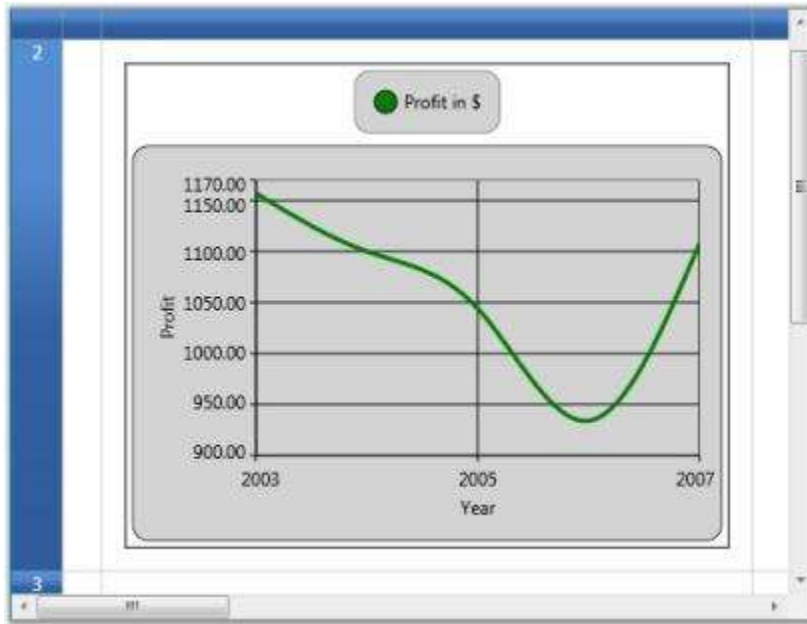
C#

```

var cell = grid.Model[2, 2];
cell.CellType = "DataTemplate";
cell.CellItemTemplateKey = "DataChart";

```

```
grid.Model.RowHeights[2] = 400d;
```



Inserting Images into Grid Cells

Grid provides inherent support to add images into grid cells. There are two possible ways to achieve this.

- **Style.Image** property—Used to insert an image alongside the text in the grid cells.
- **Style.ImageList** property—It serves the same purpose as the Image property, points to a collection of images and lets you share the same ImageSource for a group of cells. Once you have selected the images by using the ImageList property, you must set the ImageIndex property for individual cells, to indicate the specific image (from the ImageList) to be inserted into the cell.

Note: If you use both Image and ImageList properties, then the most recent property applied will be considered.

The following code example illustrates how to use these properties.

C#

```
// Using Vector Images.
ResourceDictionary dictionary = new ResourceDictionary();
dictionary.Source = new Uri(vectorImgSrcUri, UriKind.RelativeOrAbsolute);
ObservableCollection<Image> imgList = new ObservableCollection<Image>();
// Loading ImageList.
foreach (string key in dictionary.Keys)
{
    Image img = new Image();
    img.Source = (DrawingImage)dictionary["Technology"];
    imgList.Add(img);
}
// Setting ImageList.
grid.Model.TableStyle.ImageList = imgList;
// Image property setting.
```

```

grid.Model[0, 2].Text = "Technology";
Image img = new Image();
img.Source = new BitmapImage(new Uri("Technology.png", UriKind.Absolute));
grid.Model[0, 2].Image = img;
// Setting ImageIndex.
grid.Model[1, 2].Text = "Business";
grid.Model[1, 2].ImageIndex = 0;
grid.Model[2, 2].Text = "Software";
grid.Model[2, 2].ImageIndex = 1;

```

The following output is generated using the code above.



Formula Cells in WPF GridControl

Setting the CellType of a cell to a FormulaCell will allow you to enter algebraic expressions using formulas and cell references. Cell references are entries such as A11 for column A row 11 or BA3 for column BA row 3. A formula is a defined calculation from the Formula Library which, is included with Essential Grid. This Formula Library is extensible and allows you to add additional formulas.

This section comprises the following topics:

Defining a FormulaCell

You can use FormulaCells for every cell in a grid or for just a few cells. Even if you assign a CellType FormulaCell to every cell in a grid, the default behavior is to treat such cells as text box cells unless you start the cell entry with an equal sign. If the cell value starts with an equal sign then, the cell is considered as a formula cell and its contents are treated as such.

To make all cells present in a grid as potential formula cells, you will have to set the CellType of the standard BaseStyle to a FormulaCell using the following code.

C#

```

// Set up a Formula Cell.
this.gridControl1.BaseStylesMap["Standard"].StyleInfo.CellType =
"FormulaCell";

```

VB.NET

```
' Set up a Formula Cell.
Me.GridControl1.BaseStylesMap("Standard").StyleInfo.CellType =
"FormulaCell"
```

Using the Formula Library

Essential Grid's Formula Library contains the mathematical functions that are available in the .NET Framework's System.Math class. In addition, there are Sum and Avg members. For a complete list of these library functions, please see the Class Reference for GridFormulaEngine. You can also add additional functions to this library using your own code.

45	34	=a1+b1
56	5	=a2+b2
12	-5	=a3+b3
=sum(a1:a3)	=avg(b1:b3)	=a4+b4

In the above image, cell A2 has a formula that uses four different library functions: Sqrt, Pow, Cos, and Sin.

Supported Arithmetic Operators and Calculation Precedence

The current formula support will allow you to enter well-formed parenthetical algebraic expressions using operators and operands. The nine supported operators are shown in the precedence table given below, with operators on the same level being calculated as encountered when the expression is scanned from left to right.

Operations	Symbols	Calculation Precedence
Multiplication, Division	/ *	1st
Addition, Subtraction	+ -	2nd
Less Than, Greater Than, Equal, Less Than Or Equal, Greater Than Or Equal, Not Equal	< > = <= >= <>	3rd

The supported operands include those listed in the table given below. An operand by itself is also a well-formed algebraic expression that can serve as an entire formula in a cell.

Operand	Examples
number	532.1, -10.2, or 18.
cell reference	A12, BB1010, or Q18.
library formula with valid arguments	Abs(E14), Cos(-3.14), or Sum(A1:A14).
any well formed algebraic expression	E1+E2, Cos(2)

Within a formula cell, a case is ignored. So, a1 is the same as A1, and Cos(3) is the same as COS(3).

Adding Formulas to the Formula Library

Here are the steps that are required to add a function to the Function Library.

1. First, define a method that has this signature.

C#

```
// Define a method whose name is the FormulaName.
public string MyLibraryFormulaName(string args)
```

VB.NET

```
'Define a method whose name is the FormulaName.
Public Function MyLibraryFormulaName(ByVal args As String) As String
```

Here MyLibraryFormulaName must be a name that has not been already used in the Function Library and must include only letters and digits. If you want to replace an existing formula with a formula of the same name, first remove the existing formula before adding the new formula. Use the GridFormulaEngine.RemoveFunction method to remove a formula.

Then, write an implementation for your method. Here we use code to implement a function that will sum only positive numbers that are passed in as either a range like A1:A5 and/or a list such as A1, A4, A10. The code uses the FormulaEngine helper method to extract these values from the cells. The GetCellsFromArgs method will return an array of cells from a range such as A1:A5, and GetValueFromArgs method will take cells such as A3 and return a value such as 123.3.

C#

```
// Implement Your Method.
public string ComputeSumPosNums(string args)
{
    GridFormulaCellModel model = this.gridControl1.CellModels["FormulaCell"] as
    GridFormulaCellModel;
    if(model != null)
    {
        GridFormulaEngine engine = model.Engine;
        double sum = 0d;
        double d;
        string s1;
        // Loop through any arguments and sum up the positive values.
        foreach(string r in args.Split(new char[] { ',' }))
        {
            // Cell range
            if(r.IndexOf(':') > -1)
            {
                foreach(string s in engine.GetCellsFromArgs(r))
                {
                    // s is a cell line a21 or c3, and so on.
                    try
                    {
                        s1 = engine.GetValueFromArg(s);
                    }
                }
            }
        }
    }
}
```

```

catch(Exception ex)
{
    return ex.Message;
}
if(s1 != "")
{
    // Add only if positive.
    if(double.TryParse(s1, NumberStyles.Number, null, out d)
    && d > 0)
    {
        sum += d;
    }
}
}
else
{
    try
    {
        s1 = engine.GetValueFromArg(r);
    }
    catch(Exception ex)
    {
        return ex.Message;
    }
    if(s1 != "")
    {
        if(double.TryParse(s1, NumberStyles.Number, null, out d) && d > 0)
        {
            sum += d;
        }
    }
}
return sum.ToString();
}
return "";
}

```

VB.NET

```

' Implement Your Method.
Public Function ComputeSumPosNums(args As String) As String
Dim model As GridFormulaCellModel =
Me.gridControl1.CellModels("FormulaCell")
If Not (model Is Nothing) Then
Dim engine As GridFormulaEngine = model.Engine
Dim sum As Double = 0.0
Dim d As Double
Dim s1 As String
' Loop through any arguments and sum up the positive values.
Dim r As String
For Each r In args.Split(New Char() {"", "c"})
' Cell range
If r.IndexOf(":c") > -1 Then
Dim s As String

```

```

For Each s In engine.GetCellsFromArgs(r)
    ' s is a cell line a21 or c3, and so on.
    Try
    s1 = engine.GetValueFromArg(s)
    Catch ex As Exception
    Return ex.Message
    End Try
    If s1 <> "" Then
        ' Add only if positive.
        If Double.TryParse(s1, NumberStyles.Number, Nothing, d) And d > 0 Then
            sum += d
        End If
    End If
Next s
Else
    Try
    s1 = engine.GetValueFromArg(r)
    Catch ex As Exception
    Return ex.Message
    End Try
    If s1 <> "" Then
        If Double.TryParse(s1, NumberStyles.Number, Nothing, d) And d > 0 Then
            sum += d
        End If
    End If
End If
Next r
Return sum.ToString()
End If
Return ""
End Function

```

The last step is to actually add your formula to the library. You should do this after the grid has been created, say in a Form.Load event handler.

C#

```

GridFormulaCellModel cellModel = this.gridControl1.CellModels["FormulaCell"]
as GridFormulaCellModel;
// Add a formula named SumPosNums to the Library.
cellModel.Engine.AddFunction("SumPosNums", new
GridFormulaEngine.LibraryFunction(ComputeSumPosNums));

```

VB.NET

```

Dim cellModel As GridFormulaCellModel =
Me.gridControl1.CellModels("FormulaCell")
' Add a formula named SumPosNums to the Library.
cellModel.Engine.AddFunction("SumPosNums", New
GridFormulaEngine.LibraryFunction(AddressOf ComputeSumPosNums))

```

Function Reference Section

The Syncfusion Grid control supports 370 formulas under various categories including math, statistical, logical, engineering, information, date, time, text, web, financial, lookup, and database functions.

Math & Trigonometry functions

Name	Description	Syntax
ABS	Returns the number without its sign.	ABS(number)
ACOS	Returns the arccosine, or inverse cosine, of a number. The returned angle is given in radians in the range 0 (zero) to pi.	ACOS(number)number must be in the range of -1 to 1.
ACOSH	Returns the inverse hyperbolic cosine of a number. The inverse hyperbolic cosine is the value whose hyperbolic cosine is number.	ACOSH(number)number must be a real number.
ACOT	Returns the principal value of the arc cotangent, or inverse cotangent, of a number.	ACOT(number)number must be a real number.
ACOTH	Returns the inverse hyperbolic cotangent of a number.	ACOTH(number)number must be greater than 1.
ARABIC	Converts a Roman numeral to an Arabic numeral.	ARABIC(text)
ASIN	Returns the arcsine, or inverse sine, of a number. The returned angle is given in radians in the range -pi/2 to pi/2.	ASIN(number)number must be in the range of -1 to 1.
ASINH	Returns the inverse hyperbolic sine of a number.	ASINH(number)number must be a real number.
ATAN	Returns the arctangent, or inverse tangent, of a number. The returned angle is given in radians in the range -pi/2 to pi/2.	ATAN(number)

ATAN2	Returns the arctangent, or inverse tangent, of the specified x- and y-coordinates.	ATAN2(xnum, ynum)
ATANH	Returns the inverse hyperbolic tangent of a number.	ATANH(number)number must be in the range of -1 and 1.
BASE	Converts a number into a text representation with the given radix.	<i>BASE(Number, Radix, [Min_length])Number must be an integer greater than or equal to 0 and less than 2^53.Radix must be an integer greater than or equal to 2 and less than or equal to 36.Min_Length must be an integer greater than or equal to 0. (Optional)</i>
CEILING	Returns number rounded up to the nearest multiple of significance.	CEILING(number, significance)
CEILING.MATH	Rounds a number up to the nearest integer or to the nearest multiple of significance.	CEILING(number, [significance], [mode])number must be less than 9.99E+307 and greater than -2.229E-308.Significance: The multiple to which the number is to be rounded. (Optional)Mode: For negative numbers, controls whether number is rounded toward or away from zero. (Optional)
COMBIN	Returns the number of combinations for a given number of items.	COMBIN(number, number_chosen)
COMBINA	Returns the number of combinations (with repetitions) for a given number of items.	COMBINA(number, number_chosen)
COS	Returns the cosine of the given angle.	COS(number)
COSH	Returns the hyperbolic cosine of a number.	COSH(number)
COT	Returns the cotangent of an angle specified in radians.	COT(number)
COTH	Returns the hyperbolic cotangent of a hyperbolic angle.	COTH(number)

CSC	Returns the cosecant of an angle specified in radians.	CSC(number)
CSCH	Returns the hyperbolic cosecant of an angle specified in radians.	CSCH(number)
DECIMAL	Converts a text representation of a number in a given base into a decimal number.	DECIMAL(text, radix)Radix must be an integer.
DEGREES	Converts radians into degrees.	DEGREES(angle)
EVEN	Returns number rounded up to the nearest even integer.	EVEN(number)
EXP	Returns e raised to the power of number. The constant e equals 2.71828182845904, the base of the natural logarithm.	EXP(number)
FACT	Returns the factorial of a number.	FACT(number)
FACTDOUBLE	Returns the double factorial of a number.	FACTDOUBLE(number)
FLOOR	Rounds number down, toward zero, to the nearest multiple of significance.	FLOOR(number, significance)
GCD	Returns the greatest common divisor of two or more integers.	GCD(number1, [number2], ...)
INT	Rounds a number down to the nearest integer.	INT(number)
LCM	Returns the least common multiple of integers.	LCM(number1, [number2], ...)
LN	Returns the natural logarithm of a number.	LN(number)

LOG	Returns the logarithm of a number to the base you specify.	LOG(number, [base])Number must be a positive real number.Base: Optional, if base is omitted it is assumed to be 10.
LOG10	Returns the base 10 logarithm of a number.	LOG10(number)
MDETERM	Returns the matrix determinant of an array.	MDETERM(array)Array is a numeric array with an equal number of rows and columns.
MINVERSE	Returns the inverse matrix for the matrix stored in an array.	MINVERSE(array)Array is a numeric array with an equal number of rows and columns.
MMULT	Returns the matrix product of two arrays.	MMULT(array1, array2)
MOD	Returns the remainder after number is divided by divisor. The result has the same sign as divisor.	MOD(number, divisor)
MROUND	Returns a number rounded to the desired multiple.	MROUND(number, multiple)
MULTINOMIAL	Returns the ratio of the factorial of a sum of values to the product of factorials.	MULTINOMIAL(number1, [number2], ...)
MUNIT	Returns the unit matrix for the specified dimension.	MUNIT(dimension)Dimension must be an integer greater than 0.
ODD	Returns number rounded up to the nearest odd integer.	ODD(number)
PI	Returns the number 3.14159265358979	PI()
POW	Returns the result of a number raised to a power.	POW(number, power)
POWER	Returns the result of a number raised to a power.	POWER(number, power)

PRODUCT	Multiplies all of the numbers given as arguments and returns the product.	PRODUCT(number1, [number2], ...)
QUOTIENT	Returns the integer portion of a division.	QUOTIENT(numerator, denominator)
RADIANS	Converts degrees to radians.	RADIANS(angle)
ROMAN	Converts an Arabic numeral to a Roman numeral as text.	ROMAN(number, [form]) Number is an Arabic numeral you want converted. Form: Optional. A number specifying the type of Roman numeral you want
ROUND	Rounds a number to a specified number of digits.	ROUND(number, num_digits)
ROUNDDOWN	Rounds a number down, toward zero.	ROUNDDOWN(number, num_digits)
ROUNDUP	Rounds a number up, away from zero.	ROUNDUP(number, num_digits)
SEC	Returns the secant of an angle.	SEC(number)
SECH	Returns the hyperbolic secant of an angle.	SECH(number)
SIGN	Determines the sign of a number. Returns 1 if the number is positive, zero (0) if the number is 0, and -1 if the number is negative.	SIGN(number) Number must be a real number.
SIN	Returns the sine of the given angle.	SIN(number)
SINH	Returns the hyperbolic sine of a number.	SINH(number) Number must be a real number.
SQRT	Returns a positive square root.	SQRT(number)
SQRTPI	Returns the square root of (number * pi).	SQRTPI(number)
SUBTOTAL	Returns a subtotal in a list or database.	SUBTOTAL(functionnum, ref1, [ref2], ...) <i>functionnum</i> specifies which function to use in calculating subtotals.

		Function_num		Function
		1	101	AVERAGE
		2	102	COUNT
		3	103	COUNTA
		4	104	MAX
		5	105	MIN
		6	106	PRODUCT
		7	107	STDEV
		8	108	STDEVP
		9	109	SUM
		10	110	VAR
		11	111	VARP
		The function_num 1-11 includes hidden cell values, and the 101-111 ignores the hidden values.Ref1: NamedRange or ReferencesRef2: Optional , named range or cell references.		
SUM	Adds all the numbers that have been specified as arguments.	SUM(number1,[number2],...)		
SUMIFS	Adds the cells in a range that meet multiple criteria.	SUMIFS(sumrange,criteriarange1, criteria1, [criteriarange2, criteria2], ...)Sumrange: One or more cells to sum, including numbers or names, ranges, or cell references that contain numbers. Blank and text values are ignored.Criteriarange1: <i>The first range in which to evaluate the associated criteria.Criteria1: The criteria in the form of a number, expression etc.Criteriarange2 , Criteria2: Optional.</i>		
SUMPRODUCT	Multiplies corresponding components in the given arrays, and returns the sum of those products.	SUMPRODUCT(array1, [array2], [array3], ...)Array1: The first array argument whose components you want to multiply and then add.Array2, array3,... Optional.		

SUMSQ	Returns the sum of the squares of the arguments.	SUMSQ(number1, [number2], ...)
SUMX2MY2	Returns the sum of the difference of squares of corresponding values in two arrays.	SUMX2MY2(arrayx, arrayy)
SUMX2PY2	Returns the sum of the sum of squares of corresponding values in two arrays.	SUMX2PY2(arrayx, arrayy)
SUMXMY2	Returns the sum of squares of differences of corresponding values in two arrays.	SUMXMY2(arrayx, arrayy)
TAN	Returns the tangent of the given angle.	TAN(number)
TANH	Returns the hyperbolic tangent of a number.	TANH(number)
TRUNC	Truncates a number to an integer by removing the fractional part of the number.	TRUNC(number, [num_digits])
TRUNCATE	Truncates a number to an integer by removing the fractional part of the number.	TRUNCATE(number, [num_digits])

Statistical functions

Name	Description	Syntax
AVEDEV	Returns the average of the absolute deviations of data points from their mean.	AVEDEV(number1, [number2], ...)
AVERAGE	Returns the average (arithmetic mean) of the arguments.	AVERAGE(number1, [number2], ...)
AVERAGEA	Calculates the average (arithmetic	AVERAGEA(value1, [value2], ...)

	mean) of the values in the list of arguments.	
AVE	Equivalent to Average. Added for compatibility with older versions.	AVE(number1, [number2], ...)
BINOM.DIST	Returns the individual term binomial distribution probability.	BINOM.DIST(numbers, trials, probability, cumulative) <i>Numbers:</i> The number of successes in trials. <i>Trials:</i> The number of independent trials. <i>Probability:</i> The probability of success on each trial. <i>Cumulative:</i> A logical value (TRUE/FALSE)
BINOM.INV	Returns the smallest value for which the cumulative binomial distribution is greater than or equal to a criterion value.	BINOM.INV(trials, probability, alpha) <i>Trials:</i> The number of Bernoulli trials. <i>Probability:</i> The probability of a success on each trial. <i>Alpha:</i> The criterion value.
BINOMDIST	Equivalent to BINOM.DIST, added for compatibility with older versions.	BINOMDIST(numbers, trials, probability, cumulative)
CHIDIST	Equivalent to CHISQ.DIST, added for compatibility with older versions.	CHIDIST(x, deg_freedom)
CHIINV	Equivalent to CHISQ.INV, added for compatibility with older versions.	CHIINV(probability, deg_freedom)
CHISQ.DIST.RT	Returns the right-tail probability of the chi-squared distribution.	CHISQ.DIST.RT(x, degfreedom) <i>X:</i> The value at which you want to evaluate the distribution. <i>Degfreedom:</i> The number of degrees of freedom.
CHISQ.INV	Returns the inverse of the left-tail probability of the chi-squared distribution.	CHISQ.INV(probability, degfreedom) <i>Probability:</i> A probability associated with the chi-squared distribution. <i>Degfreedom:</i> The number of degrees of freedom.
CHISQ.INV.RT	Returns the inverse of the right-tail probability of the chi-squared distribution.	CHISQ.INV.RT(probability, degfreedom) <i>Probability:</i> A probability associated with the chi-squared distribution. <i>Degfreedom:</i> The number of degrees of freedom.

CHISQ.TEST	Returns the value from the chi-squared (χ^2)	CHISQ.TEST(actualrange,expectedrange) <i>Actualrange: The range of data that contains observations to test against expected values.Expectedrange: The range of data that contains the ratio of the product of row totals and column totals to the grand total.</i>
CHITEST	Equivalent to CHISQ.TEST, added for compatibility with older versions.	CHITEST(actualrange,expectedrange)
CONFIDENCE	Returns the confidence interval for a population mean, using a normal distribution.	CONFIDENCE(alpha,standarddev,size) <i>Alpha: The significance level used to compute the confidence level. Standarddev: The population standard deviation for the data range, it is assumed to be known.Size: The sample size.</i>
CONFIDENCE.NORM	Returns the confidence interval for a population mean, using a normal distribution.	CONFIDENCE.NORM(alpha,standarddev,size) <i>Alpha: The significance level used to compute the confidence level. Standarddev: The population standard deviation for the data range, it is assumed to be known.Size: The sample size.</i>
CONFIDENCE.T	Returns the confidence interval for a population mean, using a student's t distribution.	CONFIDENCE.T(alpha,standard_dev,size)
CORREL	Returns the correlation coefficient of the Array1 and Array2 cell ranges.	CORREL(array1, array2) <i>Array1: A cell range of values.Array2: A second cell range of values.</i>
COUNT	Counts the number of cells that contain numbers.	COUNT(value1, [value2], ...) <i>Value1, value2: Cell Reference or Range</i>
COUNTA	Counts the number of cells that are not empty in a range.	COUNTA(value1, [value2], ...) <i>value1: The first argument representing the values that you want to count.value2: Additional arguments representing the values that you want to count. (Optional)</i>
COUNTBLANK	Counts the empty cells in a specified range of cells.	COUNTBLANK(range)

COUNTIF	Counts the number of cells within a range that meet a single criterion that you specify.	COUNTIF(range, criteria)range: One or more cells to count.criteria: A number, expression, cell reference, or text string.
COVAR	Returns covariance, the average of the products of deviations for each data point pair in two data sets.	COVAR(array1,array2)Array1: The first cell range of integers.Array2: The second cell range of integers.
COVARIANCE.P	Returns population covariance, the average of the products of deviations for each data point pair in two data sets.	COVARIANCE.P(array1,array2)Array1: The first cell range of integers.Array2: The second cell range of integers.
COVARIANCE.S	Returns the sample covariance, the average of the products of deviations for each data point pair in two data sets.	COVARIANCE.S(array1,array2)Array1: The first cell range of integers.Array2: The second cell range of integers.
CRITBINOM	Equivalent to BINOM.INV, added for compatibility with older versions.	CRITBINOM(trials,probability_s,alpha)
DEVSQ	Returns the sum of squares of deviations of data points from their sample mean.	DEVSQ(number1, [number2], ...)
EXPON.DIST	Returns the exponential distribution.	EXPON.DIST(x,lambda,cumulative)X: The value of the function.Lambda: The parameter value.Cumulative: A logical value. If cumulative is TRUE, EXPON.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.
EXPONDIST	Equivalent to EXPON.DIST, added for Compatibility with older versions.	EXPONDIST(x,lambda,cumulative)

F.DIST	Returns the F probability distribution.	F.DIST(x,degfreedom1,degfreedom2,cumulative)X: The value at which to evaluate the function.Degfreedom1: <i>The numerator degrees of freedom</i> .Degfreedom2: The denominator degrees of freedom.Cumulative: A logical value. If cumulative is TRUE, F.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.
F.DIST.RT	Returns the (right-tailed) F probability distribution for two data sets.	F.DIST.RT(x,degfreedom1,degfreedom2)X :The value at which to evaluate the function.Degfreedom1: <i>The numerator degrees of freedom</i> .Degfreedom2: The denominator degrees of freedom.
F.INV.RT	Returns the inverse of the (right-tailed) F probability distribution.	F.INV.RT(probability,degfreedom1,degfreedom2)Probability: A probability associated with the F cumulative distribution.Degfreedom1: <i>The numerator degrees of freedom</i> .Degfreedom2: The denominator degrees of freedom.
FDIST	Equivalent to F.DIST, added for compatibility with older versions.	FDIST(x,degfreedom1,degfreedom2)
FINV	Equivalent to F.INV, added for compatibility with older versions.	FINV(probability,degfreedom1,degfreedom2)
FISHER	Returns the Fisher transformation at x.	FISHER(x)X: A numeric value for which you want the transformation.
FISHERINV	Returns the inverse of the Fisher transformation.	FISHERINV(y)Y:Â The value for which you want to perform the inverse of the transformation.
FORECAST	Calculates, or predicts, a future value using existing values.	FORECAST(x, knowny's, knownx's)X: The data point for which you want to predict a value.Knowny's: <i>The dependent array or range of data</i> .Knownx's: The independent array or range of data.
GAMMA.DIST	Returns the gamma distribution.	GAMMA.DIST(x,alpha,beta,cumulative)X: The value at which you want to evaluate the distribution.Alpha: A parameter to the distribution.Beta: A parameter to the distribution. Cumulative: If cumulative is TRUE, GAMMA.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.
GAMMA.INV	Returns the inverse of the gamma cumulative distribution.	GAMMA.INV(probability,alpha,beta)Probability: The probability associated with the gamma distribution.Alpha: A parameter to the distribution.Beta: A parameter to the

		distribution. If beta = 1, GAMMA.INV returns the standard gamma distribution.
GAMMADIST	Equivalent to GAMMA.DIST, added for compatibility with older versions.	GAMMADIST(x,alpha,beta,cumulative)
GAMMAINV	Equivalent to GAMMA.INV, added for compatibility with older versions.	GAMMAINV(probability,alpha,beta)
GAMMALN	Returns the natural logarithm of the gamma function, $\Gamma(x)$.	GAMMALN(x)X: The value for which you want to calculate GAMMALN.
GAMMALN.PRECISE	Returns the natural logarithm of the gamma function, $\Gamma(x)$.	GAMMALN.PRECISE(x)X: The value for which you want to calculate GAMMALN.PRECISE.
GEOMEAN	Returns the geometric mean of an array or range of positive data.	GEOMEAN(number1, [number2], ...)
HARMEAN	Returns the harmonic mean of a data set.	HARMEAN(number1, [number2], ...)
HYPGEOM.DIST	Returns the hypergeometric distribution.	HYPGEOM.DIST(samples,numbersample,populations,numberpop, ,cumulative)Samples: <i>The number of successes in the sample.</i> Numbersample: <i>The size of the sample.</i> Populations: <i>The number of successes in the population.</i> Numberpop: <i>The population size.</i> Cumulative: If cumulative is TRUE, then HYPGEOM.DIST returns the cumulative distribution function; if FALSE, it returns the probability mass function.
HYPGEOMDIST	Equivalent to HYPGEOM.DIST, added for compatibility with older versions.	HYPGEOMDIST(samples,numbersample,populations,numberpop)
INTERCEPT	Calculates the point at which a line will intersect the y-axis by using existing x-values and y-values.	INTERCEPT(knowny's, knownx's)Knowny's: <i>The dependent set of observations or data.</i> Knownx's: <i>The independent set of observations or data.</i>

KURT	Returns the kurtosis of a data set.	KURT(number1, [number2], ...)
LARGE	Returns the k-th largest value in a data set.	LARGE(array, k) Array: The array or range of data .K: The position in the array or cell range of data to return
LOGINV	Equivalent to LOGNORM.INV, added for compatibility with older versions.	LOGINV(probability, mean, standard_dev)
LOGNORM.DIST	Returns the log-normal distribution of x.	LOGNORM.DIST(x,mean,standarddev,cumulative) <i>X: The value at which to evaluate the function.Mean: The mean of ln(x).Standarddev: The standard deviation of ln(x).Cumulative: If cumulative is TRUE, LOGNORM.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.</i>
LOGNORM.INV	Returns the inverse of the log-normal cumulative distribution function of x.	LOGNORM.INV(probability, mean, standarddev) <i>Probability: A probability associated with the log-normal distribution.Mean: The mean of ln(x).Standarddev: The standard deviation of ln(x).</i>
LOGNORMDIST	Equivalent to LOGNORM.DIST, added for Compatibility with older versions.	LOGNORMDIST(x,mean,standard_dev)
MAX	Returns the largest value in a set of values.	MAX(number1, [number2], ...)
MAXA	Returns the largest value in the list of arguments.	MAXA(value1,[value2],...)
MEDIAN	Returns the median of the given numbers.	MEDIAN(number1, [number2], ...)
MIN	Returns the smallest value in a set of values.	MIN(number1, [number2], ...)
MINA	Returns the smallest value in the list of arguments.	MINA(value1, [value2], ...)

MODE	Returns the most frequently occurring, or repetitive, value in an array or range of data.	MODE(number1,[number2],...)
NEGBINOM.DIST	Returns the negative binomial distribution.	NEGBINOM.DIST(numberf,numbers,probabilitys,cumulative) <i>Numberf</i> : The number of failures. <i>Numbers</i> : The threshold number of successes. <i>Probabilitys</i> : The probability of a success. <i>Cumulative</i> : If cumulative is TRUE, NEGBINOM.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.
NEGBINOMDIST	Equivalent to NEGBINOM.DIST, added for compatibility with old versions.	NEGBINOMDIST(numberf,numbers,probability_s)
NORM.DIST	Returns the normal distribution for the specified mean and standard deviation.	NORM.DIST(x,mean,standarddev,cumulative) <i>X</i> : The value for which you want the distribution. <i>Mean</i> : The arithmetic mean of the distribution. <i>Standarddev</i> : The standard deviation of the distribution. <i>Cumulative</i> : If cumulative is TRUE, NORM.DIST returns the cumulative distribution function; if FALSE, it returns the probability mass function.
NORM.INV	Returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.	NORM.INV(probability,mean,standarddev) <i>Probability</i> : A probability corresponding to the normal distribution. <i>Mean</i> : The arithmetic mean of the distribution. <i>Standarddev</i> : The standard deviation of the distribution.
NORM.S.DIST	Returns the standard normal distribution.	NORM.S.DIST(z,cumulative) <i>Z</i> : The value for which you want the distribution. <i>Cumulative</i> : If cumulative is TRUE, NORMS.DIST returns the cumulative distribution function; if FALSE, it returns the probability mass function.
NORM.S.INV	Returns the inverse of the standard normal cumulative distribution.	NORM.S.INV(probability) <i>Probability</i> : A probability corresponding to the normal distribution.
NORMDIST	Equivalent to NORM.DIST, added for compatibility with old versions.	NORMDIST(x,mean,standard_dev,cumulative)
NORMINV	Equivalent to NORM.INV, added for	NORMINV(probability,mean,standard_dev)

	compatibility with old versions.	
NORMSDIST	Equivalent to NORM.S.DIST, added for compatibility with old versions.	NORMSDIST(z,cumulative)
NORMSINV	Equivalent to NORM.S.INV, added for compatibility with old versions.	NORMSINV(z,cumulative)
PEARSON	Returns the Pearson product moment correlation coefficient.	PEARSON(array1, array2)Array1: A set of independent values.Array2: A set of dependent values.
PERCENTILE	Returns the k-th percentile of values in a range.	PERCENTILE(array,k)Array: The array or range of data that defines relative standing.K: The percentile value in the range of 0 to 1, inclusively.
PERCENTILE.EXC	Returns the k-th percentile of values in a range, where k is in the range of 0 to 1, exclusively.	PERCENTILE.EXC(array,k)Array: The array or range of data that defines relative standing.K: The percentile value in the range of 0 to 1, exclusively.
PERCENTILE.INC	Returns the k-th percentile of values in a range, where k is in the range of 0 to 1, inclusively.	PERCENTILE.INC(array,k)Array: The array or range of data that defines relative standing.K: The percentile value in the range 0 to 1, exclusively.
PERCENTRANK	Returns the rank of a value in a data set as a percentage of the data set.	PERCENTRANK(array,x,[significance])Array: The array or range of data with numeric values that defines relative standing.X: The value for which you want to know the rank.Significance: A value that identifies the number of significant digits for the returned percentage value.
PERCENTRANK.EXC	Returns the rank of a value in a data set as a percentage (0 to 1, exclusively) of the data set.	PERCENTRANK.EXC(array,x,[significance])Array: The array or range of data with numeric values that defines relative standing.X: The value for which you want to know the rank.Significance: A value that identifies the number of significant digits for the returned percentage value.
PERCENTRANK.INC	Returns the rank of a value in a data set as a percentage (0 to 1, inclusively) of the data set.	PERCENTRANK.INC(array,x,[significance])Array: The array or range of data with numeric values that defines relative standing.X: The value for which you want to know the rank.Significance: A value that identifies the number of significant digits for the returned percentage value.

PERMUT	Returns the number of permutations for a given number of objects that can be selected from number objects.	PERMUT(number, numberchosen) <i>Number: An integer that describes the number of objects.</i> <i>Numberchosen: An integer that describes the number of objects in each permutation.</i>
PERMUTATIONA	Returns the number of permutations for a given number of objects (with repetitions) that can be selected from the total objects.	PERMUTATIONA(number, number-chosen) <i>Number: An integer that describes the number of objects.</i> <i>Number_chosen: An integer that describes the number of objects in each permutation.</i>
POISSON	This function has been replaced with the POISSON.DIST function. This function is available for compatibility with older versions.	POISSON(x,mean,cumulative)
POISSON.DIST	Returns the Poisson distribution.	POISSON.DIST(x,mean,cumulative) <i>X: The number of events.</i> <i>Mean: The expected numeric value.</i> <i>Cumulative: If cumulative is TRUE, POISSON.DIST returns the cumulative Poisson distribution. If FALSE, it returns the Poisson probability mass function.</i>
PROB	Returns the probability that values in a range are between two limits.	PROB(xrange, probrange, [lowerlimit], [upperlimit]) <i>Xrange: The range of numeric values of x with which there are associated probabilities.</i> <i>Probrange: A set of probabilities associated with values in xrange.</i> <i>Lowerlimit: The lower limit on the value for which you want a probability.</i> <i>Upper_limit: The optional upper limit on the value for which you want a probability.</i>
QUARTILE	This function has been replaced with QUARTILE.EXC and QUARTILE.INC. However, this function is available for compatibility with older versions.	QUARTILE(array,quart)
QUARTILE.EXC	Returns the quartile of the data set, based on percentile values	QUARTILE.EXC(array, quart) <i>Array: The array or cell range of numeric values for which you want the quartile value.</i> <i>Quart: Indicates which value to return.</i>

	from 0 to1, exclusively.	
QUARTILE.INC	Returns the quartile of a data set, based on percentile values from 0 to 1, inclusively.	QUARTILE.INC(array,quart)Array: The array or cell range of numeric values for which you want the quartile value.Quart: Indicates which value to return.
RANK.AVG	Returns the rank of a number in a list of numbers.	RANK.AVG(number,ref,[order])Number: The number whose rank you want to find.Ref: An array of, or a reference to, a list of numbers. Non numeric values in Ref are ignored.Order: A number specifying how to rank number.
RANK.EQ	Returns the rank of a number in a list of numbers.	RANK.EQ(number,ref,[order])Number: The number whose rank you want to find.Ref: An array of, or a reference to, a list of numbers. Non numeric values in Ref are ignored.Order: A number specifying how to rank number.
RANK	This function has been replaced with the RANK.AVG and RANK.EQ functions.This is available for compatibility with older versions.	RANK(number,ref,[order])
RSQ	Returns the square of the Pearson product moment correlation coefficient through data points in <i>knowny's and knownx's</i> .	RSQ(<i>knowny's,knownx's</i>) <i>Knowny's: An array or range of data points.Knownx's: An array or range of data points.</i>
SKEW	Returns the skewness of a distribution.	SKEW(number1, [number2], ...)
SKEW.P	Returns the skewness of a distribution based on a population.	SKEW.P(number 1, [number 2],â€¦)
SLOPE	Returns the slope of the linear regression line through data points in <i>knowny's and knownx's</i> .	SLOPE(<i>knowny's, knownx's</i>) <i>Knowny's: An array or cell range of numeric dependent data points.Knownx's: The set of independent data points.</i>

SMALL	Returns the k-th smallest value in a data set.	SMALL(array, k)Array: An array or range of numerical data for which you want to determine the k-th smallest value.K: The position (from the smallest) in the array or range of data to return.
STANDARDIZE	Returns a normalized value from a distribution characterized by mean and standard_dev.	STANDARDIZE(x, mean, standarddev)X: The value you want to normalize.Mean: The arithmetic mean of the distribution.Standarddev: The standard deviation of the distribution.
STDEV	This function has been replaced with the STDEV.S function.This function is available for compatibility with older versions.	STDEV(number1,[number2],...)
STDEV.P	Calculates standard deviation based on the entire population given as arguments.	STDEV.P(number1,[number2],...)
STDEV.S	Estimates standard deviation based on a sample (ignores logical values and text in the sample).	STDEV.S(number1,[number2],...)
STDEVA	Estimates standard deviation based on a sample.	STDEVA(value1, [value2], ...)
STDEVP	This function has been replaced with the STDEV.P function. It is available for compatibility with older versions.	STDEVP(number1,[number2],...)
STDEVPA	Calculates standard deviation based on the entire population given as arguments, including text and logical values.	STDEVPA(value1, [value2], ...)

STEYX	Returns the standard error of the predicted y-value for each x in the regression.	STEYX(knowny's, knownx's)Knowny's: An array or range of dependent data points.Knownx's: An array or range of independent data points.
T.INV	Returns the left-tail inverse of the Student's t-distribution.	T.INV(probability,degfreedom)Probability: The probability associated with the Student's t-distribution.Degfreedom: The number of degrees of freedom with which to characterize the distribution.
TRIMMEAN	Returns the mean of the interior of a data set.	TRIMMEAN(array, percent)Array: The array or range of values to trim and average.Percent: The fractional number of data points to exclude from the calculation.
VAR	This function has been replaced with the VAR.S function.This is available for compatibility with older versions.	VAR(number1,[number2],...)
VARA	Estimates variance based on a sample.	VARA(value1, [value2], ...)
VARP	This function has been replaced with the VAR.P function.This is available for compatibility with older versions.	VARP(number1,[number2],...)
VARPA	Calculates the variance based on the entire population.	VARPA(value1, [value2], ...)
WEIBULL	This function has been replaced with the WEIBULL.DIST function.This is available for compatibility with older versions.	WEIBULL(x,alpha,beta,cumulative)
WEIBULL.DIST	Returns the Weibull distribution.	WEIBULL.DIST(x,alpha,beta,cumulative)X: The value at which to evaluate the function.Alpha: A parameter for the distribution.Beta: A parameter for the distribution.Cumulative: Determines the form of the function.

Z.TEST	Returns the one-tailed P-value of a z-test.	Z.TEST(array,x,[sigma]) Array: The array or range of data against which to test x. x: The value to test. Sigma: The population (known) standard deviation.
ZTEST	This function has been replaced with the Z.TEST function. This is available for compatibility with older versions.	ZTEST(array,x,[sigma])

Logical functions

Name	Description	Syntax
AND	Returns TRUE if all of its arguments evaluate to TRUE. Returns FALSE if one or more arguments evaluate to FALSE.	AND(logical1, [logical2], ...) logical1: The first condition that you want to test that can evaluate to either TRUE or FALSE. logical2: Additional conditions that you want to test that can evaluate to either TRUE or FALSE, up to a maximum of 255 conditions. (Optional)
IF	The IF function returns one value if a specified condition evaluates to TRUE, and another value if that condition evaluates to FALSE.	IF(logicaltest,[valueiftrue],[valueiffalse]) Logicaltest: Any value or expression that can be evaluated to TRUE or FALSE. Valueiftrue: The value that you want to be returned if the logicaltest argument evaluates to TRUE. (Optional) Valueiffalse: The value that you want to be returned if the logicaltest argument evaluates to FALSE. (Optional)
IFERROR	Returns a value you specify if a formula evaluates to an error; otherwise, returns the result of the formula.	IFERROR(value, valueiferror) Value: The argument that is checked for an error. Valueiferror: The value to return if the formula evaluates to an error.
NOT	Reverses the value of its argument. Use NOT when you want to make sure a value is not equal to one particular value.	NOT(logical) Logical: A value or expression that can be evaluated to TRUE or FALSE.
OR	Returns TRUE if any argument is TRUE; returns FALSE if all arguments are FALSE.	OR(logical1, [logical2], ...) Logical1, logical2: 1 to 255 conditions you want to test that can be either TRUE or FALSE.
FALSE	Returns the logical value FALSE.	FALSE()
TRUE	Returns the logical value TRUE.	TRUE()

Engineering functions

Name	Description	Syntax
BESSELI	Returns the modified Bessel function.	BESSELI(X, N)X: The value at which to evaluate the function.N: The order of the Bessel function. If n is not an integer, it is truncated.
BESSELJ	Returns the Bessel function.	BESSELJ(X, N)X: The value at which to evaluate the function.N: The order of the Bessel function. If n is not an integer, it is truncated.
BESSELK	Returns the modified Bessel function, which is equivalent to the Bessel functions evaluated for purely imaginary arguments.	BESSELK(X, N)X: The value at which to evaluate the function.N: The order of the function. If n is not an integer, it is truncated.
BESSELY	Returns the Bessel function, which is also called the Weber function or the Neumann function.	BESSELY(X, N)X: The value at which to evaluate the function.N: The order of the function. If n is not an integer, it is truncated.
BIN2DEC	Converts a binary number to decimal.	BIN2DEC(number)Number: The binary number you want to convert.
BIN2HEX	Converts a binary number to hexadecimal.	BIN2HEX(number, [places])Number: The binary number you want to convert.Places: The number of characters to use.
BIN2OCT	Converts a binary number to octal.	BIN2OCT(number, [places])Number: The binary number you want to convert.Places: The number of characters to use.
BITAND	Returns a bitwise 'AND' of two numbers.	BITAND(number1, number2)Number1,Number2: Must be in decimal form and greater than or equal to 0.
BITLSHIFT	Returns a number shifted left by the specified number of bits.	BITLSHIFT(number, shiftamount)Number: Number must be an integer greater than or equal to 0.Shiftamount: Shift_amount must be an integer.
BITOR	Returns a bitwise 'OR' of two numbers.	BITOR(number1, number2)Number1, Number2: Must be in decimal form and greater than or equal to 0.
BITRSHIFT	Returns a number shifted right by the specified number of bits.	BITRSHIFT(number, shiftamount)Number: Number must be an integer greater than or

		<i>equal to 0.</i> Shiftamount: Shift_amount must be an integer.
BITXOR	Returns a bitwise 'XOR' of two numbers.	BITXOR(number1, number2)Number1,Number2: Must be in decimal form and greater than or equal to 0.
COMPLEX	Converts real and imaginary coefficients into a complex number of the form $x + yi$ or $x + yj$.	COMPLEX(realnum, inum, [suffix])Realnum: <i>The real coefficient of the complex number.</i> Inum: The imaginary coefficient of the complex number.Suffix: The suffix for the imaginary component of the complex number. If omitted, suffix is assumed to be "i".
CONVERT	Converts a number from one measurement system to another.	CONVERT(number,fromunit,tounit)Number: the value in fromunits to convert.Fromunit: the units for number.To_unit: the units for the result.
DEC2BIN	Converts a decimal number to binary.	DEC2BIN(number, [places])Number: The decimal integer you want to convert.Places: The number of characters to use.
DEC2HEX	Converts a decimal number to hexadecimal.	DEC2HEX(number, [places])Number: The decimal integer you want to convert.Places: The number of characters to use.
DEC2OCT	Converts a decimal number to octal.	DEC2OCT(number, [places])Number: The decimal integer you want to convert.Places: The number of characters to use.
ERF	Returns the error function integrated between lowerlimit and upperlimit.	ERF(lowerlimit,[upperlimit])Lowerlimit: <i>The lower bound for integrating ERF.</i> Upperlimit: The upper bound for integrating ERF. (Optional)
ERF.PRECISE	Returns the error function.	ERF.PRECISE(x)X: The lower bound for integrating ERF.PRECISE.
ERFC.PRECISE	Returns the complementary ERF function integrated between x and infinity.	ERFC.PRECISE(x)X: The lower bound for integrating ERFC.PRECISE.
GESTEP	Returns 1 if number \geq step; returns 0 (zero) otherwise.	GESTEP(number, [step])Number: The value to test against step.Step: The threshold value. If you omit a value for step, GESTEP uses zero. (Optional)
HEX2BIN	Converts a hexadecimal number to binary.	HEX2BIN(number, [places])Number: The hexadecimal number you want to

		convert.Places: The number of characters to use.
HEX2DEC	Converts a hexadecimal number to decimal.	HEX2DEC(number)Number: The hexadecimal number you want to convert.
HEX2OCT	Converts a hexadecimal number to octal.	HEX2OCT(number, [places])Number: The hexadecimal number you want to convert.Places: The number of characters to use.
IMABS	Returns the absolute value of a complex number in $x + yi$ or $x + yj$ text format.	IMABS(inumber)Inumber: A complex number for which you want the absolute value.
IMAGINARY	Returns the imaginary coefficient of a complex number in $x + yi$ or $x + yj$ text format.	IMAGINARY(inumber)Inumber: A complex number for which you want the imaginary coefficient.
IMARGUMENT	Returns the argument Theta.	IMARGUMENT(inumber)Inumber: A complex number for which you want the argument Theta.
IMCONJUGATE	Returns the complex conjugate of a complex number in $x + yi$ or $x + yj$ text format.	IMCONJUGATE(inumber)Inumber: A complex number for which you want the conjugate.
IMCOS	Returns the cosine of a complex number in $x + yi$ or $x + yj$ text format.	IMCOS(inumber)Inumber: A complex number for which you want the cosine.
IMCOSH	Returns the hyperbolic cosine of a complex number in $x+yi$ or $x+yj$ text format.	IMCOSH(inumber)Inumber: A complex number for which you want the hyperbolic cosine.
IMCOT	Returns the cotangent of a complex number in $x+yi$ or $x+yj$ text format.	IMCOT(inumber)Inumber: A complex number for which you want the cotangent.
IMCSC	Returns the cosecant of a complex number in $x+yi$ or $x+yj$ text format.	IMCSC(inumber)Inumber: A complex number for which you want the cosecant.
IMCSCH	Returns the hyperbolic cosecant of a complex number in $x+yi$ or $x+yj$ text format.	IMCSCH(inumber)Inumber: A complex number for which you want the hyperbolic cosecant.
IMDIV	Returns the quotient of two complex numbers in $x + yi$ or $x + yj$ text format.	IMDIV(inumber1, inumber2)Inumber1: The complex numerator or dividend.Inumber2: The complex denominator or divisor.
IMEXP	Returns the exponential of a complex number in $x + yi$ or $x + yj$ text format.	IMEXP(inumber)Inumber: A complex number for which you want the exponential.

IMLN	Returns the natural logarithm of a complex number in $x + yi$ or $x + yj$ text format.	IMLN(inumber)Number: A complex number for which you want the natural logarithm.
IMLOG10	Returns the common logarithm (base 10) of a complex number in $x + yi$ or $x + yj$ text format.	IMLOG10(inumber)Number: A complex number for which you want the common logarithm.
IMLOG2	Returns the base-2 logarithm of a complex number in $x + yi$ or $x + yj$ text format.	IMLOG2(inumber)Number: A complex number for which you want the base-2 logarithm.
IMPOWER	Returns a complex number in $x + yi$ or $x + yj$ text format raised to a power.	IMPOWER(inumber, number)Number: A complex number you want to raise to a power.Number: The power to which you want to raise the complex number.
IMPRODUCT	Returns the product of 1 to 255 complex numbers in $x + yi$ or $x + yj$ text format.	IMPRODUCT(inumber1, [inumber2], ...)
IMREAL	Returns the real coefficient of a complex number in $x + yi$ or $x + yj$ text format.	IMREAL(inumber)Number: A complex number for which you want the real coefficient.
IMSEC	Returns the secant of a complex number in $x+yi$ or $x+yj$ text format.	IMSEC(inumber)Number: A complex number for which you want the secant.
IMSECH	Returns the hyperbolic secant of a complex number in $x+yi$ or $x+yj$ text format.	IMSECH(inumber)Number: A complex number for which you want the hyperbolic secant.
IMSIN	Returns the sine of a complex number in $x + yi$ or $x + yj$ text format.	IMSIN(inumber)Number: A complex number for which you want the sine.
IMSINH	Returns the hyperbolic sine of a complex number in $x+yi$ or $x+yj$ text format.	IMSINH(inumber)Number: A complex number for which you want the hyperbolic sine.
IMSQRT	Returns the square root of a complex number in $x + yi$ or $x + yj$ text format.	IMSQRT(inumber)Number: A complex number for which you want the square root.
IMSUB	Returns the difference of two complex numbers in $x + yi$ or $x + yj$ text format.	IMSUB(inumber1, inumber2)Number1: The complex number from which to subtract inumber2.Number2: The complex number to subtract from inumber1.
IMSUM	Returns the sum of two or more complex numbers in $x + yi$ or $x + yj$ text format.	IMSUM(inumber1, [inumber2], ...)
IMTAN	Returns the tangent of a complex number in $x+yi$ or $x+yj$ text format.	IMTAN(inumber)Number: A complex number for which you want the tangent.

OCT2BIN	Converts an octal number to binary.	OCT2BIN(number, [places]) Number: The octal number you want to convert. Places: The number of characters to use.
OCT2DEC	Converts an octal number to decimal.	OCT2DEC(number) Number: The octal number you want to convert.
OCT2HEX	Converts an octal number to hexadecimal.	OCT2HEX(number, [places]) Number: The octal number you want to convert. Places: The number of characters to use.

Information functions

Name	Description	Syntax	
CELL	Returns information about the formatting, location, or contents of a cell.	CELL(<i>infotype</i> , [<i>reference</i>]) <i>Infotype</i> : A text value that specifies what type of cell information you want to return.	
		Info_type	Returns
		'col'	Column Number of ref.
		'color'	1 for negative values, otherwise 0.
		'contents'	Value of the upper left cell in ref.
		'filename'	Full path of the file that contains ref.
		'format'	Text value of the number format specified in the referenced cell.
		'parenthesis'	1 if the cell is formatted with parentheses for positive or all values; otherwise returns 0.
		'prefix'	Label prefix of the cell.
		'protect'	0 if the cell is not locked, otherwise 1.
		'row'	Row number of the cell in reference.

		'type'	B if the cell is empty.L if the cell contains a label.V if the cell contains a value.
		'width'	Column width of the ref cell.
ERROR.TYPE	Returns a number corresponding to one of the error values in Microsoft Excel, or returns the #N/A error if no error exists.	ERROR.TYPE(error_val)	
		error_val	Returns
		#NULL!	1
		#DIV/0!	2
		#VALUE!	3
		#REF!	4
		#NAME?	5
		#NUM!	6
		#N/A	7
		#GETTING_DATA	8
		Anything else	#N/A
INFO	Returns information about the current operating environment.	INFO(type_text)	
		Type_text	Returns
		'directory'	Path of the current directory or folder.
		'num file'	Number of active worksheets in the open workbooks.
		'origin'	Returns the absolute cell reference of the top and leftmost cell visible in the window.
		'OS version'	Current operating system version, as text.
		'recalc'	Current recalculation mode; returns 'Automatic' or 'Manual'.
		'release'	Version of Microsoft Excel, as text.

		'system'	Name of the operating environment: Macintosh = "mac" Windows = "pcdos"
ISBLANK	Returns TRUE if the cell is blank, otherwise returns FALSE.	ISBLANK(value)	
ISERR	Returns TRUE if value refers to any error value except #N/A.	ISERR(value)	
ISERROR	Returns TRUE if value refers to any error value (#N/A, #VALUE!, #REF!, #DIV/0!, #NUM!, #NAME?, or #NULL!).	ISERROR(value)	
ISEVEN	Returns TRUE if number is even, or FALSE if number is odd.	ISEVEN(number)	
ISFORMULA	Returns TRUE if referenced to a cell that contains a formula, otherwise FALSE.	ISFORMULA(reference)	
ISLOGICAL	Returns TRUE if value refers to a logical value.	ISLOGICAL(value)	
ISNA	Returns TRUE if value refers to the #N/A (value not available) error value.	ISNA(value)	
ISNONTEXT	Returns TRUE if value refers to any item that is not text.	ISNONTEXT(value)	
ISNUMBER	Returns TRUE if value refers to a number.	ISNUMBER(value)	
ISODD	Returns TRUE if number is odd, or FALSE if number is even.	ISODD(number)	
ISTEXT	Returns TRUE if value refers to text.	ISTEXT(value)	
N	Returns a value converted to a number.	N(value)	
NA	Returns the error value #N/A. #N/A is the error	NA()	

	value that means "no value is available."		
TYPE	Returns the type of value.	TYPE(value)	
		value	returns
		Number	1
		Text	2
		Logical value	4
		Error value	16
		Array (array: Used to build single formulas that produce multiple results or that operate on a group of arguments that are arranged in rows and columns. An array range shares a common formula; an array constant is a group of constants used as an argument.)	64

Date & Time functions

Name	Description	Syntax
DATE	Returns the sequential serial number that represents a particular date.	DATE(year,month,day) Year: The value of the year argument can include one to four digits. Month: A positive or negative integer representing the month of the year from 1 to 12 (January to December). Day: A positive or negative integer representing the day of the month from 1 to 31.
DATEVALUE	Converts a date that is stored as text into a serial number that Excel recognizes as a date.	DATEVALUE(datetext) Datetext: Text that represents a date in an Excel date format.
DAY	Returns the day of a date, represented by a serial number.	DAY(serialnumber) Serialnumber: The date of the day you are trying to find.
DAYS	Returns the number of days	DAYS(enddate, startdate) Startdate and Enddate are the two dates between which you want to know the number of days.

	between two dates.	
DAYS360	The DAYS360 function returns the number of days between two dates based on a 360-day year (twelve 30-day months).	DAYS360(<i>startdate</i> , <i>enddate</i> ,[<i>method</i>]) <i>Startdate, enddate</i> : The two dates between which you want to know the number of days. <i>Method</i> : A logical value that specifies whether to use the U.S. or European method in the calculation.
EDATE	Returns the serial number that represents the date that is the indicated number of months before or after a specified date.	EDATE(<i>startdate, months</i>) <i>Startdate</i> : A date that represents the start date. <i>Months</i> : The number of months before or after <i>start_date</i> .
EOMONTH	Returns the serial number for the last day of the month that is the indicated number of months before or after <i>start_date</i> .	EOMONTH(<i>startdate, months</i>) <i>Startdate</i> : A date that represents the starting date. <i>Months</i> : The number of months before or after <i>start_date</i> .
HOUR	Returns the hour of a time value.	HOUR(<i>serialnumber</i>) <i>Serialnumber</i> : The time that contains the hour you want to find.
ISOWEEKNUM	Returns number of the ISO week number of the year for a given date.	ISOWEEKNUM(<i>date</i>) <i>Date</i> : Date is the date-time code used by Excel for date and time calculation.
MINUTE	Returns the minutes of a time value. The minute is given as an integer, ranging from 0 to 59.	MINUTE(<i>serialnumber</i>) <i>Serialnumber</i> : The time that contains the minute you want to find.
MONTH	Returns the month of a date represented by a serial number.	MONTH(<i>serialnumber</i>) <i>Serialnumber</i> : The date of the month you are trying to find.

NETWORKDAYS.INTL	Returns the number of whole workdays between two dates using parameters to indicate which and how many days are weekend days.	NETWORKDAYS.INTL (startdate, enddate, [weekend], [holidays]) Startdate and enddate: The dates for which the difference is to be computed. Weekend is a weekend number or string that specifies when weekends occur.
NOW	Returns the serial number of the current date and time.	NOW()
SECOND	Returns the seconds of a time value.	SECOND(serialnumber) Serialnumber: The time that contains the seconds you want to find.
TIME	Returns the decimal number for a particular time.	TIME(hour, minute, second) Hour: A number from 0 (zero) to 32767 representing the hour. Minute: A number from 0 to 32767 representing the minute. Second: A number from 0 to 32767 representing the second.
TIMEVALUE	Returns the decimal number of the time represented by a text string.	TIMEVALUE(timetext) Timetext: A text string that represents a time in any of the time formats.
TODAY	Returns the serial number of the current date.	TODAY()
WEEKDAY	Returns the day of the week corresponding to a date.	WEEKDAY(serialnumber,[returntype]) Serialnumber: A sequential number that represents the date you are trying to find. Returntype: A number that determines the type of return value. (Optional)
WEEKNUM	Returns the week number of a specific date.	WEEKNUM(serialnumber,[returntype]) Serialnumber: A date within the week. Returntype: A number that determines on which day the week begins. The default is 1. (Optional)
WORKDAY	Returns a number that represents a date that is the indicated number of working days	WORKDAY(startdate, days, [holidays]) Startdate: A date that represents the start date. Days: The number of non-weekend and non-holiday days before or after start_date. Holidays: An optional list of one or more dates to exclude from the working calendar, such as state and federal holidays and floating holidays.

	before or after a date.	
WORKDAY.INTL	Returns the serial number of the date before or after a specified number of workdays with custom weekend parameters.	WORKDAY.INTL(<i>startdate</i> , <i>days</i> , [<i>weekend</i>], [<i>holidays</i>]) <i>Startdate</i> : The start date, truncated to integer. <i>Days</i> : The number of workdays before or after the start_date. <i>Weekend</i> : Indicates the days of the week that are weekend days and are not considered working days. (Optional)
YEAR	Returns the year corresponding to a date.	YEAR(<i>serialnumber</i>) <i>Serialnumber</i> : The date of the year you want to find.
YEARFRAC	Calculates the fraction of the year represented by the number of whole days between two dates.	YEARFRAC(<i>startdate</i> , <i>enddate</i> , [<i>basis</i>]) <i>Startdate</i> : A date that represents the start date. <i>Enddate</i> : A date that represents the end date. <i>Basis</i> : The type of day count basis to use. (Optional)

Financial functions

Name	Description	Syntax
CUMIPMT	Returns the cumulative interest paid on a loan between <i>startperiod</i> and <i>endperiod</i> .	CUMIPMT(<i>rate</i> , <i>nper</i> , <i>pv</i> , <i>startperiod</i> , <i>endperiod</i> , <i>type</i>) <i>Rate</i> : The interest rate. <i>Nper</i> : The total number of payment periods. <i>Pv</i> : The present value. <i>Startperiod</i> : The first period in the calculation. <i>Endperiod</i> : The last period in the calculation. <i>Type</i> : The timing of the payment.

		0 (zero)	Payment at the end of the period.
		1	Payment at the beginning of the period.
CUMPRINC	Returns the cumulative principal paid on a loan between <i>startperiod</i> and <i>endperiod</i> .	CUMPRINC(rate, nper, pv, <i>startperiod</i> , <i>endperiod</i> , type)Rate: The interest rate.Nper: The total number of payment periods.Pv: The present value. <i>Startperiod</i> : The first period in the calculation. <i>Endperiod</i> : The last period in the calculation.Type: The timing of the payment.	
		Type	Timing
		0 (zero)	Payment at the end of the period.
		1	Payment at the beginning of the period.
DB	Returns the depreciation of an asset for a specified period using the fixed-declining balance method.	DB(cost, salvage, life, period, [month])Cost: The initial cost of the asset.Salvage: The value at the end of the depreciation Life: The number of periods over which the asset is being depreciated.Period: The period for which you want to calculate the depreciation. Period must use the same units as life.Month: The number of months in the first year. If month is omitted, it is assumed to be 12. (Optional)	
DDB	Returns the depreciation of an asset for a specified period using the double-declining balance method or some other method you specify.	DDB(cost, salvage, life, period, [factor])Cost: The initial cost of the asset.Salvage: The value at the end of the depreciation This value can be 0.Life: The number of periods over which the asset is being depreciated.Period: The period for which you want to calculate the depreciation. Period must use the same units as life.Factor: The rate at which the balance declines. (Optional)	
DISC	Returns the discount rate for a security.	DISC(settlement, maturity, pr, redemption, [basis])Settlement: The security's settlement date. Maturity: The security's maturity date. The maturity date is the date when the security expires.Pr: The security's price per \$100 face value.Redemption: The security's redemption value per \$100 face value.Basis: The type of day count basis to use. (Optional)	
DOLLARDE	Converts a dollar price expressed as an integer part and a fraction part, such as 1.02, into a dollar price expressed as a decimal number.	DOLLARDE(fractionaldollar, fraction)Fractionaldollar: A number expressed as an integer part and a fraction part, separated by a decimal symbol.Fraction: The integer to use in the denominator of the fraction.	

DOLLARFR	Use DOLLARFR to convert decimal numbers to fractional dollar numbers, such as securities prices.	DOLLARFR(decimaldollar, fraction) <i>Decimaldollar</i> : A decimal number. <i>Fraction</i> : The integer to use in the denominator of a fraction.	
DURATION	Returns the Macaulay duration for an assumed par value of \$100.	DURATION(settlement, maturity, coupon, yield, frequency, [basis]) <i>Settlement</i> : The security settlement date is the date after the issue date when the security is traded to the buyer. <i>Maturity</i> : The maturity date is the date when the security expires. <i>Coupon</i> : The security's annual coupon rate. <i>Yield</i> : The security's annual yield. <i>Frequency</i> : The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. <i>Basis</i> : The type of day count basis to use. (Optional)	
		Basis	Day count basis
		0 or omitted	US (NASD) 30/360
		1	Actual/actual
		2	Actual/360
		3	Actual/365
		4	European 30/360
FV	Returns the future value of an investment based on periodic, constant payments and a constant interest rate.	FV(rate, nper, pmt, [pv], [type]) <i>Rate</i> : The interest rate per period. <i>Nper</i> : The total number of payment periods in an annuity. <i>Pmt</i> : The payment made each period. <i>Pv</i> : The present value, or the lump-sum amount that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0. (Optional) <i>Type</i> : The number 0 or 1, which indicates when payments are due. If type is omitted, it is assumed to be 0. (Optional)	
FVSCHEDULE	Returns the future value of an initial principal after applying a series of compound interest rates.	FVSCHEDULE(principal, schedule) <i>Principal</i> : The present value. <i>Schedule</i> : An array of interest rates to apply.	
INTRATE	Returns the interest rate for a fully invested security.	INTRATE(settlement, maturity, investment, redemption, [basis]) <i>Settlement</i> : The security settlement date is the date after the issue date when the security is traded to the buyer. <i>Maturity</i> : The maturity date is the date when the security expires. <i>Investment</i> : The amount invested in the	

		security.Redemption: The amount to be received at maturity.Basis: The type of day count basis to use.	
		Basis	Day count basis
		0 or omitted	US (NASD) 30/360
		1	Actual/actual
		2	Actual/360
		3	Actual/365
		4	European 30/360
IPMT	Returns the interest payment for a given period for an investment based on periodic, constant payments and a constant interest rate.	IPMT(rate, per, nper, pv, [fv], [type])Rate: The interest rate per period.Per: The period for which you want to find the interest and must be in the range 1 to nper.Nper: The total number of payment periods in an annuity.Pv: The present value, or the lump-sum amount that a series of future payments is worth right now.Fv: The future value or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).Type: The number 0 or 1.	
		Type	Payment Due
		0	At the end of the period.
		1	At the beginning of the period.
IRR	Returns the internal rate of return for a series of cash flows represented by the numbers in values.	IRR(values, [guess])Values: An array or a reference to cellsGuess: A number that you guess is close to the result of IRR. (Optional)	
ISPMT	Calculates the interest paid during a specific period of an investment. This function is provided for compatibility with Lotus 1-2-3.	ISPMT(rate, per, nper, pv)Rate: The interest rate for the investment.Per: The period for which you want to find the interest, and must be between 1 and nper.Nper: The total number of payment periods for the investment.Pv: The present value of the investment. For a loan, pv is the loan amount.	
MIRR	Returns the modified internal rate of return for	MIRR(values, financerate, reinvestrate)Values: An array or a reference to cells that contain numbers.Financerate: The interest	

	a series of periodic cash flows.	<i>rate you pay on the money used in the cash flows.Reinvestrate:</i> The interest rate you receive on the cash flows as you reinvest them.
NPER	Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.	NPV(rate,pmt,pv,[fv],[type])Rate : The interest rate per period.Pmt : The payment made each period; Pv : The present value, or the lump-sum amount that a series of future payments is worth right now.Fv : The future value, or a cash balance you want to attain after the last payment is made. (Optional)Type : The number 0 or 1 and indicates when payments are due. (Optional)
NPV	Calculates the net present value of an investment by using a discount rate and a series of future payments and income.	NPV(rate,value1,[value2],...)Rate: The rate of discount over the length of one period.
PMT	Calculates the payment for a loan based on constant payments and a constant interest rate.	PMT(rate, nper, pv, [fv], [type])Rate: The interest rate for the loan.Nper: The total number of payments for the loan.Pv: The present value, or the total amount that a series of future payments is worth now.Fv: The future value, or a cash balance you want to attain after the last payment is made. Type: The number 0 (zero) or 1, it indicates when payments are due. (Optional)
PPMT	Returns the payment on the principal for a given period for an investment based on periodic, constant payments and a constant interest rate.	PPMT(rate, per, nper, pv, [fv], [type])Rate: The interest rate per period.Per: Specifies the period and must be in the range 1 to nper.Nper: The total number of payment periods in an annuity.Pv: The present value "the total amount that a series of future payments is worth now.Fv: The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (zero), that is, the future value of a loan is 0.Type: The number 0 or 1, it indicates when payments are due. (Optional)
PV	Returns the present value of an investment. The present value is the total amount that a series of future payments is worth now.	PV(rate, nper, pmt, [fv], [type])Rate: The interest rate per period.Nper: The total number of payment periods in an annuity. Pmt: The payment made each period and cannot change over the life of the annuity. Fv: The future value, or a cash balance you want to attain after the last payment is made.Type: The number 0 or 1, it indicates when payments are due. (Optional)
RATE	Returns the interest rate per period of an annuity.	RATE(nper, pmt, pv, [fv], [type], [guess])Nper: The total number of payment periods in an annuity.Pmt: The payment made each period and cannot change over the life of the annuity. Pv: The present value "the total amount that a series of future payments is worth now.Fv: The future value, or a cash balance

		you want to attain after the last payment is made. Type: The number 0 or 1 and indicates when payments are due. (Optional)	
		Type	Payment Due
		0 or omitted	At the end of the period.
		1	At the beginning of the period.
		Guess: Your guess for what the rate will be. (Optional)	
RRI	Returns an equivalent interest rate for the growth of an investment.	RRI(nper, pv, fv)Nper: Nper is the number of periods for the investment.Pv: Pv is the present value of the investment.Fv: Fv is the future value of the investment.	
SLN	Returns the straight-line depreciation of an asset for one period.	SLN(cost, salvage, life)Cost: The initial cost of the asset.Salvage: The value at the end of the depreciation.Life: The number of periods over which the asset is depreciated.	
SYD	Returns the sum-of-years' digits depreciation of an asset for a specified period.	SYD(cost, salvage, life, per)Cost: The initial cost of the asset.Salvage: The value at the end of the depreciation.Life: The number of periods over which the asset has depreciated.Per: The period, it must use the same units as life.	
VDB	Returns the depreciation of an asset for any specified period, including partial periods.	VDB(cost, salvage, life, startperiod, endperiod, [factor], [noswitch])Cost: The initial cost of the asset.Salvage: The value at the end of the depreciation This value can be 0.Life: The number of periods over which the asset is depreciated.Startperiod: The starting period for which you want to calculate the depreciation.Endperiod: The ending period for which you want to calculate the depreciation. Factor: The rate at which the balance declines. (Optional) Noswitch: A logical value specifying whether to switch to straight-line depreciation when depreciation is greater than the declining balance calculation. (Optional)	
XIRR	Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic.	XIRR(values, dates, [guess])Values: A series of cash flows that corresponds to a schedule of payments in dates. Dates: A schedule of payment dates that corresponds to the cash flow paymentsGuess: A number that you guess is close to the result of XIRR. (Optional)	

Lookup & Reference functions

Name	Description	Syntax
AREA	Returns the number of areas in a reference. An	AREAS(reference)Reference: A reference to a cell or range of cells, it can refer to multiple areas.

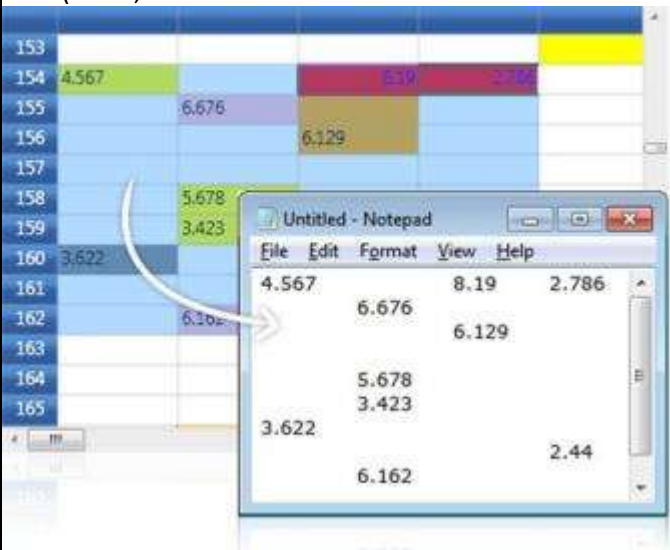
	area is a range of contiguous cells or a single cell.		
COLUMN	Returns the column number of the given cell reference.	COLUMN([reference])Reference: The cell or range of cells for which you want to return the column number. (Optional)	
COLUMNS	Returns the number of columns in an array or reference.	COLUMNS(array)Array: An array or array formula, or a reference to a range of cells for which you want the number of columns.	
FORMULATEXT	Returns a formula as a string.	FORMULATEXT(reference)Reference: A reference to a cell or range of cells.	
HLOOKUP	Searches for a value in the top row of a table or an array of values, and then returns a value in the same column from a row you specify in the table or array.	HLOOKUP(lookupvalue,tablearray, rowindexnum, [rangelookup])Lookupvalue: The value to be found in the first row of the table. Tablearray: A table of information in which data is looked up. Rowindexnum: The row number in tablearray from which the matching value will be returned. Range_lookup: A logical value that specifies whether you want HLOOKUP to find an exact match or an approximate match.	
HYPERLINK	Creates a shortcut or jump that opens a document stored on a network server, an intranet, or the Internet.	HYPERLINK(linklocation, [friendlyname])Linklocation: The path and file name to the document to be opened.Friendlyname: The jump text or numeric value that is displayed in the cell.	
INDIRECT	Returns the reference specified by a text string. References are immediately evaluated to display their contents.	INDIRECT(reftext, [a1])Reftext: A reference to a cell that contains an A1-style reference, an R1C1-style reference, a name defined as a reference, or a reference to a cell as a text string.A1: A logical value that specifies what type of reference is contained in the cell ref_text.	
MATCH	The MATCH function searches for a specified item in a range of cells, and then returns the relative position of that item in the range.	MATCH(lookupvalue, lookuparray, [matchtype])lookupvalue: The value that you want to match in lookuparray. The lookupvalue argument can be a value or a cell reference to a number, text, or logical value.lookuparray: The range of cells being searched.matchtype: The number -1, 0, or 1.	
		Match_type	Behavior
		1 or omitted	MATCH finds the largest value that is less than or equal to lookupvalue_

		0	MATCH finds the first value that is exactly equal to <i>lookupvalue_</i>
		-1	MATCH finds the smallest value that is greater than or equal to <i>lookupvalue_</i>
OFFSET	Returns a reference to a range that is a specified number of rows and columns from a cell or range of cells.	OFFSET(reference, rows, cols, [height], [width])Reference: The reference from which you want to base the offset. Rows: The number of rows, up or down, that you want the upper-left cell to refer to. Cols: The number of columns, to the left or right, that you want the upper-left cell of the result to refer to. Height: The height, in number of rows, that you want the returned reference to be. Height must be a positive number. (Optional)Width: The width, in number of columns, that you want the returned reference to be. Width must be a positive number. (Optional)	
ROW	Returns the row number of a reference.	ROW([reference])Reference: The cell or range of cells for which you want the row number.	
ROWS	Returns the number of rows in a reference or array.	ROWS(array)Array: An array, an array formula, or a reference to a range of cells for which you want the number of rows.	
SHEET	Returns the sheet number of the reference sheet.	SHEET(value)Value: Value is the name of a sheet or a reference for which you want the sheet number.	
SHEETS	Returns the number of sheets in a reference.	SHEETS(reference)Reference: Reference is a reference for which you want to know the number of sheets it contains. (Optional)	
TRANSPOSE	The TRANSPOSE function returns a vertical range of cells as a horizontal range, or vice versa.	TRANSPOSE(array)array: An array or range of cells on a worksheet that you want to transpose.	
VLOOKUP	You can use the VLOOKUP function to search the first column of a range of cells, and then return a value from any cell on the same row of the range.	VLOOKUP(lookupvalue, tablearray, colindexnum, [range/lookup])lookupvalue: The value to search in the first column of the table or range. tablearray: <i>The range of cells that contains the data.</i> colindexnum: <i>The column number in the tablearray argument from which the matching value must be returned.</i> range_lookup: A logical value that specifies whether you want VLOOKUP to find an exact match or an approximate match. (Optional)	

Text functions

Name	Description	Syntax
CODE	Returns a numeric code for the first character in a text string.	CODE(text)Text: The text for which you want the code of the first character.
CONCATENATE	The CONCATENATE function joins up to 255 text strings into one text string.	CONCATENATE(text1, [text2], ...)Text1: The first text item to be concatenated.Text2: Additional text items, up to a maximum of 255 items.
DOLLAR	The function described in this Help topic converts a number to text format and applies a currency symbol.	DOLLAR(number, [decimals])Number: A number, a reference to a cell containing a number, or a formula that evaluates to a number.Decimals: The number of digits to the right of the decimal point. (Optional)
FINDB	FINDB counts each double-byte character as 2 when editing of a language that supports DBCS is enabled and has been set as the default language.	FINDB(findtext, withintext, [startnum])Findtext: The text you want to find.Withintext: <i>The text containing the text you want to find.</i> Startnum: Specifies the character at which to start the search. (Optional)
FIXED	Rounds a number to the specified number of decimals, formats the number in decimal format using a period and commas, and returns the result as text.	FIXED(number, [decimals], [nocommas])Number: <i>The number you want to round and convert to text.</i> Decimals: <i>The number of digits to the right of the decimal point.</i> Nocommas: A logical value that, if TRUE, prevents FIXED from including commas in the returned text. (Optional)
JIS	Converts half-width (single-byte) letters within a character string to full-width (double-byte) characters.	JIS(Text)Text: The text or a reference to a cell that contains the text you want to change.
LEFT	LEFT returns the first character or characters in a text string, based on the number of characters you specify.	LEFT(text, [num_chars])
LEFTB	LEFTB returns the first character or characters in a text string, based on the number of bytes you specify.	LEFTB(text, [numbytes])Text: <i>The text string that contains the characters you want to extract.</i> Numchars: Specifies the number of characters you want LEFT to extract. (Optional)Num_bytes: Specifies the number of characters you want LEFTB to extract, based on bytes. (Optional)
LEN	LEN returns the number of characters in a text string.	LEN(text)Text: The text whose length you want to find. Spaces count as characters.
LENB	LENB returns the number of bytes used to represent the characters in a text string.	LENB(text)Text: The text whose length you want to find. Spaces count as characters.

LOWER	Converts all uppercase letters in a text string to lowercase.	LOWER(text)Text: The text you want to convert to lowercase.
MID	MID returns a specific number of characters from a text string, starting at the position you specify, based on the number of characters you specify.	MID(text, startnum, numchars)Text: The text string containing the characters you want to extract.Startnum: <i>The position of the first character you want to extract in text. The first character in text has startnum 1, and so on.</i> Numchars: <i>Specifies the number of characters you want MID to return from text.</i> Numbytes: <i>Specifies the number of characters you want MIDB to return from text, in bytes.</i>
MIDB	MIDB returns a specific number of characters from a text string, starting at the position you specify, based on the number of bytes you specify.	MIDB(text, startnum, numchars)Text: The text string containing the characters you want to extract.Startnum: <i>The position of the first character you want to extract in text. The first character in text has startnum 1, and so on.</i> Numchars: <i>Specifies the number of characters you want MID to return from text.</i> Numbytes: <i>Specifies the number of characters you want MIDB to return from text, in bytes.</i>
NUMBERVALUE	Converts text to a number, in a locale-independent way.	NUMBERVALUE(Text,[Decimalseparator],[Groupseparator])Text: The text to convert to a number.Decimalseparator: <i>The character used to separate the integer and fractional part of the result.</i> Groupseparator : <i>The character used to separate groupings of numbers, such as thousands from hundreds and millions from thousands.</i>
PROPER	Capitalizes the first letter in a text string and any other letters in that string that follow non-letter characters.	PROPER(text)Text: Text enclosed in quotation marks, a formula that returns text, or a reference to a cell containing the text you want to partially capitalize.
REPLACE	Replaces part of a text string, based on the number of characters you specify, with a different text string.	REPLACE(oldtext, startnum, numchars, newtext)Oldtext: <i>Text in which you want to replace some characters.</i> Startnum: <i>The position of the character in oldtext that you want to replace with newtext.</i> Numchars: <i>The number of characters in oldtext for REPLACE to replace with newtext.</i> Numbytes: <i>The number of bytes in oldtext for REPLACEB to replace with newtext.</i> Newtext: <i>The text that will replace characters in oldtext.</i>
REPT	Repeats text a given number of times.	REPT(text, numbertimes)Text: <i>The text you want to repeat.</i> Numbertimes: <i>A positive number specifying the number of times to repeat text.</i>
RIGHT	RIGHT returns the last character or characters in a	RIGHT(text,[numchars])Text: <i>The text string containing the characters you want to extract.</i> Numchars: <i>Specifies</i>

	text string, based on the specified number of characters.	the number of characters you want RIGHT to extract. (Optional)
RIGHTB	RIGHTB returns the last character or characters in a text string, based on the number of bytes specified.	RIGHTB(text,[numbytes]) <i>Text</i> : The text string containing the characters you want to extract. <i>Numbytes</i> : Specifies the number of characters you want RIGHTB to extract, based on bytes. (Optional)
SEARCHB	Locates one text string within a second text string, and returns the number of the starting position of the first text string from the first character of the second text string.	SEARCHB(findtext,withintext,[startnum]) <i>findtext</i> : The text that you want to find. <i>withintext</i> : The text in which you want to search for the value of the findtext argument. <i>startnum</i> : The character number in the withintext argument at which you want to start searching.
SUBSTITUTE	Substitutes newtext for oldtext in a text string.	SUBSTITUTE(text, oldtext, newtext, [instancenum]) <i>Text</i> : The text or the reference to a cell containing text for which you want to substitute characters. <i>Oldtext</i> : The text you want to replace. <i>Newtext</i> : The text you want to replace oldtext with. <i>Instancenum</i> : Specifies which occurrence of oldtext you want to replace with new_text. (Optional)
T	Returns the text referred to by value.	T(value)Value: The value you want to test.
TEXT	The TEXT function converts a numeric value to text and lets you specify the display formatting by using special format strings.	<p>TEXT(value, format</p>  <p>text)value: A numeric value, a formula that evaluates to a numeric value, or a reference to a cell containing a numeric value.<i>format</i>: A numeric format as a text string enclosed in quotation marks.</p>

TRIM	Removes all spaces from text except for single spaces between words.	TRIM(text)Text: The text from which you want spaces removed.
UNICHAR	Returns the Unicode character that is referenced by the given numeric value.	UNICHAR(number)Number is the Unicode number that represents the character.
UNICODE	Returns the number (code point) corresponding to the first character of the text.	UNICODE(text)Text: Text is the character for which you want the Unicode value.
UPPER	Converts text to uppercase.	UPPER(text)Text: The text you want converted to uppercase.
VALUE	Converts a text string that represents a number to a number.	VALUE(text)Text: The text enclosed in quotation marks or a reference to a cell containing the text you want to convert.

Database functions

Name	Description	Syntax
DAVERAGE	Averages the values in a field of records in a list or database that match conditions you specify.	DAVERAGE(database, field, criteria)Database: The range of cells that makes up the list or database. Field: Indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: The range of cells that contains the conditions you specify.
DCOUNT	Counts the cells that contain numbers in a field of records in a list or database that match conditions that you specify.	DCOUNT(database, field, criteria)Database: The range of cells that makes up the list or database. Field: Indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: The range of cells that contains the conditions you specify.
DCOUNTA	Counts the filled cells in a field of records in a list or database that match conditions that you specify.	DCOUNTA(database, field, criteria)Database: The range of cells that makes up the list or database. Field: indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: The range of cells that contains the conditions you specify.

DGET	Extracts a single value from a column of a list or database that matches conditions that you specify.	DGET(database, field, criteria)Database: The range of cells that makes up the list or database. Field: indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: The range of cells that contains the conditions you specify.
DMAX	Returns the largest number in a field of records in a list or database that matches conditions you that specify.	DMAX(database, field, criteria)Database: The range of cells that makes up the list or database. Field: Indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: is the range of cells that contains the conditions you specify.
DMIN	Returns the smallest number in a field of records in a list or database that matches conditions that you specify.	DMIN(database, field, criteria)Database: The range of cells that makes up the list or database. Field: Indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: The range of cells that contains the conditions you specify.
DSTDEV	The standard deviation of a population based on a sample by using the numbers in a field (column) of records in a list.	DSTDEV(database, field, criteria)Database: is the range of cells that makes up the list or database. Field: Indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: is the range of cells that contains the conditions you specify.
DSTDEVP	The standard deviation of a population based on the entire population using the numbers in a field of records in a list.	DSTDEVP(database, field, criteria)Database: The range of cells that makes up the list or database. Field: Indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: Is the range of cells that contains the conditions you specify.
DSUM	The numbers in a field of records in a list or database that match conditions that you specify.	DSUM(database, field, criteria)Database: The range of cells that makes up the list or database. Field: Indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: The range of cells that contains the conditions you specify.

DVAR	The variance of a population based on a sample using the numbers in a field of records in a list.	DVAR(database, field, criteria)Database: The range of cells that makes up the list or database. Field: indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: is the range of cells that contains the conditions you specify.
DVARP	The variance of a population based on the entire population by using the numbers in a field of records in a list.	DVARP(database, field, criteria)Database: The range of cells that makes up the list or database. Field: Indicates which column is used in the function. Enter the column label enclosed between double quotation marks.Criteria: The range of cells that contains the conditions you specify.

Web functions

Name	Description	Syntax
ENCODEURL	Returns a URL-encoded string.	ENCODEURL(text)Text: A string to be URL encoded.
FILTERXML	Returns specific data from the XML content by using the specified XPath.	FILTERXML(xml, xpath)Xml: A string in valid XML format.XPath: A string in standard XPath format.
WEBSERVICE	Returns data from a web service on the Internet or Intranet.	WEBSERVICE(url)Url: The URL of the web service.

Editing in WPF GridControl

This section explains the clipboard and undo redo support of Grid control.

- Clipboard Support - Elaborates on different clipboard operations
- Undo or Redo support - Grid supports undo or redo operation to those achieved with Microsoft Office-type applications.

Clipboard Support

Essential Grid provides complete support for clipboard operations. End users can copy/cut & paste any data inside the grid and to or from other OLE [Object Linking and Embedding]-enabled applications such as Notepad. The built-in source allows us to copy the text data along with the style information and also provides hooks that let us customize the clipboard operation of pasting the custom formatted data.

Copy Paste Options

CopyPasteOption property defines the list of clipboard operations supported by the grid. It exposes the following options:

- CopyText – Copies only the text from the grid selection to clipboard.

- CopyCellData – Copies both text and style information from grid cells to clipboard.
- PasteText – Pastes only the text from clipboard.
- PasteCell – Pastes the cell text along with its style information from the clipboard.
- CutText – Moves only the text from grid to clipboard.
- CutCell – Moves the text and the style information from grid to clipboard.
- ExcludeCurrentCell – Skips current cell while doing clipboard operations.
- XmlCopyPaste – Copy the cell value along with basic styles in XML format and supported to paste in Microsoft Excel. This also supports to copy the Formula value from the Grid Control and Paste in Microsoft Excel.

Example

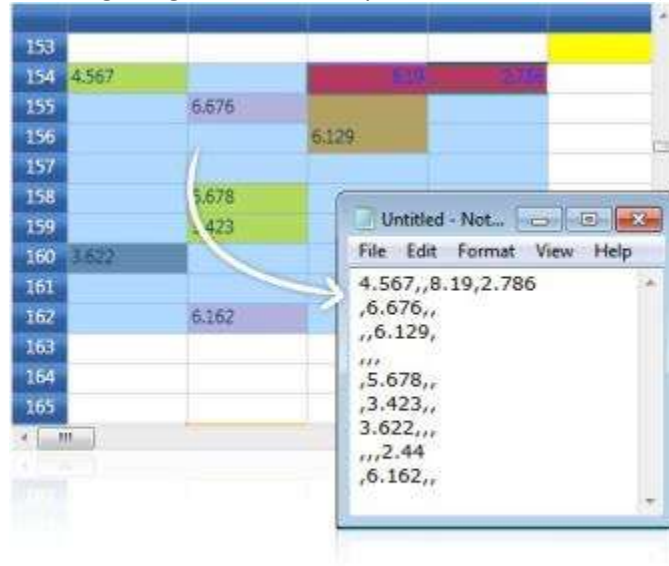
Here are the sample code snippets that define certain copy paste behaviors.

C#

```
//Copy cell data with style
gridControl.Model.Options.CopyPasteOption |= CopyPaste.CopyCellData;
//Cut cell data with style
gridControl.Model.Options.CopyPasteOption |= CopyPaste.CutCell;
//Paste cell data with style
gridControl.Model.Options.CopyPasteOption |= CopyPaste.PasteCell;
//Code to cut copy paste cell text (excluding style)
gridControl.Model.Options.CopyPasteOption = (CopyPaste) (0);
gridControl.Model.Options.CopyPasteOption |= CopyPaste.CopyText;
gridControl.Model.Options.CopyPasteOption |= CopyPaste.CutText;
gridControl.Model.Options.CopyPasteOption |= CopyPaste.PasteText;
```

{% endtabs %}

![[Pasting the grid data in notepad in WPF GridControl]](Editing



images/pastein_notepad.jpeg)

Text Data Exchange

GridModel.TextDataExchange helps you in customizing the clipboard operations. It is used as an interface that exposes the following property and methods.

- Property - TabDelimiter
- Method - CopyTextToBuffer(), PasteTextFromBuffer()

The above attributes are discussed below in detail:

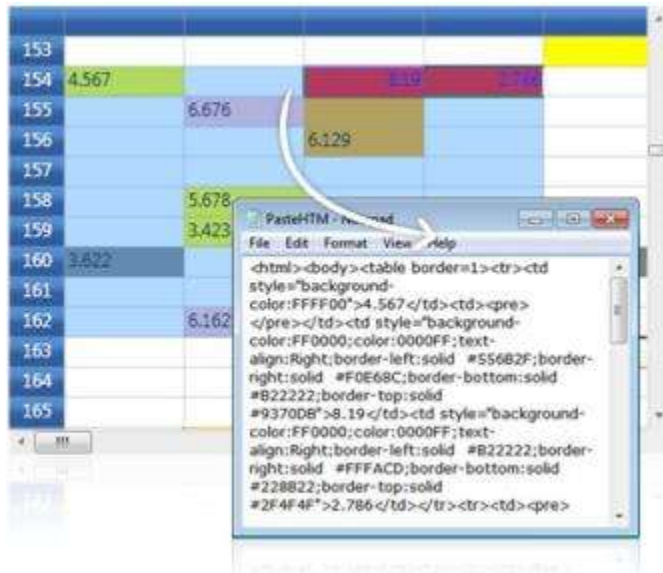
TabDelimiter Property

This property specifies a delimiter for the text to be pasted. It can be used when you want to paste the cell data in CSV (Comma-Separated Values) format.

C#

```
gridControl.Model.TextDataExchange.TabDelimiter = ",";
```

![Pasting the grid data in CSV format in WPF GridControl](Editing



images/pastein_csv.jpeg)

CopyTextToBuffer() Method

This method lets you place the cell data into an intermediate buffer, which can be customized. The method performs clipboard cut or copy operation depending on the third parameter given to it. This method accepts the following parameters:

- String buffer
- Selected range of cells
- Boolean value-This should be set to true for cut operation and should be set to false for copy operation.

The following code illustrates the CopyTextToBuffer method:

C#

```
gridControl.Model.TextDataExchange.CopyTextToBuffer(out buffer,
gridControl.Model.SelectedRanges, out row, out col, false);
```

Result of this Method Call

It returns the following values:

- Cell text
- Number of rows affected
- Number of columns affected

PasteTextFromBuffer() Method

The values returned by the CopyTextToBuffer method is passed as parameter to the PasteTextFromBuffer method using the below code:

C#

```
gridControl.Model.TextDataExchange.PasteTextFromBuffer(buffer,
gridControl.Model.SelectedRanges);
```

Result of this Method Call

It pastes the text from the given buffer into specified range of grid cells.

Events

Grid provides the following events which are available for the end user to customize the clipboard data.

- ClipboardCanCopy
- ClipboardCanCut
- ClipboardCanPaste
- ClipboardCopy
- ClipboardCut
- ClipboardPaste

IGridCopyPaste

Essential Grid defines an interface called IGridCopyPaste that exposes some methods, namely Copy(), Cut() and Paste(). Here the users can write custom code to perform cut copy or paste operations with any kind of user-defined data. Thereby, it extends its clipboard support behavior to perform clipboard operations in various forms.

For instance, let us consider performing the copy and paste operations in HTML format. The respective implementation of IGridCopyPaste is as follows:

C#

```
class HtmlCopy : IGridCopyPaste
{
    public void Copy(GridCellData gridData, GridRangeInfoList rangeList)
    {
        IDataObject iData = null;
        iData = Clipboard.GetDataObject();
        string buffer = iData.GetData(DataFormats.UnicodeText) as string;
        int top = rangeList[0].Top;
        int left = rangeList[0].Left;
```

```

int right = rangeList[0].Right;
int bottom = rangeList[0].Bottom;
string stylesheet = string.Empty;
StringBuilder sb = new StringBuilder();
GridStyleInfoStore gsis;
GridStyleInfo style;
sb.Append("<html><body><table border=1>");
for (int row = top; row <= bottom; row++)
{
    sb.Append("<tr>");
    for (int col = left; col <= right; col++)
    {
        gsis = gridData[row - top, col - left];
        style = new GridStyleInfo(gsis);
        stylesheet = "";
        if (style.HasBackground)
        {
            string backgroundColor = style.Background.ToString();
            backgroundColor = backgroundColor.Substring(3, backgroundColor.Length - 3);
            stylesheet = "\"background-color:" + backgroundColor;
        }
        if (style.HasForeground)
        {
            string foregroundColor = style.Foreground.ToString();
            foregroundColor = foregroundColor.Substring(3, foregroundColor.Length - 3);
            stylesheet = stylesheet + ";color:" + foregroundColor;
        }
        if (style.HasHorizontalAlignment)
        {
            stylesheet = stylesheet + ";text-align:" + style.HorizontalAlignment;
        }
        if (style.HasVerticalAlignment)
        {
            stylesheet = stylesheet + ";vertical-align:" + style.VerticalAlignment;
        }
        if (style.HasBorders)
        {
            string borderBrush;
            borderBrush = style.Borders.Left.Brush.ToString();
            borderBrush = borderBrush.Substring(3, borderBrush.Length - 3);
            stylesheet = stylesheet + ";border-left:solid #" + borderBrush;
            borderBrush = style.Borders.Right.Brush.ToString();
            borderBrush = borderBrush.Substring(3, borderBrush.Length - 3);
            stylesheet = stylesheet + ";border-right:solid #" + borderBrush;
            borderBrush = style.Borders.Bottom.Brush.ToString();
            borderBrush = borderBrush.Substring(3, borderBrush.Length - 3);
            stylesheet = stylesheet + ";border-bottom:solid #" + borderBrush;
            borderBrush = style.Borders.Top.Brush.ToString();
            borderBrush = borderBrush.Substring(3, borderBrush.Length - 3);
            stylesheet = stylesheet + ";border-top:solid #" + borderBrush;
        }
        stylesheet = stylesheet + "\"";
        if (!stylesheet.Equals("\"\""))
        {
            sb.Append("@<td style=" + stylesheet + ">");
        }
        else

```



```

{
    sb.Append(@"<td>");
}
if (!style.CellValue.ToString().Equals(""))
{
    sb.Append(style.CellValue.ToString());
}
else
{
    sb.Append("<pre> </pre>");
}
sb.Append("</td>");
stylesheet = string.Empty;
}
sb.Append("</tr>");
}
sb.Append("</table></body></html>");
DataObject dataObject = new DataObject();
dataObject.SetData(DataFormats.UnicodeText, sb.ToString());
Clipboard.SetDataObject(dataObject);
}
public void Cut(GridCellData gridCellData, GridRangeInfoList rangeList)
{
}
public DataObject Paste(GridRangeInfoList rangeList)
{
    return new DataObject();
}
}

```

The next step is to attach the above custom copy and paste operations to the grid control.

C#

```

HtmlCopy htmlCopy = new HtmlCopy();
gridControl.Model.GridCopyPaste = htmlCopy;

```

![[Pasting the grid data in HTML format in WPF GridControl]](Editingimages/pastein_html.jpeg)

- [ClipboardCanCopy event](#)
- [ClipboardCanCut event](#)
- [ClipboardCanPaste event](#)
- [ClipboardCopy event](#)
- [ClipboardCut event](#)
- [ClipboardPaste event](#)

Undo/Redo

Essential Grid supports undo/redo functionalities similar to those achieved with Microsoft Office-type applications. To handle this functionality, a stack is maintained internally in Essential Grid to save the changes that occur through which the following tasks can be accomplished by the users:

- Control of the stack: when to save or discard changes, and when to rollback changes
- Create new transactions and control each individual transaction (like canceling, rollback) without affecting others

The undo/redo architecture is extensible, thereby allowing users to derive the base class and add some more functionality to the grid.

The Basics

Essential Grid has a `GridModelCommandManager` class that implements support for the undo/redo commands in the Grid control. Depending upon the grid settings, as a user makes changes to the grid these changes will be tracked in stack structures which will be found in the `GridModelCommandManager` class. This class has methods that allow you to undo the last action, redo the last undone action, and batch transactions so that a series of actions can be undone or redone in a single step.

The `CommandStack` property of the `GridControl` class will return a reference to the `GridCommandStack` object that is associated with a grid. It is through this property that you can access the undo/redo support in Essential Grid. For example, you can use the `Enabled` property of the `CommandStack` to control whether or not the grid supports undo/redo at any given moment. The following code samples show you some `CommandStack` properties.

C#

```
// Turn off the Undo buffer.
this.grid.Model.CommandStack.Enabled = false;
// Turn on the Undo buffer.
this.grid.Model.CommandStack.Enabled = true;
// Execute the latest command from the undo stack.
this.grid.Model.CommandStack.Undo();
// Execute the latest command from the redo stack.
this.grid.Model.CommandStack.Redo();
// Clear the Undo buffer.
this.grid.Model.CommandStack.UndoStack.Clear();
// Clear the Redo buffer.
this.grid.Model.CommandStack.RedoStack.Clear();
// Clear both the Undo and Redo buffers.
this.grid.Model.CommandStack.Clear();
```

VB.NET

```
'Turn off the Undo buffer.
Me.grid.Model.CommandStack.Enabled = False
'Turn on the Undo buffer.
Me.grid.Model.CommandStack.Enabled = True
'Execute the latest command from the undo stack.
this.grid.Model.CommandStack.Undo()
'Execute the latest command from the redo stack.
this.grid.Model.CommandStack.Redo()
'Clear the Undo buffer.
Me.grid.Model.CommandStack.UndoStack.Clear()
'Clear the Redo buffer.
Me.grid.Model.CommandStack.RedoStack.Clear()
'Clear both the Undo and Redo buffers.
Me.grid.Model.CommandStack.Clear()
```

Transactions

A transaction is a series of steps that should be treated as a single action in the undo/redo architecture. For example, you may have a record-oriented grid where you may want to group any changes in the current row as one transaction. This way, when the user wants to undo the last change, all the changes in the row are undone. It is possible to group a series of actions into a single undo/redo step through the use of these three GridCommandStack methods: BeginTrans, CommitTrans and RollBack.

A call to BeginTrans will mark the start of a series of actions that are to be treated as a single undo/redo step. Once BeginTrans has begun, all the changes are marked as being a member of a single transaction until either CommitTrans or RollBack is called. CommitTrans signals a successful end to the transaction. A call to RollBack will cause all the changes in the current transaction to be undone and will end the transaction processing. A RollBack call will return the grid in the same state that it was in, immediately prior to the call to BeginTrans.

C#

```
// Begin the transaction.  
this.grid.Model.CommandStack.BeginTrans ("Transaction beginning");  
// Commit the transaction.  
this.grid.Model.CommandStack.CommitTrans();  
// Rollback the current transaction.  
this.grid.Model.CommandStack.Rollback();
```

VB.NET

```
'Begin the transaction.  
Me.grid.Model.CommandStack.BeginTrans ("Transaction beginning")  
'Commit the transaction.  
Me.grid.Model.CommandStack.CommitTrans()  
'Rollback the current transaction.  
Me.grid.Model.CommandStack.UndoStack.Rollback()
```

It is also possible to nest transactions. If you are in the middle of a transaction, it is okay to call BeginTrans again. But, when such nested transactions are undone, they are treated as part of a single parent transaction.

981234	2345	296	5362345	235	806
993	544	443	509	607	856
582	883	174	232	557	909
605	318	415	707	931	908
341	468	283	703	618	657
520	82	642	786	523	543
50	572	632	146	791	502
403	449	887	716	517	66
39	764	708	958	967	113
128	1	633	819	699	465
356	833	445	97	79	255
30	376	281	379	406	289
488	124	461	86	778	634
89	429	522	626	312	456
595	675	473	587	642	404
377	601	711	731	147	879

Undo/Redo Options

Undo/Redo Stack

6 Undo items

- Change Selection State Row 1
- Transaction beginning-
- > Change Cells R1C5
- > Change Cells R1C4
- > Change Selection State R1
- Change Selection State Row 1
- Change Cells R1C2

3 Redo items

- Change Selection State Row 6 Cc
- Change Cells R6C3
- Change Selection State Row 9 Cc

Derived Commands

The undo or redo architecture of Essential Grid is complete as shipped with the product. If, for some reason, you need to handle special grid requirements that cannot be performed with the standard implementation, the undo/redo architecture is extensible. To extend it, you need to derive custom command classes from either the abstract class `SyncfusionCommand` or the abstract class `GridModelCommand`. In your derived class, you will need to add whatever members you need in order to track enough state information that will allow you to undo or redo the action that is being done. Then you have to implement an execute method and other abstract members of the base class. If you do a search in the Essential Grid source code for `GridModelCommand`, you will see many examples of the derived command classes.

Once you have your derived `SyncfusionCommand` class, whenever the action is taken, you will have to create a proper instance of your derived `SyncfusionCommand` class and add it to the `GridControl.Model.CommandStack.UndoStack`. Thus, when Essential Grid needs to undo this action, your command will be popped from the `UndoStack`, and its execute method will be called indicating that this action needs to be undone (also at this point, Essential Grid will add this same instance to the `RedoStack` so that the action can later be redone if necessary).

The following code snippet demonstrates how to implement support for current cell activated action in undo/redo operations.

C#

```
public class GridCellActivatedCommand : GridModelCommand
{
    private RowColumnIndex cell;
    public GridCellActivatedCommand(GridModel model, RowColumnIndex cell)
        : base(model)
    {
        this.cell = cell;
    }
    public override void Execute()
    {

```

```

this.Grid.ActiveGridView.CurrentCell.MoveTo(cell);
}
}
void grid_CurrentCellActivated(object sender, SyncfusionRoutedEventArgs
args)
{
if (this.grid.Model.CommandStack.ShouldGenerateUndoInfo)
{
this.grid.Model.CommandStack.Push(new
GridCellActivatedCommand(this.grid.Model,
this.grid.CurrentCell.CellRowColumnIndex));
}
}
}

```

VB.NET

```

Public Class GridCellActivatedCommand
Inherits GridModelCommand
Private cell As RowColumnIndex
Public Sub New(ByVal model As GridModel, ByVal cell As RowColumnIndex)
MyBase.New(model)
Me.cell = cell
End Sub
Public Overrides Sub Execute()
Me.Grid.ActiveGridView.CurrentCell.MoveTo(cell)
End Sub
End Class
Private Sub grid_CurrentCellActivated(ByVal sender As Object, ByVal args As
SyncfusionRoutedEventArgs)
If Me.grid.Model.CommandStack.ShouldGenerateUndoInfo Then
Me.grid.Model.CommandStack.Push(New GridCellActivatedCommand(Me.grid.Model,
Me.grid.CurrentCell.CellRowColumnIndex))
End If
End Sub

```

Tables for Properties, Methods, and Events

Properties

Property	Description	Type	Data Type
Enabled	Enable undo/redo support in the Grid Control.	CLR Property	bool
UndoStack	Stack for undo commands.	CLR Property	Stack
RedoStack	Stack for redo commands.	CLR Property	Stack
InTransaction	Indicates whether BeginTrans has been called.	CLR Property	bool
IsRecording	Indicates whether the grid is in the default mode that records undo information.	CLR Property	bool

Mode	Indicates the Grid control's regular operation.	CLR Property	GridCommandMode
ShouldGenerateUndoInfo	Temporarily suspended undo.	CLR Property	bool

Methods

Method	Description	Parameters	Return Type
BeginTrans	Combines several subsequent commands into one transaction.	BeginTrans(string Name)	void
Clear	Clear both the undo and redo stacks.	Clear()	void
CommitTrans	Ends a transaction that was started with a previous BeginTrans call.	CommitTrans()	void
Push	Pushes a command onto the undo stack.	Push(SyncfusionCommand cmd)Push(SyncfusionCommand cmd, SyncfusionCommand selectionStateCommand)	void
Redo	Execute the latest command from the redo stack.	Redo()	void
Rollback	Rolls back a transaction in progress that was started with a previous BeginTrans call.	Rollback()	void
Undo	Execute the latest command from the undo stack.	Undo()	void

Note: [View sample in GitHub](#)

See also

[How to exclude header while copying](#)

[How to invoke CommitCellInfo event](#)

Managing Rows and Columns in WPF GridControl

This section explains about Managing the rows and columns of WPF GridControl.

Each Grid instance is tied to a model, which contains the data represented by the Grid control. The grid model exposes properties that allow the user to manipulate grid rows and columns.

Rows and columns count

The grid model has RowCount and ColumnCount properties. These can be set to change the number of rows and columns in the grid control, as shown below:

C#

```
// Set Row count
grid.Model.RowCount = 10;
```

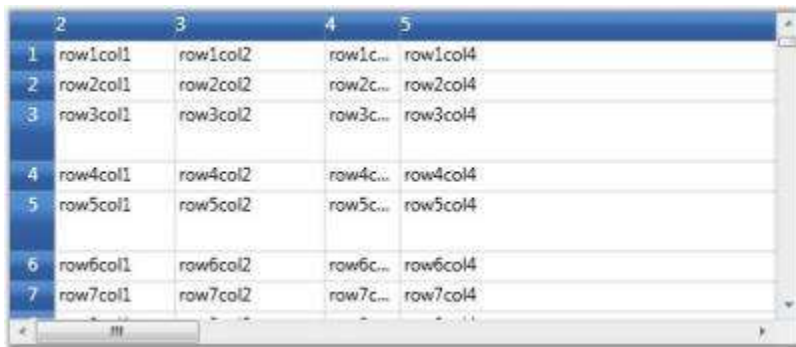
```
// Set Column count  
grid.Model.ColumnCount = 20;
```

Row heights and column widths

The grid model also stores information on row heights and column widths. Its `ColumnWidths` and `RowHeights` properties can be changed using indexers as shown below:

C#

```
// Setting column widths  
grid.Model.ColumnWidths[0] = 30;  
grid.Model.ColumnWidths[1] = 80;  
grid.Model.ColumnWidths[2] = 100;  
grid.Model.ColumnWidths[3] = 50;  
grid.Model.ColumnWidths[4] = 250;  
// Setting row heights  
grid.Model.RowHeights[5] = 40;  
grid.Model.RowHeights[3] = 40;
```



You can also specify the `DefaultLineSize` setting on `ColumnWidths` and `RowHeights` in order to set the default width or height.

C#

```
grid.Model.RowHeights.DefaultLineSize = 20;  
grid.Model.ColumnWidths.DefaultLineSize = 100;
```

Hiding rows and columns

Essential Grid supports efficient hiding of rows and columns. You can hide and unhide ranges of rows and columns using `SetHidden` method on `ColumnWidths` and `RowHeights`.

Note: `SetHidden` method accepts the following three parameters:

- The first parameter is an integer, which specifies the starting row/column to hide/unhide_
- The second parameter is an integer, which specifies the ending row/column to hide/unhide_
- Third is a boolean parameter that determines whether to hide or unhide the specified number of rows or columns. The

rows/columns will be hidden when this parameter is set to true._

The following code illustrates the usage of `SetHidden` method:

C#

```
// Hide rows
grid.Model.RowHeights.SetHidden(2, 100, true);
grid.Model.RowHeights.SetHidden(110, 1000, true);
// Unhide rows
grid.Model.RowHeights.SetHidden(1010, 10000, false);
//Hide columns
grid.Model.ColumnWidths.SetHidden(2, 100, true);
grid.Model.ColumnWidths.SetHidden(110, 150, true);
// Unhide columns
grid.Model.ColumnWidths.SetHidden(1010, 10000, false);
```

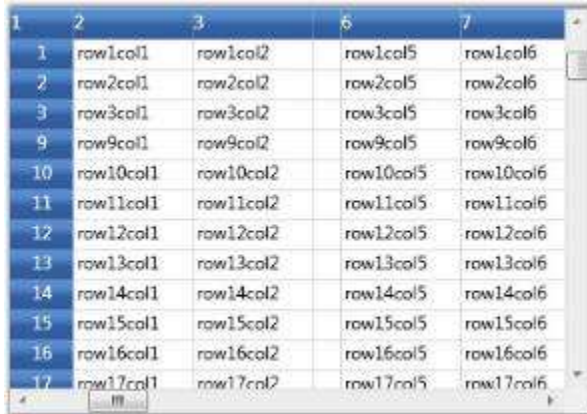
	2	102	103	104	105
1	row1col1	row1col101	row1col102	row1col103	row1
101	row101col1	row101col101	row101col102	row101col103	row1
102	row102col1	row102col101	row102col102	row102col103	row1
103	row103col1	row103col101	row103col102	row103col103	row1
104	row104col1	row104col101	row104col102	row104col103	row1
105	row105col1	row105col101	row105col102	row105col103	row1
106	row106col1	row106col101	row106col102	row106col103	row1
107	row107col1	row107col101	row107col102	row107col103	row1
108	row108col1	row108col101	row108col102	row108col103	row1
109	row109col1	row109col101	row109col102	row109col103	row1
1001	row1001col1	row1001col...	row1001col...	row1001col...	row1
1002	row1002col1	row1002col...	row1002col...	row1002col...	row1
1003	row1003col1	row1003col...	row1003col...	row1003col...	row1
1004	row1004col1	row1004col...	row1004col...	row1004col...	row1
1005	row1005col1	row1005col...	row1005col...	row1005col...	row1
1006	row1006col1	row1006col...	row1006col...	row1006col...	row1

Freeze rows and columns

It is possible to fix any number of rows and columns so that they are still visible when a grid is scrolled. This feature is called as freezing. It can be achieved in the Grid by setting the FrozenRows and FrozenColumns properties of grid model, as shown below:

C#

```
// Freeze rows and columns
grid.Model.FrozenRows = 4;
grid.Model.FrozenColumns = 3;
```

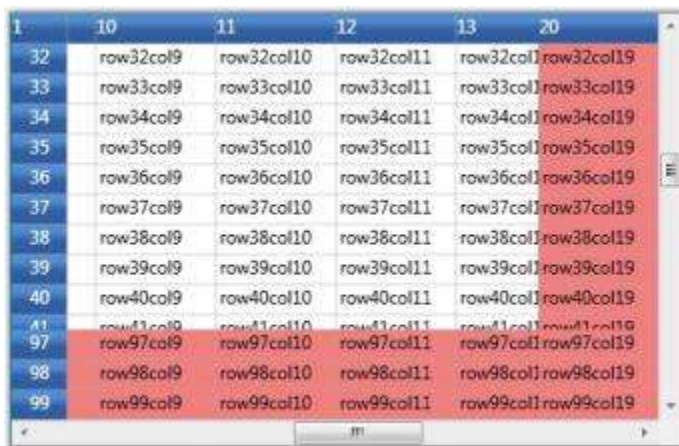



You can also fix rows to the right of the grid and columns to the bottom. Such fixed rows and columns are referred to as Footer rows and Footer columns. The properties `FooterRows` and `FooterColumns` determine the number of footer rows and footer columns. The footer row or column can be customized by using the `FooterStyle` property.

The following code illustrates the usage of `FooterRows`, `FooterColumns` and `FooterStyle` properties:

C#

```
// Footer rows and columns
grid.Model.FooterRows = 3;
grid.Model.FooterColumns = 1;
grid.Model.FooterStyle.Background = Brushes.LightCoral;
```



Header rows and columns

Grid allows the user to have any number of header rows and columns. It is done by using the `HeaderRows` and `HeaderColumns` properties of the grid model. The `HeaderStyle` property of the grid model controls the appearance of these header rows and header columns.

The following code illustrates the usage of `HeaderRows`, `HeaderColumns` and `HeaderStyle` properties:

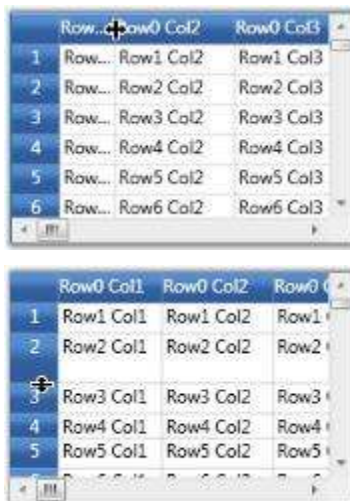
C#

```
// Header rows and columns
grid.Model.HeaderRows = 3;
grid.Model.HeaderColumns = 2;
grid.Model.HeaderStyle.Font.FontStyle = FontStyles.Italic;
```



Resize rows and columns

Grid allows the user to resize the rows and columns at run time. When this feature is enabled and if you move the mouse over the row or column divider, it will show a resize cursor using which you can resize the row or column to the required level. The following images illustrate the resizing of a column and a row:



This feature is turned on by default. To disable column or row resizing, you need to detach the corresponding mouse controllers from grid, as shown below:

C#

```
IMouseController controller = grid.MouseControllerDispatcher.Find
("ResizeRowsMouseController");
grid.MouseControllerDispatcher.Remove(controller);
controller = grid.MouseControllerDispatcher.Find
("ResizeColumnsMouseController");
grid.MouseControllerDispatcher.Remove(controller);
```

Note: To prevent resizing of specific row or column, it is required to handle ResizingRows and ResizingColumns events.

Inserting rows and columns

New columns and rows can be inserted at run time by using the following APIs:

- InsertColumns()
- InsertRows()

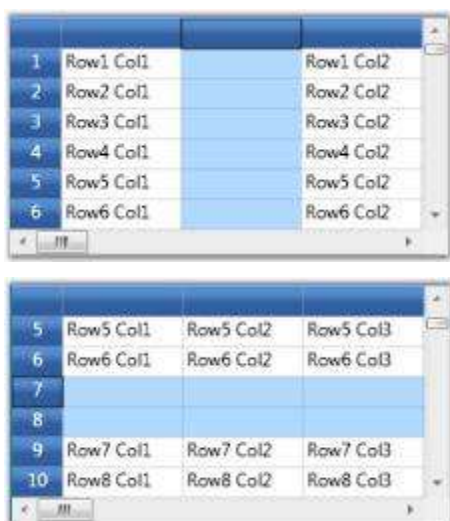
Both these methods accept the following two parameters:

1. Position index
2. Number of rows or columns to insert

The following code illustrates the usage of InsertColumns and InsertRows methods:

C#

```
//Insert a column at position 2.  
grid.Model.InsertColumns(2, 1);  
//Insert 2 rows at position 5.  
grid.Model.InsertRows(5, 2);
```



Note: You can track the moment the rows or columns are inserted by handling the RowsInserted and ColumnsInserted events.

Moving rows and columns

The rows and columns can be rearranged dynamically by moving them from one position to another using the following APIs:

- MoveRows()
- MoveColumns()

These methods accept the following three parameters:

- Position from which the rows or columns should be removed
- Number of rows or columns
- The new position at which these rows or columns should be inserted

You can also achieve this by a simple drag-and-drop action on the desired rows and columns.

The following code illustrates the usage of MoveColumns and MoveRows methods:

C#

```
//Move 3 rows from index 2 to index 5.
grid.Model.MoveRows(2, 3, 5);
//Move 2 columns from index 1 to index 4.
grid.Model.MoveColumns(1, 2, 4);
```

	3	4	5	1	2
1	Row1 Col3	Row1 Col4	Row1 Col5	Row1 Col1	Row1 Col2
2	Row5 Col3	Row5 Col4	Row5 Col5	Row5 Col1	Row5 Col2
3	Row6 Col3	Row6 Col4	Row6 Col5	Row6 Col1	Row6 Col2
4	Row7 Col3	Row7 Col4	Row7 Col5	Row7 Col1	Row7 Col2
5	Row2 Col3	Row2 Col4	Row2 Col5	Row2 Col1	Row2 Col2
6	Row3 Col3	Row3 Col4	Row3 Col5	Row3 Col1	Row3 Col2
7	Row4 Col3	Row4 Col4	Row4 Col5	Row4 Col1	Row4 Col2
8	Row8 Col3	Row8 Col4	Row8 Col5	Row8 Col1	Row8 Col2
9	Row9 Col3	Row9 Col4	Row9 Col5	Row9 Col1	Row9 Col2
10	Row10 Col3	Row10 Col4	Row10 Col5	Row10 Col1	Row10 Col2
11	Row11 Col3	Row11 Col4	Row11 Col5	Row11 Col1	Row11 Col2
12	Row12 Col3	Row12 Col4	Row12 Col5	Row12 Col1	Row12 Col2
13	Row13 Col3	Row13 Col4	Row13 Col5	Row13 Col1	Row13 Col2

Note: You can track the moment the rows or columns are moved by handling the RowsMoved and ColumnsMoved event.

Removing rows and columns

It is possible to remove a range of rows and columns from the grid. The APIs RemoveRows() and RemoveColumns() are used to achieve this. They accept two parameters:

1. An index, from which the rows or columns should be removed
2. Total number of rows or columns to be removed

The following code illustrates the usage of RemoveColumns and RemoveRows methods:

C#

```
//Remove 4 rows from position 3.
grid.Model.RemoveRows(3, 4);
//Remove 3 columns from position 2.
grid.Model.RemoveColumns(2, 3);
```

	1	5	6
1	Row1 Col1	Row1 Col5	Row1 Col6
2	Row2 Col1	Row2 Col5	Row2 Col6
3	Row7 Col1	Row7 Col5	Row7 Col6
4	Row8 Col1	Row8 Col5	Row8 Col6
5	Row9 Col1	Row9 Col5	Row9 Col6
6	Row10 Col1	Row10 Col5	Row10 Col6
7	Row11 Col1	Row11 Col5	Row11 Col6
8	Row12 Col1	Row12 Col5	Row12 Col6
9	Row13 Col1	Row13 Col5	Row13 Col6

You can track the moment the rows or columns are inserted by handling the RowsRemoved and ColumnsRemoved events.

Cell Layout Customization in WPF GridControl

This section explains the Covered Cells, Banner Cells, Overlapping Cells and Graphic cell of the WPF GridControl.

Covered Cells

Covered Cells are cells that span over neighboring cells. The combined cells will act as if they are one single cell visually and programmatically. There are different possible options to form a covered range. You can combine the cells in adjacent rows or columns or both.

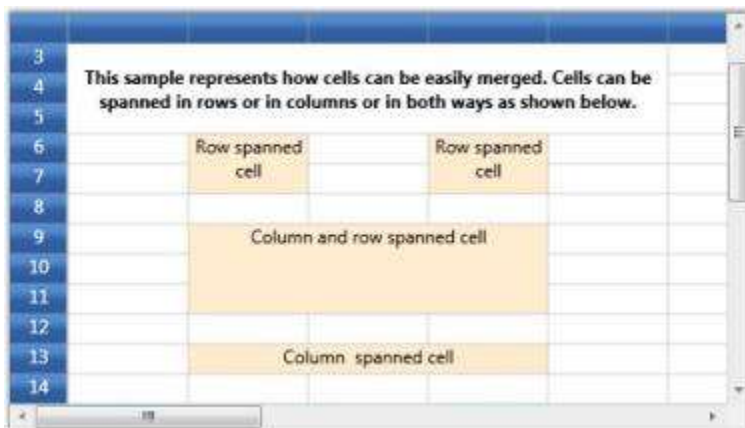
Creating Covered Range

C#

```
this.grid.Model.CoveredCells.Add (new CoveredCellInfo (6, 4, 7, 4));
this.grid.Model.CoveredCells.Add (new CoveredCellInfo (6, 6, 7, 6));
cell = this.grid.Model[6, 4];
cell.CellValue = "Row spanned cell";
cell.Background = Brushes.BlanchedAlmond;
cell.HorizontalAlignment = HorizontalAlignment.Center;
cell = this.grid.Model[6, 6];
cell.CellValue = "Row spanned cell";
cell.HorizontalAlignment = HorizontalAlignment.Center;
cell.Background = Brushes.BlanchedAlmond;
this.grid.Model.CoveredCells.Add (new CoveredCellInfo (9, 4, 11, 6));
cell = this.grid.Model[9, 4];
cell.CellValue = "Column and row spanned cell";
cell.HorizontalAlignment = HorizontalAlignment.Center;
cell.Background = Brushes.BlanchedAlmond;
this.grid.Model.CoveredCells.Add (new CoveredCellInfo (13, 4, 13, 6));
cell = this.grid.Model[13, 4];
cell.CellValue = "Column spanned cell";
cell.Background = Brushes.BlanchedAlmond;
cell.HorizontalAlignment = HorizontalAlignment.Center;
```

Output

The following output is generated using the code above.



Banner Cells

You can create custom range of cells inside a Grid, which is termed as banner cells. Let us see how to create Banner Cells.

Cell Spanned Backgrounds

Essential Grid lets you span the given background across multiple cells either row-wise, column-wise or both. The information about all the cell spans for a given grid is maintained by the `GridModel.CellSpanBackgrounds`. Each entry represents an object of `CellSpanBackgroundInfo` class that defines a cell span. This class exposes properties such as background, border, and more to customize the cell span.

You can also trigger `QueryCellSpanBackgrounds` event to create and customize cell spans.

Creating Cell Spans

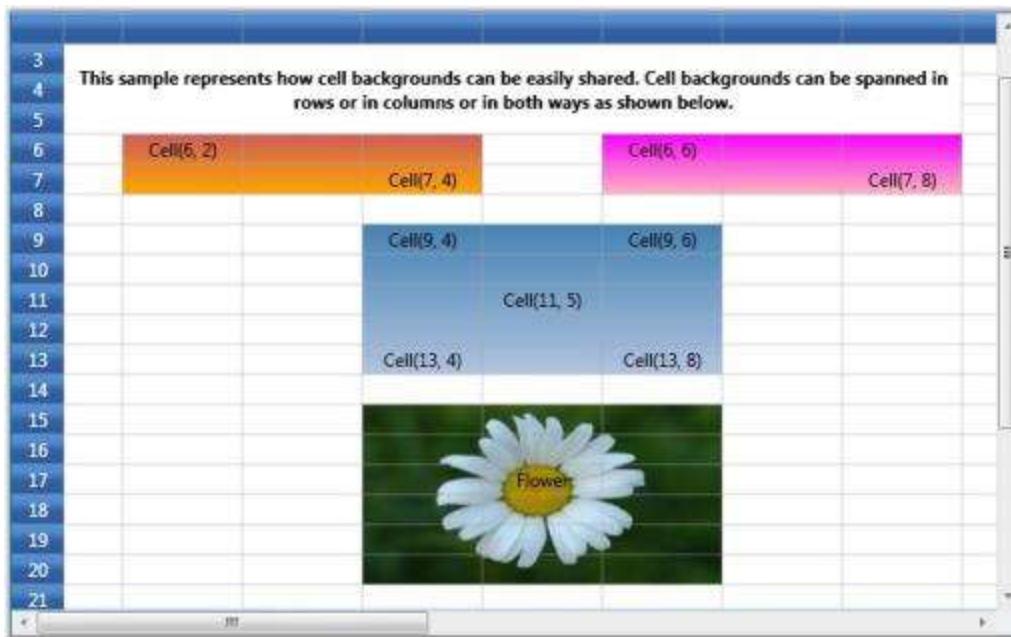
This example creates three cell spans with gradient backgrounds and a fourth cell span with an image background created through the `QueryCellSpanBackgrounds` event.

C#

```
CellSpanBackgroundInfo cellspan1 = new CellSpanBackgroundInfo(6, 2, 7, 4);
cellspan1.Background = new LinearGradientBrush(Colors.IndianRed,
Colors.Orange, 90);
grid.Model.CellSpanBackgrounds.Add(cellspan1);
CellSpanBackgroundInfo cellspan2 = new CellSpanBackgroundInfo(6, 6, 7, 8);
cellspan2.Background = new LinearGradientBrush(Colors.Magenta,
Colors.LightPink, 90);
grid.Model.CellSpanBackgrounds.Add(cellspan2);
CellSpanBackgroundInfo cellspan3 = new CellSpanBackgroundInfo(9, 4, 13, 6);
cellspan3.Background = new LinearGradientBrush(Colors.SteelBlue,
Colors.LightSteelBlue, 90);
grid.Model.CellSpanBackgrounds.Add(cellspan3);
grid.QueryCellSpanBackgrounds += new
GridQueryCellSpanBackgroundsEventHandler (grid_QueryCellSpanBackgrounds);
void grid_QueryCellSpanBackgrounds(object sender,
GridQueryCellSpanBackgroundsEventArgs e)
{
    if (e.CellRowColumnIndex.ColumnIndex == 4 && e.CellRowColumnIndex.RowIndex
    == 15)
    {
        CellSpanBackgroundInfo item = new
        CellSpanBackgroundInfo(e.CellRowColumnIndex.RowIndex,
        e.CellRowColumnIndex.ColumnIndex, 20, 6);
        item.Background = new
        ImageBrush(GetImage(@"common\Images\Grid\BannerCells\back2.jpg"));
        e.Range = new List<CellSpanBackgroundInfo>();
        e.Range.Add(item);
        e.Handled = true;
    }
}
```

Output

The following output is generated using the code above.



Overlapping Cells

Overlapping cells behavior occurs when the text exceeds the length of the cell and will float to the adjacent cell in non-editing mode. Flooding behavior specifies whether a previous cell can be allowed to float over the corresponding cell even if it is empty. Floating cell is to enable the cell to float over the next cell while editing despite of the flooding or overlapping behavior.

To assign the FloatingCell behavior to one particular cell or a certain range of cells. The FloatingCell behavior can be assigned to one particular cell or a certain range of cells as follows:

C#

```
//Provided as CellStyle
grid.Model[4, 1].EnableFloatingCell = true;
grid.Model[4, 1].FloatCellMode = GridFloatCellsMode.OnDemandCalculation;
//Provided as ColumnStyle
grid.Model.ColStyles[2].EnableFloatingCell = true;
grid.Model.ColStyles[2].FloatCellMode =
GridFloatCellsMode.OnDemandCalculation;
grid.Model.ColStyles[2].FloodCell = false;
//Provided as TableStyle
grid.Model.Options.EnableFloatingCell = true;
grid.Model.Options.FloatCellMode = GridFloatCellsMode.OnDemandCalculation;
grid.Model.Options.FloodCell = false;
```

	A	B	C	D	E	F	G
1	Floating Cell Demo						
2							
3							
4	Control overview						
5	Public	Gets the AccessibleObject assigned to the control					
6	Control	Initializes a new instance of the Control class.					
7	Public Properties						
8	Accessibility	Gets the AccessibleObject assigned to the control.					
9	AccessibleDef	Gets or sets the default action description of the control for use by accessibility client appli					
10	AccessibleDes	Gets or sets the description of the control used by accessibility client applications.					
11	AccessibleNa	Gets or sets the name of the control used by accessibility client applications.					
12	AccessibleRole	Gets or sets the accessible role of the control					
13	AllowDrop	Gets or sets a value indicating whether the control can accept data that the user drags onto					
14	Anchor	Gets or sets which edges of the control are anchored to the edges of its container.					

Properties

Property	Description	Type	Data Type	Referenc e links
EnableFloatCellFloatCellModeFloodCell	This allows the user to float the cell while typing. This floats the cell in non-editing mode. When the user specifies the property as false it will not allow the previous cell to float over it.	Static PropertyStati c PropertyStati c Property	BooleanGridFloatCellsMod e (enum type) Boolean	

Features of Overlapping Cells

Overlapping Cells

To overlap the cell when it is not in edit mode and calculate according to the flooding and length of the text.

C#


```
this.grid.Model.Options.FloatCellMode =
GridFloatCellsMode.OnDemandCalculation;
```

Flooding

To prevent the overlapping of previous cells.

C#

```
this.grid.Model.Options.FloodCell = false;
```

Floating

To enable floating cell behavior by calculating while editing the text.

C#

```
this.grid.Model.Options.EnableFloatingCell = true;
```

Graphic Cell

A graphic cell is a special type of cell that helps users render any content over the Grid control regardless of the underlying cell. Graphic cells have a separate style info class and model (GraphicModel) which is used to decide the styles and behaviors of graphic cells and their content.

GraphicStyleInfo Properties

You can get the GraphicStyleInfo object from the GraphicModel class by passing the index position or starting row and column index of the graphic cell as shown in the following code sample.

C#

```
GraphicStyleInfo style = this.grid.Model.GraphicModel[Row, Column];
```

Graphic cells have the following properties.

Properties

Property	Description	Type	Data Type
Background	Specifies a background brush for the graphic cell.	CLR Property	Brush
BorderBrush	Specifies a border brush for the graphic cell.	CLR Property	Brush
BorderThickness	Specifies a border thickness.	CLR Property	Thickness
CellType	Specifies which controls are loaded in the graphic cells.	CLR Property	string
CellValue	Specifies values that need to load in the control.	CLR Property	object
Foreground	Specifies text color.	CLR Property	Brush
HorizontalAlignment	Specifies the horizontal alignment of the control.	CLR Property	HorizontalAlignment

ReadOnly	Specifies whether the cell is editable.	CLR Property	Bool
VerticalAlignment	Specifies the vertical alignment of the control.	CLR Property	VerticalAlignment

Cell Types

The Grid control allows any controls to be loaded inside graphic cells. This greatly improves the usability and appearance of the Grid control. This attribute of a grid cell is referred to as its cell type.

Built-in Cell Types

Graphic cells have built-in support for the following cell types:

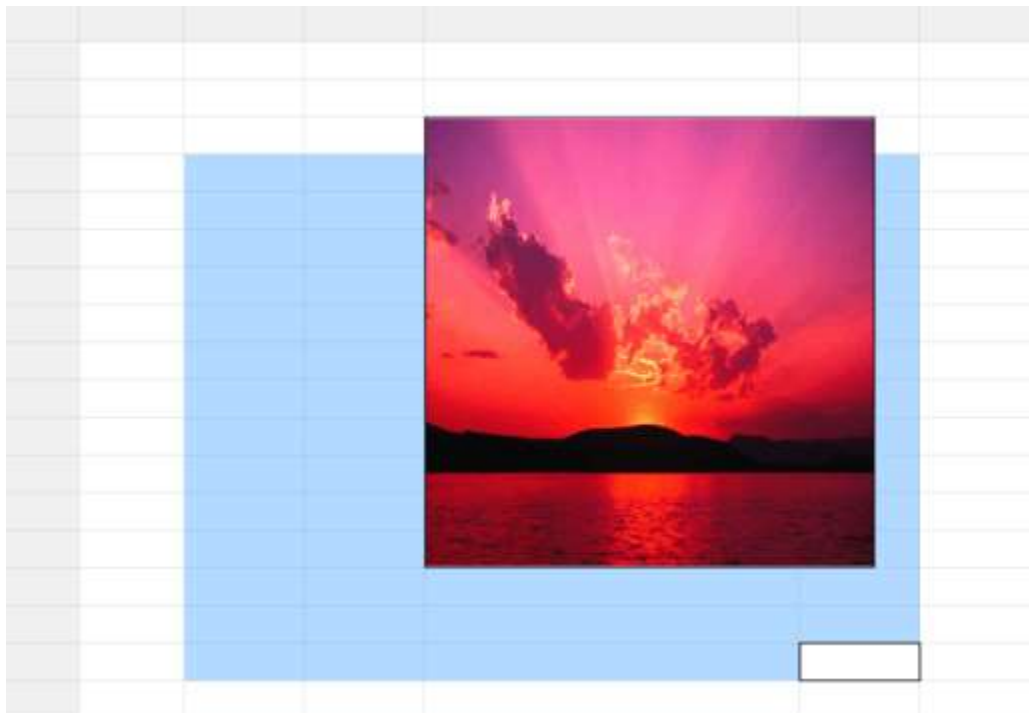
- Image Cells
- RichTextBox Cell

Image Cell

The image cell type is used to load images inside graphic cells. To load the graphic image cell in the Grid control you have to set the CellType as ImageCell and the CellValue as BitmapImage as shown in the following code sample.

C#

```
var style = this.grid.Model.GraphicModel[cellspan.RowIndex,
cellspan.ColumnIndex];
style.CellType = "ImageCell";
BitmapImage bi = new BitmapImage(new Uri(@"..\..\Resources\Sunset.jpg",
UriKind.RelativeOrAbsolute));
style.CellValue = bi;
```

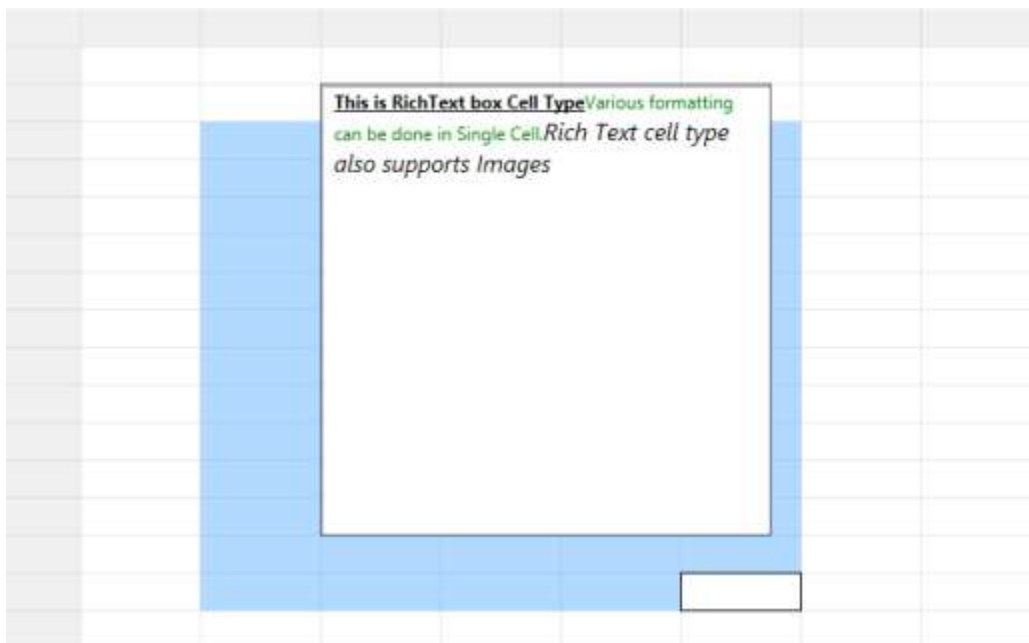


RichTextBox Cell

The RichTextBox cell type is used to load a rich text box inside graphic cells. To load the rich text box over the Grid control, you have to set the CellType as RichTextBox and the CellValue as FlowDocument as shown in the following code sample.

C#

```
FlowDocument _flowDocument = new FlowDocument();
Paragraph _paragraph = new Paragraph();
Run _run1 = new Run();
_run1.Text = "This is RichText box Cell Type";
_run1.TextDecorations = TextDecorations.Underline;
Run _run2 = new Run();
_run2.Text = "Various formatting can be done in Single Cell.";
_run1.FontWeight = FontWeights.Bold;
_run2.Foreground = Brushes.Green;
Run _run3 = new Run();
_run3.Text = "Rich Text cell type also supports Images";
_run3.FontSize = 16;
_run3.FontStyle = FontStyles.Italic;
_paragraph.Inlines.Add(_run1);
_paragraph.Inlines.Add(_run2);
_paragraph.Inlines.Add(_run3);
_flowDocument.Blocks.Add(_paragraph);
//Cell type as RichTextBox and Cell Value as FlowDocument
var style = this.grid.Model.GraphicModel[cellspan.RowIndex,
cellspan.ColumnIndex];
style.CellType = "RichTextBox";
style.CellValue = _flowDocument;
```



Custom Cell Types

The graphic cell model allows you to create custom derived controls to use additional cell types. This requires a cell model class and a cell renderer class. The cell model class creates the actual cell control

while the cell renderer class handles the UI requirements of the cell control. The custom cell type can be created by registering the cell model to the corresponding GraphicModel by naming this cell type. It can be enabled by assigning its name to the style.CellType property.

The following code can be used to register the chart cell type in the graphic model.

C#

```
this.grid.Model.GraphicModel.CellModels.Add("Chart", new
GraphicChartCellModel());
```

In general, the built-in cell types are also constructed only in this way. Every such cell type has its own cell model and renderer classes in the code base. These cell model and renderer classes originate from the GraphicCellModelBase and GraphicCellRendererBase classes. These two classes define the basic functionality for a cell type.

The GraphicCellRendererBase class has the following virtual methods you can override in you custom cell render class.

Methods

Method	Description	Parameters	Return Type
CreateUIElement	Create the UI element that needs to be loaded inside the graphic cell.	CreateUIElement(GraphicStyleInfo cellInfo)	Generic
OnArrange	Arrange the graphic cell in a particular rectangle.	OnArrange(UIElement uiElement, Rect rect, GraphicStyleInfo style)	void
OnInitializeContent	Initialize the properties from style info to the UI element.	OnInitializeContent(T element, GraphicStyleInfo style)	void
ShouldTryToHandlePreviewKeyDown	Handles KeyDown.	ShouldTryToHandlePreviewKeyDown(KeyEventArgs e)	bool

UnloadUIElements	When the control unloads, this method is executed.	UnloadUIElements(int index, T uiElement)	void
UnWireEvents	Unwire the wired events from the graphic cell control.	UnWireEvents(T element)	void
WireEvents	Wire the events in graphic cell.	WireEvents(T element)	void

Custom Graphic Chart Cell Renderer

This cell displays a chart control over the grid cells as seen in Microsoft Excel when importing an Excel file to the Grid control. To render the Chart control over the Grid control, you need to derive from the GraphicCellModel and GraphicCellRendererBase classes.

CellModel class

C#

```
public class
GraphicChartCellModel:GraphicCellModel<GraphicChartCellRenderer>
{
    public GraphicChartCellModel()
    {
    }
}
```

CellRenderer Class

C#

```
public class GraphicChartCellRenderer : GraphicCellRendererBase<Chart>
{
    public GraphicChartCellRenderer()
    {
    }
    // Create the chart by using the style info.
    protected override Chart CreateUIElement(GraphicStyleInfo cellInfo)
    {
        if (cellInfo.CellValue != null && cellInfo.CellValue is IChartShape)
        {
            IChartShape chartShape = cellInfo.CellValue as IChartShape;
            return chartShape.CreateChart();
        }
    }
}
```

```

return base.CreateUIElement(cellInfo);
}
// Initialize the Chart control.
protected override void OnInitializeContent(Char element, GraphicStyleInfo
style)
{
element.Background = style.Background;
element.BorderBrush = style.BorderBrush;
element.BorderThickness = style.BorderThickness;
base.OnInitializeContent(element, style);
}
}

```

The ChartExtensions class is used to create the Chart control (Syncfusion.Windows.Chart) from the IChartShape object (Syncfusion.XlsIO.IChartShape). While importing the Excel file to the Grid control, the chart in the Excel file is added to the graphic cell collection and the cell value is set as IChartShape.

C#

```

public static class ChartExtensions
{
// Create the chart Control.
public static Chart CreateChart(this IChartShape chartShape)
{
Chart chart = new Chart();
ChartArea chartArea = new ChartArea();
CreateChartSeries(chartArea, chartShape);
chart.Areas.Add(chartArea);
return chart;
}
// Create chart series.
private static void CreateChartSeries(CharArea chartArea, IChartShape
chartShape)
{
for (int count = 0; count < chartShape.Series.Count; count++)
{
IChartSerie XlsIOChartSerie = chartShape.Series[count];
ChartSeries UIChartSeries = new ChartSeries();
switch (XlsIOChartSerie.SeriesType)
{
case ExcelChartType.Column_Clustered:
UIChartSeries.Type = ChartTypes.Column;
break;
case ExcelChartType.Column_Stacked:
UIChartSeries.Type = ChartTypes.StackingColumn;
break;
case ExcelChartType.Column_Stacked_100:
UIChartSeries.Type = ChartTypes.StackingColumn100;
break;
case ExcelChartType.Line:
UIChartSeries.Type = ChartTypes.Line;
break;
case ExcelChartType.Line_Markers:
UIChartSeries.Type = ChartTypes.Line;
UIChartSeries.EnableEffects = true;
break;
}
}
}
}

```

```

case ExcelChartType.Pie:
    UIChartSeries.Type = ChartTypes.Pie;
break;
case ExcelChartType.Pie_Exploded:
    UIChartSeries.Type = ChartTypes.Pie;
    ChartPieType.SetExplodedAll(UIChartSeries, true);
break;
case ExcelChartType.Bar_Clustered:
    UIChartSeries.Type = ChartTypes.Bar;
break;
case ExcelChartType.Bar_Stacked:
    UIChartSeries.Type = ChartTypes.StackingBar;
break;
case ExcelChartType.Bar_Stacked_100:
    UIChartSeries.Type = ChartTypes.StackingBar100;
break;
case ExcelChartType.Area:
    UIChartSeries.Type = ChartTypes.Area;
break;
case ExcelChartType.Area_Stacked:
    UIChartSeries.Type = ChartTypes.StackingArea;
break;
case ExcelChartType.Scatter_Markers:
    UIChartSeries.Type = ChartTypes.Scatter;
break;
case ExcelChartType.Stock_HighLowClose:
    UIChartSeries.Type = ChartTypes.HiLo;
break;
case ExcelChartType.Stock_OpenHighLowClose:
    UIChartSeries.Type = ChartTypes.HiLoOpenClose;
break;
case ExcelChartType.Stock_VolumeHighLowClose:
    UIChartSeries.Type = ChartTypes.Candle;
break;
case ExcelChartType.Doughnut:
    UIChartSeries.Type = ChartTypes.Doughnut;
break;
case ExcelChartType.Doughnut_Exploded:
    UIChartSeries.Type = ChartTypes.Doughnut;
    ChartPieType.SetExplodedAll(UIChartSeries, true);
break;
case ExcelChartType.Bubble:
    UIChartSeries.Type = ChartTypes.Bubble;
break;
case ExcelChartType.Radar:
    UIChartSeries.Type = ChartTypes.Radar;
break;
default:
return;
}
DataTable ct =
XlsIOChartSerie.CategoryLabels.Worksheet.ExportDataTable(XlsIOChartSerie.CategoryLabels, ExcelExportDataTableOptions.ComputedFormulaValues);
DataTable vt =
XlsIOChartSerie.Values.Worksheet.ExportDataTable(XlsIOChartSerie.Values,
ExcelExportDataTableOptions.ComputedFormulaValues);

```

```

List<ChartDataPoint> data = ConvertDataTableToDataPoints(ct, vt,
chartShape.IsSeriesInRows);
UIChartSeries.DataSource = data;
UIChartSeries.BindingPathX = "X";
UIChartSeries.BindingPathsY = new List<string> { "Y" };
UIChartSeries.Label = XlsIOChartSerie.Name;
chartArea.Series.Add(UIChartSeries);
}
if (chartShape.HasLegend)
{
ChartLegend cl = new ChartLegend();
chartArea.Legend = cl;
switch (chartShape.Legend.Position)
{
case ExcelLegendPosition.Bottom:
Chart.SetDock(cl, ChartDock.Bottom);
break;
case ExcelLegendPosition.Corner:
Chart.SetDock(cl, ChartDock.Floating);
break;
case ExcelLegendPosition.Left:
Chart.SetDock(cl, ChartDock.Left);
break;
case ExcelLegendPosition.NotDocked:
Chart.SetDock(cl, ChartDock.Floating);
break;
case ExcelLegendPosition.Right:
Chart.SetDock(cl, ChartDock.Right);
break;
case ExcelLegendPosition.Top:
Chart.SetDock(cl, ChartDock.Top);
break;
default:
Chart.SetDock(cl, ChartDock.Right);
break;
}
}
chartArea.PrimaryAxis.Interval = 1;
}
// Create the itemsource for chart series.
private static List<ChartDataPoint> ConvertDataTableToDataPoints(DataTable
CategoryTable, DataTable ValuesTable, bool IsSeriesInRows)
{
List<ChartDataPoint> data = new List<ChartDataPoint>();
if (IsSeriesInRows)
{
for (int col = 0; col < CategoryTable.Columns.Count; col++)
{
double y = 0;
string x = CategoryTable.Rows[0][col].ToString();
double.TryParse(ValuesTable.Rows[0][col].ToString(), out y);
data.Add(new ChartDataPoint(x, y));
}
}
else
{
for (int row = 0; row < CategoryTable.Rows.Count; row++)

```



```

{
    double y = 0;
    string x = CategoryTable.Rows[row][0].ToString();
    double.TryParse(ValuesTable.Rows[row][0].ToString(), out y);
    data.Add(new ChartDataPoint(x, y));
}
}
return data;
}
}
public class ChartDataPoint
{
    public ChartDataPoint(string x, double y)
    {
        X = x;
        Y = y;
    }
    public string X { get; set; }
    public double Y { get; set; }
}

```

Graphic Cell Events

The GraphicModel class has the following two events for customizing the graphic cells.

Events

Event	Parameters	Description
GraphicQueryCellInfo	Â GraphicQueryCellInfoEventArgs	This event is similar to the QueryCellInfo event in GridModel. It is used to provide the cell values on demand.
GraphicCommittedCellInfo	GraphicCommitCellInfoEventArgs	This event is similar to the CommittedCellInfo event in GridModel. The changes made in the graphic cell will be saved by the GraphicCommittedCellInfo event.

Adding Graphic Cell to the Grid Control

To add the graphic cell to the Grid control, you have to create the instance for GraphicCellSpanInfo and add that object to the GraphicCells collection in the GraphicModel. For the GraphicCellSpanInfo constructor you have to pass the row index and column index (which will decide the starting point of the graphic cell) along with the height and width of the graphic cell.

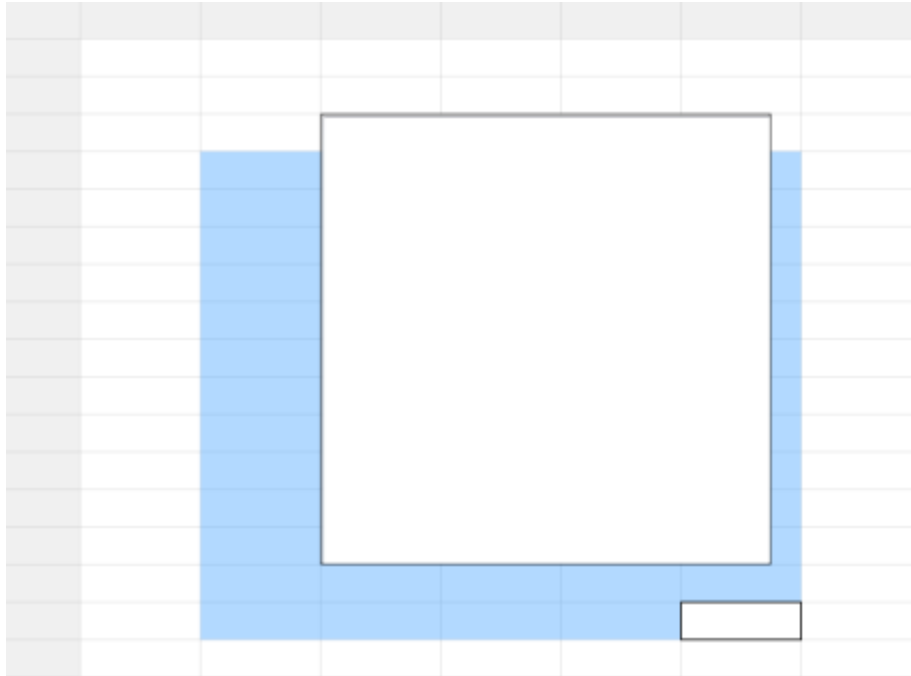
The following code sample demonstrates adding GraphicCell to the Grid control through GraphicModel.

C#

```

GraphicCellSpanInfo cellspan = new GraphicCellSpanInfo(rowIndex, colIndex,
300, 300);
this.grid.Model.GraphicModel.GraphicCells.Add(cellspan);

```

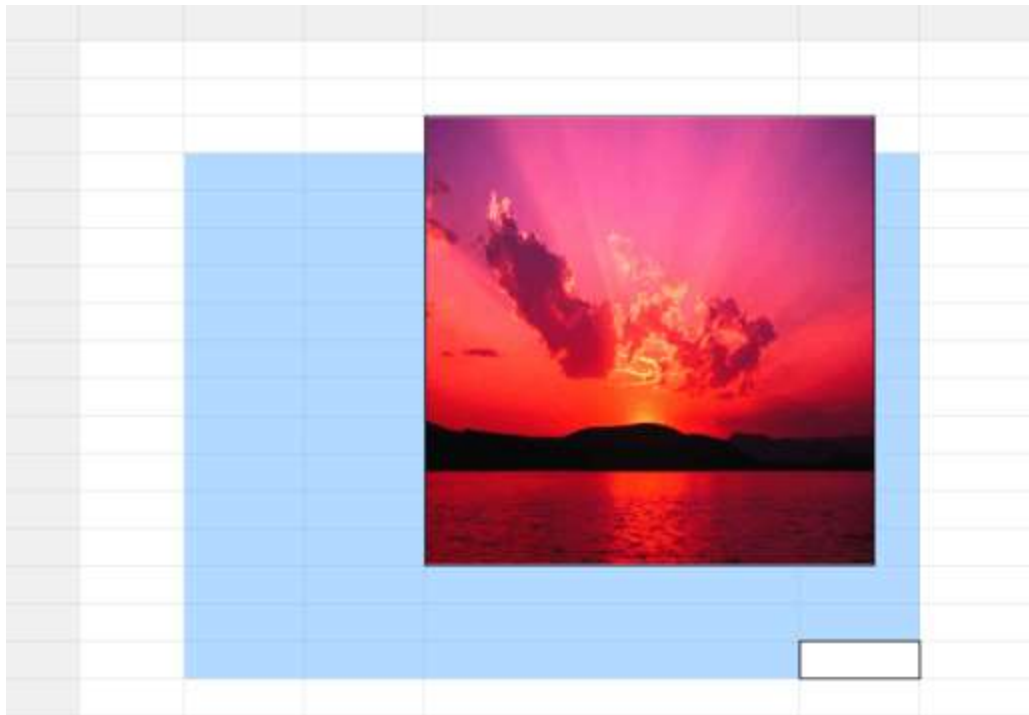


Setting Cell Type to the Graphic Cell

After adding the graphic cell to GraphicModel you can see the empty cell over the Grid control in the specified position. Now you have set the CellType, CellValue, and other style information by accessing the GraphicStyleInfo from GraphicModel. You can get the graphic cell style from the GraphicModel by passing the row and column index. Then by using that style you can set the cell type, cell values and other customization options for the Graphic cell, as shown in the following code sample.

C#

```
var style = this.grid.Model.GraphicModel[cellspan.RowIndex,
cellspan.ColumnIndex];
style.CellType = "ImageCell";
BitmapImage bi = new BitmapImage(new Uri(@"..\..\Resources\Sunset.jpg",
UriKind.RelativeOrAbsolute));
style.CellValue = bi;
```



If you do not set the offset value in the `GraphicCellSpanInfo` object, the control inside the graphic cells will be loaded in the starting position of the row and column index. By setting the offset value in the `GraphicCellSpanInfo` you can place the control anywhere in the cell.

Covered Ranges in WPF GridControl

This section explains the covered range of cells in WPF GridControl.

Creating covered cells

The range of cells can be covered by adding the [CoveredCellInfo](#) to the [CoveredRanges](#) collection. The [CoveredRanges](#) will be maintained in the [GridCoveredCellInfoCollection](#) collection.

C#

```
grid.Model.CoveredRanges.Add(new CoveredCellInfo(2, 2, 5, 5));
```

Creating covered cells using QueryCoveredRange event

You can also covered the range of cells by using [QueryCoveredRange](#) event. This event will be raised for all the cells and you can set the range of cells by using [Range](#) property.

C#

```
//Triggering the QueryCoveredRange event
grid.QueryCoveredRange += Grid_QueryCoveredRange;
private void Grid_QueryCoveredRange(object sender,
Syncfusion.Windows.Controls.Grid.GridQueryCoveredRangeEventArgs e)
{
    //Checking the cell to start covered range.
    if(e.CellRowIndex.RowIndex == 2 && e.CellRowIndex.ColumnIndex == 2)
    {
        //Set the range to be covered.
        e.Range = new CoveredCellInfo(2, 2, 5, 5);
    }
}
```

```
//Handled property has to be enabled to perform this customization.
e.Handled = true;
}
}
```

	7	13	9	8	15	15	7	10
	15	15				13	7	4
	3					9	10	5
	8					5	14	15
	10					16	7	3
	17	11	3	5	8	9	4	7
	4	6	7	7	6	11	10	16
	4	6	17	9	3	14	10	11
	2	14	13	10	12	8	4	5

Note: [View sample in GitHub](#)

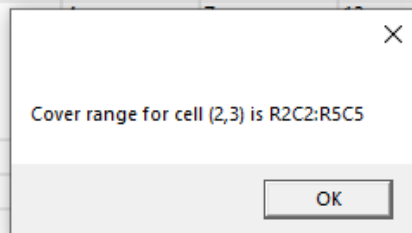
Find whether a cell in covered range

When you want to find a cell in covered ranges, you can use the [GetCoveredCell\(\)](#) method. If the specified cell with row index and column index is inside in [GetCoveredCell](#), a range will be returned.

C#

```
// Adding covered ranges
grid.Model.CoveredRanges.Add(new CoveredCellInfo(2, 2, 5, 5));
//Find the covered ranges
CoveredCellInfo coverRanges = grid.Model.CoveredRanges.GetCoveredCell(2, 3);
MessageBox.Show("Cover range for cell (2,3) is " + "R" + coverRanges.Left +
"C" + coverRanges.Top + ":" + "R" + coverRanges.Bottom + "C" +
coverRanges.Right);
```

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
Row 1	14	10	5	12	5	11	5	17
Row 2	17	9						
Row 3	2							
Row 4	5							
Row 5	9							
Row 6	4	15	8	13	10			
Row 7	4	7	7	3	4			
Row 8	3	5	8	10	9			
Row 9	5	2	2	3	6	4	17	13



Note: [View sample in GitHub](#)

Remove covered range at run time

You can remove the covered range at run time by using [Clear\(\)](#) method.

C#

```
//Removing the covered range from GridControl.
grid.Model.CoveredRanges.Clear();
```

For example, If you want to remove the covered range at run time, create one button and set the `Clear()` method for covered range in click event of button.

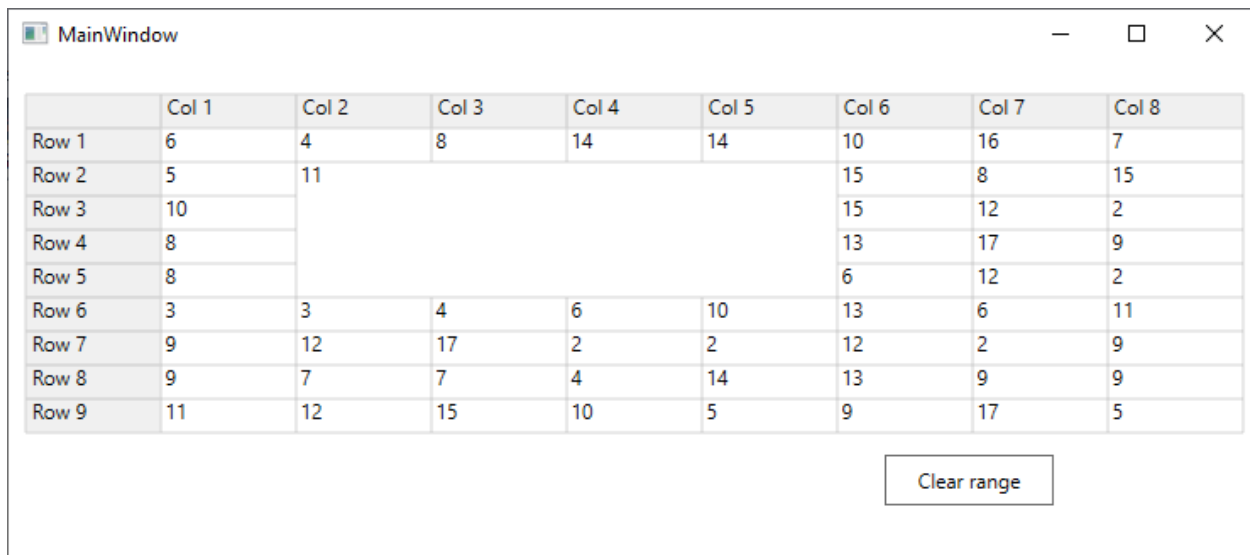
XML

```
<Grid>
<syncfusion:GridControl Name="grid" Margin="10,20,0,0" />
<Button Height="50" Width="100" Margin="400,200,0,0" Click="Button_Click" />
</Grid>
```

C#

```
grid.Model.CoveredRanges.Add(new CoveredCellInfo(2, 2, 5, 5));
private void Button_Click(object sender, RoutedEventArgs e)
{
    grid.Model.CoveredRanges.Clear();
    grid.InvalidateCells();
}
```

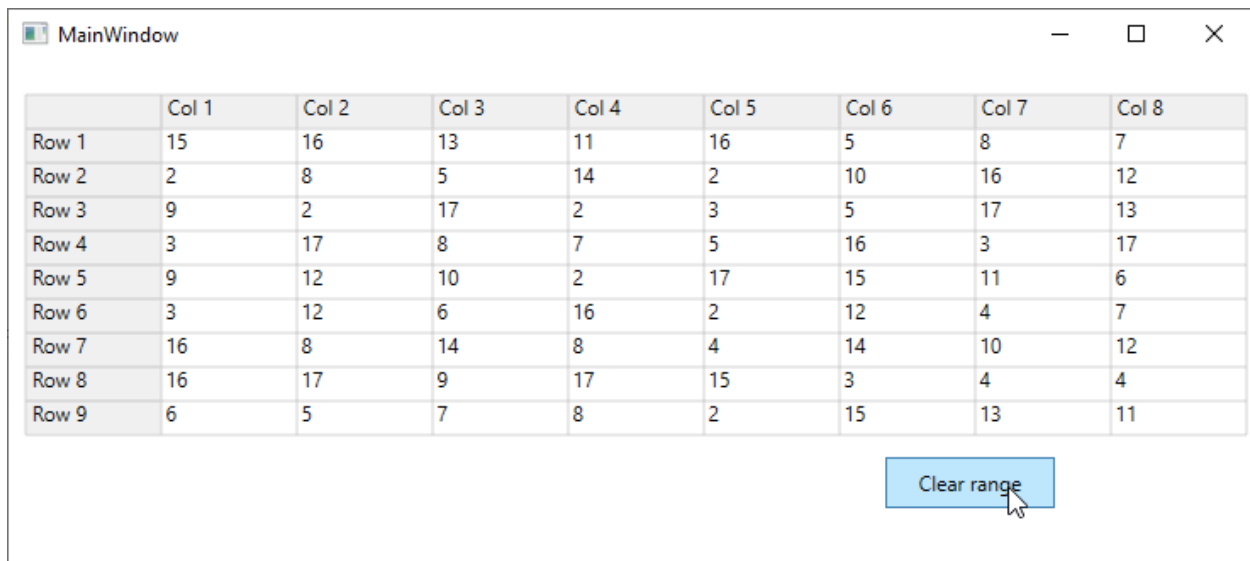
The below image provides covered range at run time before removing



	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
Row 1	6	4	8	14	14	10	16	7
Row 2	5	11				15	8	15
Row 3	10					15	12	2
Row 4	8					13	17	9
Row 5	8					6	12	2
Row 6	3	3	4	6	10	13	6	11
Row 7	9	12	17	2	2	12	2	9
Row 8	9	7	7	4	14	13	9	9
Row 9	11	12	15	10	5	9	17	5

Clear range

The below image provides covered range at run time after removing



Note: [View sample in GitHub](#)

Extend covered range at run time

You can extend the covered range at run time by using [Add\(\)](#) method.

For example, Create one button. Next, clear the current covered cell collection using `Clear()` method and create new covered cell ranges by using `Add()` method in this click event.

Note: Before extend the covered cell range, you need to clear the covered range.

C#

```
//Remove the current covered cell range
grid.Model.CoveredRanges.Clear();
grid.InvalidateCells();
//Add new covered cell range
grid.Model.CoveredRanges.Add(new CoveredCellInfo(2, 2, 7, 7));
```

The below image provides covered range at run time before extending.

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
Row 1	8	8	5	8	15	7	7	8
Row 2	11	5				10	6	7
Row 3	5					10	13	14
Row 4	7					16	2	12
Row 5	12					5	3	11
Row 6	4	7	11	16	5	12	10	10
Row 7	9	5	15	8	2	10	11	16
Row 8	12	9	2	15	2	2	3	13
Row 9	4	2	17	11	5	4	8	2

Clear range

The below image provides covered range at run time after extending.

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
Row 1	16	16	13	9	15	2	14	13
Row 2	2	7						3
Row 3	15							7
Row 4	5							2
Row 5	4							5
Row 6	16							3
Row 7	2							15
Row 8	9	8	6	2	7	3	9	10
Row 9	4	16	2	4	16	7	15	3

Clear range

Note: [View sample in GitHub](#)

Selection in WPF GridControl

WPF GridControl supports for range and record selection modes. Selection will highlight the specified range.

Selection modes

There are two modes of selection available in the GridControl. They are,

- Range selection
- Record selection

Range selection

1. In Range selection, you will be able to select cell ranges; but the selections will have no knowledge of nested tables, grouping or sorting and hence the functionality is limited like a data bound grid (GridData control).
2. To use the model selection capability, set AllowSelections to any flag except none.
3. Selection can be made through keyboard and mouse.

Record selection

1. It is designed specifically for the data bound grids.
2. In Record selection, the complete grid records (rows) will be selected and these selections function properly with nested tables, sorting, and so on.
3. To use the record selections, you must set AllowSelections to none and then set ListBoxSelectionMode to any flag except none.
4. Selection can be made through keyboard and mouse with some restriction. For more details, see Record-based Selection in this topic.

Range selection

Range selection is a cell-based selection mode that allows you to do a selection across the cell by using the [AllowSelection](#) property. It accepts value from [GridSelectionFlags](#) enumeration. Default value for [AllowSelection](#) is [Any](#) and the specified range of Cell/Row/Column or Table can be highlighted using the range section.

The possible values for this type of selection are defined by the enum GridSelectionFlags. To control the selection behavior of the grid, set any of the following flags to the AllowSelection property.

Selection flags

Flag	Description
None	Disables selecting of cells.
Row	Allows selection of rows.
Column	Allows selection of columns.
Table	Allows selection of the whole table.
Cell	Allows selection of an individual cell.
Multiple	Allows selection of multiple ranges of cells. The user has to press CTRL Key to select multiple ranges.
Shift	Allows extending existing selection when user holds SHIFT Key and clicks on a cell.
Keyboard	Allows extending existing selection when user holds SHIFT Key and presses arrow keys.
MixRangeType	Allows both rows and columns to be selected at the same time when Multiple is specified. By default, the grid does not allow row and column ranges to be selected at the same time.

Any	Allows selection of rows, columns, table, cell and multiple ranges of cells; also extends SHIFT Key support and alpha blending.
-----	---

Note: You can combine more than one flag to customize the current selection behavior.

C#

```
grid.Model.Options.AllowSelection = GridSelectionFlags.Multiple |
GridSelectionFlags.Column;
```



Record selection

This type of selection mechanism allows selection in terms of record (entire row). It is not cell-based. This selection mode is specifically designed for a data-bound grid, in which the grid data can be organized as a collection of record rows.

Grid offers the following three types of record selections which are together called as `ListBoxSelectionMode`.

- `SelectionMode-One`
- `SelectionMode-MultiSimple`
- `SelectionMode-MultiExtended`

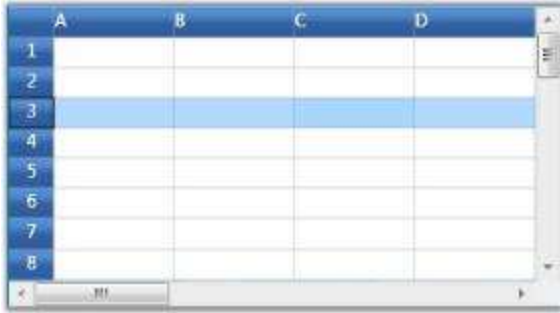
To enable record selection, set the [ListBoxSelectionMode](#) property to any of the above specified List Box Selection Mode values. To enable list box selection, turn off the range selection by setting the `AllowSelection` property to `Row`. Below is a detailed description of list box selection modes.

SelectionMode-One

It allows you to select only one item (record). For example, you have selected a record. Now if you select some other record, the previous record selection will be cleared. Hence it is a **One** record selection mode.

C#

```
grid.AllowSelection = GridSelectionFlags.Row;
grid.Model.Options.ListBoxSelectionMode = GridSelectionMode.One;
```



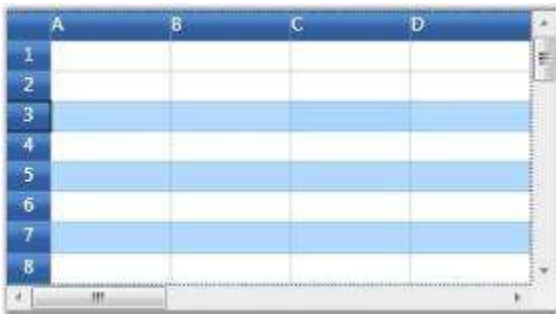
Note: Record can be selected using a single mouse click or using UP or DOWN Arrow Keys

SelectionMode - MultiSimple

In this selection mode, you will be able to select multiple items individually. For instance, you have selected a record using mouse and you want to select one more record. Click another record and you will notice that the previous selection is not cleared. Hence You can select multiple records without the need of SHIFT or CTRL keys.

C#

```
grid.AllowSelection = GridSelectionFlags.Row;  
grid.Model.Options.ListBoxSelectionMode = GridSelectionMode.MultiSimple;
```



Note: It does not support the use of SHIFT, CTRL and arrow keys to extend the selection.

SelectionMode - MultiExtended

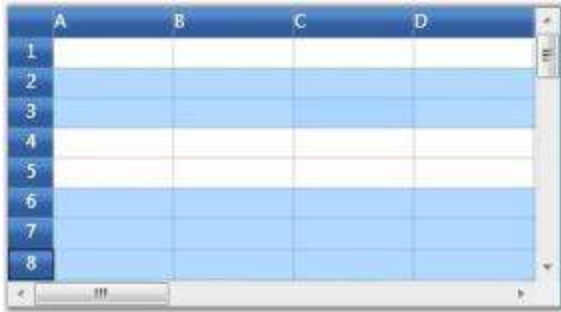
This selection type allows multiple items selection through SHIFT, CTRL and arrow keys.

You can do any of the following when this selection mode is enabled:

- Select a record, hold down the SHIFT key and select fourth record, for example. You will notice all the records in between 1st and the 4th records are also selected.
- You can make random selection by holding down the CTRL key.
- Hold down the Shift key and select the records using the UP or DOWN ARROW keys.

C#

```
grid.AllowSelection = GridSelectionFlags.Row;  
grid.Model.Options.ListBoxSelectionMode = GridSelectionMode.MultiExtended;
```



Selecting rows and columns programmatically

The entire grid selections are managed by the [GridModel.Selections](#) collection. It exposes several APIs that let you to add, remove and operate on different grid selections. Below is the description of some important properties and APIs:

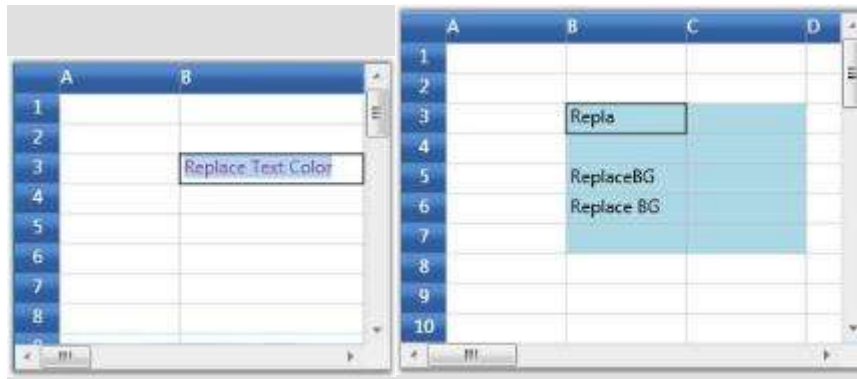
Property/Method	Description
Add , Remove	Adds or removes the specified range to/from the collection.
{{' InsertRows ' markdownify }}, {{' InsertColumns ' markdownify }}	Inserts new rows or columns into the collection.
{{' RemoveRows ' markdownify }}, {{' RemoveColumns ' markdownify }}	Removes the specified rows or columns from the collection.
Ranges	A GridRangeInfoList collection that stores all the selected ranges for the grid.
SelectRange	Adds or removes a range to/from the collection.
{{' GetSelectedRanges ' markdownify }}	Retrieves a list of selected ranges and if there are no selected ranges, returns the current cell.
GetSelectedRows	Returns the number of selected rows.
GetSelectedCols	Returns the number of selected columns.

Selection appearance customization

It is possible to modify the appearance of the selection through property settings. The following properties work in combinations to produce some special effects.

Property	Description
DrawSelectionOptions	Defines the selection behavior for the grid. Important options are: AlphaBlendReplaceBackgroundReplaceTextColor
HighlightSelectionAlphaBlend	Specifies the alpha blend color used for selection.
HighlightSelectionBackground	Specifies the background color for selection.

HighlightSelectionForeground	Specifies the foreground color for selection.
--	---



Customizing



AlphaBlendSelection background.

C#

```
LinearGradientBrush brush = new LinearGradientBrush(new
GradientStopCollection()
{
    new GradientStop(GridUtil.GetXamlConvertedValue<Color>("#A0E01020"), 0d),
    new GradientStop(GridUtil.GetXamlConvertedValue<Color>("#A0E01020"),
0.318681d),
    new GradientStop(GridUtil.GetXamlConvertedValue<Color>("#A0E08000"),
0.604396d),
    new GradientStop(GridUtil.GetXamlConvertedValue<Color>("#A0E08000"), 1d)
});
brush.StartPoint = new Point(0.5, -0.0430693);
brush.EndPoint = new Point(0.5, 0.928826);
grid.Model.Options.HighlightSelectionAlphaBlend = brush;
grid.Model.Options.DrawSelectionOptions =
GridDrawSelectionOptions.AlphaBlend;
```

![Alpha-blended-selection](Selection-Images/AlphaBlendSelection.jpeg)

Customizing the background of selected ranges.

C#

```
grid.Model.Options.DrawSelectionOptions =  
GridDrawSelectionOptions.ReplaceBackground;  
grid.Model.Options.HighlightSelectionBackground = Brushes.LightBlue;
```

![Selection Background set to blue](Selection-Images/SelectionBackground.jpeg)

Customizing foreground color of selected ranges.

C#

```
grid.Model.Options.DrawSelectionOptions =  
GridDrawSelectionOptions.ReplaceTextColor;  
grid.Model.Options.HighlightSelectionForeground = Brushes.Red;
```

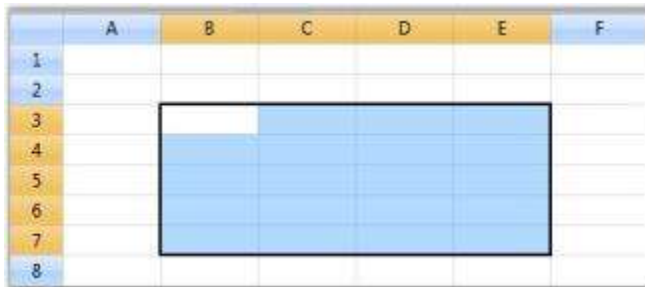
![Foreground of the selection set to pink](Selection-Images/SelectionForeground.jpeg)

Excel-like selection frame

The active selection can be outlined with a selection frame by setting the [GridModelOptions.ExcelLikeSelectionFrame](#) property to `true`.

C#

```
grid.Model.Options.ExcelLikeSelectionFrame = true;
```



Note: If multiple ranges are selected, the selection frame is applicable only for `ActiveRange`.

CurrentCell

When a cell is activated current cell is outlined with a border. You can show or hide current cell outline by setting [ShowCurrentCell](#) property.

C#

```
//To disable the current cell.  
this.gridControl.Model.Options.ShowCurrentCell = true;
```



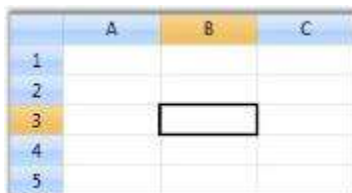
	Name	Id	Date	Country	Ship City	
	Jay	E2	5/23/2012 1...	Pune	Resende	
	Jay	E19	5/23/2012 1...	Pune	Resende	
	Smith	E26	5/23/2012 1...	India	Aires	
	Jay	E18	5/23/2012 1...	Pune	Bruxelles	
	Smith	E22	5/23/2012 1...	UK	Graz	
	Peter	E17	5/23/2012 1...	India	Bruxelles	
	Krish	E27	5/23/2012 1...	China	Aires	
	Jay	E11	5/23/2012 1...	UK	Bruxelles	
	Krish	E6	5/23/2012 1...	China	Aires	
	John	E24	5/23/2012 1...	USA	Graz	
	Peter	E29	5/23/2012 1...	USA	Bruxelles	
	John	E1	5/23/2012 1...	India	Rio de janeiro	
	Smith	E8	5/23/2012 1...	UK	Resende	
	Krish	E29	5/23/2012 1...	China	Resende	
	Jay	E23	5/23/2012 1...	UK	Bruxelles	
	Smith	E16	5/23/2012 1...	UK	Bruxelles	
	Jay	E7	5/23/2012 1...	India	Graz	
	Jay	E18	5/23/2012 1...	USA	Graz	

Excel-like CurrentCell

You can select a current cell in the Grid, similar to the current cell behavior in Microsoft Excel(borders with thickness). This feature can be enabled by setting [GridModelOptions.ExcelLikeCurrentCell](#) property to true.

C#

```
grid.Model.Options.ExcelLikeCurrentCell = true;
```



	A	B	C
1			
2			
3			
4			
5			

Note: If you have selected a current cell within a specified range, the range will be cleared, when you move the current cell selection out of this range.

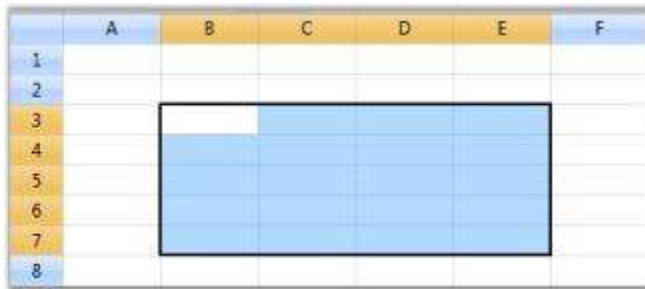
Highlighting row and column header based on selection

In Excel, whenever a selection is made, the headers of those rows and columns which are involved in the selection will be highlighted. You can get a similar behavior in the Grid by overriding the [OnPrepareRenderCell](#) method.

`OnPrepareRenderCell` method will be invoked for every cell in the grid, when they are about to be rendered. Hence, using this method, the cells which are going to be rendered are identified and their headers are highlighted.

C#

```
class ExcelGrid : GridControl
{
    protected override void OnPrepareRenderCell(GridPrepareRenderCellEventArgs e)
    {
        base.OnPrepareRenderCell(e);
        if (e.Cell.RowIndex == 0 &&
            Model.SelectedRanges.AnyRangeIntersects(GridRangeInfo.Col(e.Cell.ColumnIndex)))
        {
            e.Style.Background = this.excelOrange;
        }
        else if (e.Cell.ColumnIndex == 0 &&
            Model.SelectedRanges.AnyRangeIntersects(GridRangeInfo.Row(e.Cell.RowIndex)))
        {
            e.Style.Background = this.excelOrange;
        }
    }
    private Brush excelOrange = new SolidColorBrush(Color.FromRgb(244, 198, 111));
}
```



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

See also

[How to get first and last selected row index](#)

[How to set extended selection mode to select a row when user-click-a-cell](#)

[How to avoid selection header cell](#)

[How to programmatically invert selection](#)

[How to exclude header while copying](#)

[How to invoke CommitCellInfo event](#)

[How to change the selected cell border color](#)

[How to highlighted selected cells with border color for each cell](#)

[How to select a row or column when click a cell](#)

[How to select a row or column programmatically in code](#)

Autofit Cells in WPF GridControl

GridControl provides support to autofit rows and columns based on the content of cells.

Autofit row height

GridControl provides the support to auto fit the row height based on content of the cells using [ResizeRowsToFit](#) method which accepts the following parameters,

- [GridRangeInfo](#) - Specifies the range where **GridControl** auto fits the rows based on the cell content.
- [GridResizeToFitOptions](#) - Specifies the auto fit settings to customize the auto fit behavior.

C#

```
//To auto fit single row 2,
grid.Model.ResizeRowsToFit(GridRangeInfo.Row(2),
GridResizeToFitOptions.NoShrinkSize);
//To auto fit range of rows from 3 to 6,
grid.Model.ResizeRowsToFit(GridRangeInfo.Rows(3,6),
GridResizeToFitOptions.NoShrinkSize);
//To auto fit range of cell's(including Covered cells) row height,
this.grid.Model.ResizeRowsToFit(GridRangeInfo.Cells(1, 1, 2,
2),GridResizeToFitOptions.IncludeCellsWithinCoveredRange);
//To auto fit entire grid's row height,
this.grid.Model.ResizeRowsToFit(GridRangeInfo.Table(),
GridResizeToFitOptions.None);
```

Note: [View sample in GitHub](#)

Autofit column width

GridControl provides the support to auto fit the column width based on content of the cells using [ResizeColumnsToFit](#) method which accepts the following parameters,

- [GridRangeInfo](#) - Specifies the range where **GridControl** auto fits the columns based on the cell content.
- [GridResizeToFitOptions](#) - Specifies the auto fit settings to customize the auto fit behavior.

C#

```
//To auto fit single column 2,
grid.Model.ResizeColumnsToFit(GridRangeInfo.Col(2),
GridResizeToFitOptions.NoShrinkSize);
//To auto fit range of Columns from 3 to 6,
grid.Model.ResizeColumnsToFit(GridRangeInfo.Cols(3,6),
GridResizeToFitOptions.NoShrinkSize);
//To auto fit range of cell's(including Covered cells) column width,
this.grid.Model.ResizeColumnsToFit(GridRangeInfo.Cells(1, 1, 2,
2),GridResizeToFitOptions.IncludeCellsWithinCoveredRange);
//To auto fit entire grid's column width,
this.grid.Model.ResizeColumnsToFit(GridRangeInfo.Table(),
GridResizeToFitOptions.None);
```


Note: [View sample in GitHub](#)

Autofit Options

The auto fitting of rows and columns can be customized by using [GridResizeToFitOptions](#) enum. This enum provides the following options to control the autofit behavior of cells.

Options	Description
None	Default option to autofit the cells. Also, this option while auto fitting, shrinks the size and does not include covered cells, header cells.
ResizeCoveredCells	This option includes covered cells while auto fitting the cells. When using this option, only the last row or column of a covered range is resized.
NoShrinkSize	This option autofit the cells without shrinking the original size. For example, while auto fitting the cells, if the size is reduced than the normal size of the cell due to the content, you can use this option to retain the original size without shrinking.
IncludeHeaders	This option includes row/column header while auto fitting the cells.
IncludeCellsWithinCoveredRange	<code>ResizeCoveredCells</code> option only autofit the last row or column of a covered range. But this option autofit the columns or rows before the last one also.
IncludeHiddenCells	This option includes the hidden cells while auto fitting the cells. By default, visible cells only will be auto fitted and if you want to autofit all the cells including hidden cells in <code>GridControl</code> , then need to use this option.

Autofit Cells based on Wrap Text

To autofit the cell's height based on the applied wrap text, need to use [ResizeRowsToFit](#) method.

C#

```
this.grid.Model[2, 2].TextWrapping = TextWrapping.Wrap;
this.grid.Model.ResizeRowsToFit(GridRangeInfo.Cell(2,
2), GridResizeToFitOptions.NoShrinkSize);
```

Note: [View sample in GitHub](#)

How to avoid rendering issues due to fractions when auto fit the cells

When you autofit the cells, you may face scrolling and rendering issues in GridControl. You can set [GridColumnAutoSizer.CanRoundCalculation](#) to `true` to avoid rendering issues if needed.

C#

```
GridColumnAutoSizer.CanRoundCalculation = true;
```

Change the size of row height and column width to fit all cells in View

When GridControl is placed inside custom control and if you want to auto fit the row/column size of GridControl based on custom control resized position, then you can invoke [SizeChanged](#) event of GridControl and set the [RowHeights](#) and [ColumnWidths](#) property of [GridModel](#) to the resized height/width.

C#

```
grid.SizeChanged += grid_SizeChanged;
//To autofit the cells based on custom control resizing,
void grid_SizeChanged(object sender, SizeChangedEventArgs e)
{
    double width = (e.NewSize.Width - grid.Model.ColumnWidths.DefaultLineSize - 1) / (grid.Model.ColumnCount - 1);
    double height = (e.NewSize.Height - grid.Model.RowHeights.DefaultLineSize - 1) / (grid.Model.RowCount - 1);
    for (int i = 1; i < grid.Model.ColumnCount; ++i)
        grid.Model.ColumnWidths[i] = width;
    for (int j = 1; j < grid.Model.RowCount; ++j)
        grid.Model.RowHeights[j] = height;
}
```

Note: [View sample in GitHub](#)

See also

[How to auto fit all the columns in a GridControl based on the Window size](#)

Virtualization in WPF GridControl

This section covers the below grid virtualization topics:

- Virtual Mode-This section discusses how grid works in a virtual mode
- Virtual Cells-This section discusses about virtual cells in a grid when under virtual mode

Virtual Mode

Essential Grid for WPF supports virtual mode, which lets you dynamically provide data to the grid by handling an event, `QueryCellInfo`. This means that the grid does not store any data in its internal data structures. A virtual grid can display millions of rows as easily as it displays a dozen. The grid also exposes a second event, `CommitCellInfo` that lets you save the changes made in the UI, to the external data source.

Example

In this example, the Grid Control displays 99,000,000 x 1,000,000 cells (i.e., 99 million rows and 1 million columns). It is also possible to resize millions of rows instantly without any performance hits. The data is loaded only on demand through the `QueryCellInfo` event and the changes are saved back to the data source by the `CommitCellInfo` event.

C#

```
// a really large row and column count.
grid.Model.RowCount = 99000000; // 99 million
grid.Model.ColumnCount = 1000000; // 1 million
//Resize millions of rows instantly - pixel scrolling is updated accordingly.
```

```
grid.Model.RowHeights.SetRange(10, 1999999, 28);
grid.Model.RowHeights.SetRange(21111111, 21999999, 36);
// fill cell contents on demand.
grid.Model.QueryCellInfo += new
GridQueryCellInfoEventHandler(Model_QueryCellInfo);
// save back cell value into dictionary
grid.Model.CommitCellInfo += new
GridCommitCellInfoEventHandler(Model_CommitCellInfo);
Dictionary<RowColumnIndex, object> committedValues = new
Dictionary<RowColumnIndex, object>();
void Model_CommitCellInfo(object sender, GridCommitCellInfoEventArgs e)
{
    if (e.Style.HasCellValue)
    {
        committedValues[e.Cell] = e.Style.CellValue;
        e.Handled = true;
    }
}
void Model_QueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    if (e.Cell.RowIndex == 0)
    {
        if (e.Cell.ColumnIndex > 0)
            e.Style.CellValue = e.Cell.ColumnIndex;
    }
    else if (e.Cell.RowIndex > 0)
    {
        if (e.Cell.ColumnIndex == 0)
            e.Style.CellValue = e.Cell.RowIndex;
        else if (e.Cell.ColumnIndex > 0)
        {
            if (committedValues.ContainsKey(e.Cell))
                e.Style.CellValue = committedValues[e.Cell];
            else
                e.Style.CellValue = String.Format("{0}/{1}", e.Cell.RowIndex,
                e.Cell.ColumnIndex);
        }
    }
}
```

Output

	1	2	3	4	5	6
98999982	98999982/1	98999982/2			98999982/5	98999982/6
98999983	98999983/1	98999983/2	98999983/3	98999983/4	98999983/5	98999983/6
98999984	98999984/1	98999984/2	98999984/3	98999984/4	98999984/5	98999984/6
98999985	98999985/1	98999985/2	98999985/3	98999985/4	98999985/5	98999985/6
98999986	98999986/1	98999986/2			98999986/5	98999986/6
98999987	98999987/1	98999987/2	98999987/3	98999987/4	98999987/5	98999987/6
98999988	98999988/1	98999988/2	98999988/3	98999988/4	98999988/5	98999988/6
98999989	98999989/1	98999989/2	98999989/3	98999989/4	98999989/5	98999989/6
98999990	98999990/1	98999990/2			98999990/5	98999990/6
98999991	98999991/1	98999991/2	98999991/3	98999991/4	98999991/5	98999991/6
98999992	98999992/1	98999992/2	98999992/3	98999992/4	98999992/5	98999992/6
98999993	98999993/1	98999993/2	98999993/3	98999993/4	98999993/5	98999993/6
98999994	98999994/1	98999994/2			98999994/5	98999994/6
98999995	98999995/1	98999995/2	98999995/3	98999995/4	98999995/5	98999995/6
98999996	98999996/1	98999996/2	98999996/3	98999996/4	98999996/5	98999996/6
98999997	98999997/1	98999997/2	98999997/3	98999997/4	98999997/5	98999997/6
98999998	98999998/1	98999998/2			98999998/5	98999998/6
98999999	98999999/1	98999999/2	98999999/3	98999999/4	98999999/5	98999999/6

Virtual Cells

The Grid control supports virtual cell architecture where the cell contents are drawn statically until a live cell is required. For example, when you move the mouse over the grid, the cells under the mouse pointer needs to handle mouse inputs. Dynamically, the static cells are turned into live cells that can handle those mouse interactions, as required. These live cells stay in scope until they are no longer needed (which is usually when they are scrolled off the screen). Using static drawing for cells, and thus minimizing the need for large numbers of live cells, provides an optimal way to display large data sources very quickly.

Example

The given cell model and the renderer hosts a virtual cell editor inside the grid cell and this cell type is used or activated only when you move your mouse over any grid cell. By default, a grid cell displays a text that is set in OnRender overridden method. When you move the mouse over this cell, it will become a live UIElement editor and not render the cell anymore. This cell is now a virtual cell that will display the cell value stored in the internal cell structure, say, "Edit Me".

But, when you scroll the cell outside the view port, it will switch back to a normal renderer cell. When you move the mouse over this cell again, it will display "Edit Me".

Placing a UIElement as soon as a cell becomes visible is a time consuming process, while static rendering of a text is faster. The UI element which is required to edit the cell will be placed only on demand (that is when you hover or click a cell for editing). Hence, this approach will greatly improve the scrolling speed.

This mechanism will be enabled only if you set SupportsRenderOptimization property to true in the constructor.

C#

```
public class VirtualizedCellModel : GridCellModel<VirtualizedCellRenderer>
{
}

```

```

public class VirtualizedCellRenderer : GridVirtualizingCellRenderer<TextBox>
{
    public VirtualizedCellRenderer()
    {
        SupportsRenderOptimization = true;
        AllowRecycle = true;
        IsControlTextShown = true;
        IsFocusable = true;
    }
    protected override void OnRender(DrawingContext dc, RenderCellArgs rca,
        GridRenderStyleInfo cellInfo)
    {
        if (rca.CellUIElements != null)
            return;
        // Only if SupportsRenderOptimization is true, otherwise rca.CellVisuals is
        // never null.
        string s = String.Format("Render{0}/{1}", rca.RowIndex, rca.ColumnIndex);
        GridTextBoxPaint.DrawText(dc, rca.CellRect, s, cellInfo);
    }
    public override void OnInitializeContent(TextBox textBox,
        GridRenderStyleInfo style)
    {
        base.OnInitializeContent(textBox, style);
        Thickness margins = style.TextMargins.ToThickness();
        // TextBoxView always has a minimum margin of 2 for left and right.
        // Margin is hard coded below so that text box behavior is properly
        // emulated.
        margins.Left = Math.Max(0, margins.Left - 2);
        margins.Right = Math.Max(0, margins.Right - 2);
        textBox.Padding = margins;
        textBox.BorderThickness = new Thickness(0);
        VirtualizingCellsControl.SetWantsMouseInput(textBox, true);
        textBox.Text = GetControlText(style);
    }
    protected override string GetControlTextFromEditorCore(TextBox uiElement)
    {
        return uiElement.Text;
    }
    protected override void OnInitialize()
    {
        base.OnInitialize();
        ControlText = GetControlText(CurrentStyle);
    }
    protected override void OnWireUIElement(TextBox textBox)
    {
        base.OnWireUIElement(textBox);
        textBox.TextChanged += new TextChangedEventHandler(textBox_TextChanged);
    }
    protected override void OnUnwireUIElement(TextBox textBox)
    {
        base.OnUnwireUIElement(textBox);
        textBox.TextChanged -= new TextChangedEventHandler(textBox_TextChanged);
    }
    void textBox_TextChanged(object sender, TextChangedEventArgs e)
    {
        TextBox textBox = (TextBox)sender;
        if (!this.IsInArrange && IsCurrentCell(textBox))

```

```

{
    TraceUtil.TraceCurrentMethodInfo(textBox.Text);
    if (!SetControlText(textBox.Text))
        RefreshContent(); // reverses change.
}
}

protected override void OnGridPreviewTextInput(TextCompositionEventArgs e)
{
    CurrentCell.ScrollInView();
    CurrentCell.BeginEdit(true);
}

protected override bool ShouldGridTryToHandlePreviewKeyDown(KeyEventArgs e)
{
    if (CurrentCellUIElement.IsFocused && e.Key != Key.Escape)
        return false;
    return true;
}
}

```

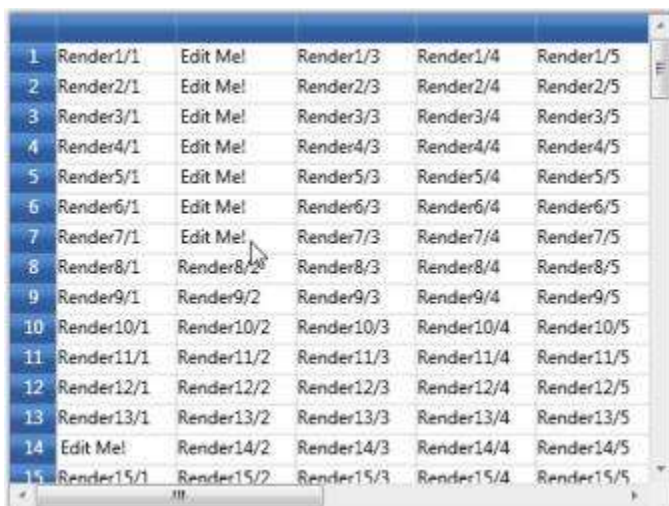
Here is the code to bind the above virtual cell to the grid:

C#

```

grid.Model.CellModels.Add("VirtualizedCell", new VirtualizedCellModel());
grid.Model.TableStyle.CellType = "VirtualizedCell";
grid.Model.TableStyle.CellValue = "Edit Me!";

```



Performance in WPF GridControl

Essential Grid is well known for its optimized performance. This section points out a sample that illustrates how to do high frequency updates in Grid control.

Sample Architecture

Inside SampleGridControl.cs file, you can define various test scenarios. The basic test will load a data table with values and then modify the records inside a timer (timer is used to keep changing the grid values at regular intervals in order to illustrate run time updates) at run time. This will trigger ListChanged event. FlatDataViewGrid control listens to these events and updates the data displayed, by highlighting the cells that were changed, and also updates the resulting summaries.

Sample Features

The features of this sample are listed below:

- This FlatDataViewGrid control is implemented using a virtual grid approach by wiring the grid to the data view using the QueryCellInfo and CommitCellInfo event handlers.
- Blinking behavior in the sample is implemented by handling OnPrepareRenderCell event using the PrepareRenderCellInfo handler.

Note: Blinking Behavior-You can see random values get updated frequently and those changed fields are highlighted by colors. For ex., increase in value is indicated in green color and decrease in value in red color. It also highlights the insertion of new records.

- The grid handles all ListChanged notifications including ItemAdded, ItemMoved, ItemDeleted, Reset, PropertyDescriptorAdded, PropertyDescriptorDeleted and PropertyDescriptorChanged.
- When the current cell is active, you can continue editing and modify the contents of the currently active cell even while rows are re-arranged. This is often a requirement in applications that wish to maintain editing compatibility even while processing a large number of updates.

	A	B	C	D	E	F	G	H	I	J	K	L
1	P00000	76	63	79	64	77	98	128	80	106	141	83
2	H00006	15	21	30	22	28	36	59	23	33	32	27
2	P00001	82	76	87	84	87	74	118	88	97	73	97
3	P00002	111	66	97	73	80	77	206	61	140	107	101
4	H00001	33	38	29	27	43	42	62	60	43	33	54
14	P00010	86	23	33	63	42	61	51	25	32	35	38
7	H00012	46	47	48	49	50	51	52	53	54	55	66
15	P00011	58	56	64	66	56	49	66	74	68	86	55
9	H00014	69	74	71	72	58	74	75	76	77	78	100
16	P00012	83	40	56	46	83	111	84	11E	75	65	61
17	H00005	46	31	37	42	35	34	36	54	66	37	59
17	P00013	37	38	60	46	47	73	35	59	85	83	49
18	R00014	128	107	101	173	57	112	120	78	206	192	76
19	P00015	1	1	4	5	8	8	10	7	14	15	15
20	P00016	87	84	95	111	101	92	93	151	116	96	118
17	P00017	104	125	118	86	63	138	118	106	92	95	205
15	H00009	10	12	11	12	13	14	15	16	14	16	19
18	P00018	73	96	107	78	91	84	91	80	98	47	188
205		12045	12993	12814	13178	12907	13468	13843	14341	14323	14285	14425

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

Example

Let us consider a sample using a FlatDataViewGrid control, which is a regular grid that is bound to a flat data view (flat table, which is not nested and without relations) and is customized to handle refresh updates (refreshing the grid values, which in turn replaces old values with new values). It has a header row with field names and a footer row with summaries.

Interactive Features in WPF GridControl

This section explains the Resizing the columns, Drag and drop the columns and cells of WPF GridControl.

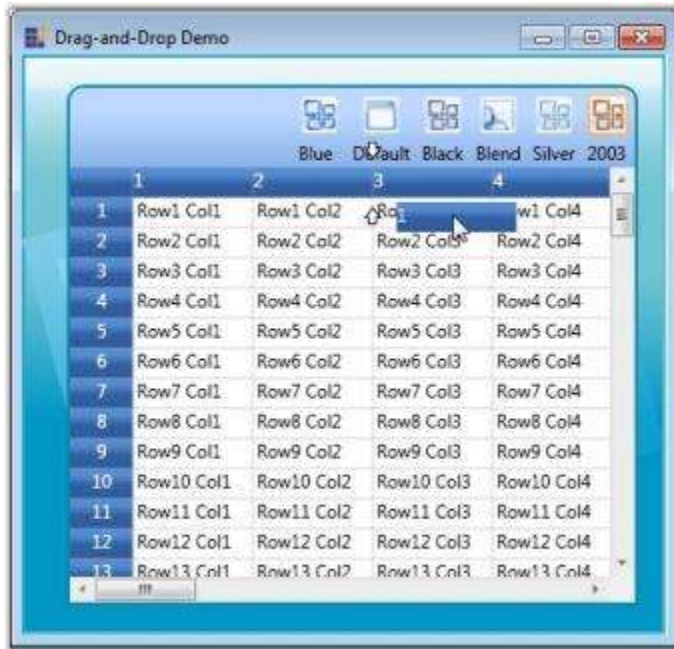
Drag-and-Drop columns

Essential Grid for WPF provides support for drag-and-drop functionality. This feature enables the user to just click a column header and drag it to a new position. It is not required to select the column header for dragging. This is an easy way of rearranging the columns dynamically. You can enable or disable this feature using AllowDragColumns property of the Grid as follows:

C#


```
//Allow column dragging
this.grid.AllowDragColumns = true;
```

The following image illustrates this feature:



In the above image, you can see the column header 1 being dragged to be placed before column 3.

Excel like - Cell drag and drop

The Excel-like Drag and Drop feature enables dragging content with their styles from cells to different locations and grids. You can use this feature to copy data to one or more locations.

C#

```
this.grid.AllowDragDrop = true;
this.grid.Model.Options.DataObjectConsumerOptions = GridDataObjectConsumerOptions.Styles;
```

Excel Like Drag and Drop has the following features:

- Uses IDataObject to copy, store, and retrieve data.
- Uses DragDrop API, which is available in WPF, to initiate drag-and-drop.

Property

Name of the Property	Description	Type of Property	Value it Accepts	PropertySyntax
AllowDragDrop	Allows dragging and dropping	Normal	Boolean	Â this.grid.AllowDragDropÂ =Â true;

	content and enables ExcelLike MouseController, which helps in the drag-and-drop operation.			
GridDataObjectConsumerOptions	Gets or sets the enum value for the DragDrop Consumer Option.	Normal	Enum	this.grid.Model.Options.DataObjectConsumerOptions = GridDataObjectConsumerOptions.Styles;
DragDropDropTargetFlags	Gets or sets the enum value for values that can be copied or moved and provides other options for DragDropTargetFlags.	Normal	Enum	this.grid.Model.Options.DragDropDropTargetFlags

Event

When is the event triggered?	How is it handled?	Method (event handler) that handles the event?	What are the event args associated?	Purpose of the Event
Occurs when the user releases the mouse over a cell at the end of an OLE drag-and-drop operation and before the data is applied to the grid.	Handled by setting the Handled flag as True.	GridOleDropAtRowColEventHandler	GridOleDropAtRowColEventArgs	This event allows you to customize the paste data behavior.
The event is initiated when the user	You can disallow the specified range to be used as the	GridExcelLikeDragRangeEventHandler	GridQueryCanDragRangeEventArgs	GridQueryCanOleDragRangeEventArgs is a custom event argument class used by the

rolls the mouse over the edge of a selected range.	OLE Data Source when assigning true to “Cancel” flag.			GridControlBase.QueryCanOLEDragRange event to determine if a specified range can serve as an OLE drag source.
This event is initiated when a user drags a range of selected cells by using the OLE drag-and-drop.	Handled by setting the Handled flag as True.	GridQueryOleDataSourceDataEventHandler	GridQueryOLEDataSourceDataEventArgs	This event allows you to provide customized clipboard formats or add support for pasting the additional clipboard content.

Excel like - Resizing

Essential Grid control supports Excel like resizing to hide or unhide columns. It also acts as a visual marker to indicate hidden columns.

Use Case Scenarios

This feature can be implemented for application which contains more rows and columns. You can also make some rows and columns to be hidden.

Adding Resizing Support to an Application

This topic explains how to implement the Resizing support to an application. The following steps explain the implementation of the Resizing support to an application.

1. Set the Resizing border properties
2. Resizing is a built-in property and there is no need to set any special property to enable it. But there are options to

set the hidden border color and thickness. Set the HiddenBorderBrush property to any color of the brush for the Model object. The assigned color will be brushed in the border color of the hidden column or row. Set the HiddenBorderThickness property to an int value say 3.

The following code snippet explains the implementation of the HiddenBorderBrush and HiddenBorderThickness properties.

C#

```
this.gridControl.Model.HiddenBorderBrush = Brushes.Red ;
this.gridControl.Model.HiddenBorderThickness = 3;
```

VB.NET

```
Me.gridControl.Model.HiddenBorderBrush = Brushes.Red
Me.gridControl.Model.HiddenBorderThickness = 3
```

Run the application

To set the rows or columns as hidden by code, you can use the SetHidden() method. It has two int type parameters to get “from Index” and “to Index”, a Boolean type which sets True for hide and False for unhide. Run the application and you will find the given rows or columns to be hidden.

The following code snippet explains the implementation of the SetHidden() method of ColumnWidths and RowHeights property.

C#

```
// To hide columns and rows.
this.gridControl.ColumnWidths.SetHidden(3, 4, true);
this.gridControl.RowHeights.SetHidden(3, 4, true);
// To unhide columns and rows.
this.gridControl.ColumnWidths.SetHidden(3, 4, false);
this.gridControl.RowHeights.SetHidden(3, 4, false);
```

VB.NET

```
// To hide columns and rows.
Me.gridControl.ColumnWidths.SetHidden(3, 4, True)
Me.gridControl.RowHeights.SetHidden(3, 4, True)
// To unhide columns and rows.
Me.gridControl.ColumnWidths.SetHidden(3, 4, False)
Me.gridControl.RowHeights.SetHidden(3, 4, False)
```

The following is a sample output of Resizing support implementation.

R0C0	R0C1	R0C2	R0C5	R0C6	
1	R1C1	R1C2	R1C5	R1C6	Hidden Columns indicated with red border
2	R2C1	R2C2	R2C5	R2C6	
5	R5C1	R5C2	R5C5	R5C6	Hidden Rows indicated with red border
6	R6C1	R6C2	R6C5	R6C6	

Hide and unhide a rows or Columns during run-time.

To hide a column or a row, hover at the line of the column or row. It shows a resizing cursor with a single bar so that you can drag the line to its next header cell. After joining to the neighbor Header Cell, the line will be darkened which means that a column or a row is hidden. To unhide the hidden rows or columns, hover on the dark marked line. The cursor will then be changed to a double bar and by double clicking, the hidden rows or columns can be resized to its original size.

The following screenshot explains how to hide and unhide a Column.

Hover over the header cell's border line. The cursor will be changed to single bar, as like in the following screenshot.

R0C0	R0C1	R0C2	R0C3	R0C4	
1	R1C1	R1C2	R1C3	R1C4	ResizeWidth Cursor
2	R2C1	R2C2	R2C3	R2C4	
3	R3C1	R3C2	R3C3	R3C4	
4	R4C1	R4C2	R4C3	R4C4	

Drag it to Column 2 so that the Border color changes as like in the following screenshot.

R0C0	R0C1	R0C3	R0C4	
1	R1C1	R1C3	R1C4	Border color changed to red after hiding the column
2	R2C1	R2C3	R2C4	
3	R3C1	R3C3	R3C4	

The following image shows the output after hiding the row by Mouse Dragging.

R0C0	R0C1	R0C3	R0C4
1	R1C1	R1C3	R1C4
2	R2C1	R2C3	R2C4
3	R3C1	R3C3	R3C4

To unhide the hidden row, hover the mouse on the hidden column border line. A double bar cursor will then be displayed as in the following screenshot.

R0C0	R0C1	R0C3	R0C4
1	R1C1	R1C3	R1C4
2	R2C1	R2C3	R2C4
3	R3C1	R3C3	R3C4

Cursor changed to double bar cues to expand the hidden Column

Double clicking on it will unhide all the hidden columns in that particular hidden column.

R0C0	R0C1	R0C2	R0C3
1	R1C1	R1C2	R1C3
2	R2C1	R2C2	R2C3
3	R3C1	R3C2	R3C3

Tables for Properties, Methods, and Events

Properties

Property	Description	Data Type	Default value	Class Name
HiddenBorderBrush	Sets the border brush color for the hidden rows or columns.	Brush	Black	GridModel
HiddenBorderThickness	Sets the border thickness for the hidden rows or columns.	int	1	GridModel

Methods

Method	Description	Parameters	Available inside the Property	Return Type
SetHidden()	Sets the specified "from" rows to "to" rows as hidden.	int from, int to, bool hide	RowHeights	void
SetHidden()	Sets the specified "from" rows to "to" rows as hidden.	int from, int to, bool hide	ColumnWidths	void

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

Events in WPF GridControl

Grid control declares a number of events that it can raise in response to an activity either by the user or by the system. An Event is a message that is triggered to notify an object or a class of the occurrence of an action. When an event is triggered, all the event handlers are notified. Following are the Grid control events:

Let us look on each event and its event handler in detail in the following topics.

QueryCellInfo and CommitCellInfo

These events are widely used to allow customization of each and every cell in the required format. QueryCellInfo accepts an argument of type GridQueryCellInfoEventArgs and CommitCellInfo accepts an argument of type GridCommitCellInfoEventArgs. The table below lists the customization properties exposed by these two event arguments.

Property	Description
Cell	Gives the cell co-ordinates.
Style	Specifies the style for the cell represented by the above Cell property.

These events are essential to operate the grid in virtual mode, where:

- QueryCellInfo is used to provide the cell values on demand and,
- Changes made in the grid will be saved back by the CommitCellInfo event.

The QueryCellInfo is used to completely customize the grid cells. The code below sets up a Virtual Grid by applying these events and also paints alternate rows using QueryCellInfo event. The QueryCellInfo event is raised for each cell that requires redrawing.

Example

These events can be triggered using the following code:

C#

```
this.grid.QueryCellInfo += new
GridQueryCellInfoEventHandler(grid_QueryCellInfo);
this.grid.CommitCellInfo += new
GridCommitCellInfoEventHandler(grid_CommitCellInfo);
```

Event Handlers

C#

```
Dictionary<RowColumnIndex, object> committedValues = new
Dictionary<RowColumnIndex, object>();
void grid_QueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    if (e.Cell.ColumnIndex > 0 && e.Cell.RowIndex > 0)
    if (e.Cell.RowIndex % 2 == 0)
        e.Style.Background = Brushes.LightGreen;
    if (e.Cell.RowIndex == 0)
    {
        if (e.Cell.ColumnIndex > 0)
            e.Style.CellValue = e.Cell.ColumnIndex;
    }
    else if (e.Cell.RowIndex > 0)
    {
        if (e.Cell.ColumnIndex == 0)
            e.Style.CellValue = e.Cell.RowIndex;
        else if (e.Cell.ColumnIndex > 0)
        {
            if (committedValues.ContainsKey(e.Cell))
```

```

e.Style.CellValue = committedValues[e.Cell];
else
e.Style.CellValue = String.Format("{0}/{1}", e.Cell.RowIndex,
e.Cell.ColumnIndex);
}
}
}
void grid_CommitCellInfo(object sender, GridCommitCellInfoEventArgs e)
{
if (e.Style.HasCellValue)
{
committedValues[e.Cell] = e.Style.CellValue;
e.Handled = true;
}
}
}

```

Output

The following output is generated using the code above.

	1	2	3	4	5
1	1/1	1/2	1/3	1/4	1/5
2	2/1	2/2	2/3	2/4	2/5
3	3/1	3/2	3/3	3/4	3/5
4	4/1	4/2	4/3	4/4	4/5
5	5/1	5/2	5/3	5/4	5/5
6	6/1	6/2	6/3	6/4	6/5
7	7/1	7/2	7/3	7/4	7/5
8	8/1	8/2	8/3	8/4	8/5
9	9/1	9/2	9/3	9/4	9/5
10	10/1	10/2	10/3	10/4	10/5
11	11/1	20/20	11/3	11/4	11/5
12	12/1	12/2	12/3	12/4	12/5
13	13/1	13/2	13/3	13/4	13/5
14	14/1	14/2	14/3	14/4	14/5
15	15/1	15/2	15/3	15/4	15/5
16	16/1	16/2	16/3	16/4	16/5
17	17/1	17/2	17/3	17/4	17/5
18	18/1	18/2	18/3	18/4	18/5

QueryBaseStyles

C#

```

void grid_QueryBaseStyles(object sender, GridQueryBaseStylesEventArgs e)
{
if (ColumnRowIndex.Text != "")
{
if (e.Cell.RowIndex == 3)
{
e.BaseStyles.Add(new GridStyleInfo() { Background = new
SolidColorBrush(Colors.Maroon) });
}
if (e.Cell.ColumnIndex == 3)
{
e.BaseStyles.Add(new GridStyleInfo() { Background = new
SolidColorBrush(Colors.Orange) });
}
}
}
}

```

```
}
}
```

This event is used to provide base styles for desired grid cells. It receives an argument of type `GridQueryBaseStylesEventArgs` that contains the following parameters.

Property	Description
BaseStyles	Holds a list of base styles applicable for current cell co-ordinates.
Cell	Represent the cell co-ordinates.
Style	Gives cell style information.

Example

This event can be triggered using the following code:

C#

```
this.grid.QueryBaseStyles += new
GridQueryBaseStylesEventHandler(grid_QueryBaseStyles);
```

Event Handler

Output

The following output is generated using the code above.



QueryCoveredRange

This event is used to define covered ranges in the required cells. It receives an argument of type `GridQueryCoveredRangeEventArgs` containing the following information about the event.

Property	Description
CellRowIndex	Represents the cell row and column indices.
Range	Defines the covered range for the cell.

Example

This event can be triggered using the following code:

C#

```
grid.QueryCoveredRange += new GridQueryBaseStylesEventArgs
(grid_QueryCoveredRange);
```

*Event Handler***C#**

```
void grid_QueryCoveredRange(object sender, GridQueryBaseStylesEventArgs e)
{
    // Combine column 2 to 4 on every 4th row.
    if (e.CellRowIndex.RowIndex % 4 == 2)
    {
        if (e.CellRowIndex.ColumnIndex >= 2 &&
            e.CellRowIndex.ColumnIndex <= 4)
        {
            e.Range = new CoveredCellInfo (e.CellRowIndex.RowIndex, 2,
            e.CellRowIndex.RowIndex, 4);
            e.Handled = true;
        }
    }
}
```

Output

The following output is generated using the code above.

	Row0 Col1	Row0 Col2	Row0 Col3	Row0 Col4	Row0 Col5	Row0 Col6
1	Row1 Col1	Row1 Col2	Row1 Col3	Row1 Col4	Row1 Col5	Row1 Col6
2	Row2 Col1	Row2 Col2	Row2 Col3	Row2 Col4	Row2 Col5	Row2 Col6
3	Row3 Col1	Row3 Col2	Row3 Col3	Row3 Col4	Row3 Col5	Row3 Col6
4	Row4 Col1	Row4 Col2	Row4 Col3	Row4 Col4	Row4 Col5	Row4 Col6
5	Row5 Col1	Row5 Col2	Row5 Col3	Row5 Col4	Row5 Col5	Row5 Col6
6	Row6 Col1	Row6 Col2	Row6 Col3	Row6 Col4	Row6 Col5	Row6 Col6
7	Row7 Col1	Row7 Col2	Row7 Col3	Row7 Col4	Row7 Col5	Row7 Col6
8	Row8 Col1	Row8 Col2	Row8 Col3	Row8 Col4	Row8 Col5	Row8 Col6
9	Row9 Col1	Row9 Col2	Row9 Col3	Row9 Col4	Row9 Col5	Row9 Col6
10	Row10 Col1	Row10 Col2	Row10 Col3	Row10 Col4	Row10 Col5	Row10 Col6
11	Row11 Col1	Row11 Col2	Row11 Col3	Row11 Col4	Row11 Col5	Row11 Col6
12	Row12 Col1	Row12 Col2	Row12 Col3	Row12 Col4	Row12 Col5	Row12 Col6
13	Row13 Col1	Row13 Col2	Row13 Col3	Row13 Col4	Row13 Col5	Row13 Col6
14	Row14 Col1	Row14 Col2	Row14 Col3	Row14 Col4	Row14 Col5	Row14 Col6

QueryCellSpanBackgrounds

This event lets you create cell spans and customize their backgrounds. It receives an argument of type `GridQueryCellSpanBackgroundsEventArgs` containing the following properties.

Property	Description
CellRowIndex	Represents the cell row and column indices.
Range	Defines the covered range for the cell.

Example

This event can be triggered using the following code:

C#

```
grid.QueryCellSpanBackgrounds += new
GridQueryCellSpanBackgroundsEventHandler (grid_QueryCellSpanBackgrounds);
```

*Event Handler***C#**

```
void grid_QueryCellSpanBackgrounds(object sender,
GridQueryCellSpanBackgroundsEventArgs e)
{
    if (e.CellRowColumnIndex.ColumnIndex == 2 && e.CellRowColumnIndex.RowIndex
    == 4)
    {
        CellSpanBackgroundInfo item = new
        CellSpanBackgroundInfo(e.CellRowColumnIndex.RowIndex,
        e.CellRowColumnIndex.ColumnIndex, 9, 4);
        item.Background = new
        ImageBrush(GetImage(@"common\Images\Grid\BannerCells\back2.jpg"));
        e.Range = new List<CellSpanBackgroundInfo>();
        e.Range.Add(item);
        e.Handled = true;
    }
}
```

Output

The following output is generated using the code above.

*ResizingRows and ResizingColumns*

These events are used to control over resizing of specific rows or columns. They are triggered when a row or column is being resized. The event handler receives an argument of type `GridResizingRowsEventArgs` or `GridResizingColumnsEventArgs` containing data related to the event. The following properties of the event arguments provide information specific to these events.

Properties of GridResizingRowsEventArgs

Property	Description
----------	-------------

AllowResize	Boolean property;Â When false, disallow the resizing action.
Rows	Used to get or set the index of range of rows being resized.
Reason	Gives a hint about user action and reason for this event.Accepts a value of type GridResizeCellsReason enumeration: <i>CancelMode</i> â€Indicates current operation was canceled. <i>DoubleClick</i> â€Indicates user double-clicked. <i>HitTest</i> â€Indicates this is a Hit-Test query. <i>MouseDown</i> â€Indicates user pressed mouse down. <i>MouseMove</i> â€Indicates user is moving the mouse. <i>MouseUp</i> â€Indicates user released the mouse. <i>ResetDefault</i> â€Indicates changed row heights will be reset back to default value. <i>ResetHide</i> â€Indicates hidden rows will be made visible.
Height	Specifies the row height.
Point	Indicates the point at which the mouse hits the row before resizing.

Properties of GridResizingColumnsEventArgs

Property	Description
AllowResize	Boolean property;Â When false, disallow the resizing action.
Columns	Used to get or set the index of range of columns being resized.
Reason	Gives a hint about user action and reason for this event.Accepts a value of type GridResizeCellsReason enumeration: <i>CancelMode</i> â€Indicates current operation was canceled. <i>DoubleClick</i> â€Indicates user double-clicked. <i>HitTest</i> â€Indicates this is a Hit-Test query. <i>MouseDown</i> â€Indicates user pressed mouse down. <i>MouseMove</i> â€Indicates user is moving the mouse. <i>MouseUp</i> â€Indicates user released the mouse. <i>ResetDefault</i> â€Indicates changed column widths will be reset back to default value. <i>ResetHide</i> â€Indicates hidden columns will be made visible.
Width	Specifies the column width.
Point	Indicates the point at which the mouse hits the column before resizing.

Example

These events can be invoked as follows:

C#

```
grid.ResizingRows += new GridResizingRowsEventHandler(grid_ResizingRows);
grid.ResizingColumns += new
GridResizingColumnsEventHandler(grid_ResizingColumns);
```

Event Handlers:

C#

```
// Disallow resizing of row 2.
void grid_ResizingRows(object sender, GridResizingRowsEventArgs args)
{
```

```

if (args.Rows.Top == 2)
args.AllowResize = false;
}
// Disallow resizing of column 3.
void grid_ResizingColumns(object sender, GridResizingColumnsEventArgs args)
{
if (args.Columns.Left == 3)
args.AllowResize = false;
}

```

GridResizingRowsEventArgs

The following table provides information on the properties of the event:

Property	Description
AllowResize	Boolean property; When false, disallow the resizing action.
Rows	Used to get or set the index of range of rows being resized.
Reason	Gives a hint about user action and reason for this event. Accepts a value of type GridResizeCellsReason enumeration: CancelMode â€œ Indicates current operation was canceled. DoubleClick â€œ Indicates user double-clicked. HitTest â€œ Indicates this is a Hit-Test query. MouseDown â€œ Indicates user pressed mouse down. MouseMove â€œ Indicates user is moving the mouse. MouseUp â€œ Indicates user released the mouse. ResetDefault â€œ Indicates changed row heights will be reset back to default value. ResetHide â€œ Indicates hidden rows will be made visible.
Height	Specifies the row height.
Point	Indicates the point at which the mouse hits the row before resizing.

GridResizingColumnsEventArgs

The following table provides information on the properties of the event:

Property	Description
AllowResize	Boolean property; When false, disallow the resizing action.
Columns	Used to get or set the index of range of columns being resized.
Reason	Gives a hint about user action and reason for this event. Accepts a value of type GridResizeCellsReason enumeration: CancelMode â€œ Indicates current operation was canceled. DoubleClick â€œ Indicates user double-clicked. HitTest â€œ Indicates this is a Hit-Test query. MouseDown â€œ Indicates user pressed mouse down. MouseMove â€œ Indicates user is moving the mouse. MouseUp â€œ Indicates user released the mouse. ResetDefault â€œ Indicates changed column widths will be reset back to default value. ResetHide â€œ Indicates hidden columns will be made visible.
Width	Specifies the column width.
Point	Indicates the point at which the mouse hits the column before resizing.

Example

This event can be triggered using the following code:

C#

```
grid.ResizingRows += new GridResizingRowsEventHandler(grid_ResizingRows);
grid.ResizingColumns += new GridResizingColumnsEventHandler
(grid_ResizingColumns);
```

*Event Handlers***C#**

```
//Disallow resizing of row 2.
void grid_ResizingRows(object sender, GridResizingRowsEventArgs args)
{
    if (args.Rows.Top == 2)
        args.AllowResize = false;
}

//Disallow resizing of column 3.
void grid_ResizingColumns(object sender, GridResizingColumnsEventArgs args)
{
    if (args.Columns.Left == 3)
        args.AllowResize = false;
}
```

RowsInserted and ColumnsInserted

These events are triggered when one or more rows or columns are inserted. The event handler receives an argument of type `GridRangeInsertedEventArgs` containing data related to this event. The following `GridRangeInsertedEventArgs` properties provide information specific to these events.

Property	Description
Count	The number of rows or columns.
InsertAt	The row or column index where the cells should be inserted before.

Example

This event can be triggered using the following code:

C#

```
grid.Model.RowsInserted += new GridRangeInsertedEventHandler
(Model_RowsInserted);
grid.Model.ColumnsInserted += new GridRangeInsertedEventHandler
(Model_ColumnsInserted);
```

*Event Handlers***C#**

```
void Model_ColumnsInserted(object sender, GridRangeInsertedEventArgs e)
{
    Console.WriteLine(e.Count + " columns are inserted at " + e.InsertAt);
}
```

```
void Model_RowsInserted(object sender, GridRangeInsertedEventArgs e)
{
    Console.WriteLine(e.Count + " rows are inserted at " + e.InsertAt);
}
```

RowsMoved and ColumnsMoved

These events are raised when a range of rows or columns are moved from one position to another. Their event handlers receive an argument of type GridRangeMovedEventArgs containing data related to these events. Following are the event argument properties information about the rows or columns migration.

Property	Description
Count	The index of the last row or column that was removed.
InsertAt	The row or column index where the cells should be inserted before.
RemoveAt	The index of the first row or column that was removed.

Example

This event can be triggered using the following code:

C#

```
grid.Model.RowsMoved += new GridRangeMovedEventHandler (Model_RowsMoved);
grid.Model.ColumnsMoved += new GridRangeMovedEventHandler
(Model_ColumnsMoved);
```

Event Handlers

C#

```
void Model_RowsMoved(object sender, GridRangeMovedEventArgs e)
{
    Console.WriteLine(e.RemoveAt + 1 - e.Count + " row(s) are moved from
position " + e.RemoveAt + " to position " + e.InsertAt);
}
void Model_ColumnsMoved(object sender, GridRangeMovedEventArgs e)
{
    Console.WriteLine(e.RemoveAt+ 1 - e.Count + " column(s) are moved from
position " + e.RemoveAt + " to position " + e.InsertAt);
}
```

RowsRemoved and ColumnsRemoved

These events are triggered when a range of rows or columns are removed from the grid. Their event handlers receive an argument of type GridRangeRemovedEventArgs containing data related to these events. Following are the event argument properties that provides information about the rows or columns removal.

Property	Description
Count	The index of the last row or column that was removed.
RemoveAt	The index of the first row or column that was removed.

Example

This event can be triggered using the following code:

C#

```
grid.Model.RowsRemoved += new GridRangeRemovedEventHandler
(Model_RowsRemoved);
grid.Model.ColumnsRemoved += new GridRangeRemovedEventHandler
(Model_ColumnsRemoved);
```

*Event Handlers***C#**

```
void Model_RowsRemoved(object sender, GridRangeRemovedEventArgs e)
{
    Console.WriteLine(e.RemoveAt + 1 - e.Count + " row(s) are removed from
position " + e.RemoveAt);
}
void Model_ColumnsRemoved(object sender, GridRangeRemovedEventArgs e)
{
    Console.WriteLine(e.RemoveAt + 1 - e.Count + " column(s) are removed from
position " + e.RemoveAt);
}
```

ClipboardCanCopy

This event is triggered when some grid data is about to be copied to the clipboard. Inside this event handler, you can check for the data and range of cells that are going to be copied, and cancel the operation if you don't want to copy those data.

1. The grid cell data and range of cells that is going to be moved to the clipboard can be accessed by DataObject and RangeList properties (refer to the Table below).
2. If you don't want to move the data to the clipboard, you can cancel the operation by setting e.Cancel to true.

It receives an argument of type GridCutCopyPasteEventArgs containing data related to this event. The following are the event argument properties.

Property	Description
DataObject	Data to be copied.
RangeList	List of cell ranges that are selected for copying.
Handled	When true, indicates that the event has been handled and no further processing of the event should happen.

Example

This event can be triggered using the following code:

C#

```
gridControl.Model.ClipboardCanCopy += new
GridCutPasteEventHandler(Model.ClipboardCanCopy);
```

Event Handler

The following event handler prevents the data in row 2 from getting copied.

C#

```
void Model_ClipboardCanCopy(object sender, GridCutPasteEventArgs e)
{
    if (e.RangeList.Contains(GridRangeInfo.Row(2)))
    {
        e.Handled = true;
    }
}
```

ClipboardCanCut

This event is triggered when some grid data is about to be moved to the clipboard. Inside this event handler, you can check for the data and range of cells going to be moved and cancel the operation if you do not want to move the data. It receives an argument of type GridCutCopyPasteEventArgs containing data related to this event. The following are the event argument properties.

Property	Description
DataObject	Data to be moved.
RangeList	List of cell ranges that are selected for moving.
Handled	When true, indicates that the event has been handled and no further processing of the event should happen.

Example

This event can be triggered using the following code:

C#

```
gridControl.Model.ClipboardCanCut += new
GridCutPasteEventHandler(Model.ClipboardCanCut);
```

Event Handler

The following event handler prevents the data in row 2 from getting cut.

C#

```
void Model_ClipboardCanCut(object sender, GridCutPasteEventArgs e)
{
    if (e.RangeList.Contains(GridRangeInfo.Row(2)))
    {
        e.Handled = true;
    }
}
```

ClipboardCanPaste

This event gets fired when some grid data is about to be pasted from the clipboard. Inside this event handler, you can check for the data and range of cells going to be pasted and cancel the operation if you don't want to paste the data. It receives an argument of type `GridCutCopyPasteEventArgs` containing data related to this event. The following are the event argument properties.

Property	Description
DataObject	Data to be pasted.
RangeList	List of cell ranges that are selected for pasting.
Handled	When true, indicates that the event has been handled and no further processing of the event should happen.

Example

This event can be triggered using the following code:

C#

```
gridControl.Model.ClipboardCanPaste += new
GridCutPasteEventHandler(Model_ClipboardCanPaste);
```

Event Handler

The following event handler prevents the data in row 2 from getting pasted.

C#

```
void Model_ClipboardCanPaste(object sender, GridCutPasteEventArgs e)
{
    if (e.RangeList.Contains(GridRangeInfo.Row(2)))
    {
        e.Handled = true;
    }
}
```

ClipboardCopy

This event gets fired when some grid data is being copied to the clipboard. Inside this event handler, you can check for the data and range of cells being copied and cancel the operation if you don't want to copy the data. You can also provide custom formatted data for copying to clipboard. It receives an argument of type `GridCutCopyPasteEventArgs` containing data related to this event. The following are the event argument properties.

Property	Description
DataObject	Data being copied.
RangeList	List of cell ranges that are selected for copying.
Handled	When true, indicates that the event has been handled and no further processing of the event should happen.

Example

This event can be triggered using the following code:

C#

```
gridControl.Model.ClipboardCopy += new
GridCutPasteEventHandler(Model_ClackbarCopy);
```

Event Handler

The following event handler sets up new data for clipboard copy.

C#

```
void Model_ClackbarCopy(object sender, GridCutPasteEventArgs e)
{
    if (e.RangeList.Contains(GridRangeInfo.Row(2)))
    {
        string newData = "Data for Row2";
        e.DataObject = new DataObject(newData);
        e.Handled = true;
    }
}
```

ClipboardCut

This event gets fired when some grid data is being moved to the clipboard. Inside this event handler, you can check for the data and range of cells being moved and cancel the operation if you don't want to move the data. You can also provide custom formatted data for moving to clipboard. It receives an argument of type GridCutCopyPasteEventArgs containing data related to this event. The following are the event argument properties.

Property	Description
DataObject	Data being moved.
RangeList	List of cell ranges that are selected for transfer.
Handled	When true, indicates that the event has been handled and no further processing of the event should happen.

Example

This event can be triggered using the following code:

C#

```
gridControl.Model.ClipboardCut += new
GridCutPasteEventHandler(Model_ClackbarCut);
```

Event Handler

The following event handler sets up new data for clipboard cut operation.

C#

```
void Model_ClackbarCut(object sender, GridCutPasteEventArgs e)
```

```

{
    if (e.RangeList.Contains(GridRangeInfo.Row(2)))
    {
        string newData = "Data for Row2";
        e.DataObject = new DataObject(newData);
        e.Handled = true;
    }
}

```

ClipboardPaste

This event gets fired when some grid data is being pasted from the clipboard. Inside this event handler, you can check for the data and range of cells being pasted and cancel the operation if you don't want to paste the data. You can also provide custom formatted data for saving into grid cells. It receives an argument of type GridCutCopyPasteEventArgs containing data related to this event. The following are the event argument properties.

Property	Description
DataObject	Data being pasted.
RangeList	List of cell ranges that are selected for pasting.
Handled	When true, indicates that the event has been handled and no further processing of the event should happen.

Example

This event can be triggered using the following code:

C#

```

gridControl.Model.ClipboardPaste += new
GridCutPasteEventHandler(Model_ClipboardPaste);

```

Event Handler

The following event handler sets up new data for clipboard paste.

C#

```

void Model_ClipboardPaste(object sender, GridCutPasteEventArgs e)
{
    if (e.RangeList.Contains(GridRangeInfo.Row(2)))
    {
        string newData = "Data for Row2";
        e.DataObject = new DataObject(newData);
        e.Handled = true;
    }
}

```

CellButtonClick

This event is triggered when a cell button is clicked. It receives an argument of type GridCellButtonClickEventArgs, which helps display the row and column indices of the cell whose button

is clicked. For example: If the cell button clicked is placed in the second row and second column, the display message will be- "Button clicked at cell [2,2]".

Example

This event can be triggered using the following code:

C#

```
grid.CellButtonClick += new
GridCellButtonClickEventHandler(grid_CellButtonClick);
```

Event Handler

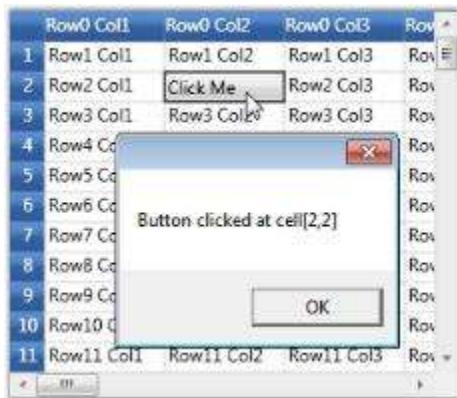
The following event handler sets up new data for clipboard paste.

C#

```
void grid_CellButtonClick(object sender, GridCellButtonClickEventArgs e)
{
    MessageBox.Show("Button clicked at cell[" + e.RowIndex + "," + e.ColumnIndex
+ "]" );
}
```

Output

The following output is generated using the code above.



CellClick

This event is triggered when a cell is clicked. It receives an argument of type GridCellClickEventArgs which helps display the row and column indices of the cell that is clicked with its click count. For example: If the cell clicked is placed in the third row and second column and clicked once, the display message will be- "Cell [3,2] is clicked 1 times".

Example

This event can be triggered using the following code:

```
{%tabs %}
```

C#

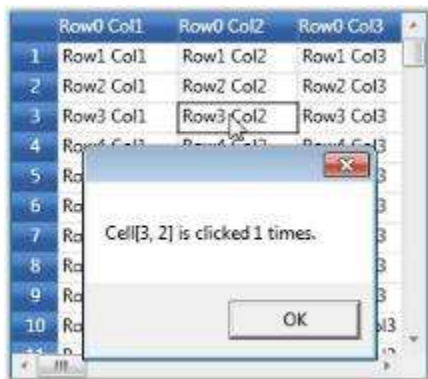
```
grid.CellClick += new GridCellClickEventHandler(grid_CellClick);
```

*Event Handler***C#**

```
void grid_CellClick(object sender, GridCellEventArgs e)
{
    MessageBox.Show("Cell[" + e.RowIndex + ", " + e.ColumnIndex + "] is clicked "
        + e.ClickCount + " times.");
}
```

Output

The following output is generated using the code above.



Cell Mouse Events

The following are the cell mouse events:

- CellMouseDown—Occurs when a mouse button is pressed in a grid cell with the click count.
- CellMouseUp—Occurs when a mouse button is released in a grid cell with the click count.
- CellMouseHover – Occurs when the mouse is hovered over a grid cell.
- CellMouseMove – Occurs when the mouse is moved around the grid cell.

These events receive an argument of type `GridCellMouseEventArgs` that provides information related to mouse events including the click position.

Example

These events can be triggered using the following code:

C#

```
grid.CellMouseDown += new
    GridCellMouseEventArgsHandler(grid_CellMouseDown);
grid.CellMouseHover += new
    GridCellMouseEventArgsHandler(grid_CellMouseHover);
grid.CellMouseMove += new
    GridCellMouseEventArgsHandler(grid_CellMouseMove);
grid.CellMouseUp += new
    GridCellMouseEventArgsHandler(grid_CellMouseUp);
```

*Event Handlers***C#**

```
void grid_CellMouseUp(object sender, GridCellMouseEventArgs args)
```

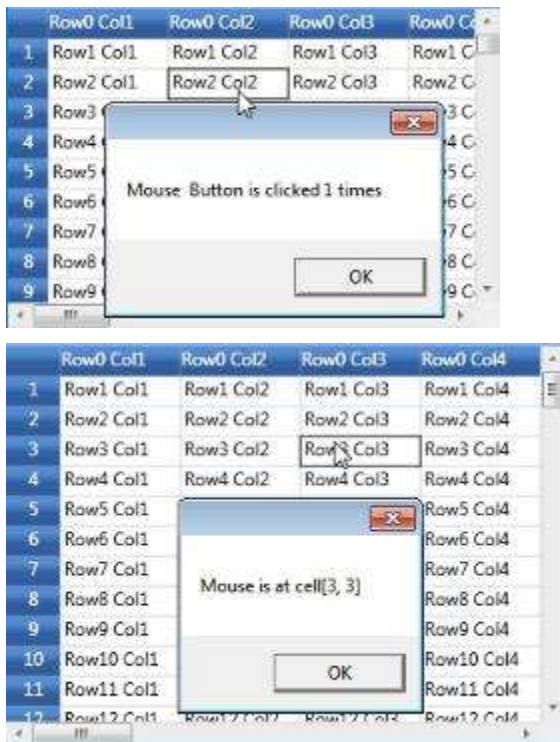
```

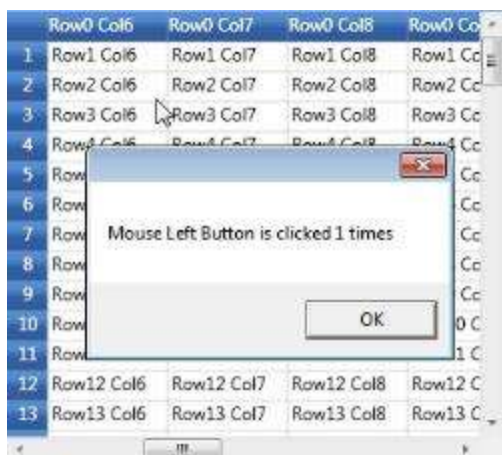
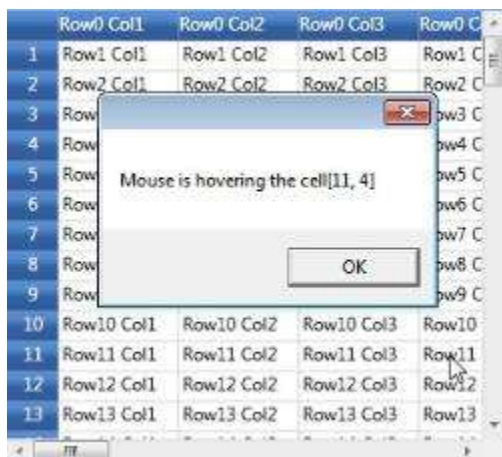
{
    MessageBox.Show("Mouse " + args.MouseControllerEventArgs.Button + " Button
    is clicked " + args.MouseControllerEventArgs.ClickCount + " times");
}
void grid_CellMouseMove(object sender, GridCellMouseControllerEventArgs
args)
{
    RowColumnIndex cell =
    grid.PointToCellRowIndex(args.MouseControllerEventArgs.Location);
    MessageBox.Show("Mouse is at cell[" + cell.RowIndex + ", " +
    cell.ColumnIndex + "]"");
}
void grid_CellMouseHover(object sender, GridCellMouseControllerEventArgs
args)
{
    RowColumnIndex cell =
    grid.PointToCellRowIndex(args.MouseControllerEventArgs.Location);
    MessageBox.Show("Mouse is hovering the cell["+cell.RowIndex+",
    "+cell.ColumnIndex+"]");
}
void grid_CellMouseDown(object sender, GridCellMouseControllerEventArgs
args)
{
    MessageBox.Show("Mouse " + args.MouseControllerEventArgs.Button + " Button
    is clicked " + args.MouseControllerEventArgs.ClickCount + " times");
}
}

```

Output

The following outputs are generated using the code above.





CurrentCellActivating

When you click a grid cell at run time, it becomes the CurrentCell (activated/designated as CurrentCell). This event is fired while activating this cell. It occurs before the grid activates the specified cell as current cell. It receives an argument of type GridCurrentCellActivatingEventArgs that let you specify – ActivateCurrentCellOptions for the given cell.

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellActivating += new
GridCurrentCellActivatingEventHandler(grid_CurrentCellActivating);
```

Event Handler

C#

```
void grid_CurrentCellActivating(object sender,
GridCurrentCellActivatingEventArgs args)
{
    args.ActivateOptions.SetCurrentCellOptions =
    GridSetCurrentCellOptions.ScrollInView;
}
```

CurrentCellActivated

It occurs after the grid activates the specified cell as current cell. It receives an argument of type `SyncfusionRoutedEventArgs` that provides the cell co-ordinates, hence the location of the cell.

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellActivated += new  
GridRoutedEventHandler(grid_CurrentCellActivated);
```

Event Handler

C#

```
void grid_CurrentCellActivated(object sender, SyncfusionRoutedEventArgs  
args)  
{  
    MessageBox.Show("CurrentCell is " + grid.CurrentCell.RowIndex + ", " +  
    grid.CurrentCell.ColumnIndex);  
}
```

CurrentCellDeactivating

It occurs before the grid deactivates the specified cell as current cell. It receives an argument of type `SyncfusionCancelRoutedEventArgs` that let you cancel this event. When you click a second cell, it first deactivates the first(current) cell and then designates the second cell as current cell.

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellDeactivating += new  
GridCancelRoutedEventHandler(grid_CurrentCellDeactivating);
```

Event Handler

C#

```
void grid_CurrentCellDeactivating(object sender,  
SyncfusionCancelRoutedEventArgs args)  
{  
    args.Cancel = true;  
}
```

CurrentCellDeactivated

It occurs after the grid activates the specified cell as current cell. It receives an argument of type `GridCurrentCellDeactivatedEventArgs` that gives the cell co-ordinates.

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellDeactivated += new  
GridCurrentCellDeactivatedEventHandler(grid_CurrentCellDeactivated);
```

Event Handler

C#

```
void grid_CurrentCellDeactivated(object sender,  
GridCurrentCellDeactivatedEventArgs args)  
{  
    MessageBox.Show("Cell deactivated:" + args.CellRowColumnIndex.RowIndex + ",  
" + args.CellRowColumnIndex.ColumnIndex);  
}
```

CurrentCellStartEditing

It occurs before the current cell switches into editing mode (when the cell is double-clicked). It receives an argument of type SyncfusionCancelRoutedEventArgs that provides an option to cancel this event.

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellStartEditing += new  
GridCancelRoutedEventArgsHandler(grid_CurrentCellStartEditing);
```

Event Handler

C#

```
void grid_CurrentCellStartEditing(object sender,  
SyncfusionCancelRoutedEventArgs args)  
{  
    args.Cancel = true;  
}
```

CurrentCellEditingComplete

It occurs when the grid completes the editing mode for active current cell. [After editing the cell, when you click to next cell or when you click any other form control or when you press escape, the cell editing mode gets stopped. This event is fired at that time.]

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellEditingComplete += new  
GridRoutedEventArgsHandler(grid_CurrentCellEditingComplete);
```

Event Handler

C#

```
void grid_CurrentCellEditingComplete(object sender,  
SyncfusionRoutedEventArgs args)  
{
```



```
Console.WriteLine(grid.CurrentCell.ToString());  
}
```

CurrentCellValidating

It occurs when the grid validates the contents of active current cell. It receives an argument of type `SyncfusionCancelRoutedEventArgs` that provides an option to cancel this event. [After editing completes, the text you entered will be checked for validity before getting applied to the cell. You can place your validation code here and if the text is invalid, you can cancel the operation by setting `args.Cancel` to `true`, which in turn will ignore the new text and will keep the old text. This event is fired during the validation operation.

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellValidating += new  
GridCancelRoutedEventHandler(grid_CurrentCellValidating);
```

Event Handler

C#

```
void grid_CurrentCellValidating(object sender,  
SyncfusionCancelRoutedEventArgs args)  
{  
    args.Cancel = true;  
}
```

CurrentCellValidated

It occurs when the grid has successfully validated the contents of active current cell.

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellValidated += new  
GridRoutedEventHandler(grid_CurrentCellValidated);
```

Event Handler

C#

```
void grid_CurrentCellValidated(object sender, SyncfusionRoutedEventArgs  
args)  
{  
    Console.WriteLine(grid.CurrentCell.ToString());  
}
```

CurrentCellChanging

It occurs when the user wants to modify the contents of current cell. It receives an argument of type `SyncfusionCancelRoutedEventArgs` that provides an option to cancel this event.

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellChanging += new
GridCancelRoutedEventHandler(grid_CurrentCellChanging);
```

*Event Handler***C#**

```
void grid_CurrentCellChanging(object sender, SyncfusionCancelRoutedEventArgs
args)
{
    args.Cancel = true;
}
```

CurrentCellChanged

It occurs when the user changes the contents of active current cell.

Example

This event can be triggered using the following code:

C#

```
grid.CurrentCellChanged += new
GridRoutedEventHandler(grid_CurrentCellChanged);
```

*Event Handler***C#**

```
void grid_CurrentCellChanged(object sender, SyncfusionRoutedEventArgs args)
{
    Console.WriteLine(grid.CurrentCell.ToString());
}
```

ToolTip in WPF GridControl

ToolTip can be added to individual cells, rows and columns to show more information about the particular cell on mouse hover. ToolTip services can be enabled by setting [GridTooltipService.SetShowTooltips](#) attached property to true. ToolTip for particular cell or row or column can be enabled by setting the [ShowToolTip](#) property of [GridStyleInfo](#).

ToolTip for specific cell

ToolTip can be displayed for any cell by setting [ShowToolTip](#) and ToolTip text can be customized by setting [ToolTip](#) property.

XML

```
//To enable tooltip for GridControl
<syncfusion:GridControl Name="grid"
syncfusion:GridTooltipService.ShowTooltips="True" />
```

C#

```
//To enable tooltip for GridControl.
GridToolTipService.SetShowTooltips(gridcontrol, true);
//Set tooltip for particular cell.
gridcontrol.Model[1, 1].ToolTip = " Grid (" + gridcontrol.Model[1,
1].CellValue + ") ";
gridcontrol.Model[1, 1].ShowTooltip = true;
```

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9
10,0	10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10,9
11,0	11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	11,9
12,0	12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	12,9
13,0	13,1	13,2	13,3	13,4	13,5	13,6	13,7	13,8	13,9
14,0	14,1	14,2	14,3	14,4	14,5	14,6	14,7	14,8	14,9
15,0	15,1	15,2	15,3	15,4	15,5	15,6	15,7	15,8	15,9
16,0	16,1	16,2	16,3	16,4	16,5	16,6	16,7	16,8	16,9
17,0	17,1	17,2	17,3	17,4	17,5	17,6	17,7	17,8	17,9
18,0	18,1	18,2	18,3	18,4	18,5	18,6	18,7	18,8	18,9
19,0	19,1	19,2	19,3	19,4	19,5	19,6	19,7	19,8	19,9

ToolTip for row and column

ToolTip can be displayed for any row or column by setting the [ShowTooltip](#) and ToolTip text can be customized by setting the [ToolTip](#).

C#

```
//Adding tooltip to the specific row
gridcontrol.Model.RowStyles[1].ToolTip = "First row";
gridcontrol.Model.RowStyles[1].ShowTooltip = true;
//Adding tooltip to the specific column
gridcontrol.Model.ColStyles[1].ToolTip = "First column";
gridcontrol.Model.ColStyles[1].ShowTooltip = true;
```

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9
10,0	10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10,9
11,0	11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	11,9
12,0	12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	12,9
13,0	13,1	13,2	13,3	13,4	13,5	13,6	13,7	13,8	13,9
14,0	14,1	14,2	14,3	14,4	14,5	14,6	14,7	14,8	14,9
15,0	15,1	15,2	15,3	15,4	15,5	15,6	15,7	15,8	15,9
16,0	16,1	16,2	16,3	16,4	16,5	16,6	16,7	16,8	16,9
17,0	17,1	17,2	17,3	17,4	17,5	17,6	17,7	17,8	17,9
18,0	18,1	18,2	18,3	18,4	18,5	18,6	18,7	18,8	18,9
19,0	19,1	19,2	19,3	19,4	19,5	19,6	19,7	19,8	19,9

Note: [View sample in GitHub](#)

Set ToolTip in QueryCellInfo event

You can set the ToolTip to a specific cell or row or column by using the [QueryCellInfo](#) event.

C#

```
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    e.Style.ShowTooltip = true;
    //Show tooltip for specific index
    if (e.Cell.RowIndex == 1 && e.Cell.ColumnIndex == 1)
    e.Style.ToolTip = " Grid (" + gridcontrol.Model[1, 1].CellValue +") ";
    // Show tooltip for row.
    if (e.Cell.ColumnIndex > 0 && e.Cell.RowIndex == 5)
    e.Style.ToolTip = " Row " + "(" + e.Cell.RowIndex + "," + e.Cell.ColumnIndex
    + ") ";
    // Show tooltip for column.
    if (e.Cell.RowIndex > 0 && e.Cell.ColumnIndex == 4)
    e.Style.ToolTip = " Column " + "(" + e.Cell.RowIndex + "," +
    e.Cell.ColumnIndex + ") ";
}
```

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9
10,0	10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10,9
11,0	11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	11,9
12,0	12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	12,9
13,0	13,1	13,2	13,3	13,4	13,5	13,6	13,7	13,8	13,9
14,0	14,1	14,2	14,3	14,4	14,5	14,6	14,7	14,8	14,9
15,0	15,1	15,2	15,3	15,4	15,5	15,6	15,7	15,8	15,9
16,0	16,1	16,2	16,3	16,4	16,5	16,6	16,7	16,8	16,9
17,0	17,1	17,2	17,3	17,4	17,5	17,6	17,7	17,8	17,9
18,0	18,1	18,2	18,3	18,4	18,5	18,6	18,7	18,8	18,9
19,0	19,1	19,2	19,3	19,4	19,5	19,6	19,7	19,8	19,9

Note: [View sample in GitHub](#)

Show or hide the ToolTip

You can show or hide the ToolTip in a specific cell or row or column by setting the [ShowToolTip](#) property to **false**.

C#

```
gridcontrol.Model[1, 1].ShowToolTip = false;
```

Setting ToolTip delay

The [SetToolTipDelay](#) method allows you to increase or decrease the time that the ToolTip waits before displaying the ToolTip.

C#

```
GridToolTipService.SetToolTipDelay(gridcontrol, 5000);
```

Handling ToolTip opening event

The [CellToolTipOpening](#) event will be triggered when the mouse hover on a cell has valid the ToolTip text.

C#

```
gridcontrol.Model[1, 1].ShowToolTip = true;
gridcontrol.Model[1, 1].ToolTip = " Grid (" + gridcontrol.Model[1, 1].CellValue + ") ";
//CellToolTipOpening event
gridcontrol.CellToolTipOpening += Gridcontrol_CellToolTipOpening;
private void Gridcontrol_CellToolTipOpening(object sender,
GridCellToolTipOpeningEventArgs e)
{
    var grids = sender as GridControl;
    if (e.Cell.RowIndex == 1 && e.Cell.ColumnIndex == 1)
```

```
grids.Model[e.Cell.RowIndex, e.Cell.ColumnIndex].ToolTip = "Hello";
}
```

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9
10,0	10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10,9
11,0	11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	11,9
12,0	12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	12,9
13,0	13,1	13,2	13,3	13,4	13,5	13,6	13,7	13,8	13,9
14,0	14,1	14,2	14,3	14,4	14,5	14,6	14,7	14,8	14,9
15,0	15,1	15,2	15,3	15,4	15,5	15,6	15,7	15,8	15,9
16,0	16,1	16,2	16,3	16,4	16,5	16,6	16,7	16,8	16,9
17,0	17,1	17,2	17,3	17,4	17,5	17,6	17,7	17,8	17,9
18,0	18,1	18,2	18,3	18,4	18,5	18,6	18,7	18,8	18,9
19,0	19,1	19,2	19,3	19,4	19,5	19,6	19,7	19,8	19,9

Hide ToolTip for disabled cell

You can disable the cell by setting **Enabled** property to **false**. If you want to hide the tooltip for this disabled cell, you need to set the **ShowToolTip** property to **false**.

C#

```
gridcontrol.Model[1, 1].Enabled = false;
gridcontrol.Model[1, 1].ToolTip = " Grid (" + gridcontrol.Model[1,
1].CellValue + ") ";
gridcontrol.Model[1, 1].ShowTooltip = false;
//Using QueryCellInfo
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    if (e.Cell.RowIndex == 1 && e.Cell.ColumnIndex == 1)
    {
        e.Style.Enabled = false;
        e.Style.ToolTip = " Grid (" + e.Cell.RowIndex + "," + e.Cell.ColumnIndex +
        ") ";
        e.Style.ShowTooltip = false;
    }
}
```

Note: [View sample in GitHub](#)

Identify whether cell has ToolTip

The **HasToolTip** property is used to identify whether a cell has ToolTip or not in a cell or row or column in GridControl. You can also highlight the row or column or cell applied to the ToolTip.

C#

```

gridcontrol.QueryCellInfo += Gridcontrol_QueryCellInfo;
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    if (e.Cell.ColumnIndex > 0 && e.Cell.RowIndex == 5)
    e.Style.ToolTip = " Row " + "(" + e.Cell.RowIndex + ", " + e.Cell.ColumnIndex
    + ") ";
    if(e.Style.HasToolTip)
    {
        e.Style.Background = Brushes.Green;
        e.Style.Foreground = Brushes.White;
    }
}

```

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9
10,0	10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10,9
11,0	11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	11,9
12,0	12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	12,9
13,0	13,1	13,2	13,3	13,4	13,5	13,6	13,7	13,8	13,9
14,0	14,1	14,2	14,3	14,4	14,5	14,6	14,7	14,8	14,9
15,0	15,1	15,2	15,3	15,4	15,5	15,6	15,7	15,8	15,9
16,0	16,1	16,2	16,3	16,4	16,5	16,6	16,7	16,8	16,9
17,0	17,1	17,2	17,3	17,4	17,5	17,6	17,7	17,8	17,9
18,0	18,1	18,2	18,3	18,4	18,5	18,6	18,7	18,8	18,9
19,0	19,1	19,2	19,3	19,4	19,5	19,6	19,7	19,8	19,9

Customize the ToolTip

The tooltip appearance can be customized by defining DataTemplate. The DataTemplate can be assigned to the [GridStyleInfo.ToolTipTemplateKey](#) or [GridStyleInfo.ToolTipTemplate](#) property. If you are using tooltipTemplate1 then you need to assign template to its corresponding template key property namely [GridStyleInfo.ToolTipTemplate](#) or [GridStyleInfo.ToolTipTemplateKey](#).

[GridStyleInfo](#) which holds cell information is the [DataContext](#) for data template of ToolTip.

Using ToolTipTemplateKey

XML

```

<Window.Resources>
<DataTemplate x:Key="tooltipTemplate1">
<Border Name="Border"
Background="Green"
BorderBrush="Black"
BorderThickness="1" Width="60" Height="20"
CornerRadius="0">
<TextBlock Background="Transparent" Text="{Binding Path=ToolTip}"
Padding="2" />

```

```

</Border>
</DataTemplate>
</Window.Resources>

```

C#

```

//Set the template key to a particular index
gridcontrol.Model[1, 1].ToolTipTemplateKey = "tooltipTemplate1";
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    if (e.Cell.RowIndex == 1 && e.Cell.ColumnIndex == 1)
    {
        e.Style.ToolTip = " Grid (" + e.Cell.RowIndex + "," + e.Cell.ColumnIndex +
        ") ";
        e.Style.ToolTipTemplateKey = "tooltipTemplate1";
    }
}

```

Using ToolTipTemplate**XML**

```

<Window.Resources>
<DataTemplate x:Key="tooltipTemplate1">
<Border Name="Border"
Background="Green"
BorderBrush="Black"
BorderThickness="1" Width="60" Height="20"
CornerRadius="0">
<TextBlock Background="Transparent" Text="{Binding Path=ToolTip}"
Padding="2" />
</Border>
</DataTemplate>
</Window.Resources>

```

C#

```

//Set the template key to a particular index
gridcontrol.Model[1, 1].ToolTipTemplate =
(DataTemplate) this.Resources["tooltipTemplate1"];
//Using QueryCellInfo event
private void Gridcontrol_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    if (e.Cell.RowIndex == 1 && e.Cell.ColumnIndex == 1)
    {
        e.Style.ToolTip = " Grid (" + e.Cell.RowIndex + "," + e.Cell.ColumnIndex +
        ") ";
        e.Style.ToolTipTemplate = (DataTemplate) this.Resources["tooltipTemplate1"];
    }
}

```


0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9
10,0	10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10,9
11,0	11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	11,9
12,0	12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	12,9
13,0	13,1	13,2	13,3	13,4	13,5	13,6	13,7	13,8	13,9
14,0	14,1	14,2	14,3	14,4	14,5	14,6	14,7	14,8	14,9
15,0	15,1	15,2	15,3	15,4	15,5	15,6	15,7	15,8	15,9
16,0	16,1	16,2	16,3	16,4	16,5	16,6	16,7	16,8	16,9
17,0	17,1	17,2	17,3	17,4	17,5	17,6	17,7	17,8	17,9
18,0	18,1	18,2	18,3	18,4	18,5	18,6	18,7	18,8	18,9
19,0	19,1	19,2	19,3	19,4	19,5	19,6	19,7	19,8	19,9

Note: [View sample in GitHub](#)

Remove the ToolTip

The `ResetValue` method is used to remove the ToolTip for any cell or row or column in GridControl and to reset the ToolTip value to the default values.

C#

```
gridcontrol.Model[1, 1].ResetValue(GridStyleInfoStore.ToolTipProperty);
```

Comment Tip in WPF GridControl

Comment Tip can be added to individual cells, rows and columns to show more information about the particular cell on mouse hover. You can set the comment indicator at any position (TopLeft, TopRight, BottomLeft and BottomRight) in a specific cell or row or column by using [GridCommentStyleInfo](#). Comment Tip service can be enabled by setting [GridCommentService.SetShowComment](#) attached property to `true`.

Comment Tip for specific cell

The comment tip can be added to the specific cell by setting the [Comment](#) property. The comment indicator will be shown at the top right corner of the cell.

C#

```
grid.Model[1, 2].Comment = grid.Model[1, 0].CellValue + ":\nPopulation rate  
in " + grid.Model[1, 2].ColumnIndex + " is " + grid.Model[1, 2].CellValue;
```

Country	2005	2006	2007	2008
India	8%	5%	10%	7%
China	15%	India: Population rate in 2 is 5%		8%
Japan	6%	7%	3%	4%

Comment Tip for row and column

The Comment tip can be added to specific row or column by setting the [Comment](#) property. The comment indicator will be shown at the top right corner of the cell.

C#

```
//Adding comment tip to the specific row
grid.Model.RowStyles[1].Comment = "Hello";
//Adding comment tip to the specific column
grid.Model.ColStyles[2].Comment = "Hello";
```

Country	2005	2006	2007	2008
India	4%	14%	14%	12%
China	4%	3%	Hello	9%
Japan	9%	3%	17%	7%

Another way to set the comment tip for specific row and column,

C#

```
//Add CommentTip for specific row
for (int i = 1; i <= 4; i++)
{
    string comment = grid.Model[1, 0].CellValue + " : \nPopulate rate in " +
        grid.Model[1, i].ColumnIndex + " is " + grid.Model[1, i].CellValue;
    grid.Model[1, i].Comment = comment;
}
//Add CommentTip for specific column
for (int i = 1; i < 4; i++)
{
    string comment = grid.Model[i, 0].CellValue + " : \nPopulate rate in " +
        grid.Model[i, 2].RowIndex + " is " + grid.Model[i, 2].CellValue;
    grid.Model[i, 2].Comment = comment;
}
```

Country	2005	2006	2007	2008
India	9%	10%	9%	6%
China	16%	6%	India: Populate rate in 3 is 9%	
Japan	7%	2%	16%	12%

Note: [View sample in GitHub](#)

Change comment indicator position

Setting [Comment](#) property always displays comment indicator at top right corner of the cell. You can change the comment indicator position for a specific cell by using [GridCommentStyleInfo](#). For example, you can set the comment indicator at top position for any cell by setting [GridCommentStyleInfo.TopLeftComment](#) or [GridCommentStyleInfo.TopRightComment](#) properties.

Note: You can display comment for all four corners at the same time.

C#

```
GridCommentStyleInfo styleInfo = new GridCommentStyleInfo();
// set the comment for specific cell
grid.Model[1, 2].GridCommentStyleInfo.TopLeftCommentBrush = Brushes.Red;
grid.Model[1, 2].GridCommentStyleInfo.TopLeftCommentBrush = Brushes.Red;
grid.Model[1, 2].GridCommentStyleInfo.TopLeftComment = grid.Model[1,
0].CellValue + ": \nPopulation rate in " + grid.Model[1, 2].ColumnIndex + "
is " + grid.Model[1, 2].CellValue;
//Set comment tip for specific row
for (int i = 1; i <= 4; i++)
{
    //Set comment tip for specific row
    if (grid.Model[1, i].RowIndex == 1 && grid.Model[1, i].ColumnIndex > 0)
    {
        grid.Model[1, i].GridCommentStyleInfo.TopLeftCommentBrush = Brushes.Red;
        grid.Model[1, i].GridCommentStyleInfo.TopLeftComment = grid.Model[1,
0].CellValue + ": \nPopulation rate in " + grid.Model[1, i].ColumnIndex + "
is " + grid.Model[1, i].CellValue;
    }
}
//Set comment tip for specific column
for (int i = 1; i < 4; i++)
{
    if (grid.Model[i, 2].ColumnIndex == 2 && grid.Model[i, 2].RowIndex > 0)
    {
        grid.Model[i, 2].GridCommentStyleInfo.TopLeftCommentBrush = Brushes.Red;
        grid.Model[i, 2].GridCommentStyleInfo.TopLeftComment = grid.Model[i,
0].CellValue + ": \nPopulation rate in " + grid.Model[i, 2].RowIndex + " is
" + grid.Model[i, 2].CellValue;
    }
}
```

Note: [View sample in GitHub](#)

Country	2005	2006	2007	2008
India	14%	13%	2%	4%
China	13%	11%	India: Population rate in 4 is 14%	
Japan	2%	6%	14%	9%

Note: Similarly, You can also set the comment indicator at bottom position for any cell or row or column by setting the [BottomLeftComment](#) and [BottomRightComment](#) properties.

Set CommentTip using QueryCellInfo

You can set the comment tip to a specific cell or row or column by using [QueryCellInfo](#) event.

C#

```
private void Model_QueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    GridCommentStyleInfo gridStyleInfo = new GridCommentStyleInfo();
    //Set comment tip for specific cell
    if (e.Style.RowIndex == 1 && e.Style.ColumnIndex == 2)
    {
        e.Style.GridCommentStyleInfo.TopLeftCommentBrush = Brushes.Red;
        e.Style.GridCommentStyleInfo.TopLeftComment = e.Style.GridModel[1,
0].CellValue + ": \nPopulation rate in " + e.Style.ColumnIndex + " is " +
e.Style.CellValue.ToString();
    }
    //set comment tip for specific row
    if (e.Style.RowIndex == 1 && e.Style.ColumnIndex > 0)
    {
        e.Style.GridCommentStyleInfo.TopLeftCommentBrush = Brushes.Red;
        e.Style.GridCommentStyleInfo.TopLeftComment = e.Style.GridModel[1,
0].CellValue + ": \nPopulation rate in " + e.Style.ColumnIndex + " is " +
e.Style.CellValue.ToString();
    }
    //Set comment tip for specific column
    if (e.Style.ColumnIndex == 2)
    {
        e.Style.GridCommentStyleInfo.TopLeftCommentBrush = Brushes.Red;
        e.Style.GridCommentStyleInfo.TopLeftComment =
e.Style.GridModel[e.Style.RowIndex, 0].CellValue + ": \nPopulation rate in "
+ e.Style.RowIndex + " is " + e.Style.CellValue.ToString();
    }
}
```

Country	2005	2006	2007	2008
India	16%	8%	16%	13%
China	15%	13%	17%	11%
Japan	13%	15%		

Note: [View sample in GitHub](#)

Handling CommentTip opening event

The [CellCommentOpening](#) event will be triggered when the mouse hover on the cell has a comment indicator.

C#

```
//CellCommentOpening event
grid.CellCommentOpening += Grid_CellCommentOpening;
grid.CellCommentOpening += Grid_CellCommentOpening;
private void Grid_CellCommentOpening(object sender,
GridCellCommentOpeningEventArgs e)
```

```
{
    var grids = sender as GridControl;
    if (e.Cell.RowIndex == 1 && e.Cell.ColumnIndex == 2)
        grids.Model[e.Cell.RowIndex,
            e.Cell.ColumnIndex].GridCommentStyleInfo.TopLeftComment = "Hello";
}
```

Country	2005	2006	2007	2008
India	4%	3%	17%	15%
China	5%	0%	15%	4%
Japan	10%	3%	15%	13%

Customize the CommentTip

The comment tip appearance can be customized by defining DataTemplate. The DataTemplate can be assigned to [GridStyleInfo.CommentTemplateKey](#). If you are using [TopLeftComment](#), [TopRightComment](#), [BottomRightComment](#), or [BottomLeftComment](#) then you need to assign template to its corresponding template key property namely [GridCommentStyleInfo.TopLeftCommentTemplateKey](#), [GridCommentStyleInfo.TopRightCommentTemplateKey](#), [GridCommentStyleInfo.BottomLeftCommentTemplateKey](#) or [GridCommentStyleInfo.BottomRightCommentTemplateKey](#).

[GridStyleInfo](#) which holds cell information is the **DataContext** for data template of comment.

In the below code TopLeft comment is customized.

XML

```
<Window.Resources>
<DataTemplate x:Key="TopLeftComment">
<Border x:Name="border" BorderThickness="1" BorderBrush="DarkBlue">
<TextBlock Background="Purple" Foreground="White" FontSize="14"
FontStyle="Italic" Text="{Binding Comment}" />
</Border>
</DataTemplate>
</Window.Resources>
```

C#

```
//Assign the template to TopLeftCommentTemplateKey
grid.Model[1, 2].GridCommentStyleInfo.TopLeftCommentTemplateKey =
    "TopLeftComment";
//Using QueryCellInfo event
private void Model_QueryCellInfo(object sender, GridQueryCellInfoEventArgs
e)
{
    if (e.Style.RowIndex == 1 && e.Style.ColumnIndex == 2)
    {
        //Assign the template to TopLeftCommentTemplateKey
        e.Style.GridCommentStyleInfo.TopLeftCommentTemplateKey = "TopLeftComment";
    }
}
```

Country	2005	2006	2007	2008
India	11%	17%	11%	8%
China	13%	13%	3%	14%
Japan	13%	17%	14%	13%

Note: [View sample in GitHub](#)

Input Message Tip in WPF GridControl

The input message tip can be added to individual cells, rows and columns to show external message of a specific cell when the cell is activated. The input message tip service can be enabled by setting the [GridTooltipService.SetShowTooltips](#) attached property to `true`. Tooltip for particular cell or row or column can be enabled by setting the [GridStyleInfo.ShowDataValidationTooltip](#) property.

Input Message Tip for particular cell

The input message tip displayed for any cell by setting the [ShowDataValidationTooltip](#) and the message tip can be customized by setting [DataValidationTooltip](#) property.

XML

```
<syncfusion:GridControl Name="grid"
syncfusion:GridTooltipService.ShowTooltips="True" />
```

C#

```
//To enable input message tip for GridControl
GridTooltipService.SetShowTooltips(grid, true);
grid.Model[1, 2].DataValidationTooltip = grid.Model[1, 0].CellValue +
":\nPopulation rate in " + grid.Model[1, 2].ColumnIndex + " is " +
grid.Model[1, 2].CellValue;
grid.Model[1, 2].ShowDataValidationTooltip = true;
```

Country	2005	2006	2007	2008
India	12%	17%	17%	11%
China	11%	16%	10%	10%
Japan	2%	10%	7%	7%
UK	13%	15%	14%	16%

Input Message Tip for row and column

The input message tip can be displayed for any row or column by setting the [ShowDataValidationTooltip](#) and the message tip can be customized by setting [DataValidationTooltip](#) property.

C#

```
//Adding input message tip for specific row
grid.Model.RowStyles[1].DataValidationTooltip = "Hello";
```

```
grid.Model.RowStyles[1].ShowDataValidationTooltip = true;
//Adding input message tip for specific column
grid.Model.ColStyles[1].DataValidationTooltip = "Hello";
grid.Model.ColStyles[1].ShowDataValidationTooltip = true;
```

Country	2005	2006	2007	2008
India	4%	2%	3%	6%
China	9%	17%	8%	4%
Japan	16%	2%	5%	17%
UK	7%	12%	9%	10%

An another way to set the input message tip for specific row and column.

C#

```
//Add input message tip for specific row
for (int i = 1; i <= 4; i++)
{
    string comment = grid.Model[1, 0].CellValue + " :\nPopulation rate in " +
    grid.Model[1, i].ColumnIndex + " is " + grid.Model[1, i].CellValue;
    grid.Model[1, i].DataValidationTooltip = comment;
    grid.Model[1, i].ShowDataValidationTooltip = true;
}
//Add input message tip for specific column
for (int i = 1; i <= 4; i++)
{
    string comment = grid.Model[i, 0].CellValue + " :\nPopulation rate in " +
    grid.Model[i, 2].RowIndex + " is " + grid.Model[i, 2].CellValue;
    grid.Model[i, 2].DataValidationTooltip = comment;
    grid.Model[i, 2].ShowDataValidationTooltip = true;
}
```

Country	2005	2006	2007	2008
India	6%	13%	6%	3%
China	2%	9%	14%	10%
Japan	2%	6%		14%
UK	13%	7%		16%

Note: [View sample in GitHub](#)

Set Input Message Tip using QueryCellInfo event

You can set the input message tip for specific cell or row or column by using [QueryCellInfo](#) event.

C#

```
private void Grid_QueryCellInfo(object sender,
    Syncfusion.Windows.Controls.Grid.GridQueryCellInfoEventArgs e)
```

```

{
e.Style.ShowDataValidationTooltip = true;
//Show message tip for specific cell
if (e.Cell.RowIndex == 1 && e.Cell.ColumnIndex == 1)
e.Style.DataValidationTooltip = e.Style.GridModel[1, 0].CellValue + ":
\nPopulation rate in " + e.Style.ColumnIndex + " is " +
e.Style.CellValue.ToString();
// Show message tip for row.
if (e.Cell.ColumnIndex > 0 && e.Cell.RowIndex == 1)
e.Style.DataValidationTooltip = e.Style.GridModel[1, 0].CellValue + ":
\nPopulation rate in " + e.Style.ColumnIndex + " is " +
e.Style.CellValue.ToString();
// Show message tip for column.
if (e.Cell.RowIndex > 0 && e.Cell.ColumnIndex == 2)
e.Style.DataValidationTooltip = e.Style.GridModel[e.Style.RowIndex,
0].CellValue + ": \nPopulation rate in " + e.Style.RowIndex + " is " +
e.Style.CellValue.ToString();
}

```

Country	2005	2006	2007	2008
India	9%	10%	15%	4%
China	15%	17%		13%
Japan	6%	4%		4%
UK	14%	10%	14%	11%

Note: [View sample in GitHub](#)

Show or hide Input Message Tip

You can show or hide the input message tip for specific cell or row or column by setting the [ShowDataValidationTooltip](#) property to `false`.

C#

```
grid.Model[1, 2].ShowDataValidationTooltip = false;
```

Identify whether cell has Input Message Tip

The [HasShowDataValidationTooltip](#) property is used to identify whether the cell has an input message tip in a cell or row or column in GridControl. You can also highlight the cell or row or column applied to the input message tip.

C#

```

private void Grid_QueryCellInfo(object sender,
Syncfusion.Windows.Controls.Grid.GridQueryCellInfoEventArgs e)
{
if (e.Cell.ColumnIndex > 0 && e.Cell.RowIndex == 2)
{
e.Style.DataValidationTooltip = e.Style.GridModel[1, 0].CellValue + ":
\nPopulation rate in " + e.Style.ColumnIndex + " is " +
e.Style.CellValue.ToString();
if(e.Style.HasShowDataValidationTooltip)

```



```
{
e.Style.Background = Brushes.CornflowerBlue;
e.Style.Foreground = Brushes.White;
}
}
```

Country	2005	2006	2007	2008
India	9%	3%	12%	9%
China	10%	4%	8%	9%
Japan	14%	15%		12%
UK	16%	4%		6%

Customize the Input Message Tip

The input message tip can be customized by defining DataTemplate. The DataTemplate can be assigned to the [GridStyleInfo.DataValidationTooltipTemplateKey](#) property. If you are using inputTextmessage1 then you need to assign template to its corresponding template key property namely [GridStyleInfo.DataValidationTooltipTemplateKey](#).

[GridStyleInfo](#) which holds cell information is the [DataContext](#) for data template of input message tip.

XML

```
<Window.Resources>
<DataTemplate x:Key="inputTextmessage1">
<Border BorderBrush="Gray" BorderThickness="2.5" CornerRadius="5">
<TextBlock Width="Auto" Height="Auto" HorizontalAlignment="Left"
VerticalAlignment="Center"
Background="LightBlue" FontSize="14" Foreground="Black" Text="{Binding
DataValidationTooltip}" />
</Border>
</DataTemplate>
</Window.Resources>
```

C#

```
// Set template key to a particular index
grid.Model[1, 2].DataValidationTooltipTemplateKey = "inputTextmessage1";
// Using QueryCellInfo event
private void Grid_QueryCellInfo(object sender,
Syncfusion.Windows.Controls.Grid.GridQueryCellInfoEventArgs e)
{
if(e.Cell.RowIndex == 1 && e.Cell.ColumnIndex == 2)
{
e.Style.DataValidationTooltip = e.Style.GridModel[1, 0].CellValue + ":
\nPopulation rate in " + e.Style.ColumnIndex + " is " +
e.Style.CellValue.ToString();
e.Style.ShowDataValidationTooltip = true;
e.Style.DataValidationTooltipTemplateKey = "inputTextmessage1";
}
}
```

```
}

```

Country	2005	2006	2007	2008
India	15%	14%	14%	4%
China	10%	11%		2%
Japan	6%	4%		12%
UK	17%	16%	4%	8%

Note: [View sample in GitHub](#)

Remove the Input Message Tip

The `ResetValue()` method is used to remove the input message tip for specific cell or row or column in GridControl and to reset the input message value to the default values.

C#

```
grid.Model[1, 2].DataValidationTooltip = grid.Model[1, 0].CellValue +
":\nPopulation rate in " + grid.Model[1, 2].ColumnIndex + " is " +
grid.Model[1, 2].CellValue;
grid.Model[1, 2].ShowDataValidationTooltip = true;
grid.Model[1,
2].ResetValue(GridStyleInfoStore.DataValidationTooltipProperty);
```

Export to Excel in WPF GridControl

This section explains about Exporting to Excel and its features of WPF GridControl

Exporting to Excel

The GridExcelConverter class provides support for exporting data from a Grid control to an Excel spreadsheet for verification and/or computation. This control copies the Grid's styles and formats to Excel automatically. The GridExcelConverter class is derived from GridExcelConverterBase. The XlsIO libraries are used to support the conversion of the grid contents to Excel. Following assembly files must be added along with the default assembly files in the reference folder:

- Syncfusion.XlsIO.Base
- Syncfusion.XlsIO.WPF
- Syncfusion.GridConverter.Wpf

Features

There are three options for exporting a Grid control:

- Converting the entire content of a grid
- Converting a selected content of the grid
- Pass the Excel Engine

Entire Content

You can convert the entire content of a GridData control to an Excel Spreadsheet. You can also avail the option for specifying the version of the Excel file using the ExcelVersion enum. The version can be one of the following:

- ExcelVersion.Excel97to2003
- ExcelVersion.Excel2007

The following code illustrates the conversion of the entire Grid content to an Excel Spreadsheet:

C#

```
gridControl.Model.ExportToExcel(@"Sample.xls", ExcelVersion.Excel97to2003);  
// (or)  
gridControl.Model.ExportToExcel(@"Sample.xlsx", ExcelVersion.Excel2007);
```



1	Text37	Text16	53.64	22.47	
2	14.45	Text81	56	Text67	21.31
3	21.31	22	71.29	Text73	12.15
4	30	75	79.46	46	
5	30	75.97	77.3		
6	51	46	18.74		
7	Text22	18.83	88	49.04	76.34
8	Text75	35	Text76	Text50	44.45
9	59.43	54.65	41	18	13
10	Text38	14.98	Text14	70	Text89
11	18.71	Text23	Text66	22.35	80.44
12	32.99		Text96	43.55	46.85
13	92.84	Text48	62	19.46	50.81
14	21.46	16	73.5	19.81	95
15	66		Text65	70.77	45
16	98.64	12	26.43	21.45	18.28
17	Text89	15.99	25	Text78	64.59
18	72.8	48	94.79	Text12	67.31
19	Text98	61.79	Text40	Text55	58.56
20	Text87	Text35	58	32.47	30.23

	A	B	C	D	E	F	G
1	Text37	Text16	53 64		22.47	75.75	08-20-2009
2	14	45	Text81	56	Text67	27.96	08-20-2009
3	21.31	22	71.29	Text73	12	19	08-20-2009
4	30	75	79.46	46			08-20-2009
5	30	75.97	77.3				08-20-2009
6	51	46	18.74				08-20-2009
7	Text22	18.83	88	49.04	76.34	Text47	08-20-2009
8	Text75	35	Text76	Text50	44.45	11	08-20-2009
9	59.43	54.65	41	18	13	95.19	08-20-2009
10	Text38	14.98	Text14	70	Text89	78	08-20-2009
11	18.71	Text23	Text66	22.35	80.44	29.66	08-20-2009
12	32.99	Text96	43.55	46.85	44.36	34.08	08-20-2009
13	92.84	Text48	62	19.46	50.81	37	08-20-2009
14	21.46	16	73.5	19.81	95	Text86	08-20-2009
15	66	Text65	70.77	45	Text54	Text94	08-20-2009
16	98.64	12	26.43	21.45	18.28	59	08-20-2009
17	Text89	15.99	25	Text78	64.59	97	08-20-2009
18	72.8	48	94.79	Text12	67.31	87	08-20-2009

The images above show how the entire content of the Grid control is exported to an Excel spreadsheet.

Selected Content

You can convert a selected content of the grid to the specified range in an Excel spreadsheet. It will be very useful, when you have some data like picture, chart, etc., in your spreadsheet and you want to fill a particular range, for example-the remaining part of the spreadsheet using the Grid cell data.

Use-Case Scenario

Consider that you have a chart in a spreadsheet in the range [A1:I19] and you wish to populate a part of the Spreadsheet starting from E21, with the selected cell data of Grid control. You can use the following code, to achieve the scenario above mentioned:

C#

```
ExcelEngine excelEngine = new ExcelEngine();
IApplication application = excelEngine.Excel;
IWorkbook myWorkbook = excelEngine.Excel.Workbooks.Add();
IWorksheet mySheet = myWorkbook.Worksheets[1];
IChart chartShape = mySheet.Charts.Add();
IChartSeries series1 = chartShape.Series.Add();
series1.SeriesType = ExcelChartType.Column_Clustered;
chartShape.ChartTitle = "Column_Clustered";
series1.Values = mySheet.Range["B1:B5"];
series1.CategoryLabels = mySheet.Range["A1:A5"];
Random r = new Random();
for (int i = 1; i <= 4; i++)
{
    mySheet.Range[i, 1].Number = i;
    mySheet.Range[i, 2].Number = r.Next(4000, 6000);
}
for (int i = 1; i <= 4; i++)
{

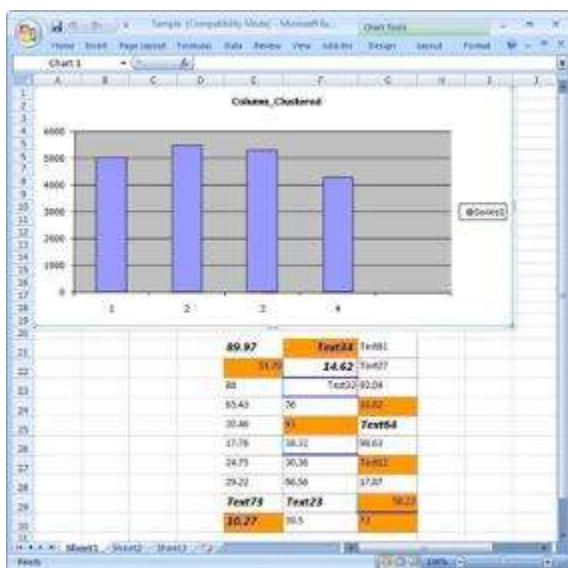
```

```

mySheet.Range[i + 5, 1].Number = i;
mySheet.Range[i + 5, 2].Number = r.Next(4000, 6000);
}
IRange excelRange = mySheet.Range[21, 5];
GridRangeInfoList rangeList = gridControl.Model.SelectedRanges;
GridRangeInfo range = rangeList[0];
gridControl.Model.ExportToExcel(range, mySheet, excelRange, @"Sample2.xls",
ExcelVersion.Excel97to2003);

```

1	63.49	89		75.94	61	58
2	Text28	62.82	70.56		10	62.1
3	Text60	83	95	93.06	77	
4	95.42		44	Text123	Text52	
5	Text32		39.45	Text46		
6	56.13		85.79	73		
7		62.33	72.35	Text82	52	55
8	90.35	Text74	59.87		71.77	48.5
9		43	89	10	58	72
10	18		19	94.5	75	16.92
11		50.81	82.86	75.9	52	74.18
12	53	Text46		53.97	98	84
13	25.55	Text30		51.1	Text26	66.27
14	28		Text72	70.66	41.77	45.01
15	31.93		87.17	80	56	84
16	86		71.32	69.64	24	41.97
17		Text46	Text24	90.83	25.63	18.82
18	11.55	30		16	Text84	89.21
19	38	26.21		45	24.52	Text70
20	86.86	44.41	28	73.89	84.57	88



The above images shows how a part of the Grid control is exported to a specific range on an Excel Spreadsheet.

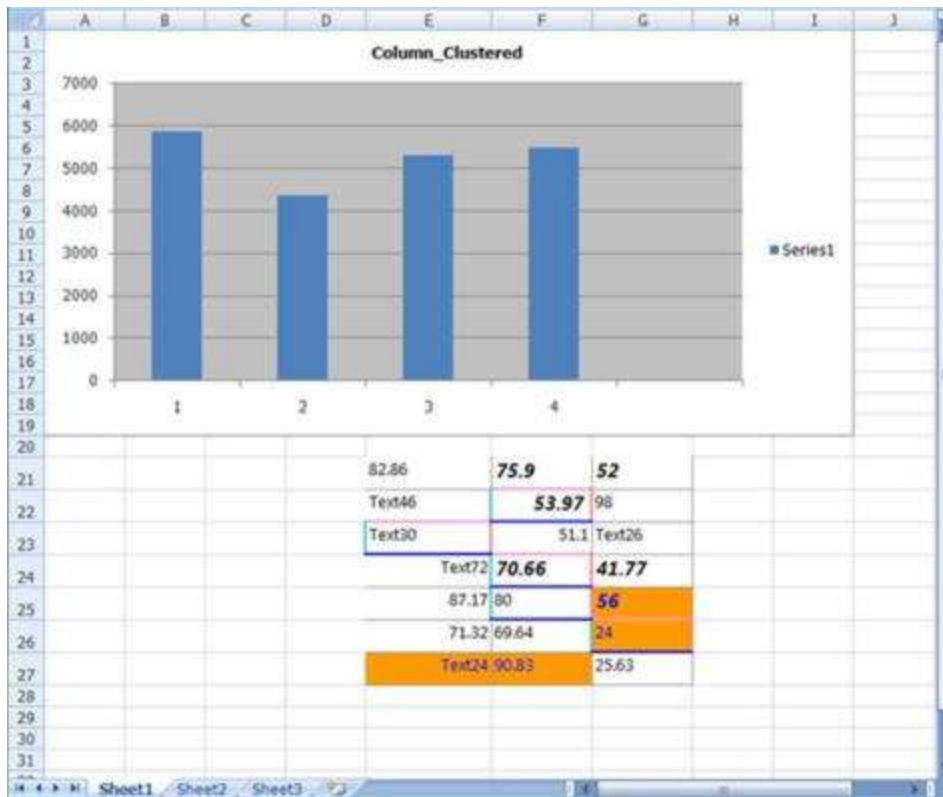
Using the Excel Engine (XlsIO)

You can also pass the Excel Engine with the worksheet number, as illustrated by the code below:

C#

```
ExcelEngine excelEngine = new ExcelEngine();
IApplication application = excelEngine.Excel;
IWorkbook myWorkbook = excelEngine.Excel.Workbooks.Add();
IWorksheet mySheet = myWorkbook.Worksheets[0];
gridControl.Model.ExportToExcel(range, excelEngine, 0, mySheet.Range[5,5],
@"Sample.xlsx", ExcelVersion.Excel2007);
```

1	63.49	89		75.94	61	58			
2	Text28	62.82	70.56		10	62.1	32		
3	Text60	83	95	93.06	77				
4	95.42		44	Text123	Text52				
5	Text32		39.45	Text46					
6	56.13		85.79	73					
7	62.33	72.35		Text82	52	55	6		
8	90.35	Text74	59.87		71.77	48.5	76		
9		43	89	30	58		72	Te	
10	18			19	94.5	75	16.92	38	
11		50.81	82.86	75.9	52	74.18	8		
12	53		Text46	53.97	98	84	94		
13	25.55		Text30	51.1	Text26	66.27	17		
14	28		Text72	70.66	41.77	45.01			
15	31.93		87.17	80	56	84	86		
16	86		71.32	69.64	24	41.97	7		
17		Text46	Text24	90.83	25.63	18.82	Te		
18	11.55	30		16	Text84	89.21	17		
19	38	26.21		45	24.52	Text70	Te		
20	86.86	44.41	28	73.89	84.57	89			



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

Exporting to CSV

The ExportToCSV method of the GridModelExportExtensions class enables a grid control to be easily exported to CSV format.

To enable exporting, the following .dll files must be added along with the default .dll files in the reference folder:

- Syncfusion.XlsIO.Base
- Syncfusion.XlsIO.WPF
- Syncfusion.GridConverter.Wpf

Export Options

There are two options for exporting a grid control:

1. Export Whole Grid – which exports an entire grid to CSV format.
2. Export Selected Range – which exports only a selected range to CSV format.

Export Whole Grid

You can convert the entire content of a grid control to a CSV file by using the following code:

C#

```
this.gc.Model.ExportToCSV("Sample.csv");
```

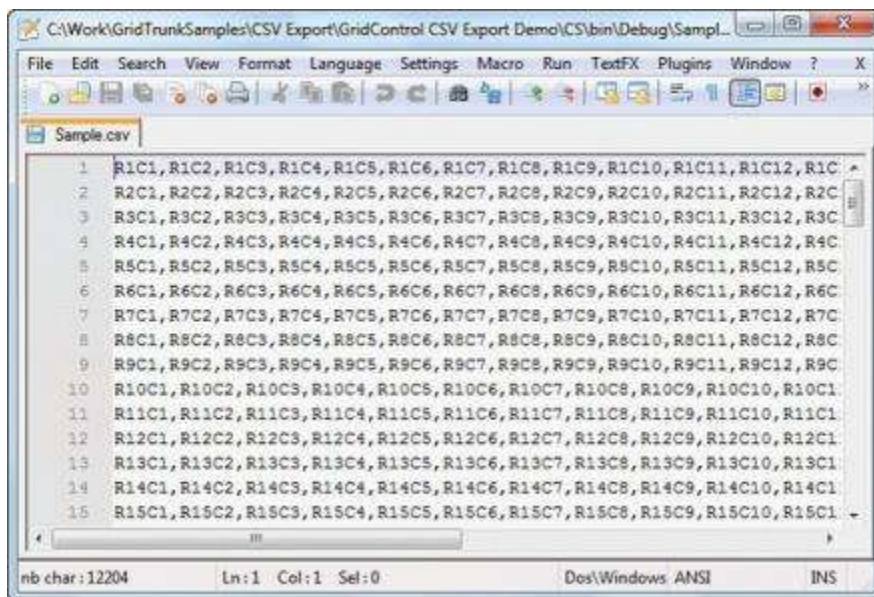
VB.NET

```
Me.gc.Model.ExportToCSV("Sample.csv")
```

When the code runs, the following output displays.



When you are ready to export the entire grid, click **Export Whole Grid**; the grid content will then be converted to CSV format.



Export Selected Range

You can convert selected grid content to CSV format by using the following code:

C#

```
GridRangeInfoList rangeList = gc.Model.SelectedRanges;
if (rangeList.Count > 0)
{
    GridRangeInfo range = rangeList[0];
}
```

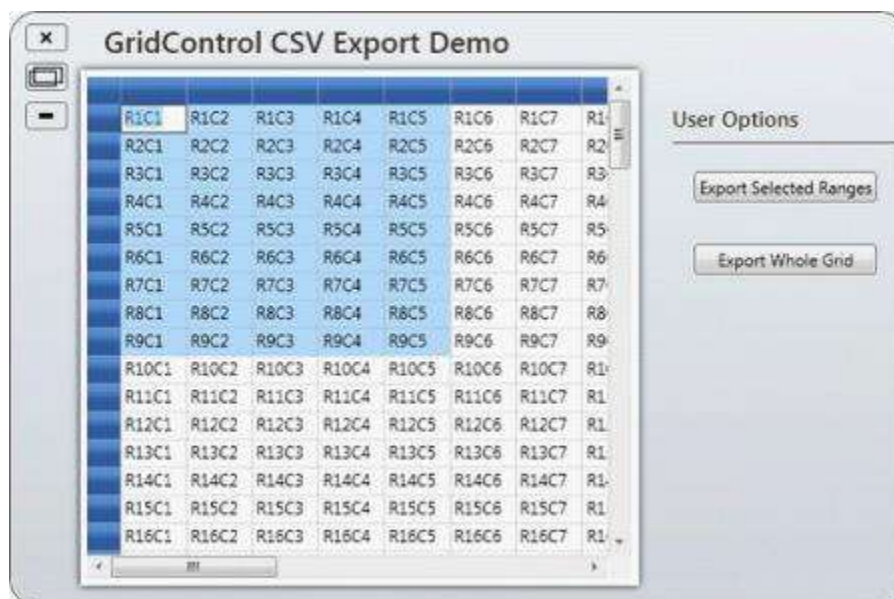


```
gc.Model.ExportToCSV(range, "Sample.csv");
}
```

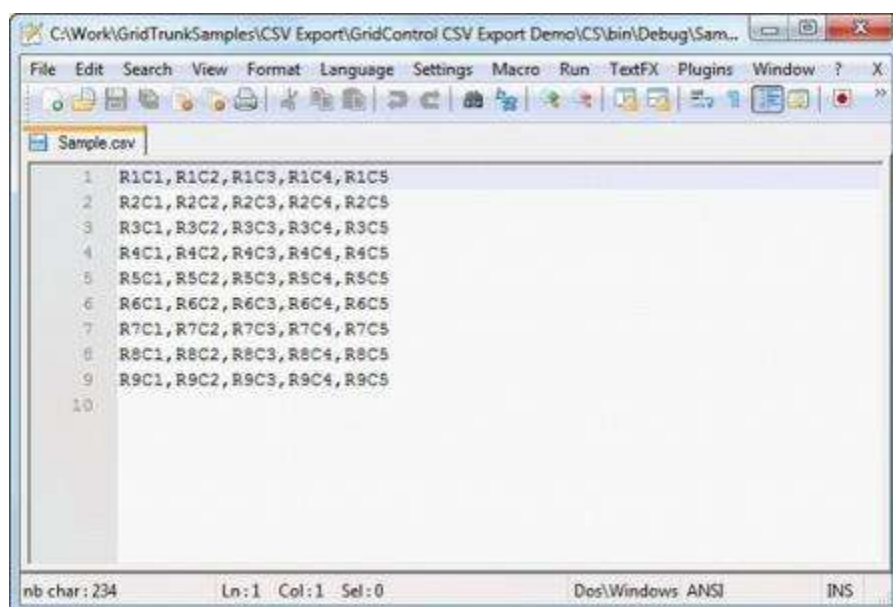
VB.NET

```
Dim rangeList As GridRangeInfoList = gc.Model.SelectedRanges
If rangeList.Count > 0 Then
Dim range As GridRangeInfo = rangeList(0)
gc.Model.ExportToCSV(range, "Sample.csv")
End If
```

When the code runs, the following output displays.



To export a selection, highlight the portion of the grid you want to export, and then click Export Selected Range; the selected grid content will then be exported to a CSV file.



Import from Excel in WPF GridControl

Essential Grid WPF provides an in-built support for Excel Importing. This feature allows you to import the Excel Workbook into a GridControl, while preserving styles, formulas, named ranges, conditional formatting, freezing pane and bookmarks.

- Importing Styles—Imports the font family, font size, font style, font weight, cell backgrounds and cell foreground.
- Importing Formulas—Imports all the formulas from the excel sheet to the GridControl. GridControl also supports the cross reference formulas as in the excel.
- Importing Conditional formatting—imports the conditional formatting (Highlight Selection Rules – Greater than, Less than, Between, Equal to, Greater than or equal to, Less than or Equal to) from the Excel workbook to grid control.
- Import the freeze pane—imports the frozen row and frozen columns to GridControl.
- Imports the Hyperlink—imports the hyperlinks (Url and the worksheet navigation) to GridControl.
- Improving Performance—Excel Importing feature supports the virtualization. By using this you can optimize the performance.
- Run-time Features—Imports the comments from the excel worksheet to GridControl.

The following assemblies are needs to be added for importing the Excel to GridControl.

- Syncfusion.GridConverter.WPF
- Syncfusion.XlsIO.Base

You can get the [GridModelImportExtensions](#) class and it's importing methods under the namespace [Syncfusion.Windows.Controls.Grid.Converter](#).

Use Case Scenarios

This feature can be used to view the Excel workbook into applications with the same set of styles and to edit the data in run time. You can also view the Excel workbook into web application with the same set of styles and borders.

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

Tables for Properties, Methods, and Events

Methods

Method	Description	Parameters	Type	Return Type	Reference links
ImportFromExcel	this method is used to import the first sheet from the stream.	ImportFromExcel(Stream fileStream)	NA	Void	NA

ImportFromExcel	this method is used to import the first sheet from the stream. With the importing event handler.	ImportFromExcel(Stream fileStream, GridCellImportFromExcelHandler importhandler)	NA	Void	NA
ImportFromExcel	this method is used to import the first sheet from the specified Excel file.	ImportFromExcel(string filename)	NA	Void	NA
ImportFromExcel	this method is used to import the first sheet from the specified Excel file with the importing event handler.	ImportFromExcel(string filename, GridCellImportFromExcelHandler importhandler)	NA	Void	NA
ImportFromExcel	this method is used to import the particular sheet from the IWorksheet.	ImportFromExcel(IWorksheet sheet)	NA	Void	NA
ImportFromExcel	this method is used to import the particular	ImportFromExcel(IWorksheet sheet, GridCellImportFromExcelHandler importhandler)	NA	Void	NA

	sheet from the IWorksheet With the importing event handler.				
ImportFromExcel	this method is used to import the list of particular worksheets into GridModel array. ArrayIndexes contains the list of worksheet indexes to be imported.	ImportFromExcel (string filename, int[] arrayIndexes, GridModel[] arrayModel)	NA	Void	NA
ImportFromExcel	this method is used to import the list of particular worksheets into GridModel array with the importing event handler.	ImportFromExcel (string filename, int[] arrayIndexes, GridModel[] arrayModel, GridCellImportFromExcelHandler importHandler)	NA	Void	NA
ImportFromExcelToVirtualGrid	this method imports the entire workbook styles to the virtual	ImportFromExcelToVirtualGrid (IWorkbook book)		GridModel[]	NA

	GridControl				
ImportFromExcelToVirtualGrid	this method imports the entire workbook styles to the virtual GridControl With the importing event handler.	ImportFromExcelToVirtualGrid (IWorkbook book, GridCellImportFromExcelHandler importhandler)		GridMode[]	NA
ConvertExcelRangeToVirtualGrid	this method imports a particular range of cells to the virtual GridControl	ConvertExcelRangeToVirtualGrid(GridStyleInfo cell,IWorksheet sheet,IRange excelRange, GridCellImportFromExcelHandler importhandler)		void	NA

Events

Event	Description	Arguments	Type	Reference links
GridCellImportFromExcelHandler	For triggering this event you have to pass the event handler method ImportFromExcel or in ConvertExcelRangeToVirtualGrid.this event is triggered while importing the each Excel cell to Grid cells.If the user handles this event for a particular set of ranges by using the e.Handle argument then the grid cell will only have the user defined style.	Importhandler(object sender, ImportingCellFromExcelEventArgs e)	NA	NA

Adding Excel Importing to an Application

You can Import the entire Excel Spreadsheet to a GridControl. You can also import the Excel97to2003 and Excel2007to2010 formats

In order to import the single sheet to grid control, open the file and pass this file as stream to the ImportFromExcel method as illustrated in the following code snippet:

C#

```
FileStream fileStream = new FileStream(@"..\..\Data\Sample.xlsx",
    FileMode.Open);
byte[] file = new byte[fileStream.Length];
fileStream.Read(file, 0, (int)fileStream.Length);
fileStream.Close();
this.gridControl.Model.ImportFromExcel(new MemoryStream(file));
```

Importing the entire workbook to a GridControl

To import the entire workbook to a GridControl, initially you have to open the workbook by using the XlsIO library as shown in the following code snippet:

C#

```
FileStream fileStream = new FileStream(@"..\..\Data\Sample.xlsx",
    FileMode.Open);
byte[] file = new byte[fileStream.Length];
fileStream.Read(file, 0, (int)fileStream.Length);
fileStream.Close();
ExcelEngine excelEngine = new ExcelEngine();
IApplication application = excelEngine.Excel;
IWorkbook workbook = application.Workbooks.Open(new MemoryStream(file),
    ExcelOpenType.Automatic);
```

Import the workbook into GridModel

After opening the workbook, you can import the workbook to GridModel by using the ImportFromExcel method. It will return the model collection; you can use it in your application. It will import all the styles, Conditional Formatting, Data Validation and book marks to model. While using this method it will take some time to import all the styles into models.

For importing the workbook you can use the following code snippet.

C#

```
GridModel[] modelCollection =
    GridModelImportExtensions.ImportFromExcel(workBook);
```

Importing the entire workbook to a virtual GridControl

To import the entire workbook to a virtual GridControl, initially you have to open the workbook by using the XlsIO library as shown in the following code snippet.

C#

```
FileStream fileStream = new FileStream(@"..\..\Data\Sample.xlsx",
    FileMode.Open);
byte[] file = new byte[fileStream.Length];
fileStream.Read(file, 0, (int)fileStream.Length);
fileStream.Close();
ExcelEngine excelEngine = new ExcelEngine();
IApplication application = excelEngine.Excel;
```

```
IWorkbook workbook = application.Workbooks.Open(new MemoryStream(file),
ExcelOpenType.Automatic);
```

Import the layout into GridModel

After that you can import the workbook by using the `ImportFromExcelToVirtualGrid` method it will return the model collection. `GridModel` have only the layout styles, to import the other styles and data for cells you have to use the `ConvertExcelRangeToVirtualGrid` method.

C#

```
GridModel[] modelCollection =
GridModelImportExtensions.ImportFromExcelToVirtualGrid(workBook);
```

Import data into GridModel

To load the data in grid cells, you have to use the `ConvertExcelRangeToVirtualGrid` method, this will import the formulas, cell value, conditional formats, data validation and the styles from the excel range to grid cells. To import the data into cells you can call the `ConvertExcelRangeToVirtualGrid` method in `QueryCellInfo` Event as shown in the following code snippet:

C#

```
void Model_QueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    GridModel gridModel = sender as GridModel;
    if (!e.Style.IsChanged)
    {
        int index = modelCollection.ToList().IndexOf(gridModel);
        IWorksheet sheet = workBook.Worksheets[index];
        if (sheet != null)
        {
            IRange range = sheet.Range;
            if (e.Cell.RowIndex >= range.Row && e.Cell.ColumnIndex >= range.Column)
            {
                IRange rangeToConvert = sheet.Range[e.Cell.RowIndex, e.Cell.ColumnIndex];
                GridModelImportExtensions.ConvertExcelRangeToVirtualGrid(e.Style, sheet,
                rangeToConvert, null);
                gridModel.Data[e.Cell.RowIndex, e.Cell.ColumnIndex] = e.Style.Store;
            }
        }
    }
}
```

How To

Change the CellType while importing the Workbook?

You can also change the cell type and other styles while importing the workbook to GridControl, for that you have to pass the delegate handler in the importing method as shown in the following code snippet.

C#

```
this.gridControl.Model.ImportFromExcel(new MemoryStream(file),
ImportHandler);
```

Then by using the `ImportHandler` method you can change the particular Cell Type and styles like background and font styles. When this event was handled, it will not import the data from the excel cell only the user specified data will be applied in the Grid. You can change the cell type as shown in the following code snippet

C#

```
private void Imoprthandler(object sender, ImportingCellFromExcelEventArgs e)
{
    if (e.Range.AddressLocal == "C5")
    {
        e.Cell.CellType = "Static";
        e.Cell.CellValue = e.Range.DisplayText;
        e.Cell.Background = new SolidColorBrush(Colors.Blue);
        e.Cell.Font.FontSize = 15;
        e.Cell.Font.FontWeight = FontWeights.Bold;
        e.Handled = true;
    }
}
```

Enable the ExcellikeFrozen Row and Column in a GridControl

To enable the thick borders to indicate the Excel like freeze panes, you have to set the `ExcellikeFreezePane` property as true as show in the following code snippet.

C#

```
grid.Model.Options.ExcellikeFreezePane = true;
```

Enable the comment service in GridControl

To enable the comment service you have to set the attached property `show comment service` as true as shown in the following code snippet.

C#

```
GridCommentService.SetShowComment(grid, true);
```

Printing in WPF GridControl

Essential Grid for WPF provides an in-built support for printing and print preview. This feature populates a print dialog that allows you to preview the output and make required modifications if necessary, before sending the grid content for printing.

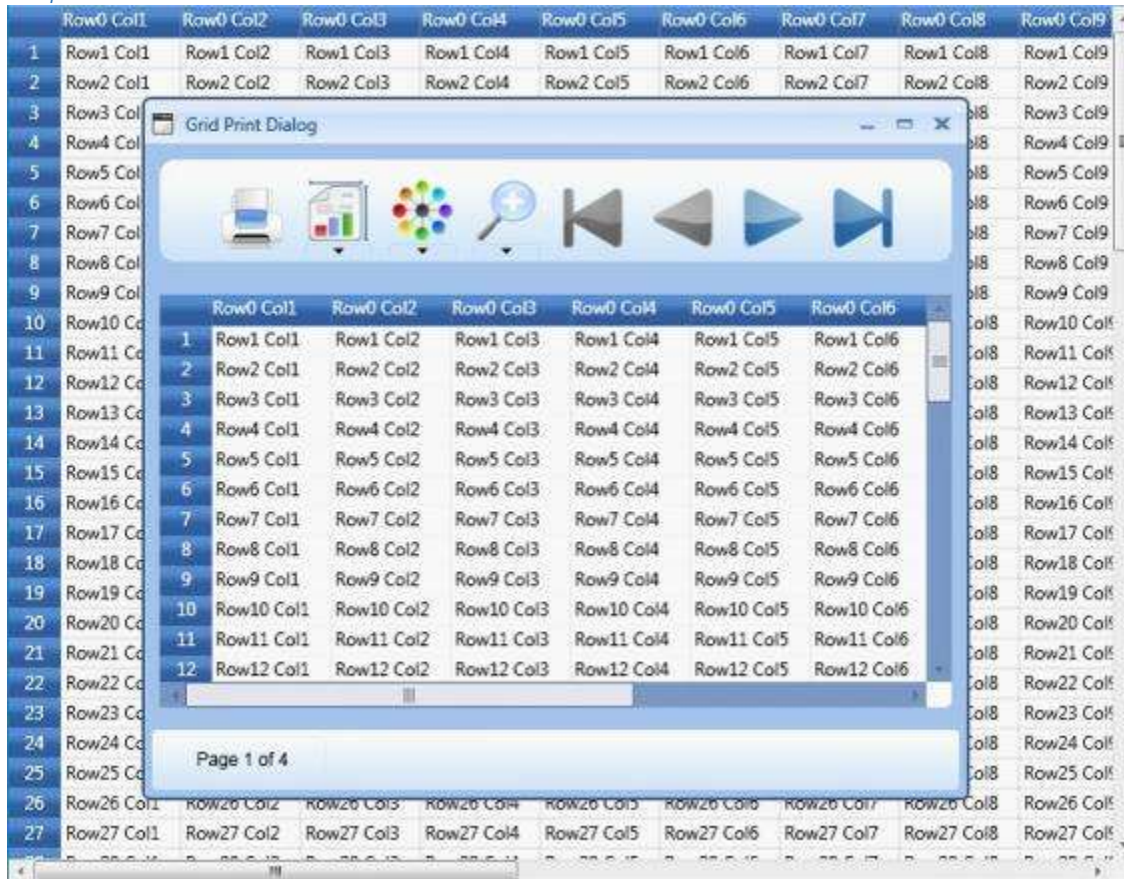
The `GridPrintDialog` class plays a vital role in the implementation of printing support. It is built based on Microsoft `PrintDialog` class that will handle the internal operations for `GridPrintDialog`. The `GridPrintDialog` class defines the designer for the Print dialog and exposes a number of properties and APIs to handle the UI requirements and define the interaction logic for the Print Dialog. The users can use these properties to configure the Print and Print Preview options.

Example

Enabling the printing feature is like invoking an API – `ShowPrintDialog()` on the instances of the grid.

C#

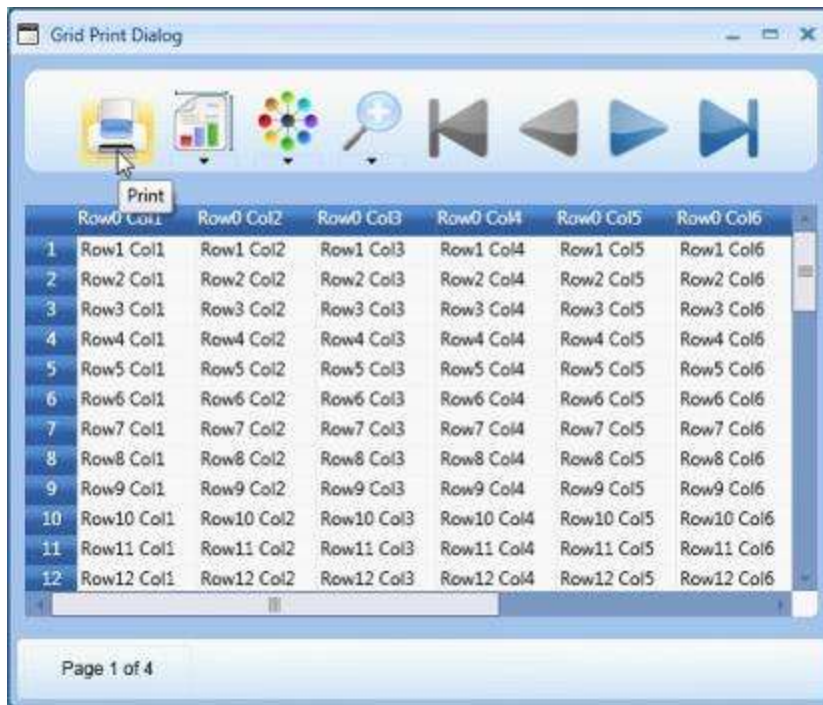
```
this.grid.ShowPrintDialog();
```


Output*Print Dialog Options*

The Print Dialog provides numerous options to configure the output.

Print Button

Clicking the Print button allows the user to send the grid content to the printer for printing.



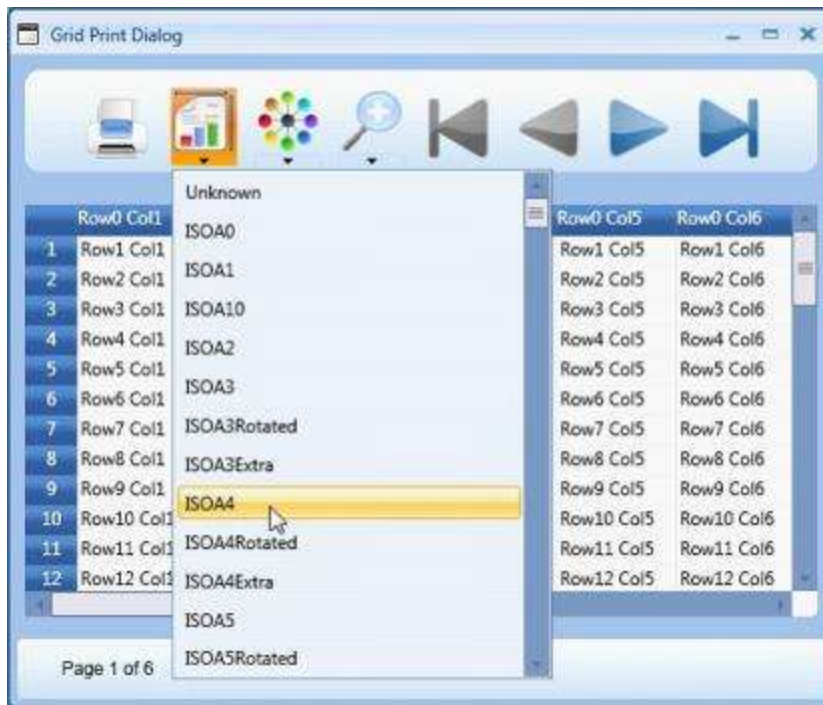
Page Size Drop-down

This drop down lists out various available paper sizes for printing. For example: ISOA4, ISOA1, ISOA3etc. Selecting the required paper size allows the user to print the grid data in the paper of same size.

Note: The paper sizes are also marked in the following way:

- Extra-Extra increases the space in addition to the predefined size of the paper, for example- A3._
- Rotated -The output is rotated by a defined angle for the selected page size._

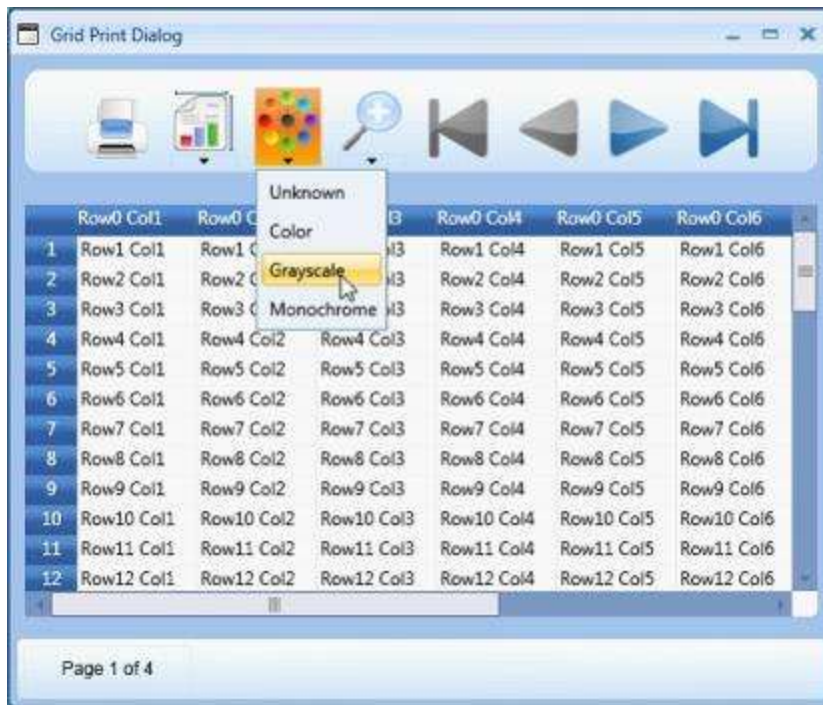
For more information, check the following link: <https://docs.microsoft.com/en-us/dotnet/api/system.printing.pagemediasizename>



Color Drop-down

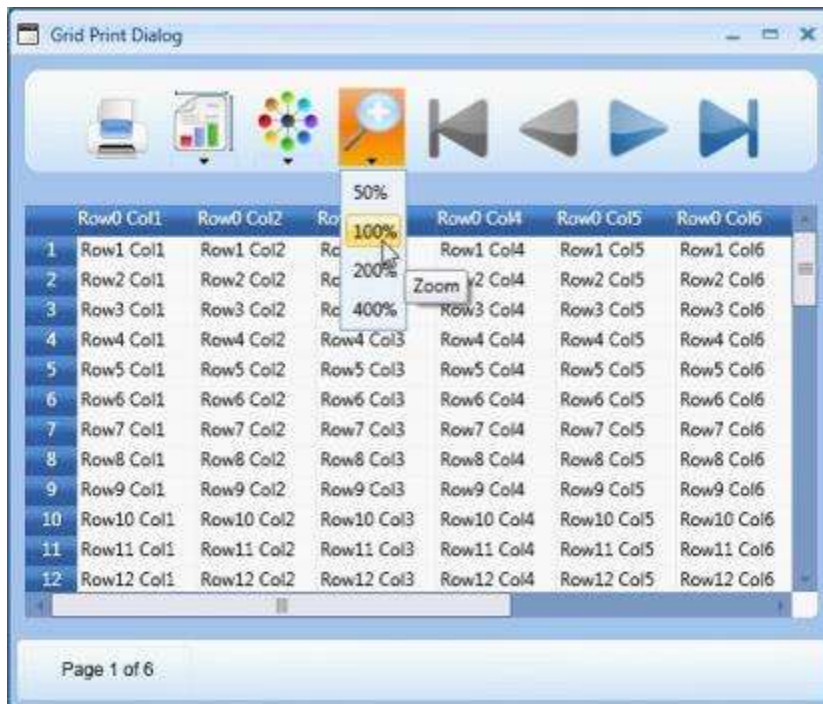
The options in this drop-down allow you select the color for the data to be printed. Following are the options provided:

- **Monochrome**—Selecting this option allows the user to print the grid data in the shades of a single color or hue.
- **Color**—Selecting this option allows the user to print the grid data in the selected color.
- **Gray scale**—Selecting this option allows the user to print the grid data in a range of shades of gray without apparent color, the darkest shade being black and the lightest shade being white.
- **Unknown**—This option can be selected in cases where you receive any print configuration settings which does not fall within the other 3 options. The application that is using this configuration will internally set the color option as unknown in such cases.



Zoom Drop-down

The options in this drop-down allow you select the required percentage magnification for viewing the grid data to be printed. Selecting the required option allows you to magnify the preview to various preset zoom levels like 50%, 100%, 200% and 400%.



Navigation Options

The Print dialog provides four navigational buttons to navigate the grid preview:



-This button allows you to move to the first page



-This button allows you to move to the last page

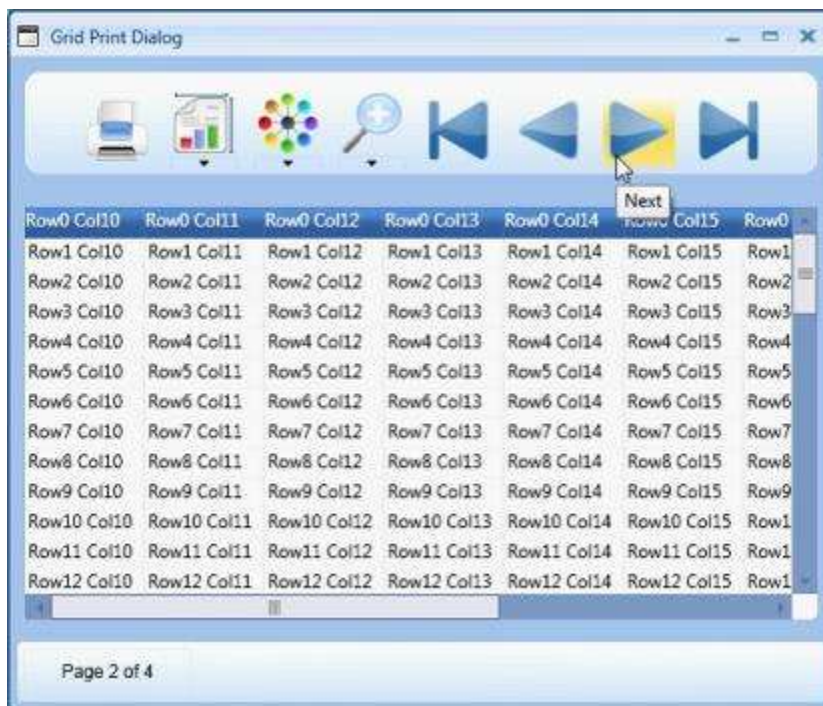


-This button allows you to move to the previous page



-This button allows you to move to the next page

The following image shows the next button highlighted for an example.



Printing Header and Footer

Headers and Footers can be added to the document to be printed.

- `GridControl.PrintHeaderTemplate` determines the content for the print header. It is a property of `GridControl` and any value assigned to this property will be provided as an output in the

Header section of the print output. For example: A text block outlined with a border is assigned to this property and hence the header also is printed with a border.

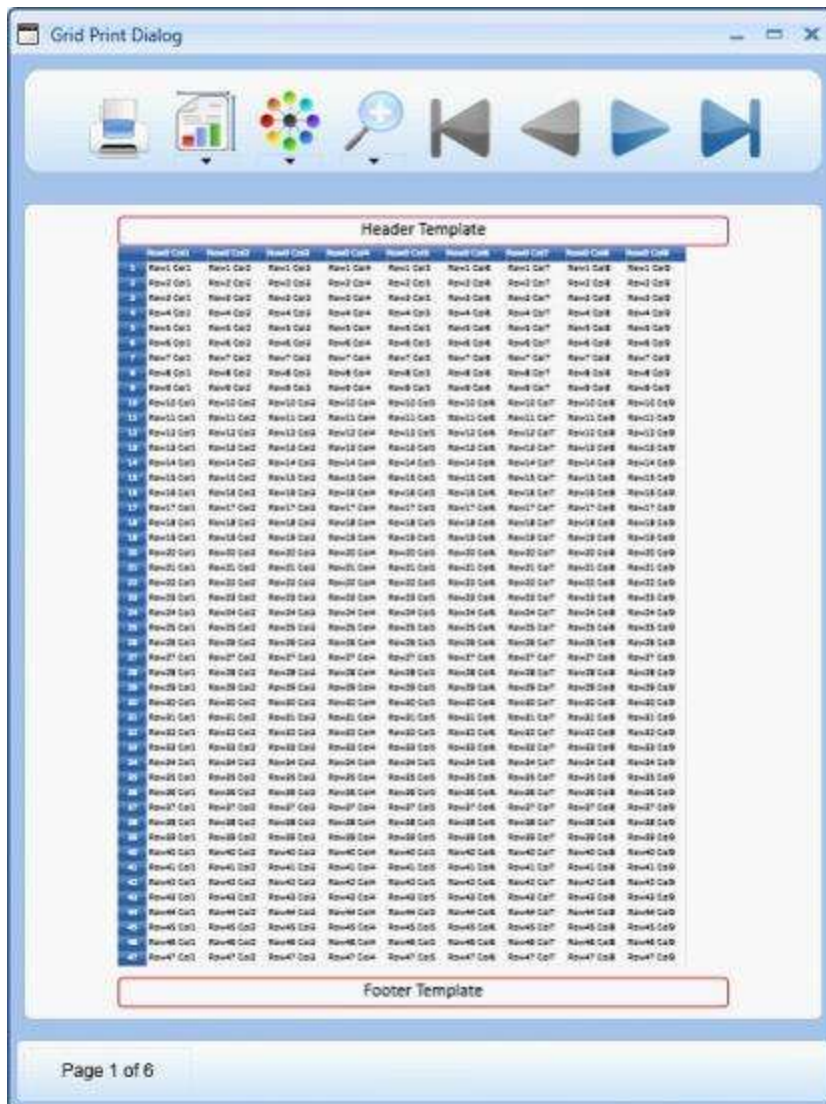
- GridControl.PrintFooterTemplate defines the print footer. The content represented by the print header will appear at the top of each printed page and likewise, the content represented by the print footer will appear at the bottom of each printed page.
- The height of the header and footer can be adjusted using the GridControl properties- PrintHeaderHeight and PrintFooterHeight.

XML

```
<syncfusion:GridControl x:Name="grid" PrintHeaderHeight="40"
PrintFooterHeight="40">
<syncfusion:GridControl.PrintHeaderTemplate>
<DataTemplate>
<Border CornerRadius="5" BorderBrush="Crimson" BorderThickness="2"
HorizontalAlignment="Stretch">
<TextBlock Text="Header Template" FontSize="24" FontFamily="Calibri"
HorizontalAlignment="Center" />
</Border>
</DataTemplate>
</syncfusion:GridControl.PrintHeaderTemplate>
<syncfusion:GridControl.PrintFooterTemplate>
<DataTemplate>
<Border CornerRadius="5" BorderBrush="Red" BorderThickness="2"
HorizontalAlignment="Stretch"
<TextBlock Text="Footer Template" FontSize="24" FontFamily="Calibri"
HorizontalAlignment="Center" />
</Border>
</DataTemplate>
</syncfusion:GridControl.PrintFooterTemplate>
```

Output

The following image is the output of the code above.



Serialization in WPF GridControl

Essential GridControl supports Serialization. The whole grid can be serialized and deserialized at run-time.

Use Case Scenarios

Serialization can be implemented for the applications which need to save its data and structure after the application is closed. Serialization supports to save the structure and data of the GridControl to an XML file and it can be loaded at any time.

Adding Serialization to an Application

The following sample application explains the implementation of the Serialization support to GridControl.

1.Create an application. Create a WPF application and add the GridControl to it. 2.Call the Serialization support methods

In the application, create three buttons. The first button to call the `Serialize()` method, the second button to make changes to the Grid and the third button is to call the `Deserialize()` method. The following code snippet explains the implementation of Serialization.

C#

```
// To Serialize the GridControl.
this.grid.Model.Serialize("Data.xml");
// To Deserialize the GridControl.
this.grid.Model.Deserialize("Data.xml");
```

3.Run the application

Run the application. Click the `Serialize` button to serialize the initial load; this creates an XML file and saves it. Click the second button `ModifyGridStyle` to make some changes in the `GridControl`. Now click the `Deserialize` button which restores the old settings of the `GridControl`.

Supported Properties for Serialization

The following properties are Serialized in the `GridControl`.

- `RowCount`
- `ColCount`
- `ActivateCurrentCellBehavior`
- `AllowSelection`
- `AllowSelectionOnShiftTab`
- `ColumnSizer`
- `CurrentCellBorder`
- `CurrentCellBorderWidth`
- `DataObjectConsumerOptions`
- `DrawSelectionOptions`
- `ExcelLikeCurrentCell`
- `ExcelLikeSelectionFrame`
- `HighlightSelectionBorder`
- `HighlightSelectionBorderWidth`
- `ListBoxModeAllowUIElementClick`
- `ListBoxSelectionMode`
- `MaxLength`
- `ScrollFrozen`
- `ShowCurrentCell`
- `WrapCell`
- `WrapCellBehavior`

Methods

Method	Description	Parameters	Type	Return Type
<code>Serialize()</code>	A virtual method called to Serialize the <code>GridControl</code> . It stores the settings in an XML file named as specified in its parameter.	<code>string fileName</code>	<code>public</code>	<code>void</code>

Deserialize()	A virtual method called to Deserialize the GridControl. It restores the settings in an XML file named as specified in its parameter.	stringÂ fileName	public	void
SerializeToStream()	Serializes the Grid to Stream.	TextWriterÂ textWriter	publicÂ	void
SerializeAsString()	Serialize the Grid as String.	NA	public	stringÂ
DeserializeFromStream	Deserialize from the given TextReader.	TextReader textReader	public	void
DeserializeFromString	Deserialize from the given String content.	string content	public	void

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

Zooming in WPF GridControl

This feature enables the user to change the zoom level of the Grid control that brings either more or fewer cells into the view. By zooming in you can get a magnified view of the grid cells, and by zooming out you can bring more cells in to the view. This does not change the underlying size of Grid control, and the printout of the Grid control remains constant, regardless of the selected zoom scale.

Use Case Scenarios

When a larger number of cells are updated in the Grid, the user can view the grid cells clearly by increasing the zoom scale. By decreasing the zoom scale, the user can display more cells in the view.

Properties

Property	Description	Type	Data Type
ZoomScale	Used to change the zoom level of the Grid control	Dependency Property	Double

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

Change Zoom Scale of the Grid Control

You can change the zoom level of the Grid control by using the ZoomScale property defined in the Grid control.

The following code illustrates how to change the ZoomScale of the Grid control:

C#

```
gridControl.ZoomScale = 1.5;
```

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate
1	10248	VINET	5	1996-07-04...	1996-08-01...	1996-07-16...
2	10249	TOMSP	6	1996-07-05...	1996-08-16...	1996-07-10...
3	10250	HANAR	4	1996-07-08...	1996-08-05...	1996-07-12...
4	10251	VICTE	3	1996-07-08...	1996-08-05...	1996-07-15...
5	10252	SUPRD	4	1996-07-09...	1996-08-06...	1996-07-11...
6	10253	HANAR	3	1996-07-10...	1996-07-24...	1996-07-16...
7	10254	CHOPS	5	1996-07-11...	1996-08-08...	1996-07-23...
8	10255	RICSU	9	1996-07-12...	1996-08-09...	1996-07-15...
9	10256	WELLI	3	1996-07-15...	1996-08-12...	1996-07-17...
10	10257	HILAA	4	1996-07-16...	1996-08-13...	1996-07-22...
11	10258	ERNSH	1	1996-07-17...	1996-08-14...	1996-07-23...
12	10259	CENTC	4	1996-07-18...	1996-08-15...	1996-07-25...
13	10260	OTTIK	4	1996-07-19...	1996-08-16...	1996-07-29...
14	10261	QUEDE	4	1996-07-19...	1996-08-16...	1996-07-30...
15	10262	RATTC	8	1996-07-22...	1996-08-19...	1996-07-25...

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipPostalCo...
1	10248	VINET	5	1996-07-04	1996-08-01	1996-07-16	3	32.38	Vins et alco...	59 rue de l'...	Reims		51100
2	10249	TOMSP	6	1996-07-05	1996-08-16	1996-07-10	1	11.61	Toms Specia...	Luisenstr. 48	Münster		44087
3	10250	HANAR	4	1996-07-08	1996-08-05	1996-07-12	2	65.83	Hanari Carnes	Rua do Paço	Rio de Janeiro RJ		05454-876
4	10251	VICTE	3	1996-07-08	1996-08-05	1996-07-15	1	41.34	Victualies e...	2, rue du Co...	Lyon		69004
5	10252	SUPRD	4	1996-07-09	1996-08-06	1996-07-11	2	51.3	Suprêmes d...	Boulevard TL	Charleroi		8-6000
6	10253	HANAR	3	1996-07-10	1996-07-24	1996-07-16	2	58.17	Hanari Carnes	Rua do Paço	Rio de Janeiro RJ		05454-876
7	10254	CHOPS	5	1996-07-11	1996-08-08	1996-07-23	2	22.98	Chop-uey...	Hauptstr. 31	Bern		3012
8	10255	RICSU	9	1996-07-12	1996-08-09	1996-07-15	3	148.33	Richmar Sup...	Statenweg 5	Genève		1204
9	10256	WELLI	3	1996-07-15	1996-08-12	1996-07-17	2	13.97	Wellington I...	Rua do Mer...	Retende	SP	08737-363
10	10257	HILAA	4	1996-07-16	1996-08-13	1996-07-22	3	81.91	HILARION-A...	Carrera 22 c...	San Cristóbal	Tachira	5022
11	10258	ERNSH	1	1996-07-17	1996-08-14	1996-07-23	1	140.51	Ernst Handel	Kirchgasse 6	Graz		8010
12	10259	CENTC	4	1996-07-18	1996-08-15	1996-07-25	3	3.25	Centro com...	Sierras de G...	México D.F.		05022
13	10260	OTTIK	4	1996-07-19	1996-08-16	1996-07-28	1	55.09	Ottiles Käse	Mehrheimer...	Köln		50739
14	10261	QUEDE	4	1996-07-19	1996-08-16	1996-07-30	2	3.05	Que Delicia	Rua da Panif...	Rio de Janeiro RJ		02389-673
15	10262	RATTC	8	1996-07-22	1996-08-19	1996-07-25	3	48.29	Rattlesnake...	2817 Milton	Albuquerque	NM	87110
16	10263	ERNSH	9	1996-07-23	1996-08-20	1996-07-31	3	146.06	Ernst Handel	Kirchgasse 6	Graz		8010
17	10264	FOLKO	6	1996-07-24	1996-08-21	1996-08-23	3	3.67	Folk och fa...	Åkergatan 24	Bräcke		S-844 67
18	10265	BLOMP	2	1996-07-25	1996-08-22	1996-08-12	1	55.28	Bondal pér...	24, place Klé...	Strasbourg		67000
19	10266	WARTH	3	1996-07-26	1996-09-06	1996-07-31	3	25.73	Wartian Her...	Torikatu 38	Oulu		90110
20	10267	FRANK	4	1996-07-29	1996-08-26	1996-08-06	1	208.58	Frankenvers...	Berliner Plat...	München		80805
21	10268	GROSR	8	1996-07-30	1996-08-27	1996-08-02	3	66.29	GROSELLA...	5ª Ave. Los...	Caracas	DF	1061
22	10269	WHITC	5	1996-07-31	1996-08-14	1996-08-09	1	4.56	White Clove	1029 - 12th	Seattle	WA	98124
23	10270	WARTH	1	1996-08-01	1996-08-29	1996-08-02	1	136.54	Wartian Her...	Torikatu 38	Oulu		90110
24	10271	SPLIR	6	1996-08-01	1996-08-29	1996-08-30	2	4.54	Split Rail Be...	P.O. Box 555	Lander	WY	82520
25	10272	RATTC	6	1996-08-02	1996-08-30	1996-08-06	2	98.03	Rattlesnake...	2817 Milton	Albuquerque	NM	87110
26	10273	QUICK	3	1996-08-05	1996-09-02	1996-08-12	3	76.07	QUICK-Stop	Taucherstra...	Cunewalde		01307
27	10274	VINET	6	1996-08-06	1996-09-03	1996-08-16	1	6.01	Vins et alco...	59 rue de l'...	Reims		51100
28	10275	MAGAA	1	1996-08-07	1996-09-04	1996-08-09	1	26.93	Magazzini A...	Via Ludovic...	Bergamo		24100
29	10276	TORTU	8	1996-08-08	1996-08-22	1996-08-14	3	13.84	Tortuga Res...	Avda. Aztec...	México D.F.		05033
30	10277	MORGK	2	1996-08-09	1996-09-06	1996-08-13	3	125.77	Morgenster...	Heerstr. 22	Leipzig		04179
31	10278	BERGS	8	1996-08-12	1996-09-09	1996-08-16	2	92.69	Berglunds s...	Berguvsväg...	Luleå		S-958 22
32	10279	LEHMS	8	1996-08-13	1996-09-10	1996-08-16	2	25.83	Lehmanns...	Magazinwe...	Frankfurt a.M.		60528
33	10280	BERGS	2	1996-08-14	1996-09-11	1996-09-12	1	8.98	Berglunds s...	Berguvsväg...	Luleå		S-958 22
34	10281	ROMEY	4	1996-08-14	1996-08-28	1996-08-21	1	2.94	Romero y to...	Gran Vía 1	Madrid		28001
35	10282	ROMEY	4	1996-08-15	1996-09-12	1996-08-21	1	12.69	Romero y to...	Gran Vía 1	Madrid		28001
36	10283	LIAS	3	1996-08-16	1996-09-13	1996-08-23	3	84.81	LIAS-Super...	Carrera 52 c...	Bogotá		3508
37	10284	LEHMS	4	1996-08-19	1996-09-16	1996-08-27	1	76.56	Lehmanns...	Magazinwe...	Frankfurt a.M.		60528
38	10285	QUICK	1	1996-08-20	1996-09-17	1996-08-26	2	76.83	QUICK-Stop	Taucherstra...	Cunewalde		01307
39	10286	QUICK	8	1996-08-21	1996-09-18	1996-08-30	3	229.24	QUICK-Stop	Taucherstra...	Cunewalde		01307
40	10287	RICAR	8	1996-08-22	1996-09-19	1996-08-28	3	12.76	Ricardo Ado...	Av. Copacab...	Rio de Janeiro RJ		02389-890
41	10288	REGGC	4	1996-08-23	1996-09-20	1996-09-03	1	7.45	Reggiani Ca...	Strada Provi...	Reggio Emilia		42100
42	10289	BISBEV	7	1996-08-26	1996-09-23	1996-08-28	3	22.77	B's Beverages	Fauntleroy...	London		EC1 5NT
43	10290	COMMI	8	1996-08-27	1996-09-24	1996-09-03	1	79.7	Comércio M...	Av. dos Lus...	Sao Paulo	SP	05432-045
44	10291	QUEDE	6	1996-08-27	1996-09-24	1996-09-04	2	6.4	Que Delicia	Rua da Panif...	Rio de Janeiro RJ		02389-673

Real Time Applications in WPF GridControl

Grids can be adopted in many real time applications where the database is of crucial importance. As such applications are widely spread; the grids are indispensably used world-wide. This section elaborates on some of the real time applications which can use Essential Grid.

Applications with High Frequency Updates

Grid can be used in applications with frequent updates, for example stock values in share market. When grid is switched over to virtual mode, it reforms itself as a light weight control that consumes a very little memory and processing power, and provides a very small latency under heavy load. Such virtual grids are typically useful when there is a need to display enormous data very quickly.

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

Excel-like UI Applications

Another important application is Excel like UI that simulates Microsoft Excel 2007 and gives an appearance that resembles excel. This application exhibits the following excel characteristics:

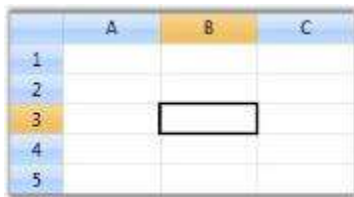
- Excel like Current Cell
- Formula Cells
- Excel like Selection Frame
- Markup Headers
- Workbook of sheets

a. Excel-like Current Cell

You can select a current cell in the Grid, similar to the current cell behavior in Microsoft Excel. This feature can be enabled, by setting `GridModelOptions.ExcelLikeCurrentCell` property to `true`, as follows:

C#

```
grid.Model.Options.ExcelLikeCurrentCell = true;
```



Note: If you have selected a current cell within a specified range, and when you move the current cell selection out of this range, the range will be cleared.

b. Excel-like Selection Frame

The active selection can be outlined with a selection frame by setting the `GridModelOptions.ExcelLikeSelectionFrame` property to `true`, as follows:

C#

```
grid.Model.Options.ExcelLikeSelectionFrame = true;
```

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						

c. Formula Cells

As we discussed in the previous chapter, Grid control provides complete support to formula cells. It can be enabled for the grid by setting the format string, FormulaCell to the TableStyle.CellType property, as follows:

C#

```
grid.Model.TableStyle.CellType = "FormulaCell";
```

	B2	Fx	=SUM(A1:A3)	
	A	8	C	D
1	200			
2	350	670		
3	120			
4				

d. Markup Headers

In Excel, whenever a selection is made, the headers of those rows and columns which are involved in the selection will be highlighted. You can get a similar behavior in the Grid using the OnPrepareRenderCell event.

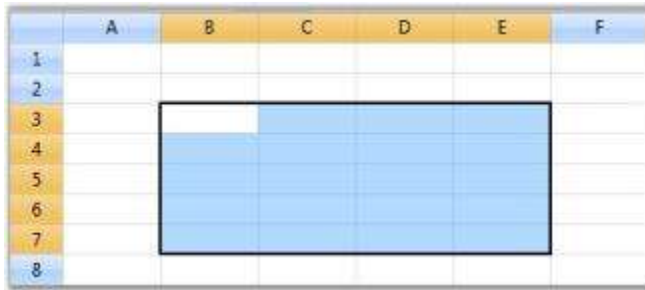
OnPrepareRenderCell event-This event will be triggered for every cell when they are about to be rendered. Hence, using this event, the cells that are going to be rendered are identified and their headers are highlighted.

The following code illustrates how to handle this event:

C#

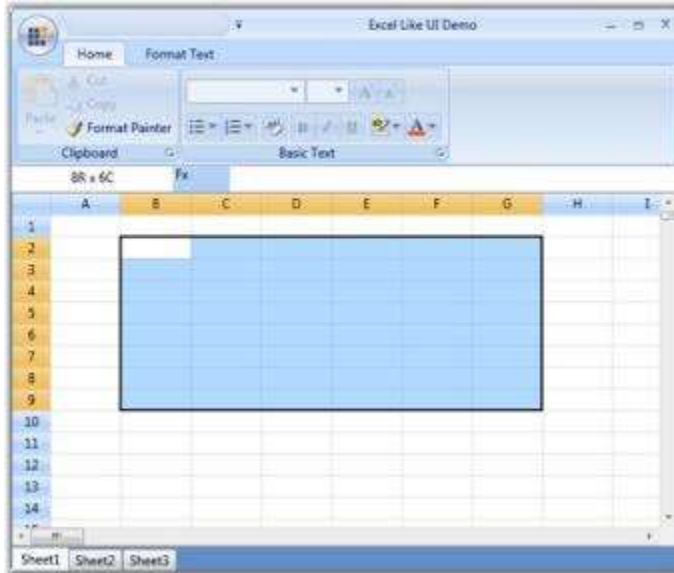
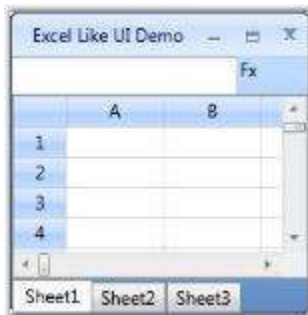
```
protected override void OnPrepareRenderCell(GridPrepareRenderCellEventArgs
e)
{
    base.OnPrepareRenderCell(e);
    if (e.Cell.RowIndex == 0 &&
        Model.SelectedRanges.AnyRangeIntersects(GridRangeInfo.Col(e.Cell.ColumnIndex
)))
    {
        e.Style.Background = this.excelOrange;
    }
    else if (e.Cell.ColumnIndex == 0 &&
        Model.SelectedRanges.AnyRangeIntersects(GridRangeInfo.Row(e.Cell.RowIndex)))
    {
        e.Style.Background = this.excelOrange;
    }
}
```

```
}
```



e. Workbook of Sheets

You can create a workbook with multiple sheets similar to excel, using a Tab control, where individual tab represents a worksheet embedded within a Grid control.



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

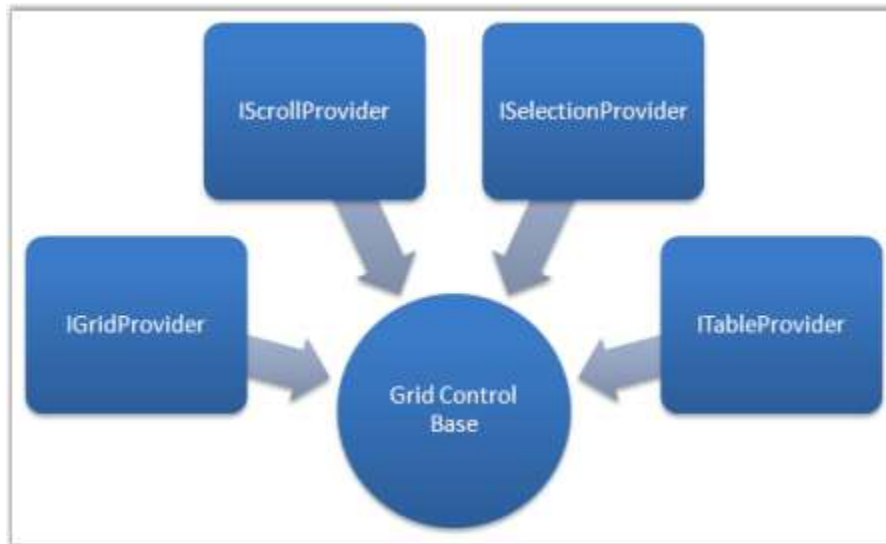
Testing in WPF GridControl

GridControl provides support for UI Automation and VS2010 Coded UI testing. This section provides the detailed description about using them with the Grid.

Grid Control UI Automation Support

Microsoft UI Automation provides a single, generalized interface that automation clients can examine or use to operate the user interfaces of a variety of platforms and frameworks. For more information, see <https://docs.microsoft.com/en-us/dotnet/desktop/wpf/controls/ui-automation-of-a-wpf-custom-control>.

With the Grid control, UI Automation is enabled for writing testable applications. It involves the patterns below.

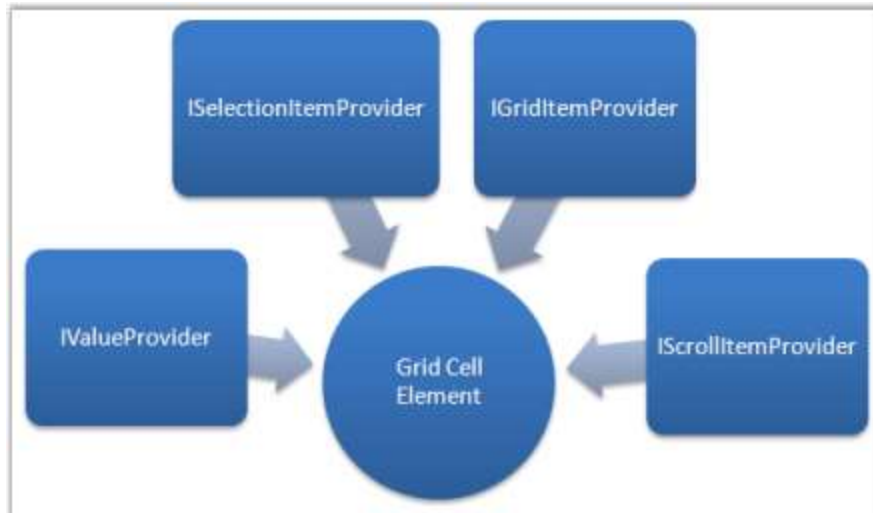


Grid control is customized for maximum performance, and thus the visuals are always virtualized. Automation Elements are generated for these live visuals alone. The different sets of automation providers implemented, provide access to the inner elements.

Following are the different sets of identifiers that can be obtained for the Grid:

- GridPattern
- TablePattern
- SelectionPattern
- ScrollPattern

Each cell in the grid is considered as an Automation Element, which in itself has some providers implemented. The following figure displays the different sets of identifiers for a Grid Cell Element.



Following are the different set of identifiers that can be obtained for each Grid Cell Element:

- GridItemPattern
- ValuePattern
- SelectionItemPattern
- ScrollItemPattern

Note: With NUnit or any other test frameworks, using TestApi from codeplex.com makes it quite easy to write unit tests. We are not recommending/fixing any issues with TestApi, it is an open source library from Microsoft.

Using UI Automation Patterns

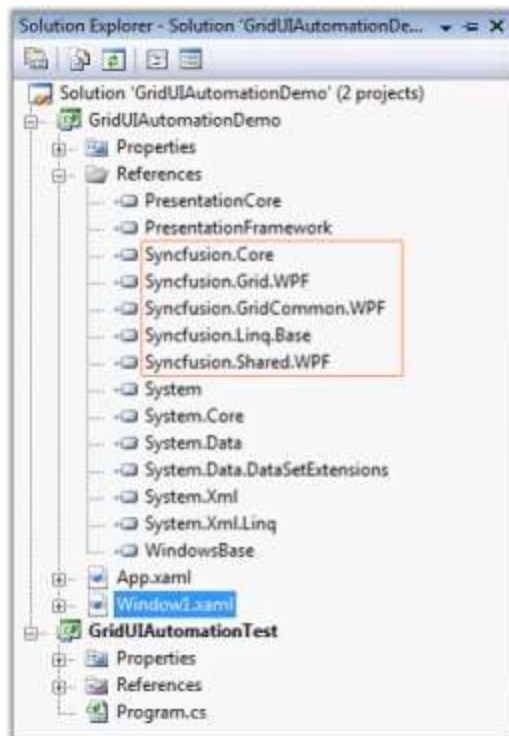
Let us walkthrough the following sample application that demonstrates the usage of UI Automation using a Console application.

API Usage

Setting up an Automation Sample is very useful to understand the usage of API for Automation Peer in Grid control. Automation Elements are run on a different thread from the main GUI thread.

The following set of instructions illustrates the same.

1. Create a WPF sample application with references added up for Syncfusion assemblies.



2. Create a console project as shown below.

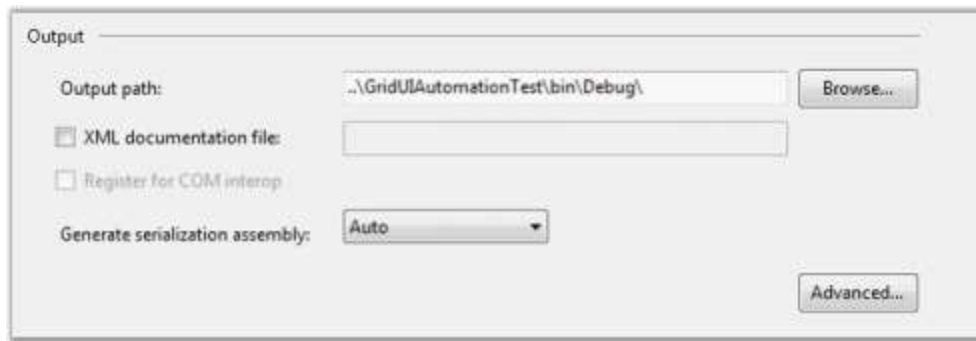
XML

```
<Grid>
<ScrollViewer CanContentScroll="True" HorizontalScrollBarVisibility="Auto"
VerticalScrollBarVisibility="Auto">
<syncfusion:GridControl x:Name="grid" />
</ScrollViewer>
</Grid>
```

C#

```
private void InitGrid()
{
    var model = this.grid.Model;
    model.RowCount = 10;
    model.ColumnCount = 10;
    model.QueryCellInfo += (s, e) =>
    {
        e.Style.CellValue = string.Format("Cell {0} / {1}", e.Cell.RowIndex,
        e.Cell.ColumnIndex);
    };
}
```

3. Enter the path where the output of the Window sample has to be saved, in the Output path field.



Note: Mention the output path as the Console application's bin\Debug directory.

The following sample code uses TestApi assemblies.

C#

```
// Initialization Code.
private static AutomationElement GetGridAut(out AutomatedApplication app)
{
    string sampleAppPath = "GridUIAutomationDemo.exe";
    app = new OutOfProcessApplication(new OutOfProcessApplicationSettings
    {
        ProcessStartInfo = new ProcessStartInfo(sampleAppPath),
        ApplicationImplementationFactory = new
        UIAutomationOutOfProcessApplicationFactory()
    });
    app.Start();
    app.WaitForMainWindow(TimeSpan.FromSeconds(15));
    var rootElement = app.MainWindow as AutomationElement;
    var grid = rootElement.AsQueryable(TreeScope.Descendants).First(o =>
    o.ClassName == "GridControl" && o.ControlType ==
    ControlType.DataGrid);
    return grid;
}
```

Note: We have added minimal set of LINQ-to-UIAutomation classes that would translate the LINQ query for searching the AutomationElement from the root hierarchy. With LINQ-To-UIAutomation library, only First method is supported now.

The Grid Automation element is obtained.

Obtaining the Automation Pattern

Once you get the actual Grid's Automation Element, you can then make use of different Patterns supported by the control. The following code example illustrates the same.

C#

```
var gridPattern = grid.GetCurrentPattern(GridPatternIdentifiers.Pattern) as
GridPattern;
var item = gridPattern.GetItem(1, 1);
if (item != null)
{
    object value = null;
    item.TryGetCurrentPattern(ValuePatternIdentifiers.Pattern, out value);
    var valueProvider = value as ValuePattern;
```

```
var cellValue = valueProvider.Current.Value;  
Console.WriteLine("Item at [1,1] - {0}", cellValue);  
}
```

VS 2010 Coded UI Testing

Essential Grid WPF now supports automated UI testing with VS 2010 Coded UI technology. The Grid Test plugin blends in with the automated UI Testing Framework in VS 2010 by implementing the following classes:

- UITechnologyManager
- UITestPropertyProvider
- UIActionFilter

You can get the Extension Project of Coded UI from [this](#) location.

The architectural diagram is as follows:



- Grid Test Plugin implements the necessary details to communicate with the VS 2010 Test Framework.
- The Grid application host runs with a .NET Remoting channel hosted internally to communicate with the Test plugin through an interface. The data is then channeled across to the VS 2010 Test Framework to identify the Cells.

Properties

Following are the properties exposed in Coded UI Testing:

- CellValue
- Text
- Description
- SelectedRanges
- Background
- Foreground
- Borders

- Format
- FormulaTag

Creating a Coded UI Test With Essential Grid WPF

Initial steps before creating the Coded UI Test project:

- Prepare the Grid sample application.
- Write Unit tests using VS 2010.
- Testing the application with generated Coded UI Tests.

Preparing the Grid application

1. Syncfusion.VisualStudio.TestTools.UITest.GridExtensions.dll contains implementation to easily change an existing

application to the test application that the plugin would require. Add a reference to this assembly.

2. Open App.xaml.

Note: The following code appears.

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 Application to Syncfusion:GridControlTestApplication as follows.

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.

C#

```
namespace WpfApplication3
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
```

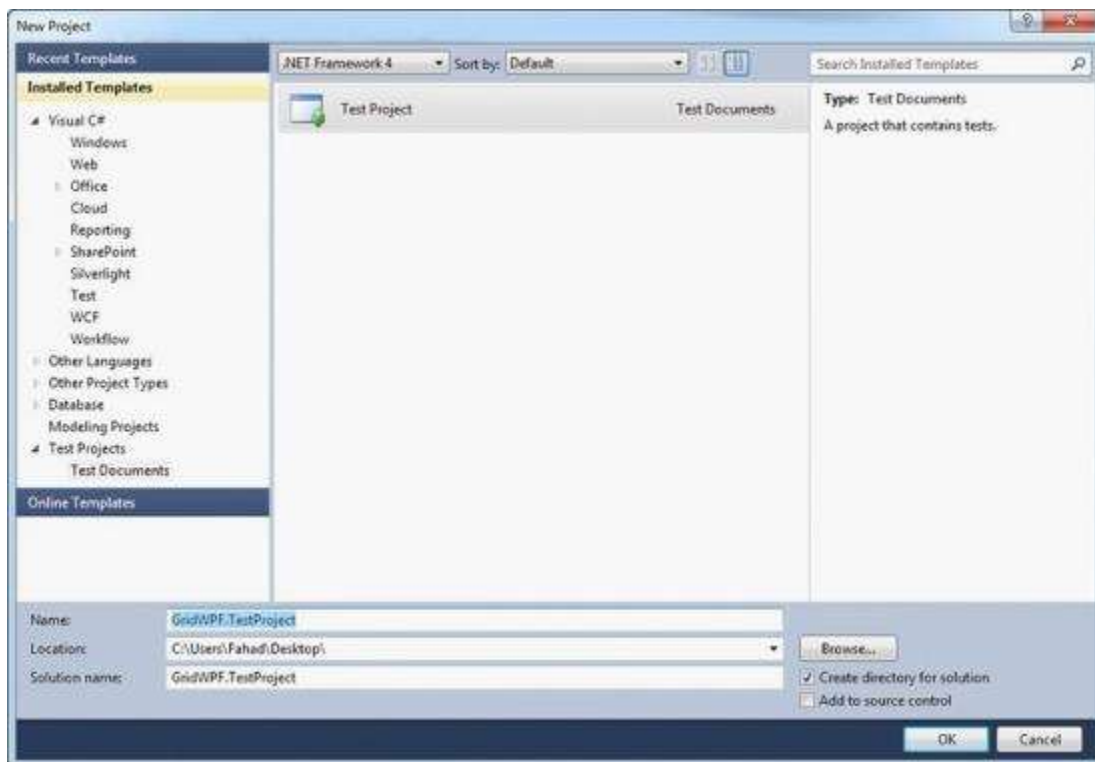
```
public partial class App : GridControlTestApplication
{
}
}
```

5. Build the application to make it ready for testing.

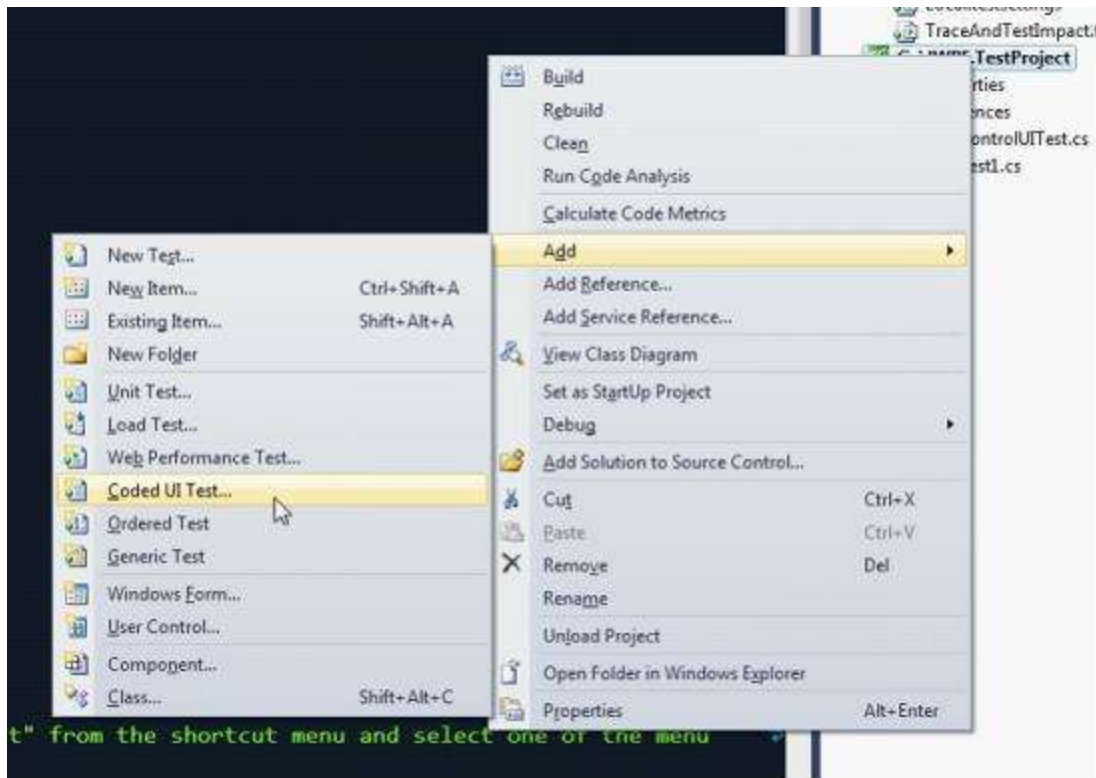
Note: The GridControlTestApplication works only with a single Grid instance in the window. For multiple instances the IGridInteropService interface has to be implemented.

Creating Unit Tests with VS2010

1. Create a new Test Project in VS2010,



2. Add a new Coded UI Test item for the project.



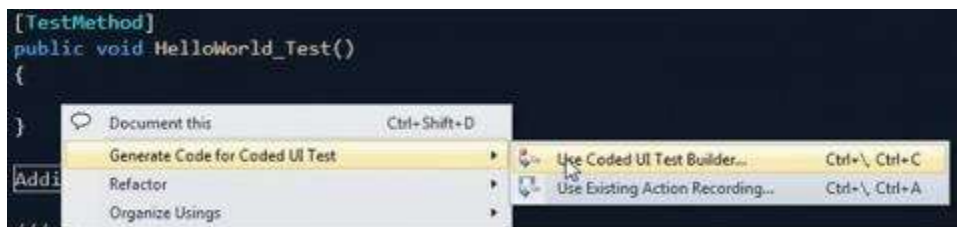
3. Add a TestMethod called HelloWorld_Test.

The following code illustrates this.

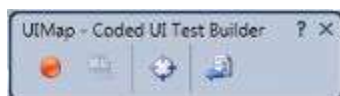
C#

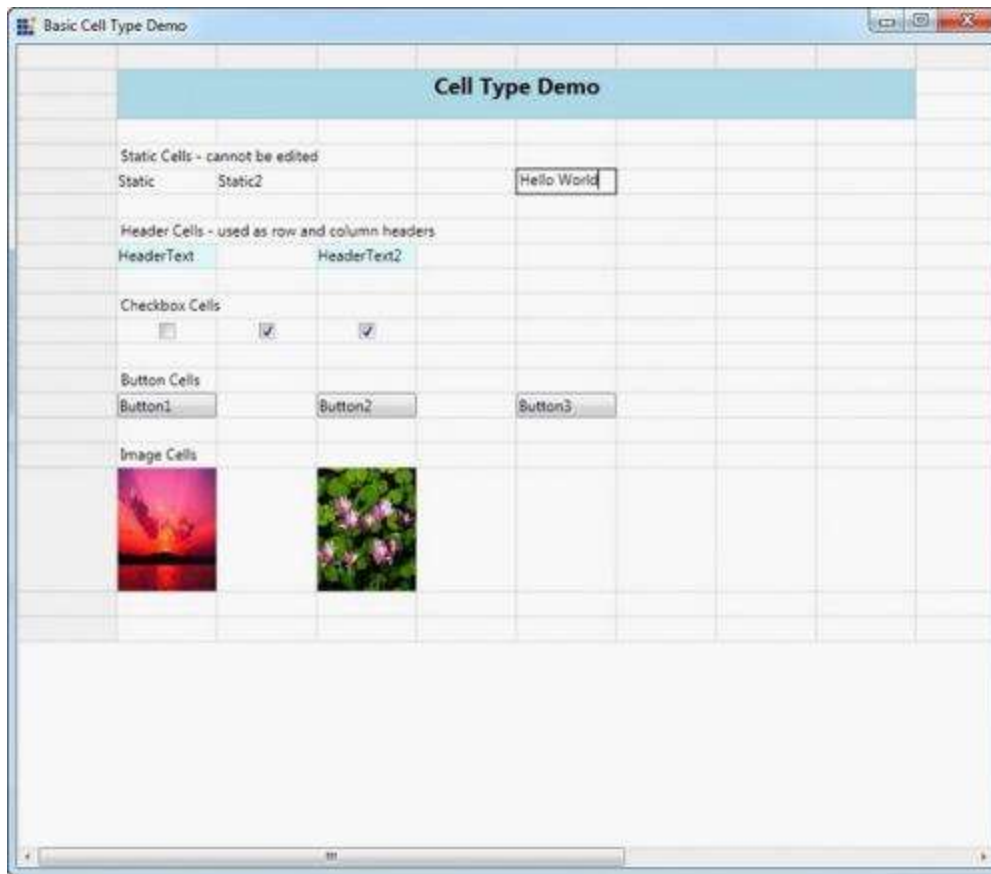
```
public void HelloWorld_Test()
{
}
```

4. Build and run the Grid application that you configured. * Right-click on the TestMethod body and then select as below,

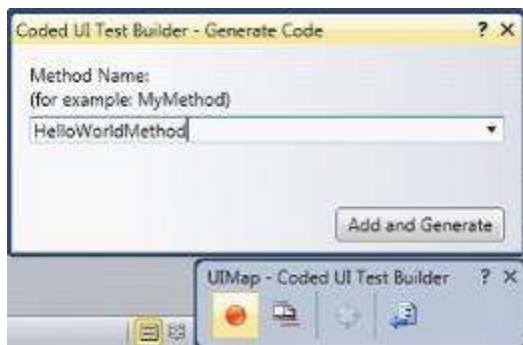


5. Click on the record button to perform actions, you can add a Hello World text in a grid cell [x, y] in this scenario.



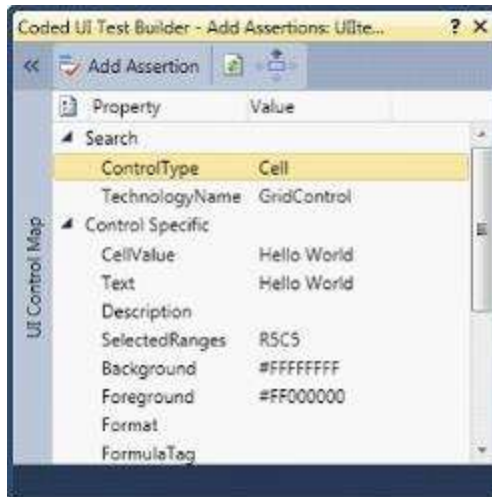


6. Click on Generate code in the Coded UI Test Builder.



7. You can assert the cell value using the cross-hair present in the Coded UI Test builder. 8. Click on the cross-hair and hover to the Hello World cell.

Note: The assert window is displayed as below.



9. Add asserts to the properties displayed in the assert window and generate the assert method.
10. Close the Coded UI Test builder.

Note: Coded UI Unit Test is created.

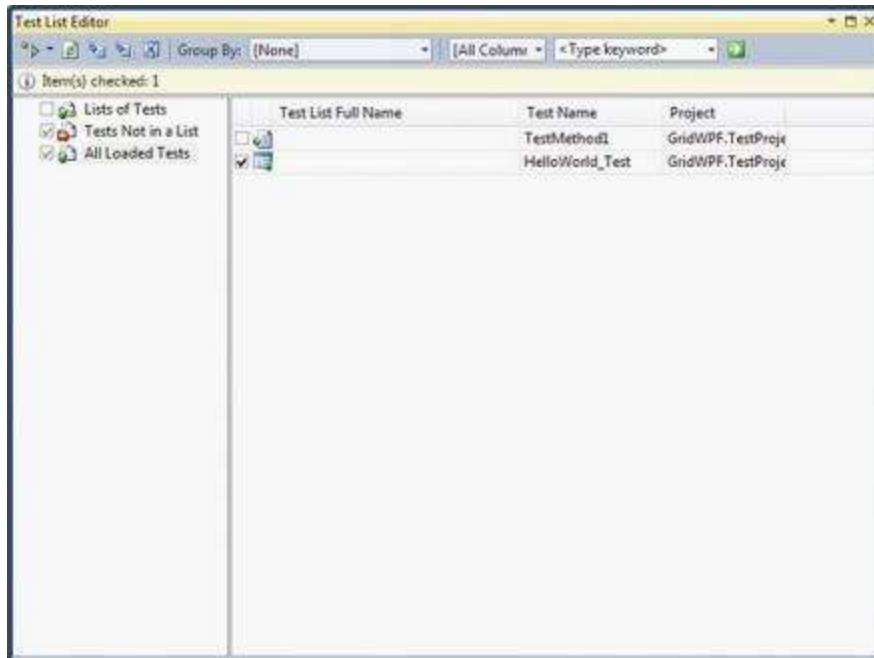
The following code generates automatically.

C#

```
public void HelloWorld_Test()  
{  
    this.UIMap.HelloWorldMethod();  
    this.UIMap.HelloWorldAssert();  
}
```

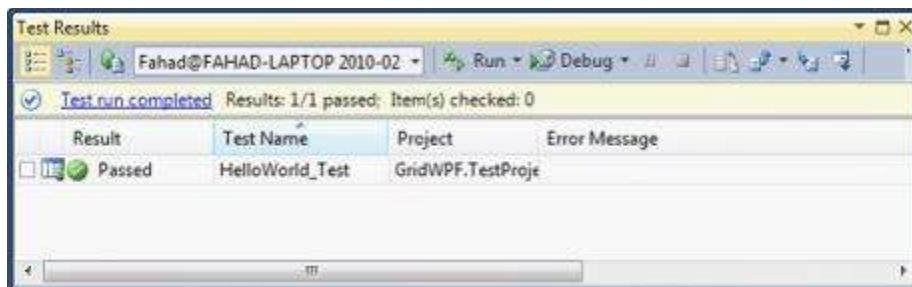
Testing the Application with Generated Coded UI Test

1. Build and run the Grid WPF application.
2. Build the test project and run the Unit tests using VS 2010.



3. Run the test HelloWorld_Test.
 - This will automatically focus the running Grid application and perform automated testing.

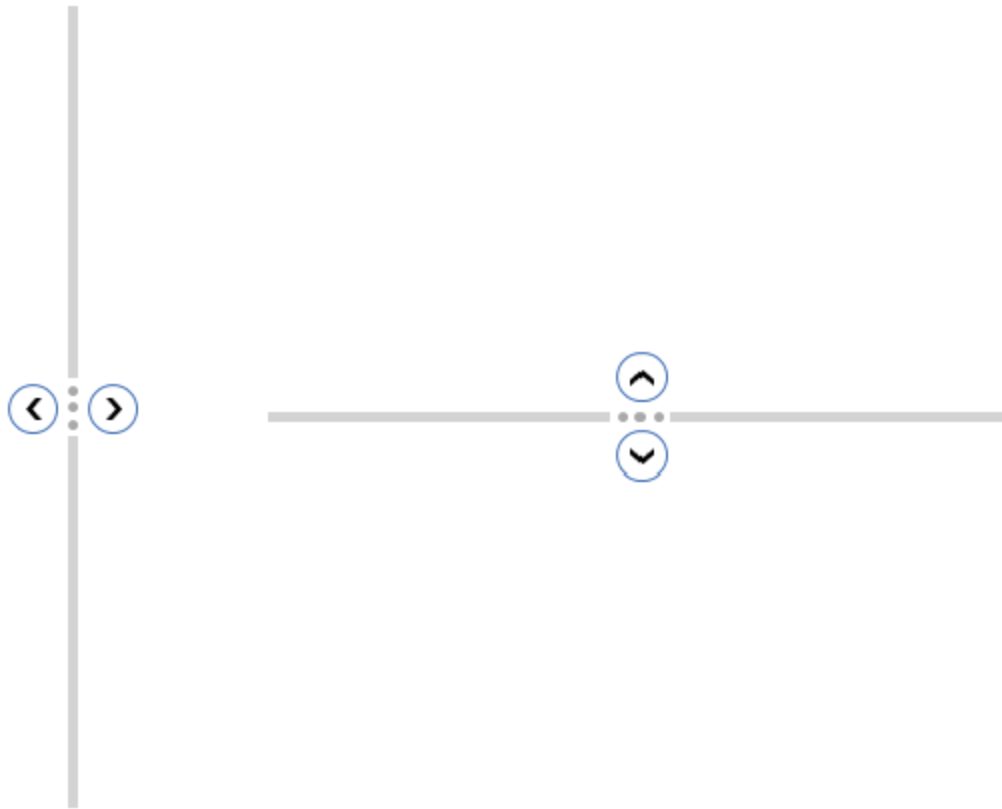
Note: The sample test procedure mentioned above requires the sample to be run individually, in normal automated testing, this would be included in the TestInitialize method to start the application.



SfGridSplitter

WPF GridSplitter (SfGridSplitter) Overview

[SfGridSplitter](#) is a container control that helps to split the available space horizontally or vertically with a movable splitter and arrange the visual elements inside it.



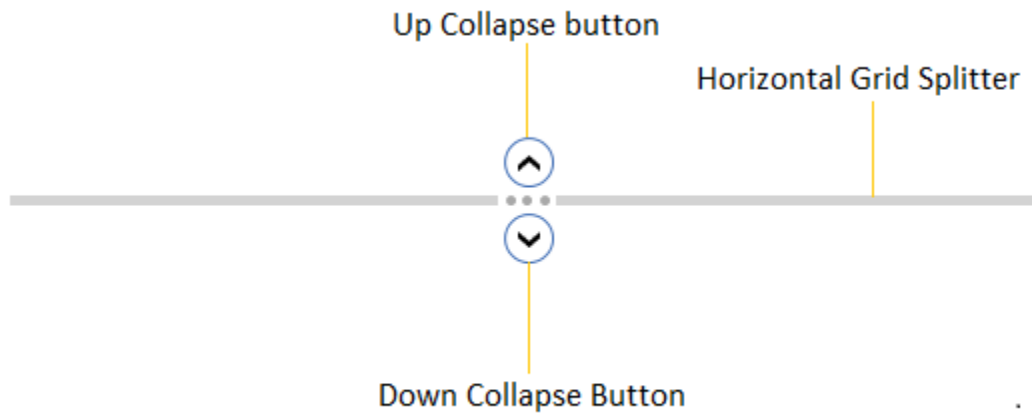
Features of SfGridSplitter

- Dynamic resizing: Splits available space with movable splitter that helps resize controls on demand.
- Expand / Collapse: Supports to expand and collapse splitter controls interactively in UI.
- Orientation: Supports both horizontal and vertical orientation to split controls based on user layout.

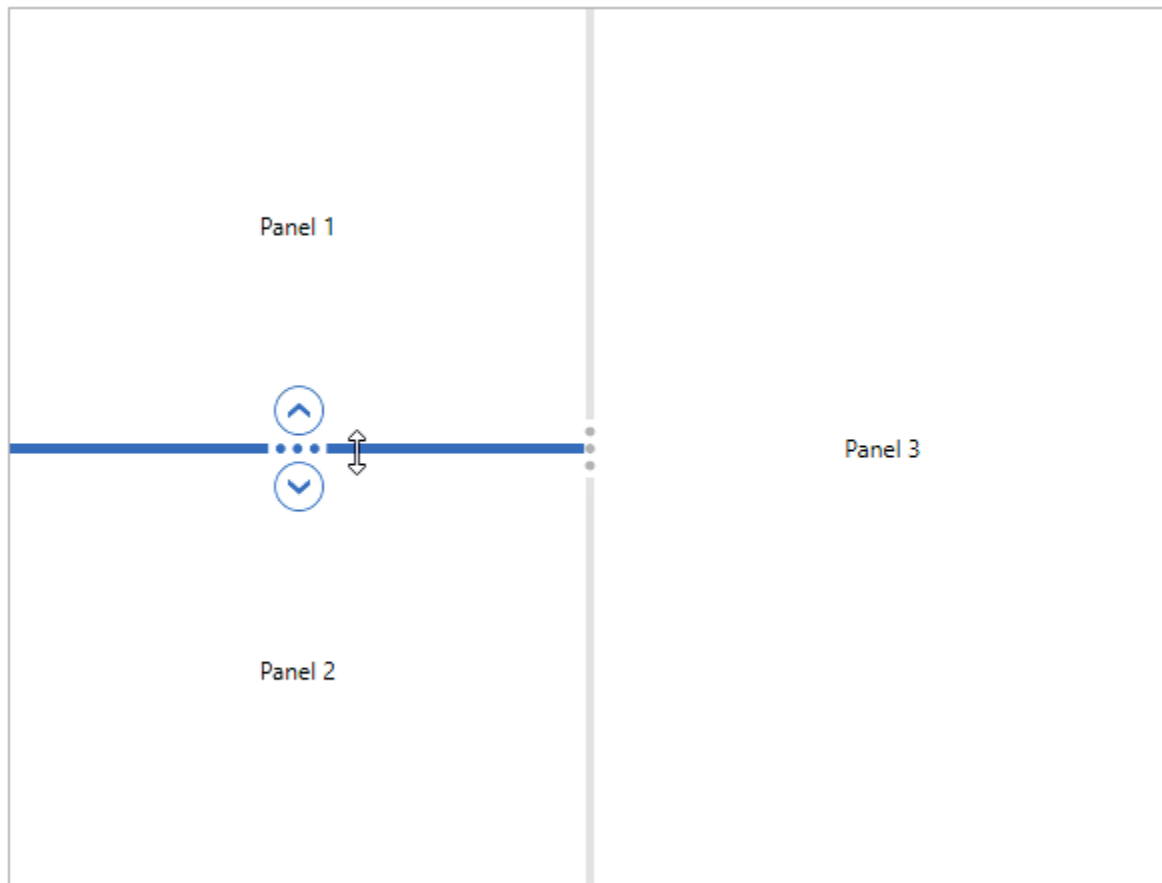
Getting Started with WPF GridSplitter (SfGridSplitter)

This section explains how to create a [WPF GridSplitter](#) (SfGridSplitter) and explains about its structure.

Structure of SfGridSplitter



Visual representation



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 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* *Syncfusion.SfShared.WPF* 2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page. 3. Declare the **SfGridSplitter** 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="SfGridSplitterSample.MainWindow"
Title="SfGridSplitter Sample" Height="350" Width="525">
<Grid>
<!-- Adding SfGridSplitter control -->
<syncfusion:SfGridSplitter x:Name="sfGridSplitter"
HorizontalAlignment="Stretch"/>
</Grid>
</Window>
```

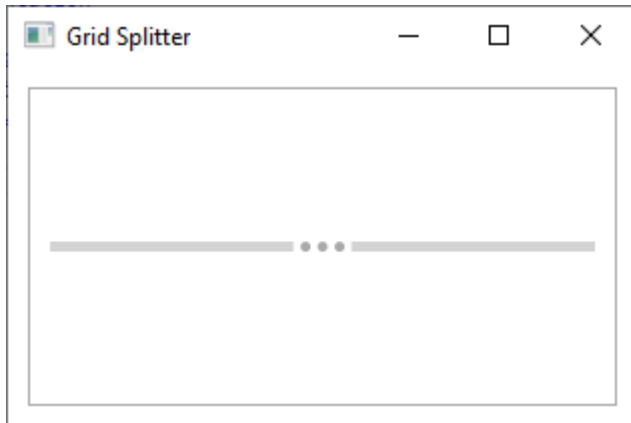
Add control manually in C\#

To add the control manually in C#, follow the given steps:

1. Add the following required assembly references to the project: *Syncfusion.SfInput.WPF* *Syncfusion.SfShared.WPF* 2. Import the **SfGridSplitter** namespace using **Syncfusion.Windows.Controls.Input**; 3. Create an **SfGridSplitter** instance, and add it to the window.

C#

```
using Syncfusion.Windows.Controls.Input;
namespace SfGridSplitterSample {
public partial class MainWindow : Window {
public MainWindow() {
InitializeComponent();
//Creating an instance of SfGridSplitter control
SfGridSplitter sfGridSplitter = new SfGridSplitter();
sfGridSplitter.HorizontalAlignment = HorizontalAlignment.Stretch;
//Adding SfGridSplitter as window content
this.Content = sfGridSplitter;
}
}
}
```



Note: [View Sample in GitHub](#)

Resize the grid rows

If we want to resize the specific grid rows, place the `SfGridSplitter` on next or previous row and set the `HorizontalAlignment` property as `Stretch` and `ResizeBehavior` property value as `PreviousAndNext`.

XML

```
<Border
Margin="10"
BorderBrush="DarkGray"
BorderThickness="1">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition Height="auto" />
      <RowDefinition />
    </Grid.RowDefinitions>
    <TextBlock Grid.Row="0"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      TextAlignment="Center"
      Text="Panel 1">
    </TextBlock>
    <TextBlock Grid.Row="2"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      TextAlignment="Center"
      Text="Panel 2">
    </TextBlock>
    <!--Grid Splitter-->
    <syncfusion:SfGridSplitter Name="gridSplitter"
      HorizontalAlignment="Stretch"
      ResizeBehavior="PreviousAndNext"
      Width="auto"
      Grid.Row="1">
    </syncfusion:SfGridSplitter>
  </Grid>
</Border>
```



Note: [View Sample in GitHub](#)

Resize the grid columns

If we want to resize the specific grid columns, place the `SfGridSplitter` on next or previous column and set the `VerticalAlignment` property as `Stretch` and `ResizeBehavior` property value as `PreviousAndNext`.

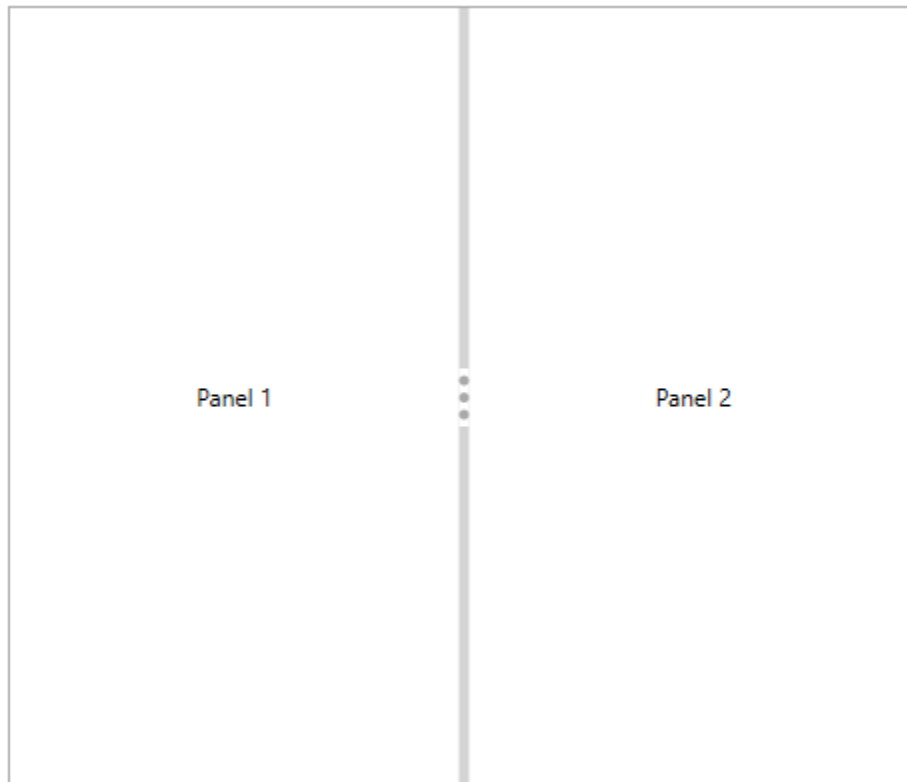
XML

```
<Border
Margin="10"
BorderBrush="DarkGray"
BorderThickness="1">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition Width="auto" />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <TextBlock Grid.Column="0"
HorizontalAlignment="Center"
VerticalAlignment="Center"
TextAlignment="Center"
Text="Panel 1">
    </TextBlock>
    <TextBlock Grid.Column="2"
HorizontalAlignment="Center"
VerticalAlignment="Center"
TextAlignment="Center"
Text="Panel 2">
```

```

</TextBlock>
<!--Grid Splitter-->
<syncfusion:SfGridSplitter Name="gridSplitter"
HorizontalAlignment="Stretch"
ResizeBehavior="PreviousAndNext"
Width="auto"
Grid.Column="1">
</syncfusion:SfGridSplitter>
</Grid>
</Border>

```



Note: [View Sample in GitHub](#)

Note: We can restrict the moving location of the grid splitter (Left or Right or Bottom or Top) by setting the value for `ResizeBehavior` property. The `ResizeBehavior` value must be assigned based on the row or column where the grid splitter placed.

Resizing the grid rows and columns with specific pixel

If we want to resize the rows or columns of grid with particular pixel interval, Set the pixel value for `DragIncrement` and `KeyboardIncrement` properties. If we move the splitter using the mouse, `DragIncrement` property values is used as resize pixel interval. If we move the splitter using the Up-Down buttons, `KeyboardIncrement` property values is used as resize pixel interval. The default value of `DragIncrement` property is `1` and `KeyboardIncrement` property is `20`.

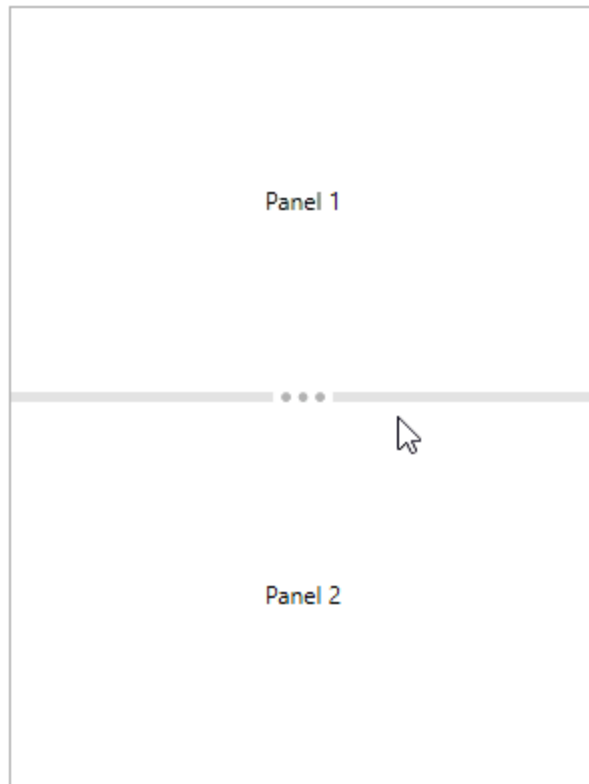
XML

```

<Border
Margin="10"

```

```
BorderBrush="DarkGray"
BorderThickness="1">
<Grid>
<Grid.RowDefinitions>
<RowDefinition />
<RowDefinition Height="auto" />
<RowDefinition />
</Grid.RowDefinitions>
<TextBlock Grid.Row="0"
HorizontalAlignment="Center"
VerticalAlignment="Center"
TextAlignment="Center"
Text="Panel 1">
</TextBlock>
<TextBlock Grid.Row="2"
HorizontalAlignment="Center"
VerticalAlignment="Center"
TextAlignment="Center"
Text="Panel 2">
</TextBlock>
<!--Grid Splitter-->
<syncfusion:SfGridSplitter DragIncrement="50"
KeyboardIncrement="50"
HorizontalAlignment="Stretch"
Width="auto"
Grid.Row="1">
</syncfusion:SfGridSplitter>
</Grid>
</Border>
```



Note: [View Sample in GitHub](#)

Resize the grid rows or columns programmatically

We can move the splitter and resize the affected columns or rows programmatically with certain pixels by passing the pixel value in [MoveSplitter\(Double\)](#) method.

XML

```
<Border
Margin="10"
BorderBrush="DarkGray"
BorderThickness="1">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition Height="auto" />
      <RowDefinition />
    </Grid.RowDefinitions>
    <TextBlock Grid.Row="0"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      TextAlignment="Center"
      Text="Panel 1">
    </TextBlock>
    <TextBlock Grid.Row="2"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      TextAlignment="Center"
      Text="Panel 2">
  </Grid>
</Border>
```



```

</TextBlock>
<!--Grid Splitter-->
<syncfusion:SfGridSplitter Name="gridSplitter"
HorizontalAlignment="Stretch"
ResizeBehavior="PreviousAndNext"
Width="auto"
Grid.Row="1">
</syncfusion:SfGridSplitter>
</Grid>
</Border>

```

C#

```

//GridSplitter moves 50 pixels.
gridSplitter.MoveSplitter(50);

```

Show or hide the grid row and columns

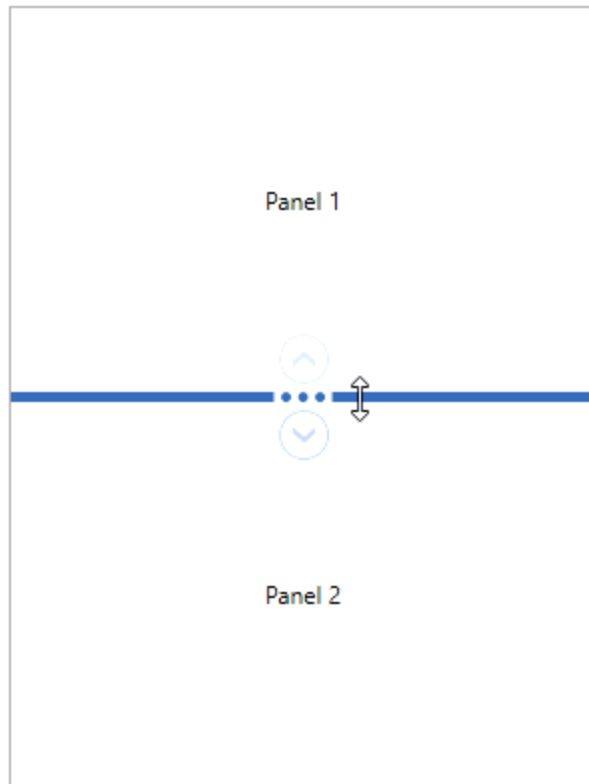
We can collapse or expands the element in either side of the splitter by clicking the collapse buttons. We can show or hide the collapse button by using the [EnableCollapseButton](#) property value as `true`. The default value of `EnableCollapseButton` is `false`.

XML

```

<Border
Margin="10"
BorderBrush="DarkGray"
BorderThickness="1">
<Grid>
<Grid.RowDefinitions>
<RowDefinition />
<RowDefinition Height="auto" />
<RowDefinition />
</Grid.RowDefinitions>
<TextBlock Grid.Row="0"
HorizontalAlignment="Center"
VerticalAlignment="Center"
TextAlignment="Center"
Text="Panel 1">
</TextBlock>
<TextBlock Grid.Row="2"
HorizontalAlignment="Center"
VerticalAlignment="Center"
TextAlignment="Center"
Text="Panel 2">
</TextBlock>
<!--Grid Splitter-->
<syncfusion:SfGridSplitter EnableCollapseButton="True"
HorizontalAlignment="Stretch"
Width="auto"
Grid.Row="1" >
</syncfusion:SfGridSplitter>
</Grid>
</Border>

```



Note: [View Sample in GitHub](#)

Custom UI for Collapse buttons

If you want to change the UI of horizontal splitter up and down collapse button separately, use the [UpButtonTemplate](#) and [DownButtonTemplate](#) properties. If you want to change the UI of vertical splitter left and right collapse button, use the [LeftButtonTemplate](#) and [RightButtonTemplate](#) properties.

Note: You can see the effect of collapse button templates only on when `EnableCollapseButton` property value is `true`.

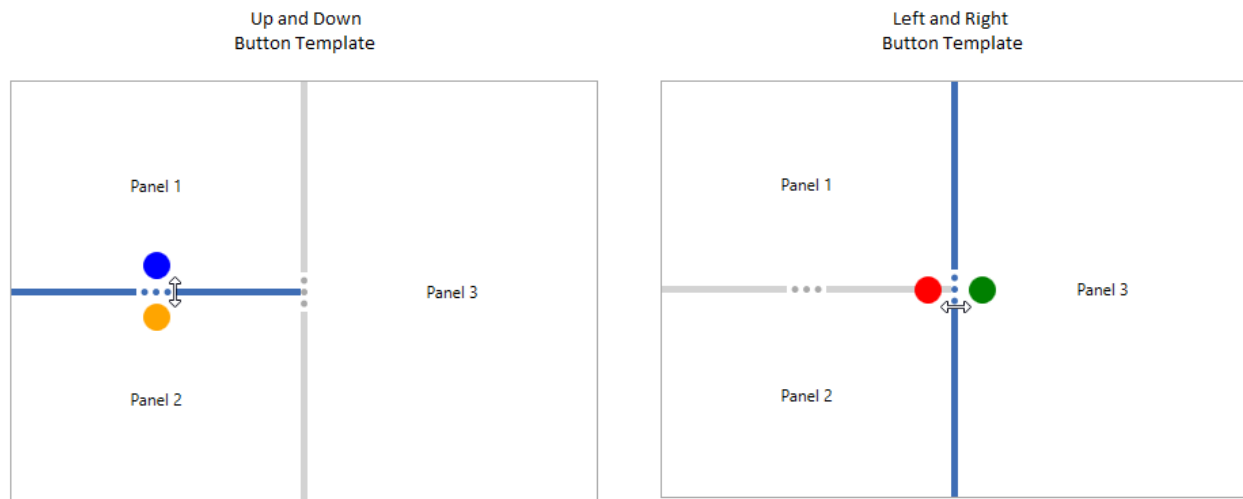
XML

```
<Border
Margin="10"
BorderBrush="DarkGray"
BorderThickness="1">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition Height="auto" />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition Width="auto" />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
```

```

Grid.Row="0"
Grid.Column="0"
TextAlignment="Center"
Text="Panel 1"/>
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Grid.Row="2"
Grid.Column="0"
Text="Panel 2"/>
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Grid.RowSpan="3"
Grid.Column="2"
Text="Panel 3"/>
<!--Horizontal Splitter-->
<syncfusion:SfGridSplitter Grid.Row="1"
Grid.Column="0"
Height="5"
HorizontalAlignment="Stretch"
EnableCollapseButton="True"
ResizeBehavior="PreviousAndNext">
<!--Up button template-->
<syncfusion:SfGridSplitter.UpButtonTemplate>
<DataTemplate>
<Ellipse Width="20" Height="20" Fill="Blue"/>
</DataTemplate>
</syncfusion:SfGridSplitter.UpButtonTemplate>
<!--Down button template-->
<syncfusion:SfGridSplitter.DownButtonTemplate>
<DataTemplate>
<Ellipse Width="20" Height="20" Fill="Orange"/>
</DataTemplate>
</syncfusion:SfGridSplitter.DownButtonTemplate>
</syncfusion:SfGridSplitter>
<!--Vertical Splitter-->
<syncfusion:SfGridSplitter Grid.RowSpan="3"
Grid.Column="1"
Width="5"
VerticalAlignment="Stretch"
EnableCollapseButton="True"
ResizeBehavior="PreviousAndNext"
ShowsPreview="True">
<!--Left button template-->
<syncfusion:SfGridSplitter.LeftButtonTemplate>
<DataTemplate>
<Ellipse Width="20" Height="20" Fill="Red"/>
</DataTemplate>
</syncfusion:SfGridSplitter.LeftButtonTemplate>
<!--Right button template-->
<syncfusion:SfGridSplitter.RightButtonTemplate>
<DataTemplate>
<Ellipse Width="20" Height="20" Fill="Green"/>
</DataTemplate>
</syncfusion:SfGridSplitter.RightButtonTemplate>
</syncfusion:SfGridSplitter>
</Grid>
</Border>

```



Note: [View Sample in GitHub](#)

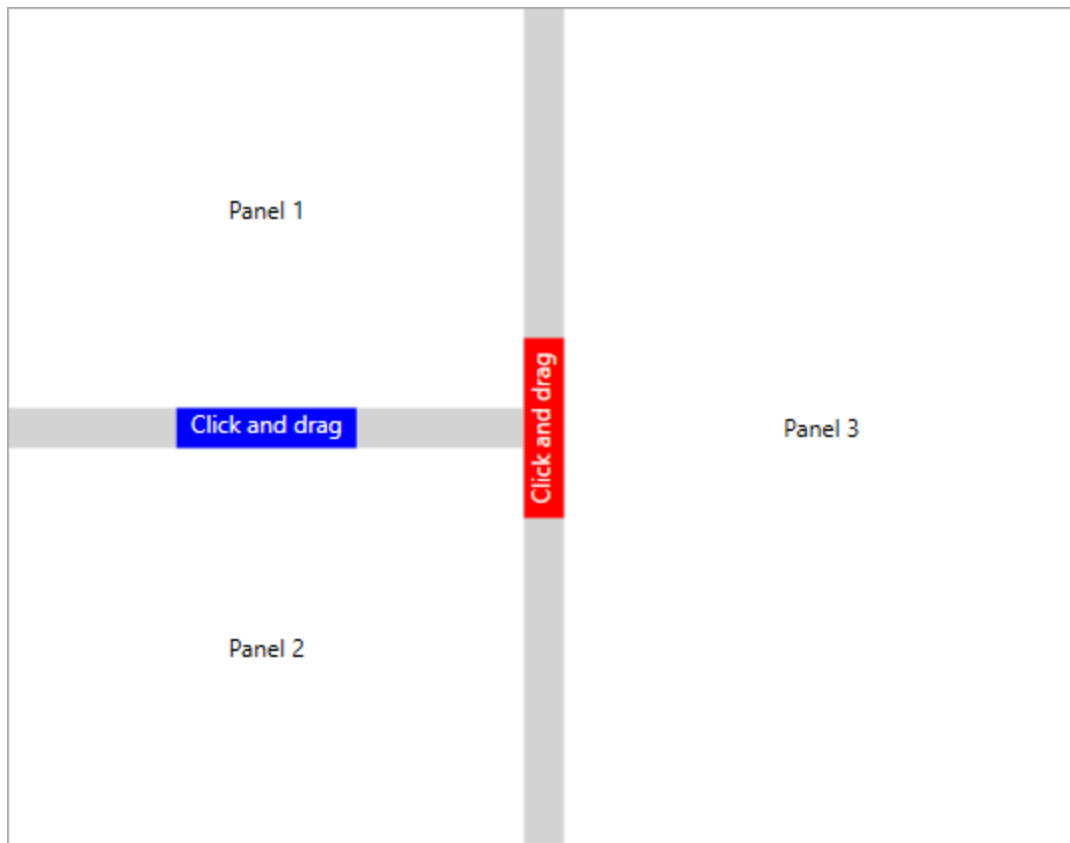
Custom UI for expander gripper

If you want to change the UI of vertical and horizontal splitter gripper separately, use the [VerticalGripperTemplate](#) and [HorizontalGripperTemplate](#) properties.

XML

```
<Border
Margin="10"
BorderBrush="DarkGray"
BorderThickness="1">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition Height="auto" />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition Width="auto" />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <TextBlock HorizontalAlignment="Center"
      VerticalAlignment="Center"
      Grid.Row="0"
      Grid.Column="0"
      Text="Panel 1" />
    <TextBlock HorizontalAlignment="Center"
      VerticalAlignment="Center"
      Grid.Row="2"
      Grid.Column="0"
      Text="Panel 2" />
    <TextBlock HorizontalAlignment="Center"
      VerticalAlignment="Center"
      Grid.RowSpan="3"
      Grid.Column="2"
      Text="Panel 3" />
```

```
<!--Horizontal Splitter-->
<syncfusion:SfGridSplitter Grid.Row="1"
Grid.Column="0"
Height="20"
HorizontalAlignment="Stretch"
EnableCollapseButton="True"
ResizeBehavior="PreviousAndNext"
ShowsPreview="True">
<!--Horizontal Gripper Template-->
<syncfusion:SfGridSplitter.HorizontalGripperTemplate>
<DataTemplate>
<TextBlock Background="Blue"
Foreground="White"
TextAlignment="Center"
VerticalAlignment="Center"
HorizontalAlignment="Center"
Text="Click and drag"
Width="90"
Height="20"/>
</DataTemplate>
</syncfusion:SfGridSplitter.HorizontalGripperTemplate>
</syncfusion:SfGridSplitter>
<!--Vertical Splitter-->
<syncfusion:SfGridSplitter Grid.RowSpan="3"
Grid.Column="1"
Width="20"
VerticalAlignment="Stretch"
EnableCollapseButton="True"
ResizeBehavior="PreviousAndNext"
ShowsPreview="True">
<!--Vertical GripperTemplate-->
<syncfusion:SfGridSplitter.VerticalGripperTemplate>
<DataTemplate>
<TextBlock Background="Red"
Foreground="White"
TextAlignment="Center"
VerticalAlignment="Center"
HorizontalAlignment="Center"
Text="Click and drag"
Width="90"
Height="20">
<TextBlock.LayoutTransform>
<RotateTransform Angle="-90"/>
</TextBlock.LayoutTransform>
</TextBlock>
</DataTemplate>
</syncfusion:SfGridSplitter.VerticalGripperTemplate>
</syncfusion:SfGridSplitter>
</Grid>
</Border>
```



Note: [View Sample in GitHub](#)

Deferred resizing

We can directly redistribute the row or columns by using `SfGridSplitter`. If we want to preview the location of redistributing row or columns before it changed, we can use the `ShowsPreview` property as `true`.

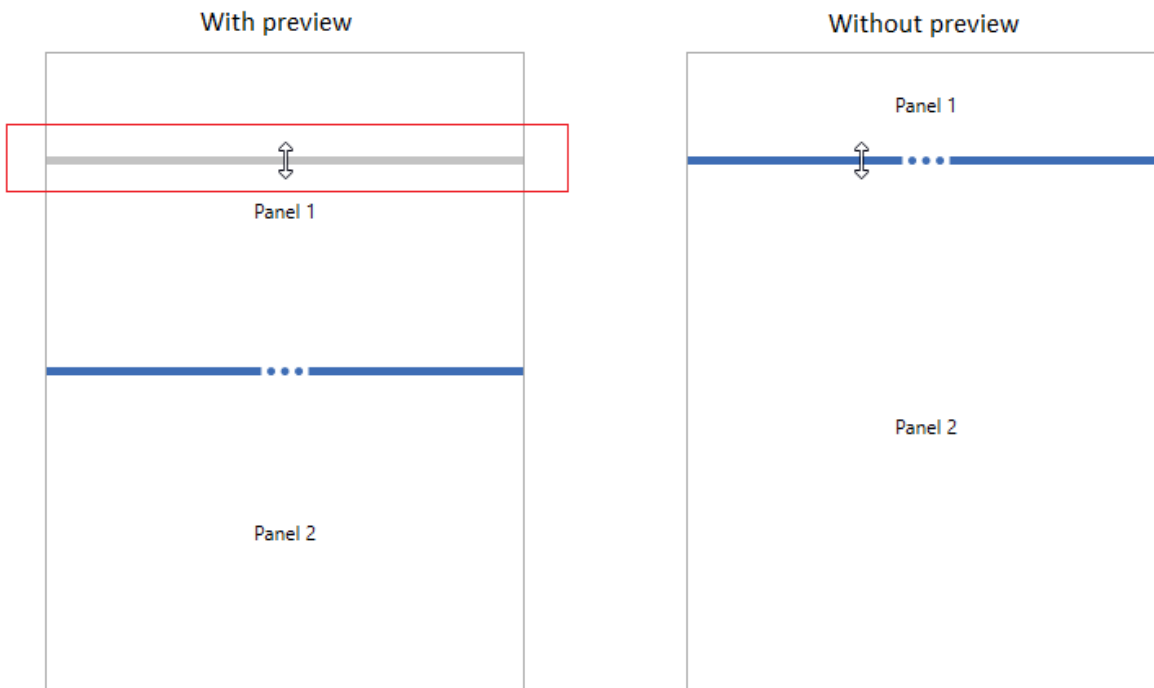
XML

```
<Border
  Margin="10"
  BorderBrush="DarkGray"
  BorderThickness="1">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition Height="auto" />
      <RowDefinition />
    </Grid.RowDefinitions>
    <TextBlock Grid.Row="0"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      TextAlignment="Center"
      Text="Panel 1">
    </TextBlock>
    <TextBlock Grid.Row="2"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
```

```

TextAlignment="Center"
Text="Panel 2">
</TextBlock>
<!--Grid Splitter-->
<syncfusion:SfGridSplitter ShowsPreview="True"
HorizontalAlignment="Stretch"
Width="auto"
Grid.Row="1" >
</syncfusion:SfGridSplitter>
</Grid>
</Border>

```



Note: [View Sample in GitHub](#)

Grid splitter for the merged columns or rows

If we want to resize the merged columns or rows, place the grid splitter on next or previous row or column of the grid.

XML

```

<Border
Margin="10"
BorderBrush="DarkGray"
BorderThickness="1">
<Grid>
<Grid.RowDefinitions>
<RowDefinition />
<RowDefinition Height="auto" />
<RowDefinition />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>

```

```
<ColumnDefinition />
<ColumnDefinition Width="auto" />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
Grid.Row="0"
Grid.Column="0" TextAlignment="Center"
Text="Panel 1">
</TextBlock>
<TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
Grid.Row="2"
Grid.Column="0" Text="Panel 2">
</TextBlock>
<TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
Grid.RowSpan="3"
Grid.Column="2" Text="Panel 3">
</TextBlock>
<!--Horizontal Splitter-->
<syncfusion:SfGridSplitter Grid.Row="1"
Grid.Column="0"
Height="5"
HorizontalAlignment="Stretch"
ResizeBehavior="PreviousAndNext">
</syncfusion:SfGridSplitter>
<!--Vertical Splitter-->
<syncfusion:SfGridSplitter Grid.RowSpan="3"
Grid.Column="1"
Width="5"
VerticalAlignment="Stretch"
ResizeBehavior="PreviousAndNext">
</syncfusion:SfGridSplitter>
</Grid>
</Border>
```

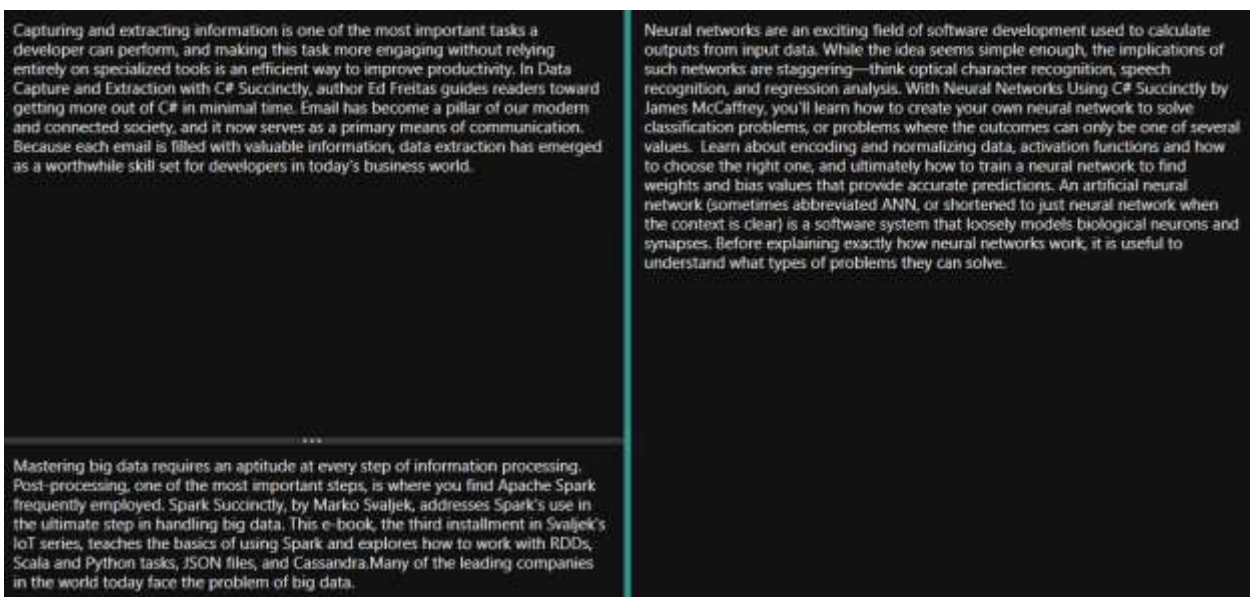



Note: [View Sample in GitHub](#)

Theme

SfGridSplitter supports various built-in themes. Refer to the below links to apply themes for the SfGridSplitter,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Appearance in WPF GridSplitter (SfGridSplitter)

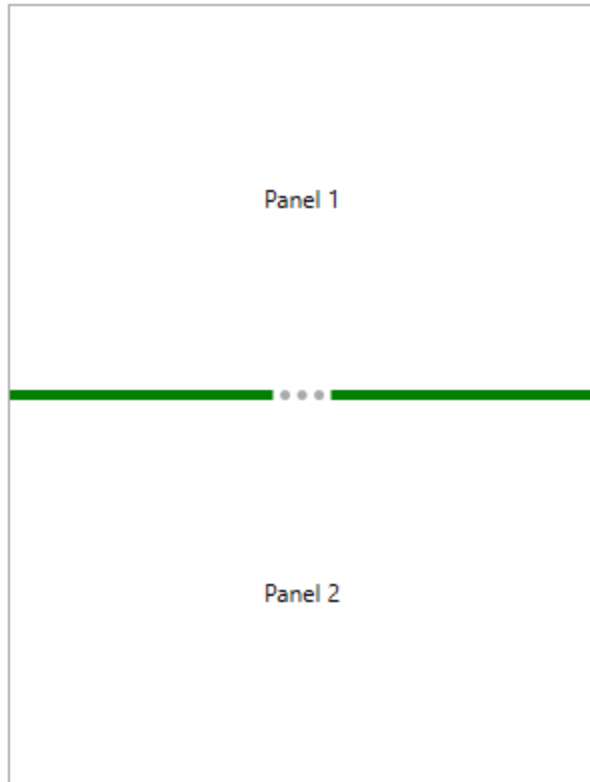
This section explains different UI customization and styling support available in [SfGridSplitter](#) control.

Setting the Background

We can change the background color of **SfGridSplitter** by setting the **Background** property. The default color value of **Background** property is **Light Gray**.

XML

```
<Border
Margin="10"
BorderBrush="DarkGray"
BorderThickness="1">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition Height="auto" />
      <RowDefinition />
    </Grid.RowDefinitions>
    <TextBlock Grid.Row="0"
HorizontalAlignment="Center"
VerticalAlignment="Center"
TextAlignment="Center"
Text="Panel 1">
    </TextBlock>
    <TextBlock Grid.Row="2"
HorizontalAlignment="Center"
VerticalAlignment="Center"
TextAlignment="Center"
Text="Panel 2">
    </TextBlock>
    <!--Grid Splitter-->
    <syncfusion:SfGridSplitter Background="Green"
HorizontalAlignment="Stretch"
Width="auto"
Grid.Row="1">
    </syncfusion:SfGridSplitter>
  </Grid>
</Border>
```



Note: [View Sample in GitHub](#)

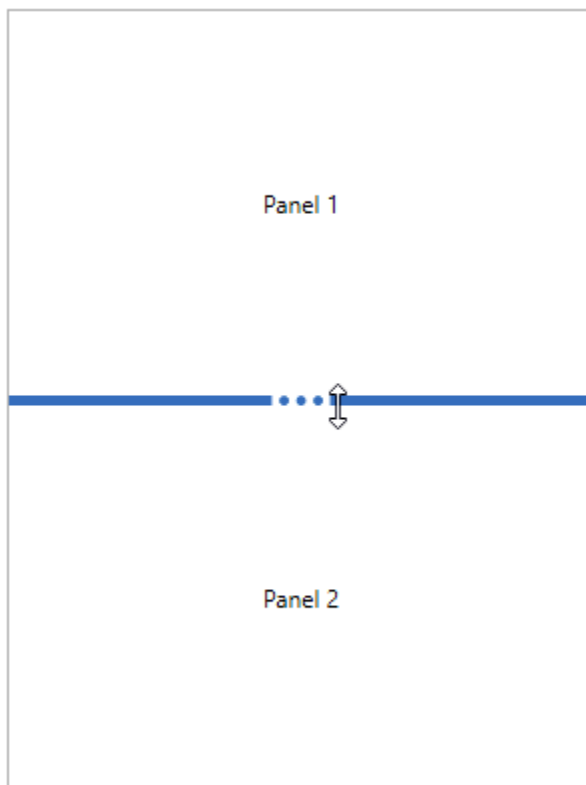
Custom drag preview

We can change the custom UI of the preview grid splitter by using the [SfGridSplitter.PreviewStyle](#) property. We can see the effect of `PreviewStyle` only on when `ShowsPreview` property value is `true`.

XML

```
<Border
Margin="10"
BorderBrush="DarkGray"
BorderThickness="1">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition Height="auto" />
      <RowDefinition />
    </Grid.RowDefinitions>
    <TextBlock Grid.Row="0"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      TextAlignment="Center"
      Text="Panel 1">
    </TextBlock>
    <TextBlock Grid.Row="2"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      TextAlignment="Center"
      Text="Panel 2">
    </TextBlock>
  </Grid>
</Border>
```

```
<!--Grid Splitter-->
<syncfusion:SfGridSplitter ShowsPreview="True"
HorizontalAlignment="Stretch"
Width="auto"
Grid.Row="1" >
<syncfusion:SfGridSplitter.PreviewStyle>
<Style TargetType="Control">
<Setter Property="Background" Value="Red"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="Control">
<Grid x:Name="Root" Opacity="0.5">
<Ellipse Fill="{TemplateBinding Background}"/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:SfGridSplitter.PreviewStyle>
</syncfusion:SfGridSplitter>
</Grid>
</Border>
```

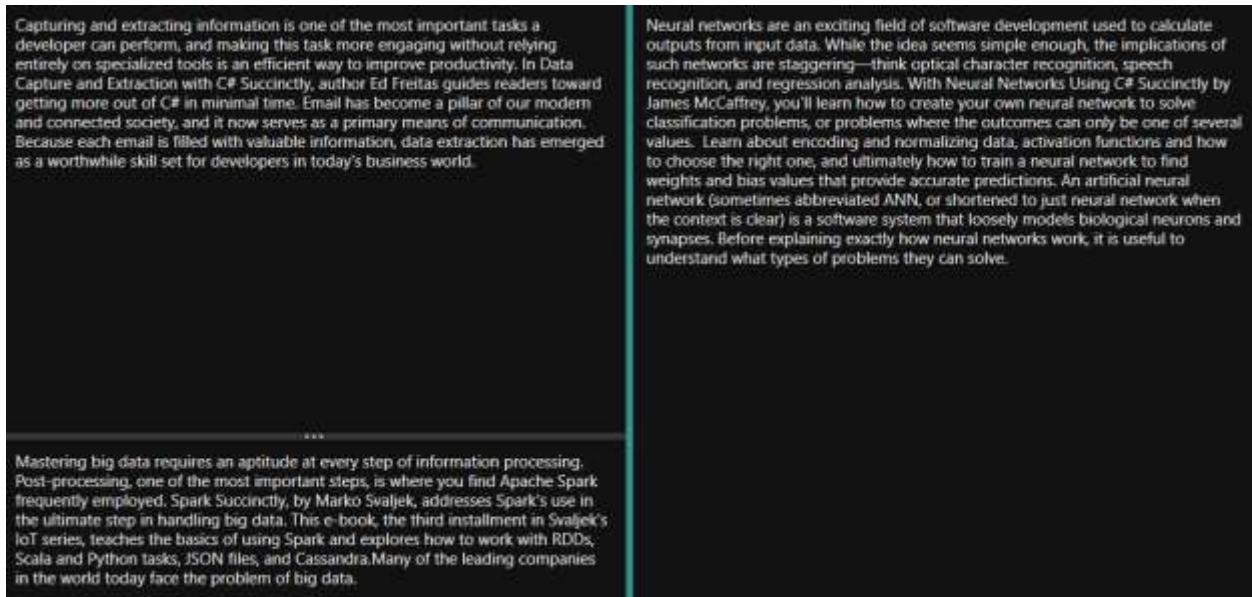


Note: [View Sample in GitHub](#)

Theme

SfGridSplitter supports various built-in themes. Refer to the below links to apply themes for the SfGridSplitter,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

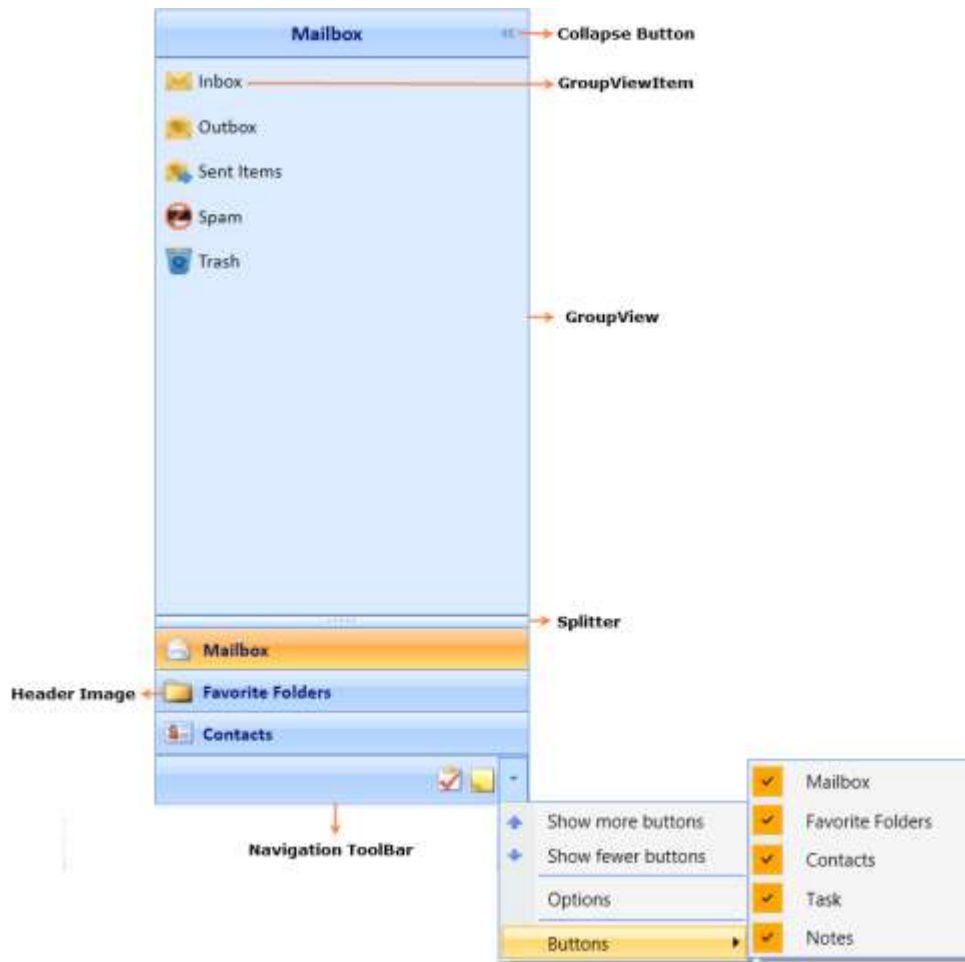


GroupBar

WPF Navigation Pane (GroupBar) Overview

The GroupBar control implements list-type controls in the UI, similar to the Microsoft Outlook Bar. It has a container to host controls within it. Use it to host a categorized collection of command items and custom controls wired to execute custom commands. It includes several customizable features which can be easily implemented in WPF applications.

Control Structure



Features

- Vertical and Horizontal layouts and orientation of GroupBar items.
- Alignment and Orientation of GroupView Items.
- Built-in visual styles - Default, Office 2003, Office 2007 Blue, Office 2007 Silver, Office 2007 Black and Blend are available for the control.
- It supports expansion of multiple tabs.
- ToolTip support for GroupView items.
- Built-In context menu that provides the user access to functionalities to customize the GroupBar items just like in VS 2005 Toolbox context menu.
- Various types of visual modes such as Multiple Expansion (VS 2005 Toolbox) and Stack Mode.
- Dynamically sort tabs / items alphabetically in ascending / descending order through context menu or programmatically.
- Dynamically add new tabs / items through context menu or programmatically.
- Dynamically alter the tab header text or item header text using context menu or programmatically.
- Dynamically move a selected tab up / down one step in the GroupBar using context menu.
- Tabs can toggle between expanded and collapsed state. Can expand multiple tabs at the same time.

- Dynamically move a selected item up / down one step in the GroupBar using context menu or programmatically.

Navigation Pane Features

- Resize the Pop up.
- Expand / Collapse the Navigation Pane.
- Set the height of Tool bar.

Getting Started with WPF Navigation Pane (GroupBar)

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 GroupBar

You can create a WPF application with GroupBar control using the following steps:

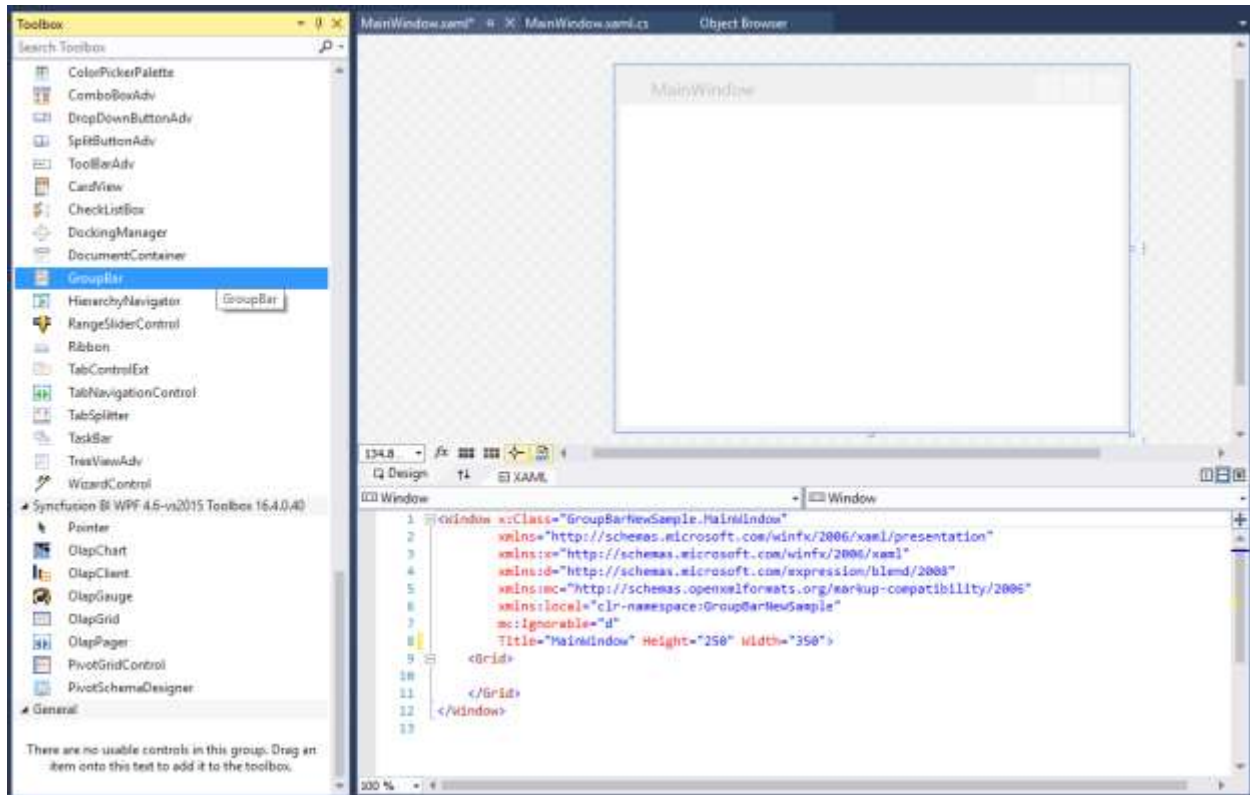
Create a project

Create a new WPF project in Visual Studio to display the GroupBar with functionalities.

Add control through designer

The GroupBar control can be added to an application by dragging it from the toolbox to a designer view. The following 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 GroupBar 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="GroupBarSample.MainWindow"
Title="GroupBar Sample" Height="350" Width="525">
<Grid>
<!-- Adding GroupBar Control -->
<syncfusion:GroupBar Name="groupBar"/>
</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* and *Syncfusion.Shared.WPF*
2. Import the GroupBar namespace using **Syncfusion.Windows.Tools.Controls**;
3. Create a GroupBar instance, and add it to the window.

C#


```
using Syncfusion.Windows.Tools.Controls;
namespace GroupBarSample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Creating an instance of GroupBar control
            GroupBar groupBar = new GroupBar();
            //Adding GroupBar as window content
            this.Content = groupBar;
        }
    }
}
```

Add items using GroupBarItem

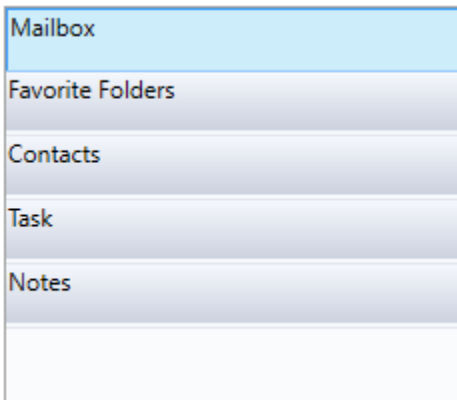
You can add the group bar items inside the GroupBar control using [GroupBarItem](#).

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem" Header="Mailbox"/>
  <syncfusion:GroupBarItem Name="groupBarItem1" Header="Favorite Folders"/>
  <syncfusion:GroupBarItem Name="groupBarItem2" Header="Contacts"/>
  <syncfusion:GroupBarItem Name="groupBarItem3" Header="Task"/>
  <syncfusion:GroupBarItem Name="groupBarItem4" Header="Notes"/>
</syncfusion:GroupBar>
```

C#

```
GroupBar groupBar = new GroupBar();
//Instance for GroupBarItem
GroupBarItem groupBarItem1 = new GroupBarItem();
GroupBarItem groupBarItem2 = new GroupBarItem();
GroupBarItem groupBarItem3 = new GroupBarItem();
GroupBarItem groupBarItem4 = new GroupBarItem();
GroupBarItem groupBarItem5 = new GroupBarItem();
//Setting text for GroupBarItem
groupBarItem1.HeaderText = "Mailbox";
groupBarItem2.HeaderText = "Favorite Folders";
groupBarItem3.HeaderText = "Contacts";
groupBarItem4.HeaderText = "Task";
groupBarItem5.HeaderText = "Notes";
//Adding group bar items into group bar
groupBar.Items.Add(groupBarItem1);
groupBar.Items.Add(groupBarItem2);
groupBar.Items.Add(groupBarItem3);
groupBar.Items.Add(groupBarItem4);
groupBar.Items.Add(groupBarItem5);
this.Content = groupBar;
```



Bind data

Items in a GroupBar can also be added by binding a collection of business objects using the [ItemsSource](#) property.

- **Model.cs**

C#

```
public class Model
{
    public string Header
    {
        get;
        set;
    }
    public string Content
    {
        get;
        set;
    }
    public bool IsSelected
    {
        get;
        set;
    }
}
```

- **ViewModel.cs**

C#

```
public class ViewModel
{
    private ObservableCollection<Model> groupBarData;
    public ObservableCollection<Model> GroupData
    {
        get { return groupBarData; }
    }
}
```

```

set
{
    groupBarData = value;
}
}
public ViewModel()
{
    GroupData = new ObservableCollection<Model>();
    populateGroupBar();
}
private void populateGroupBar()
{
    GroupData.Add(new Model() { Header = "Mailbox", Content = "GroupBarItem1",
    IsSelected = true });
    GroupData.Add(new Model() { Header = "Favorite Folder", Content =
    "GroupBarItem2", IsSelected = false });
    GroupData.Add(new Model() { Header = "Contacts", Content = "GroupBarItem3",
    IsSelected = false });
    GroupData.Add(new Model() { Header = "Task", Content = "GroupBarItem4",
    IsSelected = false });
    GroupData.Add(new Model() { Header = "Notes", Content = "GroupBarItem5",
    IsSelected = false });
}
}

```

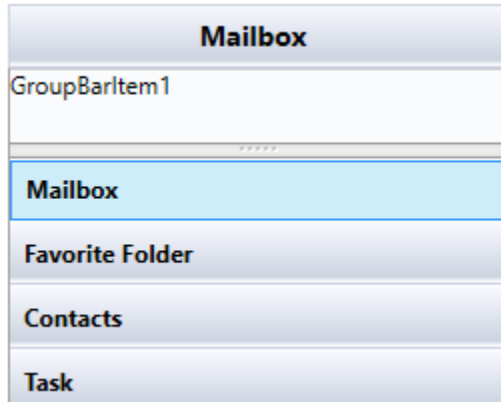
- **MainWindow.Xaml**

C#

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<syncfusion:GroupBar Name="groupBar" ItemsSource="{Binding GroupData}"
VisualMode="StackMode">
<syncfusion:GroupBar.ItemContainerStyle>
<Style TargetType="{x:Type syncfusion:GroupBarItem}">
<Setter Property="HeaderText" Value="{Binding Header}"/>
<Setter Property="Content" Value="{Binding Content}"/>
<Setter Property="IsSelected" Value="{Binding IsSelected}" />
</Style>
</syncfusion:GroupBar.ItemContainerStyle>
</syncfusion:GroupBar>

```



Add content to GroupBar Item

You can add content to a GroupBar Item using a panel or a [GroupView](#) control.

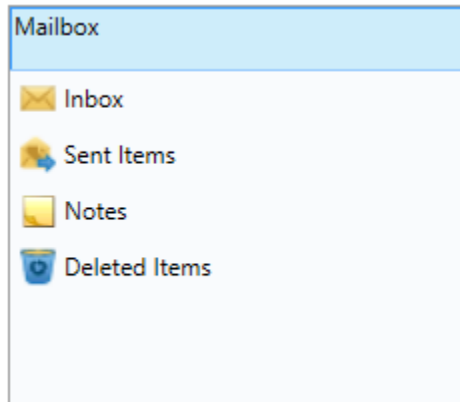
XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" Name="groupBar"
VisualMode="MultipleExpansion" >
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="Mailbox">
<!-- Adding GroupView to GroupBarItem -->
<syncfusion:GroupView Name="groupView">
<!-- Adding GroupViewItem to GroupView -->
<syncfusion:GroupViewItem Name="groupViewItem"
ImageSource="Images\inbox.png" Text="Inbox"/>
<syncfusion:GroupViewItem Name="groupViewItem1"
ImageSource="Images\sent.png" Text="Sent Items"/>
<syncfusion:GroupViewItem Name="groupViewItem2"
ImageSource="Images\Notes.png" Text="Notes"/>
<syncfusion:GroupViewItem Name="groupViewItem3"
ImageSource="Images\trash.png" Text="Deleted Items"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Create instances
GroupBar groupBar = new GroupBar();
GroupBarItem groupBarItem = new GroupBarItem();
GroupView groupView = new GroupView();
GroupViewItem item1 = new GroupViewItem();
GroupViewItem item2 = new GroupViewItem();
GroupViewItem item3 = new GroupViewItem();
GroupViewItem item4 = new GroupViewItem();
//Set header for groupbar item
groupBarItem.Header = "Mailbox";
item1.Text = "Inbox";
item2.Text = "Sent Items";
item3.Text = "Notes";
item4.Text = "Deleted Items";
groupView.Items.Add(item1);
```

```
groupView.Items.Add(item2);  
groupView.Items.Add(item3);  
groupView.Items.Add(item4);  
groupByItem.Content = groupView;  
groupByItem.Items.Add(groupBarItem);  
this.Content = groupBar;
```



Expand one or more items

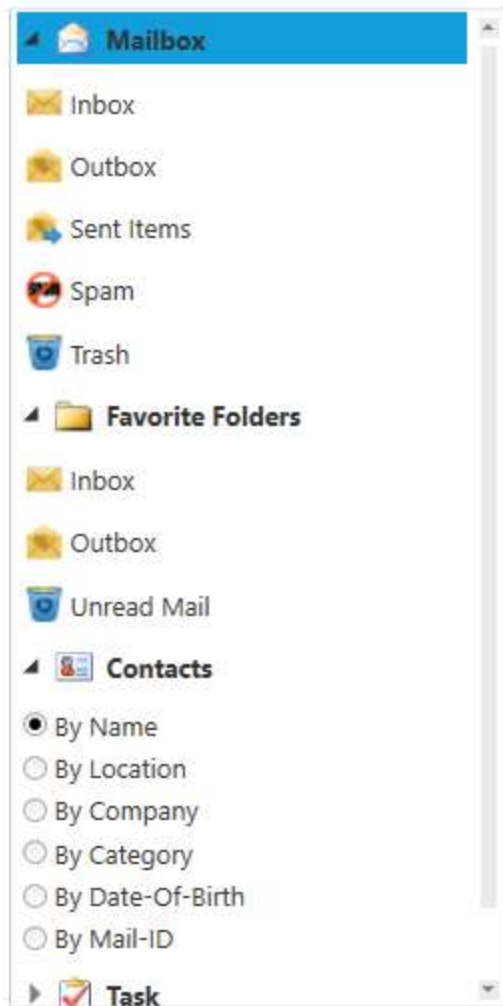
You can expand and collapse any number of groupbar items by setting the [VisualMode](#) to `MultipleExpansion`.

XML

```
<!--Expanding more groupbar items -->  
<syncfusion:GroupBar Height="200" Width="230" Name="groupByItem"  
VisualMode="MultipleExpansion"/>
```

C#

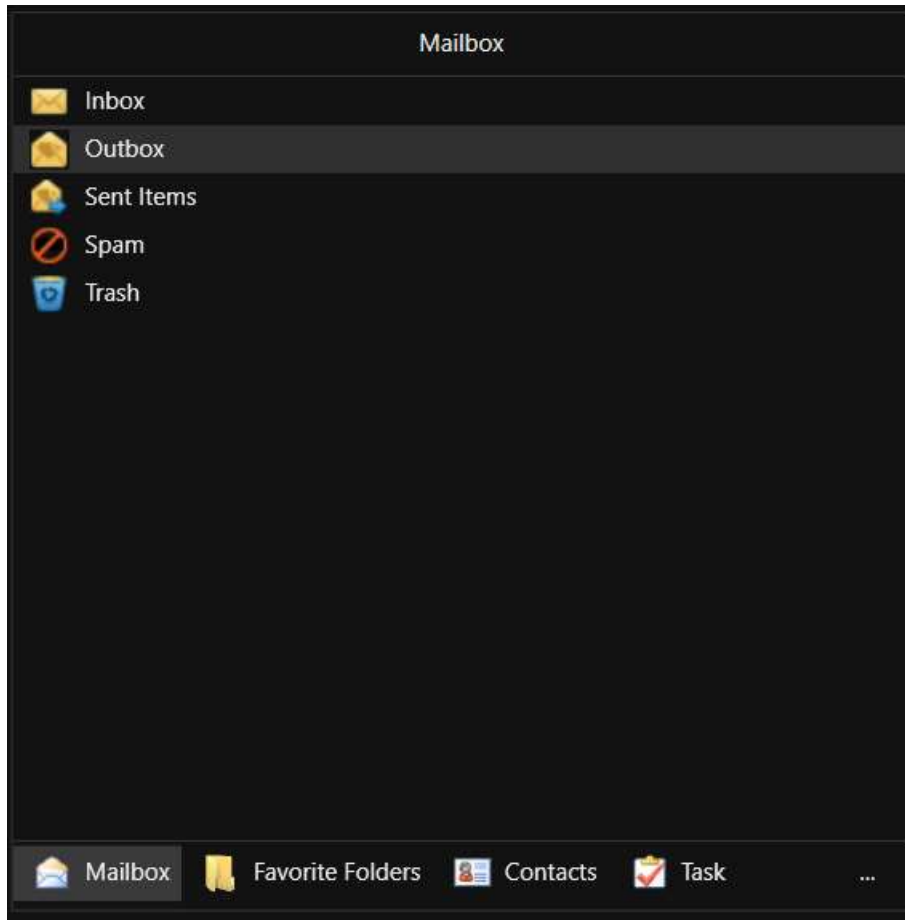
```
//Expanding more groupbar items  
groupByItem.VisualMode = Syncfusion.Windows.Tools.VisualMode.MultipleExpansion;
```



Theme

GroupBar supports various built-in themes. Refer to the below links to apply themes for the GroupBar,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Adding GroupBar Item to GroupBar in WPF Navigation Pane (GroupBar)

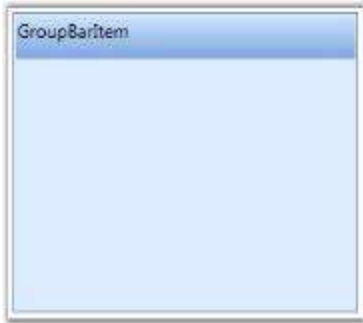
GroupBar Item is added to the GroupBar using XAML or C# code. The following code illustrates this.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem"/>
</syncfusion:GroupBar>
```

C#

```
//Creating an instance of GroupBar
GroupBar groupBar = new GroupBar();
//Creating an instance of GroupBarItem
ItemGroupBarItem groupBarItem = new GroupBarItem();
//Setting header for GroupBar itemgroup
BarItem.Header = "GroupBarItem";
//Adding GroupBar item to GroupBar
groupBar.Items.Add(groupBarItem);
//Adding GroupBar to the window
this.Content = groupBar;
```



Note: To display the GroupBar Item, you must have a GroupBar in which you are going to add the GroupBar Item.

Adding Content to GroupBar Item in WPF Navigation Pane (GroupBar)

You can add content to a GroupBar Item using a Panel or a GroupView control. Any content can be added to the GroupBar Item by adding any control inside the panel. To view the contents of a GroupBar Item in ListView mode, add the GroupView as content in the GroupBar Item.

Adding GroupView to GroupBar Item

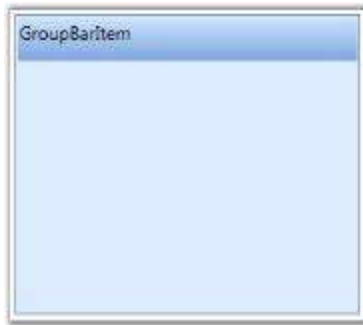
GroupView control can be added to the GroupBar Item by using XAML or C# code. Here is the code snippet.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
    <!-- Adding GroupView to GroupBarItem -->
    <syncfusion:GroupView Name="groupView"/>
  </syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Creating an instance of GroupBar
GroupBar groupBar = new GroupBar();
//Creating an instance of GroupBarItem
GroupBarItem groupBarItem = new GroupBarItem();
//Setting header for groupbar item
groupBarItem.Header = "GroupBarItem";
//Creating an instance of GroupView
GroupView groupView = new GroupView();
//Adding GroupView to GroupViewItem
groupBarItem.Content = groupView;
//Adding GroupBar item to GroupBar
groupBar.Items.Add(groupBarItem);
//Adding GroupBar to the windowthis.Content = groupBar;
```

Adding GroupView Item to GroupView

GroupView Item can be added to a GroupView using XAML or C# code. Here is the code snippet.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
    <!-- Adding GroupView to GroupBarItem -->
    <syncfusion:GroupView Name="groupView">
      <!-- Adding GroupViewItem to GroupView -->
      <syncfusion:GroupViewItem Name="groupViewItem"
Text="GroupViewItem"/>
    </syncfusion:GroupView>
  </syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Creating an instance of GroupBar
GroupBar groupBar = new GroupBar();
//Creating an instance of GroupBarItem
GroupBarItem groupBarItem = new GroupBarItem();
//Setting header for groupbar item
groupBarItem.Header = "GroupBarItem";
//Creating an instance of GroupView
GroupView groupView = new GroupView();
//Creating an instance of GroupViewItem
GroupViewItem groupViewItem = new GroupViewItem();
//Adding content to GroupViewItem
groupViewItem.Text = "GroupViewItem";
//Adding GroupViewItem to GroupView
groupView.Items.Add(groupViewItem);
//Adding content of GroupBarItem as GroupView;
groupBarItem.Content = groupView;
//Adding GroupBar item to GroupBar
groupBar.Items.Add(groupBarItem);
//Adding GroupBar to the window
this.Content = groupBar;
```



Adding Panel to GroupBar Item

You can add content to the GroupBar Item using a Panel as follows. This is achieved using the following code snippet.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
    <!-- Adding content for groupbar item using panel -->
    <StackPanel Orientation="Vertical">
      <TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
      <RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
      <RadioButton Margin="4,2,2,2">Vertical</RadioButton>
      <TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
      <RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
      <RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
    </StackPanel>
  </syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```



Note: To display the GroupView Item, you must already have a GroupView control in which you are going to add the GroupView Item.

Events to handle with GroupBar Item

The events of GroupBarItem are GroupBarItemAdded and GroupBarItemRemoved.

- GroupBarItemAdded – which is called when GroupBarItem is added. GroupBarItem can be added either by using procedural code or by using context menu to add new item.
- GroupBarItemRemoved – which is called when GroupBarItem is removed. GroupBarItem can be removed by using procedural code or by using context menu to remove an existing item.

Rotating the GroupBar in WPF Navigation Pane (GroupBar)

You can rotate the GroupBar in any direction by setting a suitable angle to the `RotationAngle` property. This dependency property sets the angle of displaying GroupBar. It returns the angle of rotation used for displaying the GroupBar. The default value is 0.

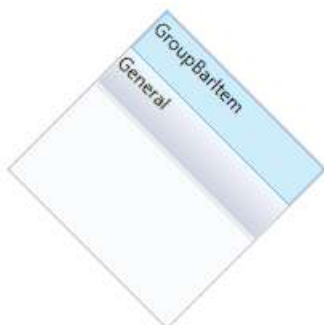
Use the below code snippet to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" RotationAngle="45"
Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
<!-- Adding content for groupbar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel></syncfusion:GroupBarItem><!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView></syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Set the rotation angle
groupBar.RotationAngle = 45;
```



Rotating the Content in the GroupBar

Rotating the Content in WPF Navigation Pane (GroupBar)

The position of the content can be changed by setting an angle to the ContentRotationAngle property. This dependency property sets the angle of rotation for displaying item contents.

The following code snippet will help you setting this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" ContentRotationAngle="45" Width="230"
Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
<!-- Adding content for groupbar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal
</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Setting the rotation angle as 45
groupBar.ContentRotationAngle = 45;
```

Rotating the GroupBar

Collapsing the GroupBar in WPF Navigation Pane (GroupBar)

Enabling this property allows the GroupBar to be collapsed when clicked. The GroupBar will collapse similar to the collapsing action in Microsoft office Outlook. AllowCollapse property is used to enable or disable the collapse button of the GroupBar.

By clicking the Navigation pane, the content of the GroupBar Items are displayed. You can resize the pop-up and also add or remove the GroupBar Item from the Toolbar using the StackItem host pop-up menu.

To enable the AllowCollapse property in GroupBar, use the following code.

XML

```
<!-- Adding GroupBar that has allow collapse property to true -->
<syncfusion:GroupBar Height="300" Width="230" Name="groupBar"
AllowCollapse="True" VisualMode="StackMode">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
```

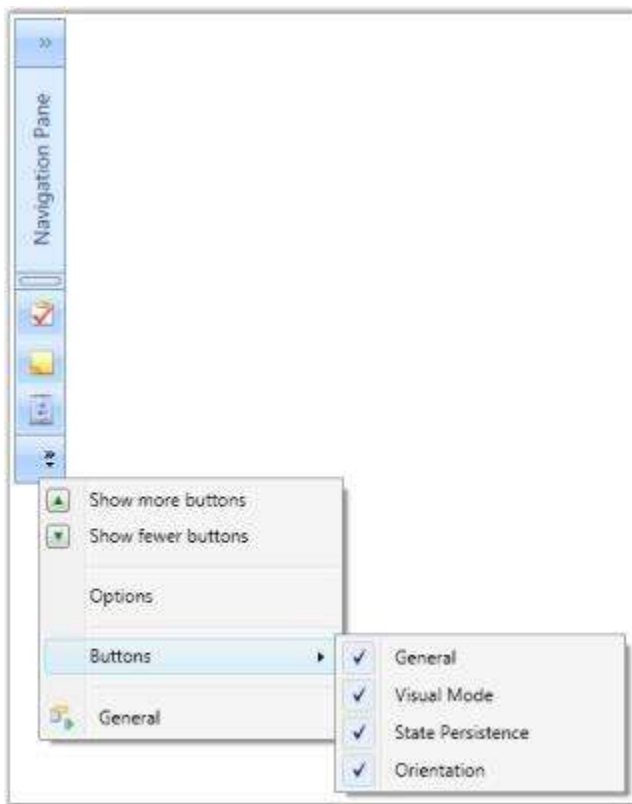
```

<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Tasks.png" Name="groupBarItem2"
Header="Visual Mode">

```

C#

```
myGroupBar.AllowCollapse = true;
```



Events to handle with AllowCollapse

The event corresponding to this property is `BeforeGroupBarItemPopupOpened`. This event is triggered when `AllowCollapse` property gets changed.

Note: `AllowCollapse` property settings works only for GroupBar in `StackMode`.

Collapsing the GroupBar in Stack Mode

Collapsing and expanding the GroupBar in `StackMode` is done by using `IsCollapsed` property. This dependency property indicates the state of GroupBar, whether collapsed or expanded. By setting this property to `true`, groupbar is collapsed. By setting `false`, groupbar is expanded.

Use the below code snippet to set this property.

XML

```

<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" IsCollapsed="True"
AllowCollapse="True" VisualMode="StackMode" Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem"
IsSelected="True">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel> </syncfusion:GroupBarItem> <!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewModel="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>

```

C#

```

//Set the visual mode as stack mode
groupBar.VisualMode = VisualMode.StackMode;
//Collapsing the GroupBar in stack mode
groupBar.IsCollapsed = true;

```

Width of the Collapsed GroupBar**Adjusting the GroupBar width in WPF Navigation Pane (GroupBar)**

Set the width of the GroupBar while it is collapsed by using the CollapsedWidth property. This dependency property sets the width of the collapsed groupbar.

Use the below code snippet to set this property.

XML

```

<!-- Adding GroupBar with AllowCollapse property = true -->

```

C#

```

//Enable AllowCollapse property for GroupBar
groupBar.AllowCollapse = true;
//Setting the width of Collapse button
groupBar.CollapsedWidth = 100;

```

Collapsing the GroupBar

Setting the FlowDirection for GroupBar in WPF Navigation Pane

Flow Direction of the GroupBar is set by using the FlowDirection property.

Property Table

Property	Description
FlowDirection	Sets the flow direction for the GroupBar control. The options provided are as follows. <i>LeftToRight</i> <i>RightToLeft</i>

Use the following code snippet to set this property.

XML

```

<!-- Adding GroupBar that has flow direction as left to right -->
<syncfusion:GroupBar Height="300" Width="230" Name="groupBar"
FlowDirection="RightToLeft">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewModel="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView></syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Tasks.png" Name="groupBarItem2"
Header="Visual Mode">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Default"/>
<syncfusion:GroupViewItem Text="Multiple Expansion"/>
<syncfusion:GroupViewItem Text="StackMode"/>
</syncfusion:GroupView></syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Notes.png" Name="groupBarItem3"
Header="State Persistence">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Save State"/>
<syncfusion:GroupViewItem Text="Load State"/>
<syncfusion:GroupViewItem Text="Reset State"/>
</syncfusion:GroupView></syncfusion:GroupBarItem>
</syncfusion:GroupBar>

```

C#

```
// Setting flow direction as RightToLeft
```

```
groupBar.FlowDirection = FlowDirection.RightToLeft;
```

Closing the GroupBar in Stack Mode in WPF Navigation Pane (GroupBar)

You can close the GroupBar in stack mode by enabling the close button. This is done by using the `IsCloseButtonEnabled` property. This dependency property sets the value indicating whether the close button is visible on the header of the item in stack mode when `IsToolBarEnabled` property is set to *false*. That is close button is visible only if toolbar is disabled. This property can work only in stack mode.

Use the below code snippet to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" IsCloseButtonEnabled="True"
    IsToolBarEnabled="False" VisualMode="StackMode" Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem"
    IsSelected="True">
<!-- Adding content for groupbar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2" >Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel> </syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
    Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Setting the visual mode as stack mode
groupBar.VisualMode = VisualMode.StackMode;
//Enabling the close button
groupBar.IsCloseButtonEnabled = true;
//Disabling the toolbargroupBar.IsToolBarEnabled = false;
```

Maximum number of Visible Items in OverflowPanel

User can customize the maximum number of Visible Items to be displayed in the StackMode OverflowPanel, using the property `StackVisibleItemsCount`.

XML

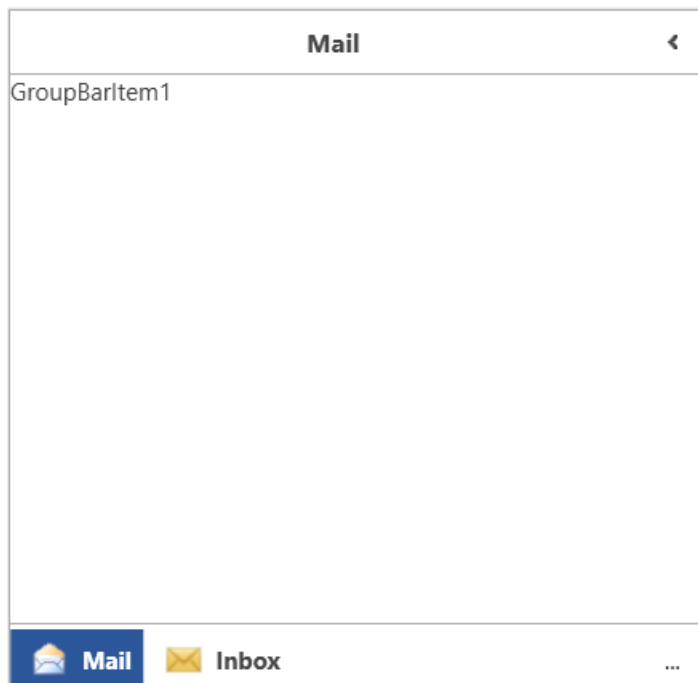
```
<syncfusion:GroupBar Name="GroupBar" Height="340" VerticalAlignment="Center"
```



```
HorizontalAlignment="Center" VisualMode="StackMode" IsToolBarEnabled="True"
StackVisibleItemsCount="2">
<syncfusion:GroupBarItem x:Name="groupitem1" HeaderText="Mail"
HeaderImageSource="Images/mail.png" >
<Border >
<TextBlock Text="GroupBarItem1" />
</Border>
</syncfusion:GroupBarItem>
<syncfusion:GroupBarItem HeaderText="Inbox"
HeaderImageSource="Images/inbox.png" >
<Border >
<TextBlock Text="GroupBarItem2" />
</Border>
</syncfusion:GroupBarItem>
<syncfusion:GroupBarItem HeaderText="Notes"
HeaderImageSource="Images/Notes.png" >
<Border >
<TextBlock Text="GroupBarItem3" />
</Border>
</syncfusion:GroupBarItem>
<syncfusion:GroupBarItem HeaderText="Sent"
HeaderImageSource="Images/sent.png" >
<Border >
<TextBlock Text="GroupBarItem4" />
</Border>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
GroupBar.StackVisibleItemsCount = 2;
```



Note: This property is applicable for SfSkinManager Office2016 themes.

Changing the orientations of GroupBar in WPF Navigation Pane

You can change the layout of the entire GroupBar. GroupBar control has the Orientation property, which lets you place the contents of GroupBar either vertically or horizontally. It provides the following options.

- **Horizontal:** the contents of the GroupBar Item in the GroupBar control are placed horizontally.
- **Vertical:** the contents of the GroupBar Item in GroupBar control are placed vertically.

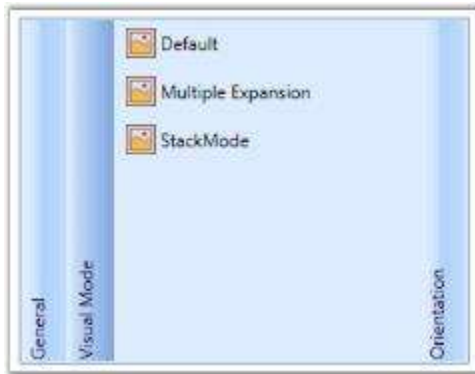
Use the following code snippet to set GroupBar Orientation to Horizontal.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="300" Width="230" Orientation="Horizontal"
Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem2" Header="Visual Mode">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Default"/>
<syncfusion:GroupViewItem Text="Multiple Expansion"/>
<syncfusion:GroupViewItem Text="StackMode"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem4" Header="Orientation">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

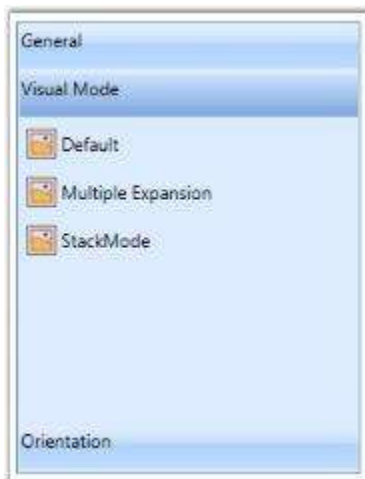
```
//Setting orientation as horizontal
groupBar.Orientation = Orientation.Horizontal;
```



Use the following code snippet to set groupbar Orientation to Vertical.

C#

```
//Setting orientation as vertical
groupBar.Orientation = Orientation.Vertical;
```



You can also change the orientation of the GroupView alone by using the Orientation property for GroupView. There are two types of orientation in GroupView control as in GroupBar.

- Horizontal
- Vertical

Use the following code snippet to set GroupView Orientation to Horizontal.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="300" Width="230" Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem1" Header="General">
    <!-- Adding content for GroupBar item using GroupView -->
    <syncfusion:GroupView Name="groupView" Orientation="Horizontal">
      <syncfusion:GroupViewItem Text="List View"/>
      <syncfusion:GroupViewItem Text="Show ContextMenu"/>
      <syncfusion:GroupViewItem Text="Show ToolTip"/>
    </syncfusion:GroupView>
  </syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

```

</syncfusion:GroupView></syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem2" Header="Visual Mode">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Default"/>
<syncfusion:GroupViewItem Text="Multiple Expansion"/>
<syncfusion:GroupViewItem Text="StackMode"/>
</syncfusion:GroupView></syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem4" Header="Orientation">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel></syncfusion:GroupBarItem>
</syncfusion:GroupBar>

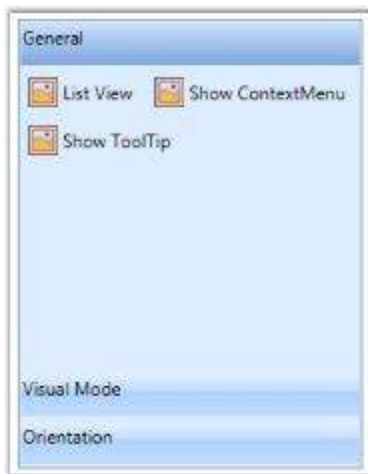
```

C#

```

//Setting the orientation of GroupView as Horizontal
groupView.Orientation = Orientation.Horizontal;

```



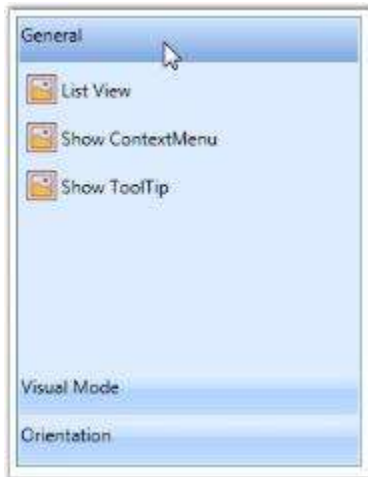
Use the following code snippet to set GroupView Orientation to Vertical.

C#

```

//Setting the orientation of GroupView as Horizontal
groupView.Orientation = Orientation.Vertical;

```



Events to Handle Orientation of Groupbar

The events corresponding to this property are [OrientationChanged]() and [OrientationChanging]().

OrientationChanged Event

This event is called when the orientation of groupbar is changed and is triggered when the Orientation property is changed.

OrientationChanging Event

This event is called when orientation of groupbar is changing and is triggered when the Orientation property is changing.

Changing Header Height in WPF Navigation Pane (GroupBar)

Header Height of the GroupBar Item

The height of the each GroupBar item header is set using the ItemHeaderHeight property. This dependency property sets the height for each groupbar items header.

Use the below code to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" ItemHeaderHeight="50"
Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem"
IsSelected="True">
    <!-- Adding content for GroupBar item using panel -->
    <StackPanel Orientation="Vertical">
      <TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
      <RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
      <RadioButton Margin="4,2,2,2">Vertical</RadioButton>
      <TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
      <RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
      <RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
    </StackPanel>
  </syncfusion:GroupBarItem>
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
    <!-- Adding content for GroupBar item using GroupView -->
    <syncfusion:GroupView Name="groupView" IsListViewMode="True">
```

```
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Set the height of GroupBar item header groupBar.ItemHeaderHeight = 50;
```

Height of the Container that hosts the Selected Header

HeaderHeight is a dependency property, which gets or sets the height of the topmost container that hosts the selected header.

Use the below code snippet to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" HeaderHeight="20" Width="230"
VisualMode="StackMode" Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem"
IsSelected="True">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel> </syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
groupBar.HeaderHeight = 20;
```

Setting Animation for GroupBar expansion collapse in WPF

Animation Speed

The expanding and collapsing operation in a GroupBar leads to an animated action. This animation is controlled using the AnimationSpeed property, which sets the speed of animation. This dependency property will be activated when the user expands or collapses the GroupBar Item of the GroupBar control. Using this property, you can control the animation speed of GroupBarItem while expanding / collapsing. It returns the speed of animation.

Use the following code snippet to set this property.

XML

```
<!-- Adding GroupBar that have animation speed -->
<syncfusion:GroupBar Height="300" Width="230" Name="groupBar"
AnimationSpeed="5" AnimationType="Fade">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewModel="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Tasks.png" Name="groupBarItem2"
Header="Visual Mode">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Default"/>
<syncfusion:GroupViewItem Text="Multiple Expansion"/>
<syncfusion:GroupViewItem Text="StackMode"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Notes.png" Name="groupBarItem3"
Header="State Persistence">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Save State"/>
<syncfusion:GroupViewItem Text="Load State"/>
<syncfusion:GroupViewItem Text="Reset State"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Getting the speed of animation
groupBar.AnimationSpeed = 5;
```

Animation Type

When the GroupBar is collapsed or expanded, it gives an animation effect. There are three types of animation, defined for the GroupBar control using AnimationType property. This dependency property sets the animation type to be applied when GroupBar is collapsed or expanded. It provides the following options.

- Fade - Animation fades out when the GroupBar is collapsed or expanded
- None - No animation when the GroupBar is collapsed or expanded
- Slide - Animation slides when the GroupBar is collapsed or expanded

Use the following code snippet to set AnimationType property.

XML

```
<!-- Adding GroupBar that have AnimationType as Fade -->
<syncfusion:GroupBar Height="300" Width="230" AnimationType="Fade"
Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
    <!-- Adding content for GroupBar item using GroupView -->
    <syncfusion:GroupView>
      <syncfusion:GroupViewItem Text="List View"/>
      <syncfusion:GroupViewItem Text="Show ContextMenu"/>
      <syncfusion:GroupViewItem Text="Show ToolTip"/>
    </syncfusion:GroupView>
  </syncfusion:GroupBarItem>
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem HeaderImageSource="Tasks.png" Name="groupBarItem2"
Header="Visual Mode">
    <!-- Adding content for GroupBar item using GroupView -->
    <syncfusion:GroupView>
      <syncfusion:GroupViewItem Text="Default"/>
      <syncfusion:GroupViewItem Text="Multiple Expansion"/>
      <syncfusion:GroupViewItem Text="StackMode"/>
    </syncfusion:GroupView>
  </syncfusion:GroupBarItem>
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem HeaderImageSource="Notes.png" Name="groupBarItem3"
Header="State Persistence">
    <!-- Adding content for GroupBar item using GroupView -->
    <syncfusion:GroupView>
      <syncfusion:GroupViewItem Text="Save State"/>
      <syncfusion:GroupViewItem Text="Load State"/>
      <syncfusion:GroupViewItem Text="Reset State"/>
    </syncfusion:GroupView>
  </syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//To set animation type as Fade
groupBar.AnimationType = AnimationsType.Fade;
//To set animation type as None
groupBar.AnimationType = AnimationsType.None;
//To set animation type as Slide
```



```
groupBar.AnimationType = AnimationsType.Slide;
```

Setting ToolTip for GroupBar in WPF Navigation Pane (GroupBar)

ToolTip is one of the important user interactive feature available in GroupBar control. You can set the tooltip for the collapse button and expand button using the properties listed below.

Property table

Property	Description
CollapseButtonToolTip	Gets or sets the tooltip to be displayed when mouse is hovered over the collapse button.
ExpandButtonToolTip	Gets or sets the tooltip to be displayed when mouse is hovered over the expand button.

Use the below code snippet to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" CollapseButtonToolTip="Expand"
VisualMode="StackMode" AllowCollapse="True" ExpandButtonToolTip="Collapse"
Width="230" Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Setting the visual mode as StackMode
groupBar.VisualMode = VisualMode.StackMode;
//Setting AllowCollapse property
groupBar.AllowCollapse = true;
```

```
//Setting the tooltip for Expand button
groupBar.ExpandButtonToolTip = "Collapse";
//Setting the tooltip for Collapse button
groupBar.CollapseButtonToolTip = "Expand";
```

Using Context menu in GroupBar in WPF Navigation Pane (GroupBar)

To enable the context menu support in the GroupBar, set `IsEnabledContextMenu` property to `true`. This dependency property sets the value that indicates whether the context menu is enabled.

Here are the benefits of using context menu in GroupBar control.

Use the following code snippet to set this property.

- Cut, copy and paste the content of GroupBar
- Sort the content of groupbar
- Add the GroupBarItem
- Rename the GroupBarItem
- Delete the GroupBarItem
- Move the content of GroupBar either up or down
- Add the content of GroupBar item.
- Rename the content of GroupBar item.
- Delete the content of GroupBar item.

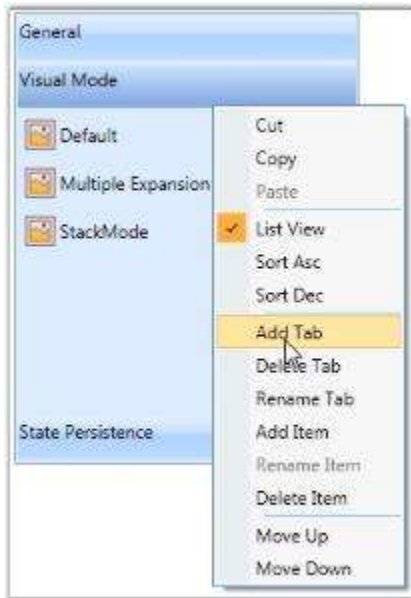
XML

```
<!-- Adding GroupBar that has context menu -->
<syncfusion:GroupBar Height="250" Width="230" IsEnabledContextMenu="True"
Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Tasks.png" Name="groupBarItem2"
Header="Visual Mode">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Default"/>
<syncfusion:GroupViewItem Text="Multiple Expansion"/>
<syncfusion:GroupViewItem Text="StackMode"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Enable the Context menu for GroupBar
```

```
groupBar.IsEnabledContextMenu = true;
```



Toolbars in GroupBar in WPF Navigation Pane (GroupBar)

We can hide the toolbar that is displayed in the GroupBar using the `IsToolBarEnabled` property. This dependency property sets the value indicating whether to hide the toolbar and items in stacked mode. It returns the bool value that indicates the state of toolbar that is hidden or visible. This property works only in stacked mode.

The following code snippet will help you to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" IsToolBarEnabled="True"
VisualMode="StackMode" Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem"
IsSelected="True">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel></syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
```

```
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
```

C#

```
//Enable the ToolBar
groupBar.IsToolBarEnabled = true;
```

Navigation Pane in GroupBar in WPF Navigation Pane (GroupBar)

Resizing the Navigation Pane Pop-up

You can resize the navigation pane pop-up in any direction using the `PopupResizeDirection` property. This dependency property sets the value indicating the resize directions of the navigation pane's pop-up. There are four built-in resize directions.

- Both - navigation pane pop-up is resized in both directions, i.e., vertically and horizontally
- Horizontal – navigation pane pop-up is resized in the horizontal direction only
- None – navigation pane pop-up is not resized
- Vertical – navigation pane pop-up is resized in the vertical direction only

Use the below code to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" PopupResizeDirection="Both"
Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem"
IsSelected="True">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewModel="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Resize the navigation pane pop-up in both direction
groupBar.PopupResizeDirection = PopupResizeDirection.Both;
```

Navigation Pane Gripper

Gripper is available in navigation pane. Using the gripper we can increase or decrease the size of the GroupBar items. You can enable the gripper in GroupBar by setting ShowGripper property to true. This property works only in stacked mode. Default value is true.

Use the below code snippet to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" VisualMode="StackMode"
ShowGripper="True" Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal
</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Enable the gripper in navigation pane
groupBar.ShowGripper = true;
```

Adding Groupview in ListView Mode in WPF Navigation Pane (GroupBar)

You can arrange GroupView Items in list view mode by using the IsListViewMode property. This property is used to enable or disable the layout of items in a GroupView in the ListView mode.

You can arrange GroupView items in list view mode using the following code.

XML

```
<!-- Adding GroupBar that have visual mode is Multiple Expansion -->
<syncfusion:GroupBar Height="300" Width="230" Name="groupBar">
```

```

<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>

```

C#

```
groupview.IsListViewMode = true;
```

GroupView Orientation in Orientation topic.

Default Image for the GroupView Item in WPF Navigation Pane (GroupBar)

You can set the default item image for the GroupView item using the DefaultItemImage property. This dependency property sets the default image for GroupView item. It returns the default image of GroupView item.

Use the below code snippet to set this property.

XML

```

<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" DefaultItemImage="App.ico" Width="230"
Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>

```

State Persistence in WPF Navigation Pane (GroupBar)

The bar state can be persisted in the GroupBar control using the bar state methods available. You can set three different state persistence in the GroupBar. The three state persistence are as follows.

- Save State – Saves the state persisted for the current GroupBar location. This can be done using the SaveBarState method.
- Load State – Loads the state that is persisted. This can be done using the LoadBarState method.
- Reset State – Resets the state that is persisted. This can be done using the ResetBarState method.

You can set the state persistence using the following code.

C#

```
// Save State
myGroupBar.SaveBarState();
// Load State
myGroupBar.LoadBarState();
registerEvents(myGroupBar);
// Reset State
myGroupBar.ResetBarState();
registerEvents(myGroupBar);
```

The ResetBarState method is used to register the events. The following code illustrates this.

C#

```
public void registerEvents(Syncfusion.Windows.Tools.Controls.GroupBar
groupBar)
{
    if (groupBar != null)
    {
        foreach (GroupBarItem tab in groupBar.Items)
        {
            SubscribeToTabEvents(tab);
            if (tab.Content is GroupView)
            {
                GroupView view = tab.Content as GroupView;
                if (view != null)
                {
                    foreach (GroupViewItem item in view.Items)
                    {
                        SubscribeToItemEvents(item);
                    }
                }
            }
        }
    }
}
```

You can also save the state persisted on loading the GroupBar. By using the SaveOriginalState property, you can enable the saving the state on loading. This dependency property sets the value indicating whether to save the state persisted on loading the GroupBar.

Use the below code snippet to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" SaveOriginalState="True"
Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem"
IsSelected="True">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewModel="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Enable the save state property
groupBar.SaveOriginalState = true;
```

GroupBar Items in WPF Navigation Pane (GroupBar)

This topic illustrates the how to work with group-bar items.

Status of the GroupBar Item

You can enable or disable the dragging of items using the DraggingItemInProgress property. This is a dependency property which gets or sets the value indicating whether the dragging of the items is in progress.

Use the below code snippet to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" DraggingItemInProgress="True" Width="230"
Name="groupBar">
```



```
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewModel="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Setting the items in progress while dragging
groupBar.DraggingItemInProgress = true;
```

Cursor Type for the GroupBar Item

You can customize the cursor type of GroupBar item using the GroupBarItemCursorType property. This dependency property sets the type of cursor for the groupbar item. There are two types of built-in cursor types available for the GroupBar item. They are as follows.

- Default – default cursor type is displayed while moving the mouse over GroupBar item
- Hand – hand cursor type is displayed while moving the mouse over GroupBar item

Use the below code snippet to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" GroupBarItemCursorType="Hand" Width="230"
Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
```

```

</StackPanel>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>

```

C#

```

//Setting the cursor type for GroupBar item
groupBar.GroupBarItemCursorType = ItemCursorType.Hand;

```

Content Length of the GroupBar Item

You can also set the length of the content in GroupBar item. This is done using the `ItemContentLength` property. This dependency property sets the height or width available for the content of the selected item, i.e., it sets the height for the content, if GroupBar layout is *vertical*, and width for the content, if GroupBar layout is *horizontal*.

Use the below code snippet to set this property.

XML

```

<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" Width="230" ItemContentLength="100"
Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem"
IsSelected="True">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>

```

```
</syncfusion:GroupBar>
```

C#

```
//Set the length of the content  
groupBar.ItemContentLength = 100;
```

Hidden GroupBar Item

You can get the hidden GroupBar items and do some operation depending upon the visibility by using the property `HiddenItemsCount`. This dependency property gets the number of GroupBar items that are hidden.

Use the below code to set this property.

C#

```
int count;  
//Get the hidden GroupBar items  
count=groupBar.HiddenItemsCount;
```

Setting Corner Radius for the GroupBar Item

It is now possible to set the corner radius of GroupBar Items in the GroupBar by using the `GroupBarItemCornerRadius` property. This helps the user to specify the degree of roundness at the tip of the GroupBar.

The following code examples illustrate how to set this property.

XML

```
<syncfusion:GroupBarItem HeaderText="Mailbox" GroupBarItemCornerRadius="20"  
ShowInGroupBar="True">
```

C#

```
// Creating an instance of GroupBar.  
GroupBar groupBar = new GroupBar();  
// Creating an instance of GroupBar Item.  
GroupBarItem groupBarItem = new GroupBarItem();  
// Setting Header Text for GroupBar Item.  
groupBarItem.HeaderText = "Mailbox";  
// Setting Corner Radius For GroupBar Item.  
groupBarItem.GroupBarItemCornerRadius = new CornerRadius(20d);  
// Setting ShowInGroupBar property for GroupBar Item.  
groupBarItem.ShowInGroupBar = true;  
// Adding GroupBar Item to GroupBar.  
groupBar.Items.Add(groupBarItem);
```

Run the code. The following output is displayed.



Appearance in WPF Navigation Pane (GroupBar)

This section deals with the appearance of GroupBar control and contains the following topics:

Visual Mode

Visual Mode feature in GroupBar is used to change the visual style of the GroupBar. This is done using the VisualMode property. This dependency property can be used to set the visual mode for GroupBar content. It returns the visual mode of the GroupBar.

There are three types of visual modes available in GroupBar.

- Default
- MultipleExpansion
- StackMode

Default

By setting this enum to the VisualMode property, expand and collapse of the GroupBarItem in GroupBar is enabled. Only one GroupBar item can be expanded at a time. On expanding another GroupBar item, the previously expanded groupbar item will be collapsed.

MultipleExpansion

By setting this enum to the VisualMode property, the style of GroupBar will be similar to the tree view structure in terms of expand and collapse. Any number of GroupBar items can be expanded or collapsed.

StackMode

By setting this enum to the VisualMode property, the GroupBar behavior is set to behave like the Outlook GroupBar.

Use the following code snippet to set the VisualMode property to *Default*.

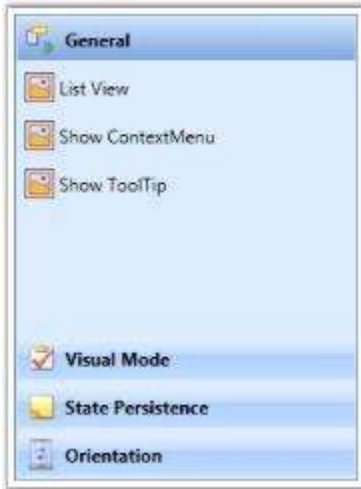
XML

```
<!-- Adding GroupBar that have visual mode is Default -->
<syncfusion:GroupBar Height="300" Width="230" VisualMode="Default"
Name="groupBar">
```

```
<!-- Adding GroupBarItem -->
```

C#

```
//Setting visual mode as Default
groupBar.VisualMode = VisualMode.Default;
```



Multiple Expansion

In Multiple Expansion mode, more than one item can be expanded as seen in a tree view structure.

Use the following code snippet to set VisualMode property to Multiple Expansion.

XML

```
<!-- Adding GroupBar that have visual mode is Multiple Expansion -->
<syncfusion:GroupBar Height="300" Width="230" VisualMode="MultipleExpansion"
Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
    <!-- Adding content for groupbar item using GroupView -->
    <syncfusion:GroupView>
      <syncfusion:GroupViewItem Text="List View"/>
      <syncfusion:GroupViewItem Text="Show ContextMenu"/>
      <syncfusion:GroupViewItem Text="Show ToolTip"/>
    </syncfusion:GroupView>
  </syncfusion:GroupBarItem>
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem HeaderImageSource="Tasks.png" Name="groupBarItem2"
Header="Visual Mode">
    <!-- Adding content for groupbar item using GroupView -->
    <syncfusion:GroupView>
      <syncfusion:GroupViewItem Text="Default"/>
      <syncfusion:GroupViewItem Text="Multiple Expansion"/>
      <syncfusion:GroupViewItem Text="StackMode"/>
    </syncfusion:GroupView>
  </syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

```

<syncfusion:GroupBarItem HeaderImageSource="Notes.png" Name="groupBarItem3"
Header="State Persistence">
<!-- Adding content for groupbar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Save State"/>
<syncfusion:GroupViewItem Text="Load State"/>
<syncfusion:GroupViewItem Text="Reset State"/>
</syncfusion:GroupView> </syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem4" HeaderImageSource="bin.png"
Header="Orientation">
<!-- Adding content for groupbar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel> </syncfusion:GroupBarItem></syncfusion:GroupBar>

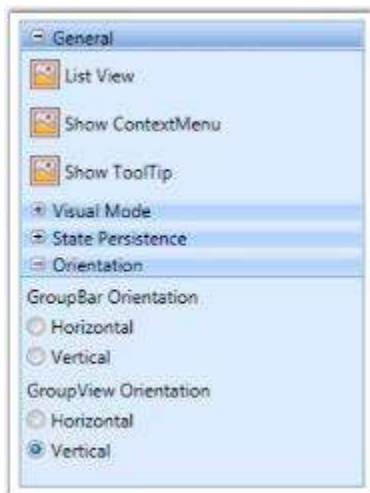
```

C#

```

//groupBar is a name of groupbar
//groupbar that have visual mode is MultipleExpansion
groupBar.VisualMode = VisualMode.MultipleExpansion;

```

**Stack Mode**

In StackMode, only one item can be expanded, Items are organized in stack-like mode. It looks like an Outlook groupbar. In stack mode, the pop-up menu is displayed at the bottom of the groupbar. This helps customize groupbar items in a GroupBar control.

Use the following code snippet to set the VisualMode property to StackMode.

XML

```

<!-- Adding GroupBar that have visual mode is StackMode-->
<syncfusion:GroupBar Height="300" Width="230" VisualMode="StackMode"
Name="groupBar">

```

```

<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for groupbar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Tasks.png" Name="groupBarItem2"
Header="Visual Mode">
<!-- Adding content for groupbar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Default"/>
<syncfusion:GroupViewItem Text="Multiple Expansion"/>
<syncfusion:GroupViewItem Text="StackMode"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Notes.png" Name="groupBarItem3"
Header="State Persistence">
<!-- Adding content for groupbar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Save State"/>
<syncfusion:GroupViewItem Text="Load State"/>
<syncfusion:GroupViewItem Text="Reset State"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem4" HeaderImageSource="bin.png"
Header="Orientation">
<!-- Adding content for groupbar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>

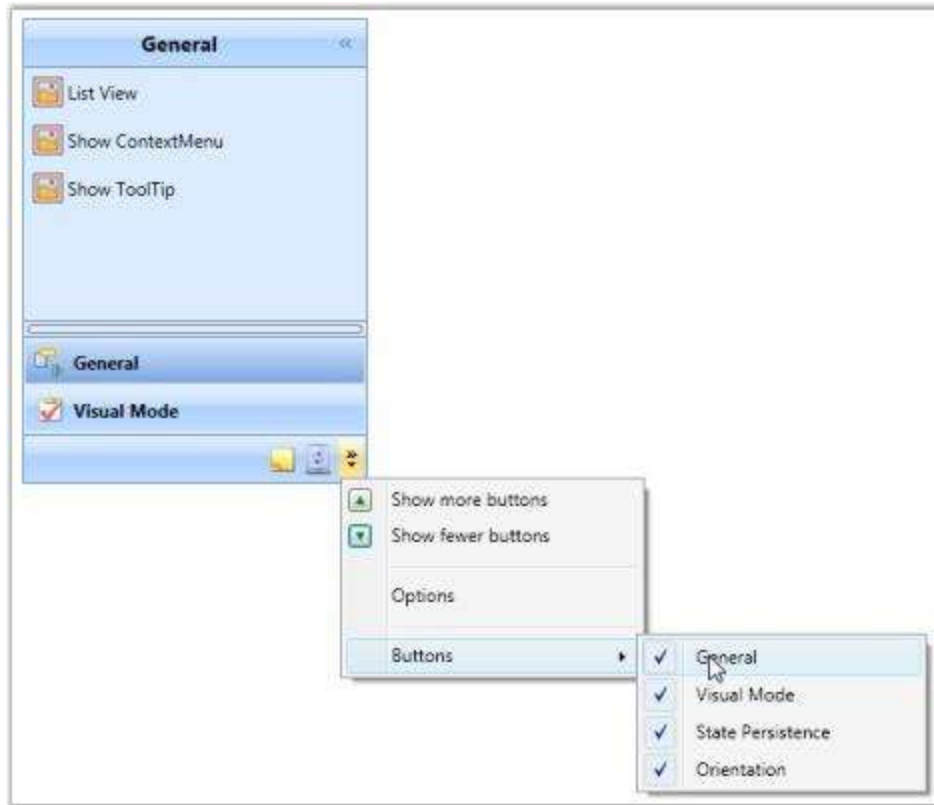
```

C#

```

//groupBar is a name of groupbar
//groupbar that have visual mode is StackMode
groupBar.VisualMode = VisualMode.StackMode;

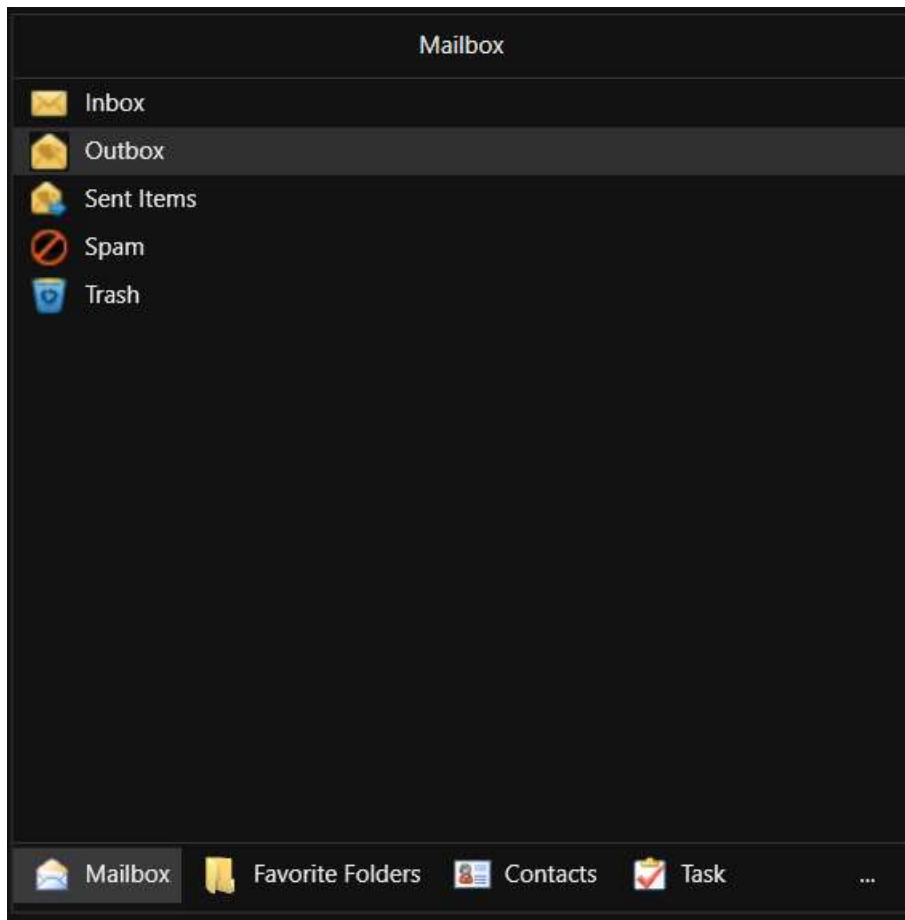
```



Theme

GroupBar supports various built-in themes. Refer to the below links to apply themes for the GroupBar,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Item Container Style

1. The `ItemContainerStyle` property of `GroupBar` sets the style for `GroupBarItem`. This style will be applied to all items in the `GroupBar` control.style for `GroupBarItem`.

XML

```
<Style TargetType="{x:Type syncfusion:GroupBarItem}"
x:Key="GroupBarItemStyle">
  <Setter Property="HeaderTemplate" >
    <Setter.Value>
      <DataTemplate >
        <Grid>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="25" />
            <ColumnDefinition Width="*" />
          </Grid.ColumnDefinitions>
          <TextBlock Text="{Binding XPath=@Name}" Margin="5" Foreground="Green"
            VerticalAlignment="Center" FontWeight="Bold" FontFamily="Bookman Old Style"
            Grid.Column="1"/>
        </Grid>
      </DataTemplate>
    </Setter.Value>
  </Setter>
  <Setter Property="ContentTemplate">
```

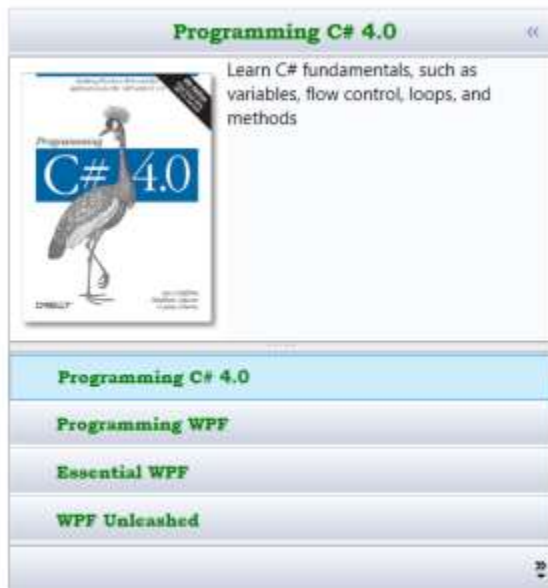
```
<Setter.Value>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="4*" />
<ColumnDefinition Width="6*" />
</Grid.ColumnDefinitions>
<Image Source="{Binding XPath=@ImagePath}" />
<TextBlock Text="{Binding XPath=@Description}" TextWrapping="Wrap"
Grid.Column="1" />
</Grid>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
```

- Set ItemContainerStyle of GroupBar as follows.

XML

```
<syncfusion:GroupBar Name="groupBar1" AllowCollapse="True"
VisualMode="StackMode"
ItemsSource="{Binding Source={StaticResource xmlSource}, XPath=Book}"
ItemContainerStyle="{StaticResource GroupBarItemStyle}"
/>
```

- This will generate the following GroupBar control.



Item Container Style Selector

The ItemContainerStyleSelector property is used to choose the ItemContainerStyle at run time based on specified conditions.

1. Create the styles in the Window's resource.

XML

```

<Style TargetType="{x:Type syncfusion:GroupBarItem}" x:Key="WpfItemStyle">
  <Setter Property="HeaderTemplate" >
    <Setter.Value>
      <DataTemplate>
        <StackPanel Orientation="Horizontal">
          <Image Margin="3"
            Source="D:\IUE_DOC\GroupBarWPF\GroupBarIUEDemo\GroupBarIUEDemo\Data\wpf.png"
            Width="20" Height="20"/>
          <TextBlock Text="{Binding XPath=@Name}" FontWeight="Bold"
            Foreground="Blue"/>
        </StackPanel>
      </DataTemplate>
    </Setter.Value>
  </Setter>
  <Setter Property="ContentTemplate">
    <Setter.Value>
      <DataTemplate>
        <Grid >
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="4*" />
            <ColumnDefinition Width="6*" />
          </Grid.ColumnDefinitions>
          <Image Source="{Binding XPath=@ImagePath}" />
          <TextBlock Text="{Binding XPath=@Description}" TextWrapping="Wrap"
            Grid.Column="1"/>
        </Grid>
      </DataTemplate>
    </Setter.Value>
  </Setter>
</Style>
<Style TargetType="{x:Type syncfusion:GroupBarItem}" x:Key="CsItemStyle">
  <Setter Property="HeaderTemplate" >
    <Setter.Value>
      <DataTemplate>
        <StackPanel Orientation="Horizontal">
          <Image Margin="3"
            Source="D:\IUE_DOC\GroupBarWPF\GroupBarIUEDemo\GroupBarIUEDemo\Data\images.jpg"
            Width="20" Height="20"/>
          <TextBlock Text="{Binding XPath=@Name}" FontWeight="Bold"
            Foreground="Green"/>
        </StackPanel>
      </DataTemplate>
    </Setter.Value>
  </Setter>
  <Setter Property="ContentTemplate">
    <Setter.Value>
      <DataTemplate>
        <Grid >
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="4*" />
            <ColumnDefinition Width="6*" />
          </Grid.ColumnDefinitions>

```

```
<Image Source="{Binding XPath=@ImagePath}"/>
<TextBlock Text="{Binding XPath=@Description}" TextWrapping="Wrap"
Grid.Column="1"/>
</Grid>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
```

2. Create the StyleSelector in the code.

CSHARP

```
public class GroupBarItemContainerStyleSelector : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        Window window = Application.Current.MainWindow;
        string bookname = (item as
        System.Xml.XmlElement).GetAttribute("Name").ToString().ToLower();
        if (bookname.Contains("wpf"))
        {
            return ((Style)window.Resources["WpfItemStyle"]);
        }
        else
        {
            return ((Style)window.Resources["CsItemStyle"]);
        }
    }
}
```

3. Now define the style selector in the Window's resource.

XML

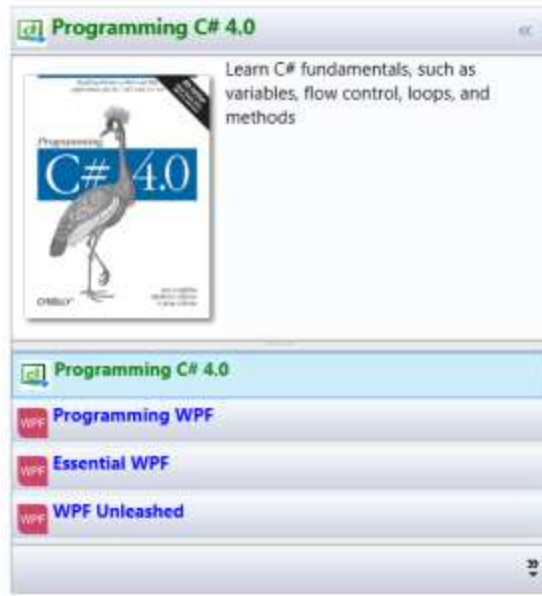
```
<local:GroupBarItemContainerStyleSelector
x:Key="groupBarItemContainerStyleSelector"/>
```

4. Now set ItemContainerStyle for the GroupBar control.

XML

```
<syncfusion:GroupBar Name="groupBar1" AllowCollapse="True"
VisualMode="StackMode"
ItemsSource="{Binding Source={StaticResource xmlSource}, XPath=Book}"
ItemContainerStyleSelector="{StaticResource
groupBarItemContainerStyleSelector}"
/>
```

This will generate the following GroupBar control.



Drag Marker Color

You can apply a custom brush for the drag marker by using the `DragMarkerBrush` property. This brush setting can be visualized while dragging the items in the GroupBar.

`DragMarkerBrush` – This dependency property sets the brush value for the drag marker.

Use the below code snippet to set this property.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="200" DragMarkerBrush="Red" Width="230"
Name="groupBar">
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem" Header="GroupBarItem">
    <!-- Adding content for GroupBar item using panel -->
    <StackPanel Orientation="Vertical">
      <TextBlock Text="GroupBar Orientation" Margin="4,4,2,2"/>
      <RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
      <RadioButton Margin="4,2,2,2">Vertical</RadioButton>
      <TextBlock Text="GroupView Orientation" Margin="4,4,2,2"/>
      <RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
      <RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
    </StackPanel>
  </syncfusion:GroupBarItem>
  <!-- Adding GroupBarItem -->
  <syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
    <!-- Adding content for GroupBar item using GroupView -->
    <syncfusion:GroupView Name="groupView" IsListViewModel="True">
      <syncfusion:GroupViewItem Text="List View"/>
      <syncfusion:GroupViewItem Text="Show ContextMenu"/>
      <syncfusion:GroupViewItem Text="Show ToolTip"/>
    </syncfusion:GroupView>
  </syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

C#

```
//Setting drag marker brush
groupBar.DragMarkerBrush = Brushes.Red;
```

Custom Template for the Collapse Button

CollapseButtonTemplate property is used to customize the collapse button by applying custom templates. The target type of template is toggle button. By using this property, the user must write their own control template of target type ToggleButton.

Use the following code snippet for setting a template for the CollapsedButton.

XML

```
<!-- Adding GroupBar that has animation speed -->
<syncfusion:GroupBar Height="300" Width="230" Name="groupBar"
AllowCollapse="True" VisualMode="StackMode">
<syncfusion:GroupBar.CollapseButtonTemplate>
<ControlTemplate TargetType="{x:Type ToggleButton}">
<Border Height="16" Width="16">
<Path Name="path" Data="M0,0L3.5,4 7,0z" Fill="Black"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Border>
<ControlTemplate.Triggers>
<Trigger Property="IsChecked" Value="True">
<Setter Property="Data" TargetName="path" Value="M 3 1 L 7 5 L 3 9 z"/>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</syncfusion:GroupBar.CollapseButtonTemplate>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
```

Collapse Button Color

You can also set the background brush and MouseOver background brush for the collapse button using the CollapseButtonBackground and CollapseButtonMouseOverBackground properties, respectively.

Use the following code snippet to set these properties.

XML

```
<!-- Adding GroupBar -->
<syncfusion:GroupBar Height="300" Width="230" Name="groupBar"
CollapseButtonBackground="AliceBlue"
```

```

CollapseButtonMouseOverBackground="Bisque" AllowCollapse="True"
VisualMode="StackMode">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewModel="True">
<syncfusion:GroupViewItem Text="List View"/>
<syncfusion:GroupViewItem Text="Show ContextMenu"/>
<syncfusion:GroupViewItem Text="Show ToolTip"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Tasks.png" Name="groupBarItem2"
Header="Visual Mode">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Default"/>
<syncfusion:GroupViewItem Text="Multiple Expansion"/>
<syncfusion:GroupViewItem Text="StackMode"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem HeaderImageSource="Notes.png" Name="groupBarItem3"
Header="State Persistence">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView>
<syncfusion:GroupViewItem Text="Save State"/>
<syncfusion:GroupViewItem Text="Load State"/>
<syncfusion:GroupViewItem Text="Reset State"/>
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>

```

C#

```

//Enable AllowCollapse property for GroupBar
groupBar.AllowCollapse = true;
//Set the Background brush for Collapse button
groupBar.CollapseButtonBackground = Brushes.AliceBlue;
//Set the MouseOverBackground brush for Collapse Button
groupBar.CollapseButtonMouseOverBackground = Brushes.Bisque;

```

Customizing Group Bar Header

Group Bar header can be customized in a required manner. Customized background color, corner radius, height, text alignment etc. for a Group Bar header can be customized in the following way.

XML

```

<Style x:Key="headerStyle" TargetType="{x:Type Border}">
<Setter Property="CornerRadius" Value="7"/>
<Setter Property="Background" Value="Pink"/>
<Setter Property="Height" Value="30" />
</Style>
<syncfusion:GroupBar GroupBarHeaderStyle="{StaticResource headerStyle}"/>

```



Populating-Data in WPF Navigation Pane (GroupBar)

Data can be populated in the GroupBar control through XAML, C#, or XML. The GroupBar control also supports binding to objects. The following sections describe how to implement these different forms of data population.

Through XAML

Create a GroupBar control in XAML, as seen below.

XML

```
<syncfusion:GroupBar Name="groupBar1">
  <syncfusion:GroupBarItem HeaderText="NewGroupBarItem1" IsSelected="True" >
    <syncfusion:GroupView>
      <syncfusion:GroupViewItem Text="New GroupViewItem" />
    </syncfusion:GroupView>
  </syncfusion:GroupBarItem>
  <syncfusion:GroupBarItem HeaderText="NewGroupBarItem2" />
  <syncfusion:GroupBarItem HeaderText="NewGroupBarItem3" />
</syncfusion:GroupBar>
```

Through C#

To create a GroupBar control in C# , include the following namespace to the directives list.

C#

```
using Syncfusion.Windows.Tools.Controls;
```

Next, create the GroupBar as follows.

C#

```
GroupBar gBar = new GroupBar();
GroupBarItem gBarItem1 = new GroupBarItem() { HeaderText
="NewGroupBarItem1", IsSelected = true };
GroupView gView = new GroupView();
GroupViewItem gViewItem = new GroupViewItem() { Text="New GroupViewItem"};
gView.Items.Add(gViewItem);
```



```

gBarItem1.Content = gView;
GroupBarItem gBarItem2 = new GroupBarItem() {
HeaderText="NewGroupBarItem2"};
GroupBarItem gBarItem3 = new GroupBarItem() {
HeaderText="NewGroupBarItem3"};
gBar.Items.Add(gBarItem1);
gBar.Items.Add(gBarItem2);
gBar.Items.Add(gBarItem3);

```

This will generate the following GroupBar control.



Data-Binding in WPF Navigation Pane (GroupBar)

You can bind the custom object to the DataContext property of the GroupBar control and their corresponding elements can be bound with the children of the GroupBar control. When a DataContext is set to a window or to a GroupBar, it gets inherited to all its logical children. The OnInitialized method will get this DataContext value. As you use this DataContext in the template with DataTriggers, you need to set that property to active host tab or null.

Use the code snippet to apply a DataContext to the GroupBar control.

XML

```

<Window x:Class="WpfApplication1.Window1"
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:local="clr-namespace:WpfApplication1" Title="Window1" Height="300"
Width="300">
<Window.Resources>
<!--Custom object which is defined in the code behind can be accessed
through the key logic in XAML-->
<local:GroupData x:Key="groupData" />
</Window.Resources>
<Grid Margin="30">
<!-- Adding GroupBar -->

```

```
<syncfusion:GroupBar Height="200" DataContext="{Binding groupData}"
VisualMode="Default" AllowCollapse="True" Width="230" Name="groupBar">
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem" Header="{Binding Header}">
<!-- Adding content for GroupBar item using panel -->
<StackPanel Orientation="Vertical">
<TextBlock Text="GroupBar Orientation" Margin="4,4,2,2" />
<RadioButton IsChecked="True" Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton Margin="4,2,2,2">Vertical</RadioButton>
<TextBlock Text="GroupView Orientation" Margin="4,4,2,2" />
<RadioButton Margin="4,2,2,2">Horizontal</RadioButton>
<RadioButton IsChecked="True" Margin="4,2,2,2">Vertical</RadioButton>
</StackPanel>
</syncfusion:GroupBarItem>
<!-- Adding GroupBarItem -->
<syncfusion:GroupBarItem Name="groupBarItem1" HeaderImageSource="Label.gif"
Header="General">
<!-- Adding content for GroupBar item using GroupView -->
<syncfusion:GroupView Name="groupView" IsListViewMode="True">
<syncfusion:GroupViewItem Text="{Binding ListView}" />
<syncfusion:GroupViewItem Text="Show ContextMenu" />
<syncfusion:GroupViewItem Text="Show ToolTip" />
</syncfusion:GroupView>
</syncfusion:GroupBarItem>
</syncfusion:GroupBar>
</Grid>
</Window>
```

C#

```
/// <summary>/// Interaction logic for the class GroupData/// </summary>
public class GroupData
{
    /// <summary>
    /// Initializes a new instance of the <see cref="GroupData"/> class.
    /// </summary>
    public GroupData()
    {
        this.Header = "GroupBarItem1";
        this.ListView = "ListViewItem";
    }
    /// <summary>
    /// Gets or sets the header.
    /// </summary>
    /// <value>The header.</value>
    public string Header
    {
        get;
        set;
    }
    /// <summary>
    /// Gets or sets the list view.
    /// </summary>
    /// <value>The list view.</value>
    public string ListView
    {

```

```
get;  
set;  
}  
}
```

Data-Binding to Objects

The GroupBar control also supports binding to objects. The following example demonstrates this.

1. Create a class that acts as a model for the GroupBar control.

CSHARP

```
public class Model  
{  
    public string Header  
    {  
        get;  
        set;  
    }  
    public string Content  
    {  
        get;  
        set;  
    }  
    public bool IsSelected  
    {  
        get;  
        set;  
    }  
}
```

2. Create a ViewModel class and initialize the items as follows.

CSHARP

```
public class ViewModel  
{  
    public ObservableCollection < Model > GroupItems  
    {  
        get;  
        set;  
    }  
    public ViewModel()  
    {  
        GroupItems = new ObservableCollection < Model > ();  
        PopulateData();  
    }  
    private void PopulateData()  
    {  
        Model model1 = new Model()  
        {  
            Header = "Item1", Content = "GroupBarItem1", IsSelected = true  
        };  
    }  
}
```

```
Model model2 = new Model()
{
    Header = "Item2", Content = "GroupBarItem2"
};
Model model3 = new Model()
{
    Header = "Item3", Content = "GroupBarItem3"
};
Model model4 = new Model()
{
    Header = "Item4", Content = "GroupBarItem4"
};
GroupItems.Add(model1);
GroupItems.Add(model2);
GroupItems.Add(model3);
GroupItems.Add(model4);
}
```

3. Create a ViewModel instance and use it as DataContext for the root window.

XML

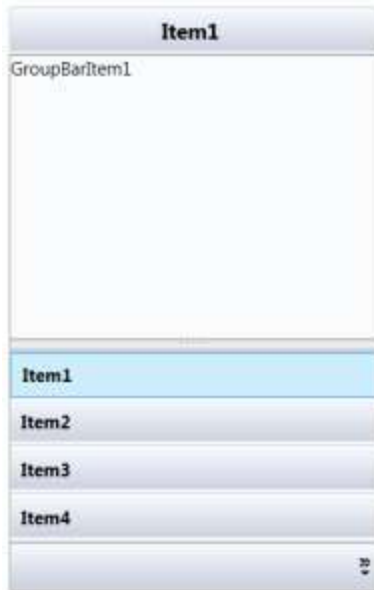
```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
```

4. Now configure the ItemsSource and ItemContainerStyle of GroupBar.

XML

```
<syncfusion:GroupBar Name="groupBar1" ItemsSource="{Binding GroupItems}"
    VisualMode="StackMode">
    <syncfusion:GroupBar.ItemContainerStyle>
        <Style TargetType="{x:Type syncfusion:GroupBarItem}">
            <Setter Property="HeaderText" Value="{Binding Header}"/>
            <Setter Property="Content" Value="{Binding Content}"/>
            <Setter Property="IsSelected" Value="{Binding IsSelected}" />
        </Style>
    </syncfusion:GroupBar.ItemContainerStyle>
</syncfusion:GroupBar>
```

This creates the following GroupBar control.



Data-Binding with XML

An XML file can also be used as the ItemsSource for the GroupBar control. The following example illustrates this.

1. Create an XML file with the following information and name it Data.xml.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Books>
  <Book Name="Programming C# 4.0" Description="Learn C# fundamentals, such as
variables, flow control, loops, and methods" ImagePath="programming-c-sharp-
four.png" />
  <Book Name="Programming WPF" Description="A tutorial on XAML, the new HTML-
like markup language for declaring Windows UI" ImagePath="programming-
wpf.png" />
  <Book Name="Essential WPF" Description="Visuals and media, including 2D, 3D,
video, and animation" ImagePath="essential_wpf.png" />
  <Book Name="WPF Unleashed" Description="Examines the WPF feature areas in
incredible depth: controls, layout, resources, data binding, styling,
graphics, animation, and more" ImagePath="wpf-unleashed.png" />
</Books>
```

2. Add the XmlDataProvider for the XML document.

XML

```
<XmlDataProvider Source="Data.xml" x:Key="xmlSource" XPath="Books" />
```

3. ItemsSource property for the GroupBar control.

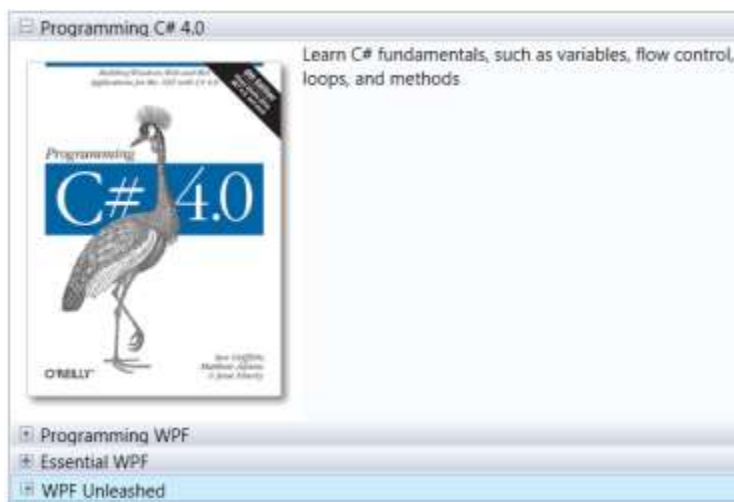
XML

```

<syncfusion:GroupBar Name="groupBar1" Margin="20" ItemsSource="{Binding
Source={StaticResource xmlSource}, XPath=Book}"
VisualMode="MultipleExpansion">
<syncfusion:GroupBar.ItemContainerStyle>
<Style TargetType="{x:Type syncfusion:GroupBarItem}">
<Setter Property="HeaderTemplate">
<Setter.Value>
<DataTemplate>
<Grid>
<TextBlock Text="{Binding XPath=@Name}" />
</Grid>
</DataTemplate>
</Setter.Value>
</Setter>
<Setter Property="ContentTemplate">
<Setter.Value>
<DataTemplate><
Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="4*" />
<ColumnDefinition Width="6*" />
</Grid.ColumnDefinitions>
<Image Source="{Binding XPath=@ImagePath}" />
<TextBlock Text="{Binding XPath=@Description}" TextWrapping="Wrap"
Grid.Column="1" />
</Grid>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:GroupBar.ItemContainerStyle>undefined</syncfusion:GroupBar>

```

This will create the following GroupBar control.



Customizing-Data-Templates in WPF Navigation Pane (GroupBar)

Data templates can be customized for items, headers, and content. The next sections explain how to customize data templates.

Item Template

You can customize how a business object is displayed by using ItemTemplate of GroupBarItem. The following example shows how to use ItemTemplate. following code, GroupBarItem's header is customized with DataTemplate. The value of Header is assigned to the TextBlock's text and TextBlock's properties like FontWeight, FontFamily and Foreground are modified for a customized look.

XML

```
<syncfusion:GroupBar Name="groupBar1" Margin="20" AllowCollapse="True"
VisualMode="StackMode" ItemsSource="{Binding GroupItems}" >
<syncfusion:GroupBar.ItemTemplate>
<DataTemplate>
<Grid>
<TextBlock Text="{Binding Header}" Margin="5" Foreground="Green"
VerticalAlignment="Center" FontWeight="Bold" FontFamily="Bookman Old Style"
Grid.Column="1"/>
</Grid>
</DataTemplate>
</syncfusion:GroupBar.ItemTemplate>
</syncfusion:GroupBar>
```

Implementing the above code will create the following GroupBar control.



Item Template Selector

Using ItemTemplateSelector, you can use different templates for items depending on specific constraints. The following example illustrates this.

1. Create the template selector in the code.

CSHARP

```
public class GroupBarItemTemplateSelector : DataTemplateSelector
{
public override DataTemplate SelectTemplate(object item, DependencyObject
container)
{
Window window = Application.Current.MainWindow;
string bookname = (item as
System.Xml.XmlElement).GetAttribute("Name").ToString().ToLower();
if (bookname.Contains("wpf"))
{
```

```

return ((DataTemplate)window.Resources["WpfBookTemplate"]);
}
else
{
return ((DataTemplate)window.Resources["CsBookTemplate"]);
}
}
}

```

2. Define the data templates in the Window's resources.

XML

```

<DataTemplate x:Key="WpfBookTemplate">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="25" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Image Source="wpf.png"/>
<TextBlock Text="{Binding XPath=@Name}" Margin="5" Foreground="Green"
VerticalAlignment="Center" FontWeight="Bold" FontFamily="Bookman Old Style"
Grid.Column="1"/>
</Grid>
</DataTemplate>
<DataTemplate x:Key="CsBookTemplate">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="25" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Image Source="images.jpg"/>
<TextBlock Text="{Binding XPath=@Name}" Margin="5" Foreground="Green"
VerticalAlignment="Center" FontWeight="Bold" FontFamily="Bookman Old Style"
Grid.Column="1"/>
</Grid>
</DataTemplate>

```

3. The template selector in the Window's resources.

XML

```

<local:GroupBarItemTemplateSelector
x:Key="groupBarItemTemplateSelector"/>

```

4. Use this template selector to choose a template for the GroupBar control.

XML

```

<syncfusion:GroupBar Name="groupBar1" Margin="20" AllowCollapse="True"
VisualMode="StackMode" ItemTemplateSelector="{StaticResource

```



```
groupBarItemTemplateSelector}" ItemsSource="{Binding Source={StaticResource
xmlSource}, XPath=Book}" >
</syncfusion:GroupBar>
```

This will generate the following GroupBar control.



Header Template

You can customize the header of a GroupViewItem by using a header template. This is illustrated in the following example.

1. Define the data template for the header as follows.

XML

```
<DataTemplate x:Key="headerTemplate">
<Grid>
<Border Background="Gray">
<TextBlock Text="{Binding XPath=@Name}" Margin="5" Foreground="White"
VerticalAlignment="Center" FontWeight="Bold" FontFamily="Bookman Old Style"
Grid.Column="1"/>
</Border>
</Grid>
</DataTemplate>
```

2. Set HeaderTemplate for GroupBarItem to the above template.

XML

```
<syncfusion:GroupBar Name="groupBar1" Margin="20" VisualMode="StackMode"
ItemsSource="{Binding Source={StaticResource xmlSource}, XPath=Book}">
<syncfusion:GroupBar.ItemContainerStyle>
<Style TargetType="{x:Type syncfusion:GroupBarItem}">
<Setter Property="HeaderTemplate" Value="{StaticResource headerTemplate}" />
```

```

</Style>
</syncfusion:GroupBar.ItemContainerStyle>
</syncfusion:GroupBar>

```

The code above applies HeaderTemplate to the GroupBar, so the headers of the group-bar items will contains a text box with a white foreground.

Content Template

You can customize the content of GroupViewItem by using ContentTemplate. This is demonstrated in the following example.

1. Define DataTemplate for the content as follows.

XML

```

<DataTemplate x:Key="contentTemplate">
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="4*" />
<ColumnDefinition Width="6*" />
</Grid.ColumnDefinitions>
<Image Source="{Binding XPath=@ImagePath}" />
<TextBlock Text="{Binding XPath=@Description}" TextWrapping="Wrap"
Grid.Column="1" />
</Grid>
</DataTemplate>

```

2. Set ContentTemplate for GroupBarItem to the above template.

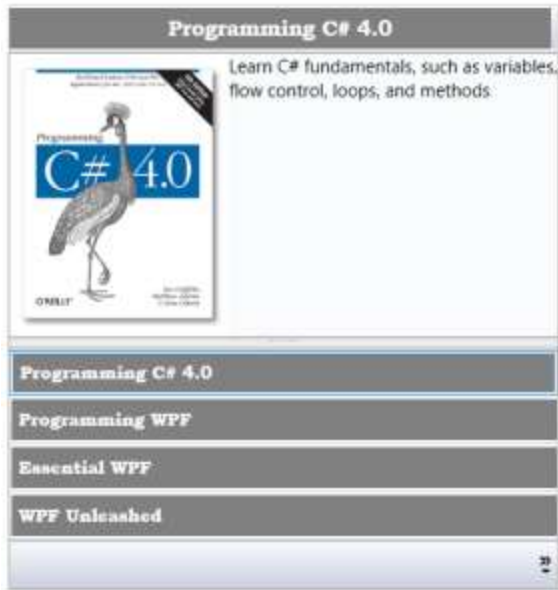
XML

```

<syncfusion:GroupBar Name="groupBar1" Margin="20" VisualMode="StackMode"
ItemsSource="{Binding Source={StaticResource xmlSource}, XPath=Book}">
<syncfusion:GroupBar.ItemContainerStyle>
<Style TargetType="{x:Type syncfusion:GroupBarItem}">
<Setter Property="HeaderTemplate" Value="{StaticResource headerTemplate}" />
<Setter Property="ContentTemplate" Value="{StaticResource
contentTemplate}" />
</Style>
</syncfusion:GroupBar.ItemContainerStyle>
</syncfusion:GroupBar>

```

This will populate the GroupBar control.



Header Template Selector

With `HeaderTemplateSelector`, you can use different templates for the `GroupBarItem`'s header depending on specific constraints. The following example illustrates this.

1. Create the template selector in code.

CSHARP

```
public class GroupBarItemHeaderTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        Window window = Application.Current.MainWindow;
        string bookname = (item as System.Xml.XmlElement).GetAttribute("Name").ToString().ToLower();
        if (bookname.Contains("wpf"))
        {
            return ((DataTemplate)window.Resources["WpfBookHeaderTemplate"]);
        }
        else
        {
            return ((DataTemplate)window.Resources["CsBookHeaderTemplate"]);
        }
    }
}
```

2. Define the data templates in the Window's resources.

XML

```
<DataTemplate x:Key="WpfBookHeaderTemplate">
<Grid>
```

```

<Grid.ColumnDefinitions>
<ColumnDefinition Width="25" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Image Source="wpficon.png"/>
<TextBlock Text="{Binding XPath=@Name}" Margin="5" Foreground="Green"
VerticalAlignment="Center" FontWeight="Bold" FontFamily="Bookman Old Style"
Grid.Column="1"/>
</Grid>
</DataTemplate>
<DataTemplate x:Key="CsBookHeaderTemplate">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="25" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Image Source="csicon.png"/>
<TextBlock Text="{Binding XPath=@Name}" Margin="5" Foreground="Blue"
VerticalAlignment="Center" FontWeight="Bold" FontFamily="Bookman Old Style"
Grid.Column="1"/>
</Grid>
</DataTemplate>

```

3. Create an instance for the template selector in the Window's resources.

XML

```

<local:GroupBarItemHeaderTemplateSelector
x:Key="groupBarItemHeaderTemplateSelector"/>

```

Now assign the key given in the above code to GroupBar's HeaderTemplateSelector.

Content Template Selector

With ContentTemplateSelector, you can use different templates for GroupBarItem's content depending on specific constraints. The following example illustrates this.

1. Create the template selector in the code as follows.

C#

```

public class GroupBarItemContentTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject
container)
    {
        Window window = Application.Current.MainWindow;
        string bookname = (item as
System.Xml.XmlElement).GetAttribute("Name").ToString().ToLower();
        if (bookname.Contains("wpf"))
        {
            return ((DataTemplate)window.Resources["WpfBookContentTemplate"]);
        }
        else
    }
}

```

```
{
    return ((DataTemplate)window.Resources["CsBookContentTemplate"]);
}
}
```

2. Define the data templates in the Window's resources.

CSHARP

```
<DataTemplate x:Key="CsBookHeaderTemplate">
    <Grid>
    <Grid.ColumnDefinitions>
    <ColumnDefinition Width="25" />
    <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Image Source="csicon.png"/>
    <TextBlock Text="{Binding XPath=@Name}" Margin="5" Foreground="Blue"
    VerticalAlignment="Center" FontWeight="Bold" FontFamily="Bookman Old Style"
    Grid.Column="1"/>
    </Grid>
</DataTemplate>
<DataTemplate x:Key="WpfBookContentTemplate">
    <Grid >
    <Grid.ColumnDefinitions>
    <ColumnDefinition Width="4*" />
    <ColumnDefinition Width="6*" />
    </Grid.ColumnDefinitions>
    <Image Source="{Binding XPath=@ImagePath}"/>
    <TextBlock Text="{Binding XPath=@Description}" TextWrapping="Wrap"
    Foreground="Green" Grid.Column="1"/>
    </Grid>
</DataTemplate>
```

3. Create an instance of the template selector in the Window's resources.

CSHARP

```
<local:GroupBarItemContentTemplateSelector
x:Key="groupBarItemContentTemplateSelector"/>
```

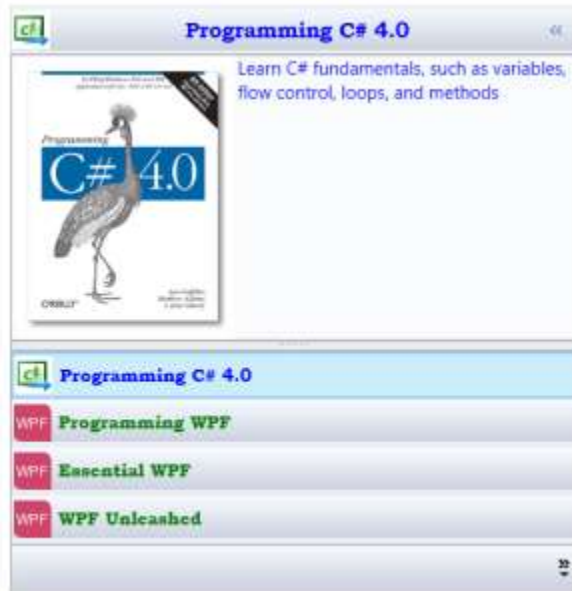
4. Now use HeaderTemplateSelector and ContentTemplateSelector.

CSHARP

```
<syncfusion:GroupBar Name="groupBar1" AllowCollapse="True"
VisualMode="StackMode" ItemsSource="{Binding Source={StaticResource
xmlSource}, XPath=Book}" >
<syncfusion:GroupBar.ItemContainerStyle>
<Style TargetType="{x:Type syncfusion:GroupBarItem}">
```

```
<Setter Property="HeaderTemplateSelector" Value="{StaticResource
groupBarItemHeaderTemplateSelector}"/>
<Setter Property="ContentTemplateSelector" Value="{StaticResource
groupBarItemContentTemplateSelector}"/>
</Style>
</syncfusion:GroupBar.ItemContainerStyle>
</syncfusion:GroupBar>
```

This will populate the GroupBar control.

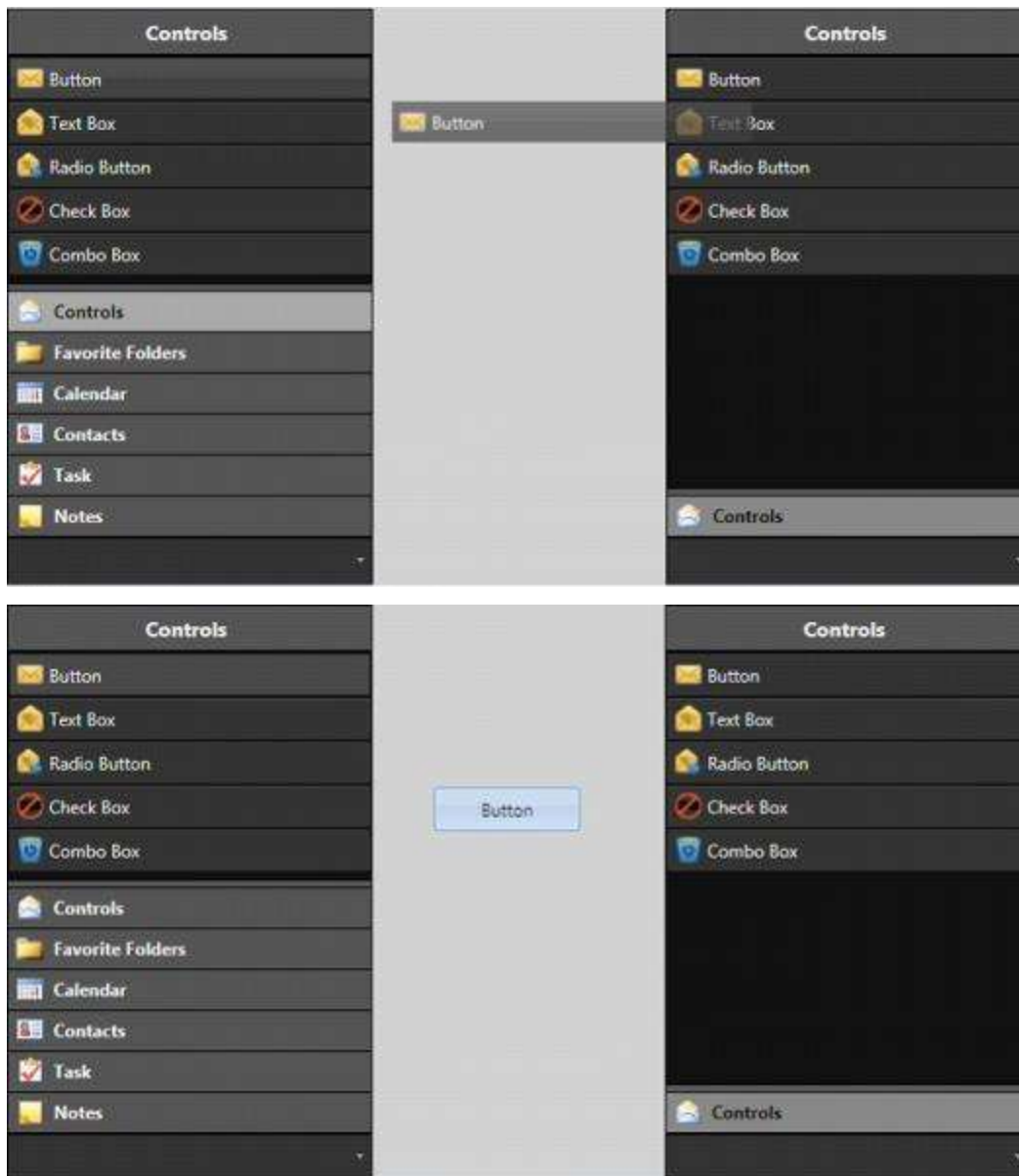


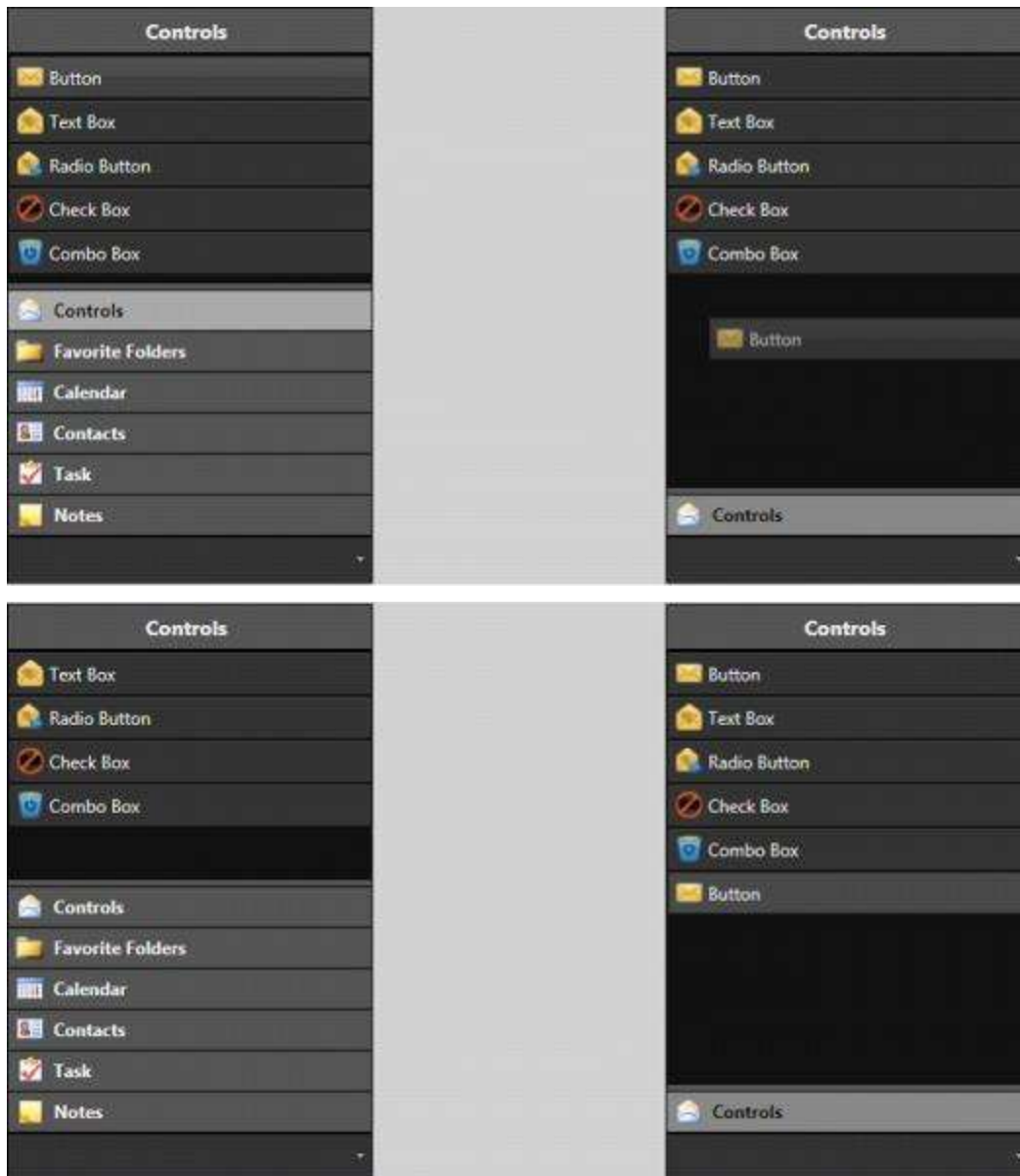
Drag-and-Drop in WPF Navigation Pane (GroupBar)

Details

Property: DragItemVisibility set to True







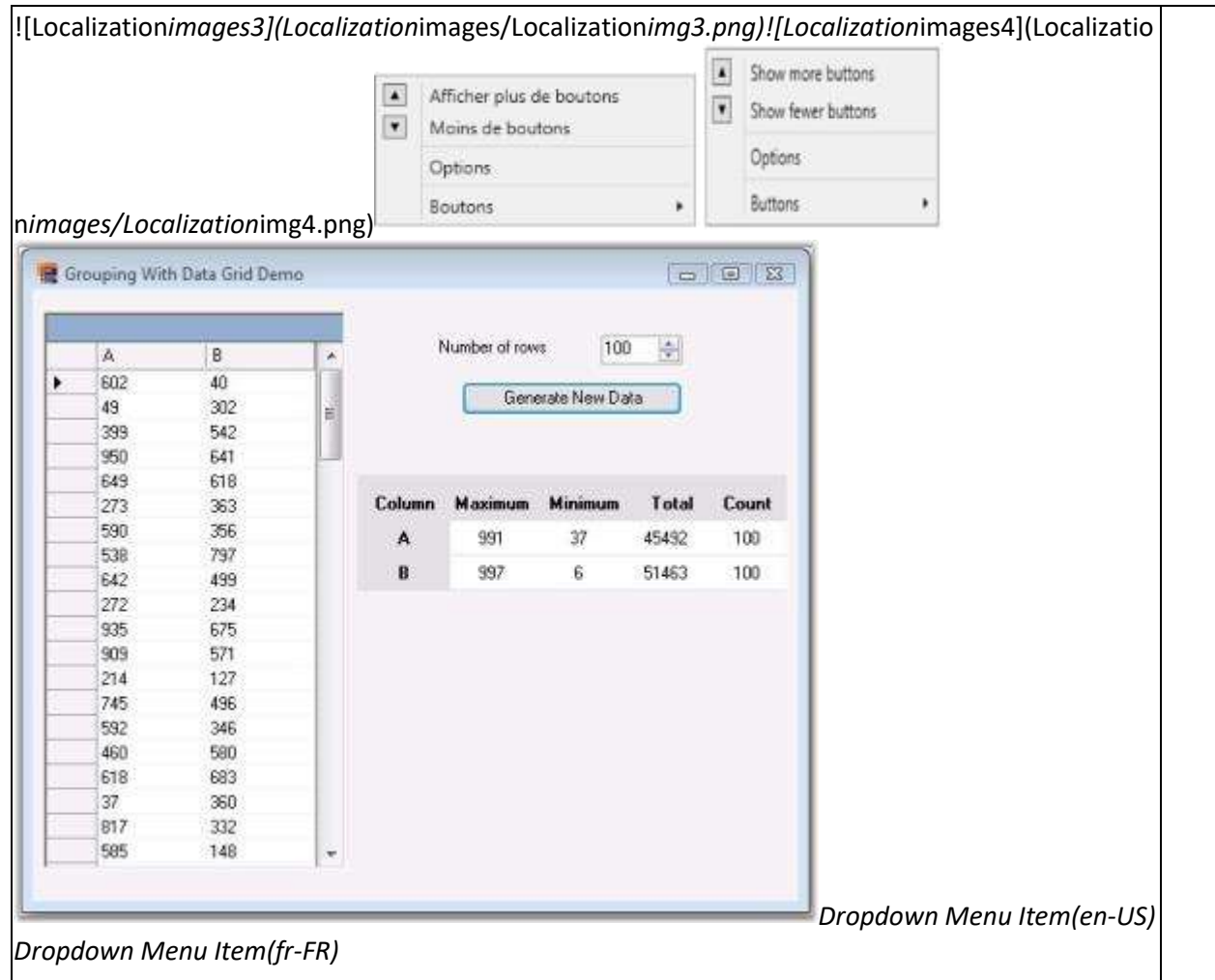


Localization in WPF Navigation Pane (GroupBar)

Localization involves the customization of the application for a specific language. It translates string values according to the specific culture. In the following table, the language is English and French

Property	Des crip tion
Options	Sets the string for the Options property Menu Item In GroupBar.
Options(en-US) Options de(fr-FR)	
GroupbarMItemCut,GroupbarMItemCopy,GroupbarMItemPaste,GroupbarMItemListView,GroupbarMItemSortAsc,GroupbarMItemSortDsc,GroupbarMItemAddTab,GroupbarMItemDeleteTab,Group	Sets the stri

<p>barMItemRenameTab,GroupbarMItemAddItem,GroupbarMItemRenameItem,GroupbarMItemDeleteItem,GroupbarMItemMoveUp,GroupbarMItemMoveDown</p>	<p>ng for the cont ext me nu ite m in Gro upB ar.</p>
<div data-bbox="194 646 734 1155"> </div> <p><i>ContextMenu(en-US) ContextMenu(en-US)</i></p>	
<p>Show fewer buttons,Show more buttonsButtons</p>	<p>Sets the stri ng for the Â Dr opd own Me nu Ite m in Gro upB ar.</p>



Grouping

WPF Grouping Overview

This section covers information on Essential Grouping, its key features, prerequisites to use the control, its compatibility with various OS and browsers, and finally the documentation details complimentary with the product. It comprises the following subsections:

Introduction to Essential Grouping

Essential Grouping is a 100% Native .NET library that provides you with support for managing and manipulating tabular information without dependencies on any particular UI component. Our Grouping Framework can be used in any .NET environment, including C#, VB.NET, and managed C++.

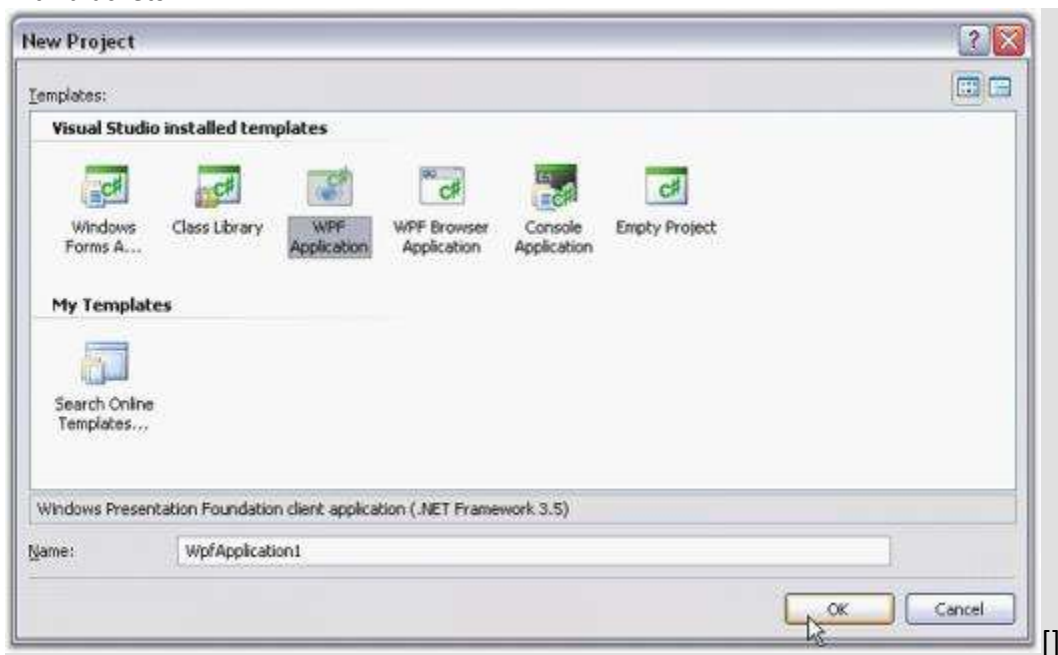
Syncfusion Essential Grouping is a data technology that allows you to easily access, manipulate, and display your data in a variety of configurations. Your data source can be any *ICollection* object whose items have public properties. You can easily sort the items on one or several of these public properties. You can display and retrieve items based on the grouping that is produced through these sorts, you can include caption information and / or summary information on these groups; you can impose filters on the items, retrieving only items that specify your filter conditions and you can also add expression properties to display calculated values depending upon other properties in the item.

![Grouping - Overview](Overviewimages/Overviewimg1.jpeg)

Key Features

Some of the key features of Essential Grouping are listed below:

- Grouping supports data presentation techniques like sorting, grouping, adding caption and summary information for the groups.
- Grouping also supports nested tables and hierarchies in the form of related tables.
- The grouping technology uses balanced binary trees as the core data structure instead of arrays. Binary trees have this advantage whereby parent branches can cache information about their children. This allows position information and summary information to be cached in parent branches facilitating quick inserts of new records honoring any sort of criteria that is applied. Inserting, removing, and moving of records takes \ only $\text{Log}_2(n)$ operations. With linear lookup structures such as an ArrayList, each of these operations would take $O(n)$ operations.
- Expressions can be any well-formed algebraic combination of property (column) names enclosed with brackets



numerical constants and literals, and the algebraic and logical operators.

- Grouping is a recursive process whereby a data source may be grouped several times. This leads to the recursive situation of groups having sub-groups and so on.

Prerequisites and Compatibility

This section covers the requirements mandatory for using Essential Grouping. It also lists operating systems and browsers compatible with the product.

Prerequisites

The prerequisites details are listed below:

Development Environments	<ul style="list-style-type: none"> • Visual Studio 2015 (Ultimate, Premium, Professional, and Express) • Visual Studio 2012 (Ultimate, Premium, Professional, and Express) • Visual Studio 2010 (Ultimate, Premium, Professional, and Express)
--------------------------	---

	<ul style="list-style-type: none"> • Visual Studio 2008 (Team System, Professional, Standard & Express) • Visual Studio 2005 (Professional, Standard & Express)
.NET Framework versions	<ul style="list-style-type: none"> • .NET 4.6 • .NET 4.5.1 • .NET 4.5 • .NET 4.0 • .NET 3.5 • .NET 2.0

Compatibility

The compatibility details are listed below:

Operating Systems	<ul style="list-style-type: none"> • Windows 8.1 (32 bit and 64 bit) • Windows 8 (32 bit and 64 bit) • Windows Server 2013 (32 bit and 64 bit) • Windows Server 2012(32bit and 64 bit) • Windows Server 2008 (32 bit and 64 bit) • Windows 7 (32 bit and 64 bit) • Windows Vista (32 bit and 64 bit) • Windows Server 2003 (32 bit and 64 bit) • Windows XP
-------------------	--

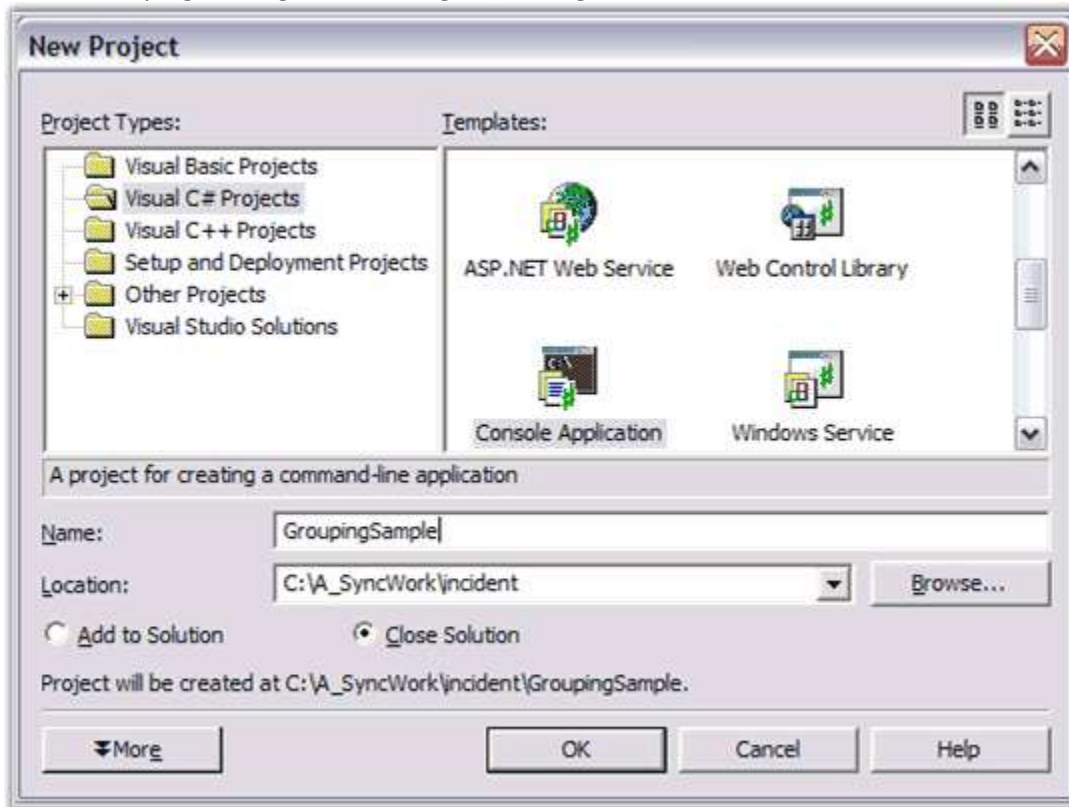
Getting Started with WPF Grouping

This section will show you how easy it is to get started using Essential Grouping. It will give you a basic introduction to the concepts you need to know before getting started with the product and some tips and ideas on how to implement Grouping into your projects to improve customization and increase efficiency. It shows how to create an IList data source and use it with Grouping. The datasource is an ArrayList of custom objects. As part of this lesson, you will see how to iterate through the data in the GroupingEngine.

Creating a WPF Application

1. Open Microsoft Visual Studio. Go to File menu and click New Project. In the New Project dialog, select WPF Application template, name the project and click OK.

![[WPF Grouping Getting-Started Image3]](Getting-Started



images/Getting-Startedimg3.jpeg)

2. A WPF application is created.
3. Now you need to deploy Essential Grouping into this WPF application. Refer WPF topic for detailed info.

Deploying Essential Grouping

This section will guide you to deploy Essential Grouping in a WPF applications.

1. In order to deploy an application that uses the Syncfusion assemblies, the referenced Syncfusion assemblies should reside in the application folder where the application executable exists, in the target machine.
2. In order to do that, go to the References folder in the Solution Explorer. Select all the Syncfusion assemblies, right-click and go to Properties. Change the Copy Local property of the Syncfusion assemblies to true and compile the project.
3. Check whether the licenses.licx file listed in the project has its Build Action property to be Embedded Resource.
4. Now you may see that the Syncfusion assemblies referenced in the project are copied to the output directory along with the application executable (bin/debug/).
5. Deploy the application executable along with the Syncfusion assemblies in that location to the target machine. Be sure that these Syncfusion assemblies reside in the same location as the application executable in the target machine.

Assemblies needed for deployment

- Syncfusion.Core.dll
- Syncfusion.Grouping.Base.dll
- Syncfusion.Grouping.Wpf.dll
- Syncfusion.Shared.Base.dll
- Syncfusion.Shared.Wpf.dll

Essential Grouping is now deployed in your WPF applications.

Data Binding in WPF Grouping

Essential Grouping lets you sort, group and summarize data. The data needs to be an IList object. For this lesson, we will use an ArrayList of custom objects which have four public properties: A, B, C, and D.

The below section illustrates how to access the data that is bound to the grouping engine.

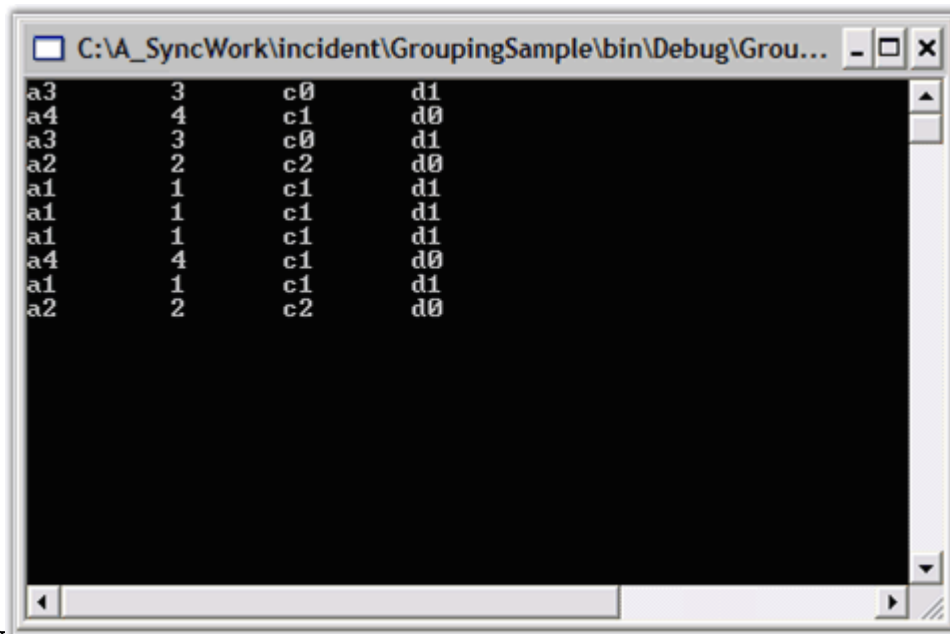
- Iterating Through the Data

This section elaborates on the procedure to setup a datasource for the Grouping engine in the below topics.

Creating an ArrayList of Objects

The first thing you need to do is to derive a class that will serve as your custom object.

1. In Visual Studio .NET, select Files -> New -> Project. Then using either C# or VB.NET, select the Console Application project template to create a new Console Application, and name it GroupingSample.



![Data-Binding
images1](Data-Bindingimages/Data-Binding_img1.png)

2. Create a custom object class and add it to the 'Class1' file generated by Step 1. Name the class 'MyObject', and let it have four string properties named A, B, C and D. Also add a constructor that

accepts an integer argument. Given below is the sample code for this. Note that you are making B a property only to hold digits. You will be using this property later to illustrate summarizing of data.

C#

```
using System;
namespace GroupingSample
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
        }
    }
    public class MyObject
    {
        private string aValue;
        private string bValue;
        private string cValue;
        private string dValue;
        public MyObject(int i)
        {
            aValue = string.Format("a{0}", i);
            // Use digit only.
            bValue = string.Format("{0}", i);
            cValue = string.Format("c{0}", i%3);
            dValue = string.Format("d{0}", i%2);
        }
        public string A
        {
            get{return aValue;}
            set{aValue = value;}
        }
        public string B
        {
            get{return bValue;}
            set{bValue = value;}
        }
        public string C
        {
            get{return cValue;}
            set{cValue = value;}
        }
        public string D
        {
            get{return dValue;}
            set{dValue = value;}
        }
        public override string ToString()
        {
            return A + "\t" + B + "\t" + C + "\t" + D;
        }
    }
}
```


VB.NET

```
Module Module1
Sub Main()
End Sub
Public Class MyObject
Private aValue As String
Private bValue As String
Private cValue As String
Private dValue As String
Public Sub New(ByVal i As Integer)
aValue = String.Format("a{0}", i)
' Use digit only.
bValue = String.Format("{0}", i)
cValue = String.Format("c{0}", i Mod 3)
dValue = String.Format("d{0}", i Mod 2)
End Sub 'New
Public Property A() As String
Get
Return aValue
End Get
Set(ByVal Value As String)
aValue = Value
End Set
End Property
Public Property B() As String
Get
Return bValue
End Get
Set(ByVal Value As String)
bValue = Value
End Set
End Property
Public Property C() As String
Get
Return cValue
End Get
Set(ByVal Value As String)
cValue = Value
End Set
End Property
Public Property D() As String
Get
Return dValue
End Get
Set(ByVal Value As String)
dValue = Value
End Set
End Property
Public Overrides Function ToString() As String
Return A + ControlChars.Tab + B + ControlChars.Tab + C + ControlChars.Tab +
D
' ToString
End Function
' MyObject
End Class
End Module
```

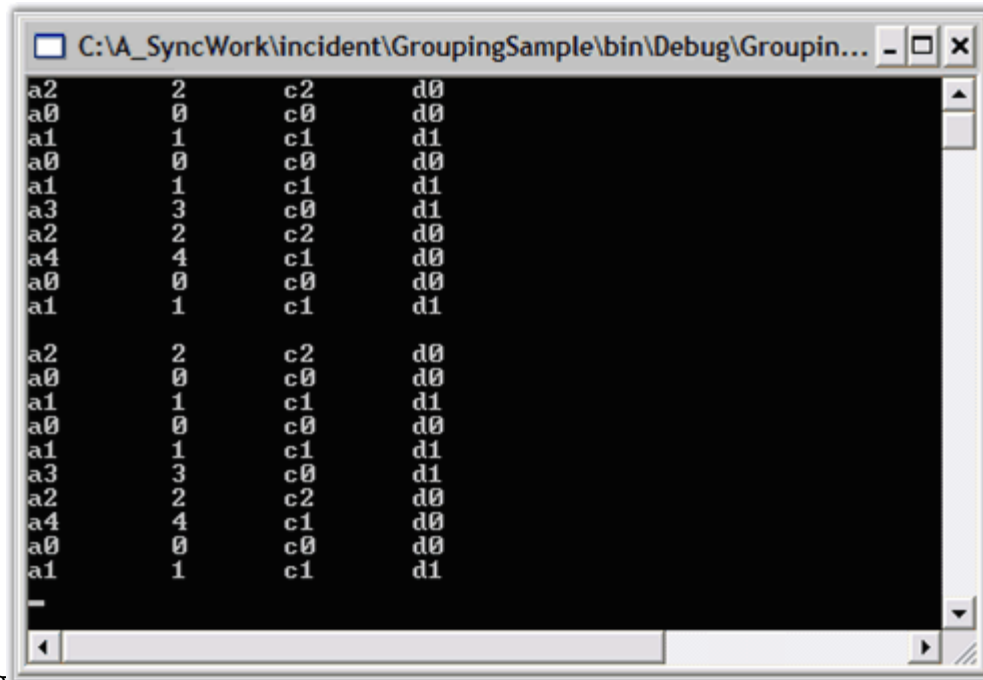
3. Add the code to the Main function as follows. This creates a random list of 'MyObject' and echoes this list to the console.

C#

```
static void Main(string[] args)
{
    // Create an arraylist of random MyObjects.
    ArrayList list = new ArrayList();
    Random r = new Random();
    for(int i = 0; i < 10; i++)
    {
        list.Add(new MyObject(r.Next(5)));
        Console.WriteLine(list[i]);
    }
    // Pause
    Console.ReadLine();
}
```

VB.NET

```
Sub Main()
    ' Create an arraylist of random MyObjects.
    Dim list As New ArrayList()
    Dim r As New Random()
    Dim i As Integer
    For i = 0 To 10
        list.Add(New MyObject(r.Next(5)))
        Console.WriteLine(list(i))
    Next
    ' Pause
    Console.ReadLine()
End Sub
```



![Data-Binding

img2](Data-Bindingimages/Data-Binding_img2.png)

4. An ArrayList of Objects is created.

Setting a Datasource In the Grouping Engine

Add the following grouping namespace for referring the assemblies deployed in the application.

Refer Deploying Essential Grouping section to know about deploying Essential Grouping.

C#

```
using Syncfusion.Grouping;
```

VB.NET

```
Imports Syncfusion.Grouping
```

Then create a Grouping.Engine object and set the ArrayList we created to be its data source.

For more details on creating array list, refer Creating an ArrayList Of Objects topic.

C#

```
// Put this code in the Main function after Console.ReadLine().
// Create a Grouping.Engine object.
Engine groupingEngine = new Engine();
// Set its data source.
groupingEngine.SetSourceList(list);
```

VB.NET

```
Imports Syncfusion.Grouping
'....
' Put this code in the Main function after Console.ReadLine().
```

```
' Create a Grouping.Engine object.
Dim groupingEngine As New Engine()
' Set its data source.
groupingEngine.SetSourceList(list)
```

ArrayList of Objects is set as the datasource for the Grouping engine.

Iterating Through the Data

Now, we have set a datasource in the Grouping Engine. Lets see how to iterate through the data.

This section will show you how to access the data through the Grouping.Engine object by using the Engine.Table.Records collection.

Add the following code to the main function. This code will iterate through the Records collection and will display the output in the Console.

Console is a text-only user interface that allows the user to interact with the operating system or text-based application by entering the text through the keyboard and reading the text output from the computer screen.

C#

```
// Access the data directly from the Engine.
foreach (Record rec in groupingEngine.Table.Records)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}
// Pause
Console.ReadLine();
```

VB.NET

```
' Access the data directly from the Engine.
Dim rec As Record
For Each rec In groupingEngine.Table.Records
Dim obj As MyObject = CType(rec.GetData(), MyObject)
If Not (obj Is Nothing) Then
    Console.WriteLine(obj)
End If
Next rec
' Pause
Console.ReadLine()
```

![Data-Bindingimg6](Data-Bindingimages/Data-Binding_img6.png)

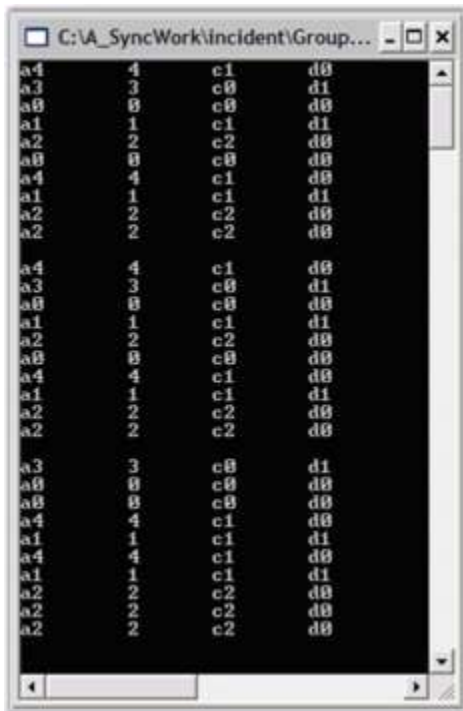
Using Grouping in WPF Grouping

The Grouping of data is one type of data analysis technique. It is natural to organize data into groups. For example, you may want to group your sales details by months and get the total sales on a month-by-month basis. The following sections elaborates on this:

Grouping a Table

In this lesson, you will start working with the `Grouping.Engine` object to see how to apply a grouping to the data as well as summarize the data. In the Data Binding section, you used the `grouping.Engine.Table.Records` collection to access the data in the `Grouping.Engine` object. The `grouping.Engine.Table` property is the property of the `Grouping.Engine` that holds the actual data needed by Essential Grouping.

You will now look at the property that holds the schema information that is associated with the data, i.e., the `grouping.Engine.TableDescriptor` property. For example, the `TableDescriptor.Columns` property holds a collection of `ColumnDescriptor` objects that define the schema information on the columns in the data.



Note: Here, the columns correspond to the public properties in our sample `MyObject` class, A, B, C, and D.

We will now continue using the same sample created in the previous section and add the corresponding code at the bottom of the `Main` method.

1. To group the '`MyObject`' `ArrayList` by a particular property, say property C, you have to add only the property name ("`C`") to the `grouping.Engine.TableDescriptor.GroupedColumns` collections. Add the following code snippet to the bottom of the `Main` method.

C#

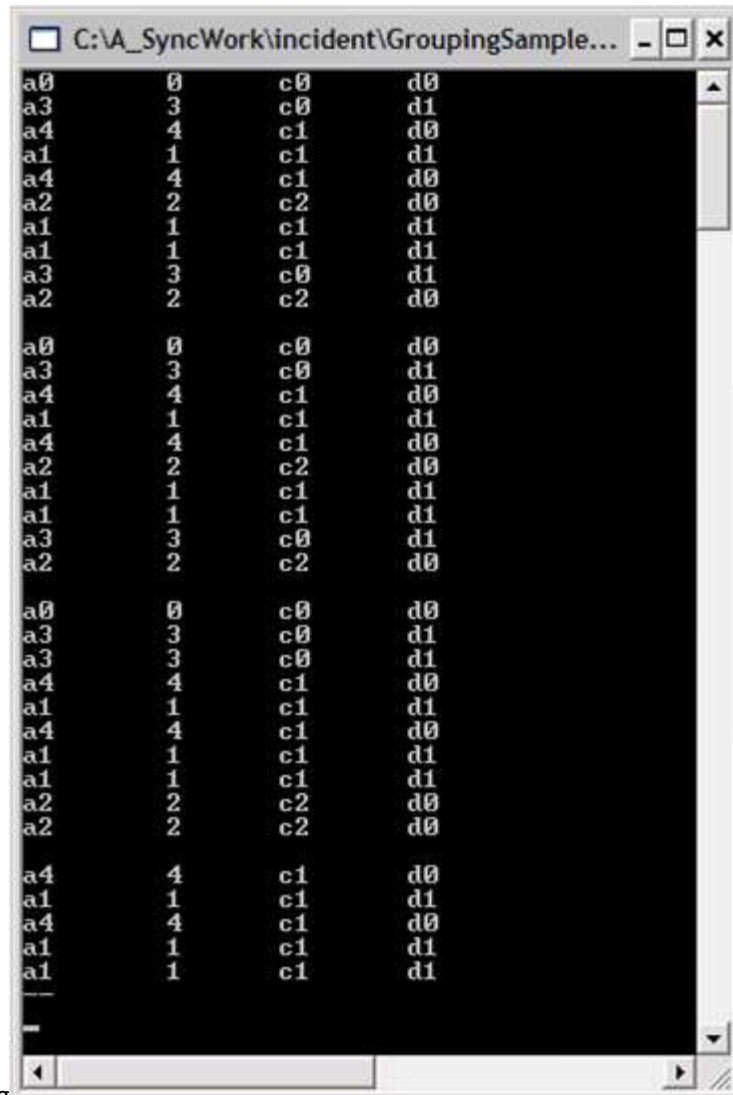
```
// Group on property C.
groupingEngine.TableDescriptor.GroupedColumns.Add("C");
// Display the records in the engine after grouping.
foreach (Record rec in groupingEngine.Table.Records)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
```

```
Console.WriteLine(obj);  
}  
}
```

VB.NET

```
' Group on property C.  
groupingEngine.TableDescriptor.GroupedColumns.Add("C")  
' Display the records in the engine after grouping.  
For Each rec In groupingEngine.Table.Records  
Dim obj As MyObject = CType(rec.GetData(), MyObject)  
If Not (obj Is Nothing) Then  
    Console.WriteLine(obj)  
End If  
Next rec
```

2. After running the code from step 1, a screen similar to the one below will be displayed. Note that the bottom list displayed is now sorted by column C. This is a one side effect of grouping by column C.



!Using-Grouping
Groupingimages/Using-Grouping_img2.png)

img2](Using-

The Grouping.TableDescriptor Class

As noted previously, the grouping.TableDescriptor is the property that maintains the schema information for the data source. Here is a table showing some collections in the TableDescriptor that you will be using as the lessons continue.

TableDescriptor Property	Description
SortedColumns	Holds sorted properties.
GroupedColumns	Holds grouped properties.
Summaries	Holds the columns that have summaries.
RecordFilters	Holds information on columns that are part of filters.
SortedColumns	Holds sorted properties.

Accessing a Particular Group

Grouping is a recursive process whereby a data source may be grouped several times. This leads to the recursive situation of groups having sub-groups and so on. In recursion, there is usually some primary node or initial starting point that you use, to work with the recursive objects. In Grouping, the initial starting point is `Engine.Table.TopLevelGroup`. This is the 'primary' Group object.

The `Grouping.Group` class has two properties that are used to recursively access nested groups and the Record objects contained in the terminal group. The properties are:

- `Group.Groups` and `Group.Records`.
- 1. `Group.Groups` is a collection of Group objects that are contained in the parent Group, and `Group.Records` is a collection of Records that are contained in the parent Group.
- 2. At most one of these collections will actually hold objects. If the Groups collections is populated, this implies that the Group has sub-groups and there are no records.
- 3. If the Records collection is populated then, it implies that this Group is a terminal group with records, but there are no sub-groups.

Your first task is to add a recursive method to either display records if the Group has records, or to recursively call itself to display any records of its child groups.

4. Add the following code below the Main method to implement a recursive method to display records in a Group.

C#

```
private static void ShowRecordsUnderGroup(Group g)
{
    if(g.Records != null && g.Records.Count > 0)
    {
        // Displaying the data for all the records in each group.
        foreach(Record rec in g.Records)
        {
            MyObject obj = rec.GetData() as MyObject;
            if(obj != null)
            {
                Console.WriteLine(obj);
            }
        }
        Console.WriteLine("--");
    }
    else if(g.Groups != null && g.Groups.Count > 0)
    {
        // Iterating through the groups.
        foreach(Group g1 in g.Groups)
        {
            // Recursive call
            ShowRecordsUnderGroup(g1);
        }
    }
}
```


VB.NET

```
Private Sub ShowRecordsUnderGroup(ByVal g As Group)
If Not (g.Records Is Nothing) And g.Records.Count > 0 Then
Dim rec As Record
' Displaying the data for all the records in each group.
For Each rec In g.Records
Dim obj As MyObject = CType(rec.GetData(), MyObject)
If Not (obj Is Nothing) Then
Console.WriteLine(obj)
End If
Next rec
Console.WriteLine("--")
Else
If Not (g.Groups Is Nothing) And g.Groups.Count > 0 Then
Dim g1 As Group
' Iterating through the groups.
For Each g1 In g.Groups
' Recursive call
ShowRecordsUnderGroup(g1)
Next g1
End If
End If
' ShowRecordsUnderGroup
End Sub
```

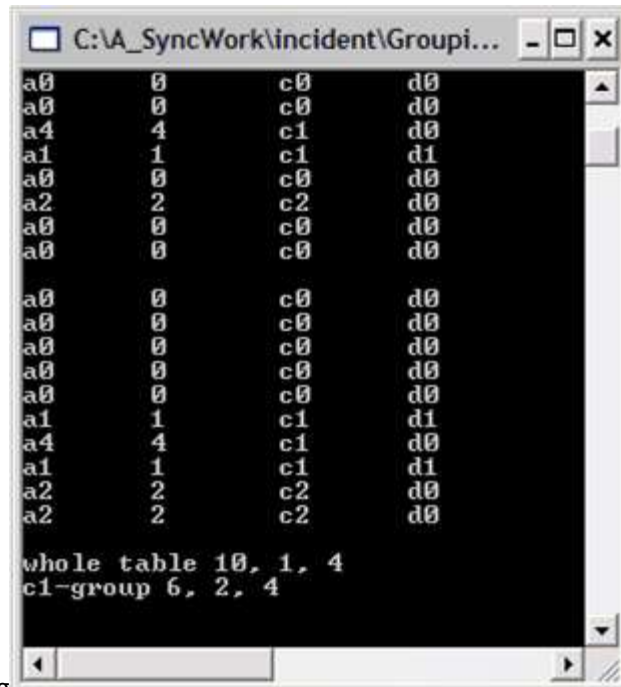
5.Once you have your ShowRecordsUnderGroup method, you only have to retrieve a particular group from the Groups collection and then call the method. So, after grouping on property C, you can view all the records whose Category is "c1" using the code like the one given below the Main method.

C#

```
// Get the Group associated with the value "c1".
Group g = groupingEngine.Table.TopLevelGroup.Groups["c1"];
ShowRecordsUnderGroup(g);
// Pause
Console.ReadLine();
```

VB.NET

```
' Get the Group associated with the value "c1".
Dim g As Group = groupingEngine.Table.TopLevelGroup.Groups("c1")
ShowRecordsUnderGroup(g)
' Pause
Console.ReadLine()
```



![(Using-Grouping-images/Using-Grouping_img3.png)](Using-Grouping-images/Using-Grouping_img3.png)

6. Similar code can be used to display all the records by passing the 'primary' group to your ShowRecordsUnderGroup method. To implement this, add the following code to the Main method.

C#

```
// Show all records under the TopLevelGroup.
ShowRecordsUnderGroup(groupingEngine.Table.TopLevelGroup);
// Pause
Console.ReadLine();
```

VB.NET

```
' Show all records under the TopLevelGroup.
ShowRecordsUnderGroup(groupingEngine.Table.TopLevelGroup)
' Pause
Console.ReadLine()
```

Adding a Summary

Essential Grouping lets you summarize your data by adding SummaryDescriptor objects to the schema information that is stored in the Engine.TableDescriptor.Summaries collection. You can have multiple summaries by adding several SummaryDescriptors.

At the bottom of the Main method, add this code to create a summary item for the Engine.

C#

```
// Create a summary that computes the Int32Aggregate calculations on
// property B.
SummaryDescriptor sdBInt32Agg = new SummaryDescriptor("BInt32Agg", "B",
SummaryType.Int32Aggregate);
// Add this summary to the Summaries collection.
```

```
groupingEngine.TableDescriptor.Summaries.Add(sdBInt32Agg);
```

VB.NET

```
' Create a summary that computes the Int32Aggregate calculations on property
B.
Dim sdBInt32Agg As New SummaryDescriptor("BInt32Agg", "B",
SummaryType.Int32Aggregate)
' Add this summary to the Summaries collection.
groupingEngine.TableDescriptor.Summaries.Add(sdBInt32Agg)
```

There are several overloads of the constructor for SummaryDescriptor. Here, we are using the overload that accepts a SummaryType enum as the third argument. This SummaryType will allow you to pick out some predefined calculations such as the Int32Aggregate functions like Max, Min, Sum, and Average. There are enums that specify double, boolean, and other aggregate types. Here, we choose Int32 as that is the type of value you will see in the B property in the data.

Retrieving Summary Values for a Particular Group

After adding the SummaryDescriptor, the GroupingEngine will maintain these summary items in a group by group basis. You can access these summary values for any group. Here we will get the values for the TopLevelGroup and for our c1 group to illustrate how this is done.

To obtain a particular group's summary values, you need to get the group and access the summary through its GetSummary method. Once you have the summary, you can cast it to its Int32AggregateSummary type.

The following code snippet illustrates this.

C#

```
// To simplify notation, add this statement at the top of the file.
using ISummary = Syncfusion.Collections.BinaryTree.ITreeTableSummary;
// At the bottom of the Main method, add these lines.
// Now go through the group to get the Summary value for the group.
ISummary groupSummary =
groupingEngine.Table.TopLevelGroup.GetSummary("BInt32Agg");
Int32AggregateSummary int32Summary = (Int32AggregateSummary) groupSummary;
Console.WriteLine("whole table {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum);
// Value for "c1" group.
groupSummary =
groupingEngine.Table.TopLevelGroup.Groups["c1"].GetSummary("BInt32Agg");
int32Summary = (Int32AggregateSummary) groupSummary;
Console.WriteLine("c1-group {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum);
// Pause
Console.ReadLine();
```

VB.NET

```
' At the bottom of the Main method, add these lines.
' Now go through the group to get the Summary value for the group.
Dim groupSummary As Syncfusion.Collections.BinaryTree.ITreeTable =
groupingEngine.Table.TopLevelGroup.GetSummary("BInt32Agg")
```

```

Dim int32Summary As Int32AggregateSummary = CType(groupSummary,
Int32AggregateSummary)
Console.WriteLine("whole table {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum)
' Value for "c1" group.
groupSummary =
groupingEngine.Table.TopLevelGroup.Groups("c1").GetSummary("BInt32Agg")
int32Summary = CType(groupSummary, Int32AggregateSummary)
Console.WriteLine("c1-group {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum)
' Pause
Console.ReadLine()

```

![[Using-Grouping

Data Manipulation in WPF Grouping

In addition to grouping data, you may want to filter it for some special criteria. For example, you may want to see the total monthly sales due to orders under some value. Essential Grouping gives you the flexibility to add calculated values to the data, and then use these values to produce other information like total monthly sales due for respective order etc.

The following data manipulation techniques are available in Essential Grouping:

Filters

Another collection that is part of the schema information in the Engine.TableDescriptor is the RecordFilters collection. This collection lets you filter the data to see only the items that are in your data list and that satisfy the criteria that is specified. You can express the criteria as a logical expression using the property names, algebraic, and logical operators. You can also use LIKE, MATCH, and IN operators.

1. In the Console Application used in lessons 1 and 2, comment out all the code that is in the Main method and add the following code to create a data object and set it into the Grouping Engine.

C#

```

static void Main(string[] args)
{
    // Create an arraylist of random MyObjects.
    ArrayList list = new ArrayList();
    Random r = new Random();
    for(int i = 0; i < 10; i++)
    {
        list.Add(new MyObject(r.Next(5)));
    }
    // Create a Grouping.Engine object.
    Engine groupingEngine = new Engine();
    // Set its datasource.
    groupingEngine.SetSourceList(list);
}

```

VB.NET

```

Sub Main()
    ' Create an arraylist of random MyObjects.
    Dim list As New ArrayList()
    Dim r As New Random()

```

```

Dim i As Integer
For i = 0 To 10
list.Add(New MyObject(r.Next(5)))
Next
' Create a Grouping.Engine object.
Dim groupingEngine As New Engine()
' Set its data source.
groupingEngine.SetSourceList(list)
End Sub

```

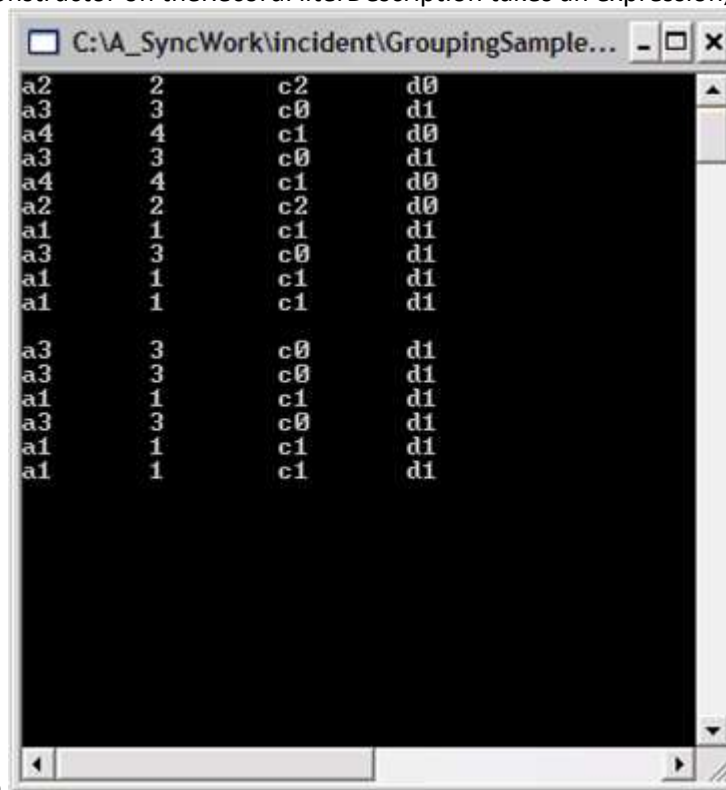
2.You are now ready to apply a filter to the data. Say for example you want to see only those items whose property D has the value d1. You must observe that D is a string that has non-numeric values. So, in this case you will need to use one of the string comparison operators (LIKE or MATCH) in your filter condition. 3.To add a filter condition, you will need to add a RecordFilterDescriptor to the Engine.TableDescriptor.RecordFilters collection.

Do the following steps:

1.List the data before the filter. 2.Apply the filter by creating a RecordFilterDescriptor. 3.Add it to the RecordFilters collection. 4.List the filtered data.

Note: To list the data, instead of accessing the Table.Records collections, you were using the Table.FilteredRecords collections. The FilteredRecords collection only includes the records that satisfy all filters in the RecordFilters collection. Add this code at the end of the Main method.

5.Note that the constructor on theRecordFilterDescription takes an expression, "[D] LIKE 'd1'". This



expression will be
items in the list where property D has the value d1.

True only for those

C#

```

// Display the data before filtering.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}
// Pause
Console.ReadLine();
// Filter on [D] = d1
RecordFilterDescriptor rfd = new RecordFilterDescriptor("[D] LIKE 'd1'");
groupingEngine.TableDescriptor.RecordFilters.Add(rfd);
// Display the data after filtering.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}
// Pause
Console.ReadLine();

```

VB.NET

```

' Display the data before filtering.
Dim rec As Record
For Each rec In groupingEngine.Table.FilteredRecords
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec
' Pause
Console.ReadLine()
' Filter on [D] = d1
Dim rfd As New RecordFilterDescriptor("[D] LIKE 'd1'")
groupingEngine.TableDescriptor.RecordFilters.Add(rfd)
' Display the data after filtering.
For Each rec In groupingEngine.Table.FilteredRecords
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec
' Pause
Console.ReadLine()

```

a2	2	c2	d0
a3	3	c0	d1
a4	4	c1	d0
a3	3	c0	d1
a4	4	c1	d0
a2	2	c2	d0
a1	1	c1	d1
a3	3	c0	d1
a1	1	c1	d1
a1	1	c1	d1
a3	3	c0	d1
a3	3	c0	d1
a1	1	c1	d1
a3	3	c0	d1
a1	1	c1	d1
a1	1	c1	d1
a2	2	c2	d0
a3	3	c0	d1
a3	3	c0	d1
a2	2	c2	d0
a1	1	c1	d1
a3	3	c0	d1
a1	1	c1	d1
a1	1	c1	d1

![Data-Manipulation

img2](Data-

Manipulationimages/Data-Manipulation_img2.png)

6.You can apply more complex filters. Here is the code that will remove any existing filters and filter the property D being d1 or property b equal 2. Note here that since you expect property B to display only numeric data you must use the = operator in the comparison.

C#

```
groupingEngine.TableDescriptor.RecordFilters.Clear();
rfd = new RecordFilterDescriptor("[D] LIKE 'd1' OR [B] = 2");
groupingEngine.TableDescriptor.RecordFilters.Add(rfd);
// Access the data directly from the Engine.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}
// Pause
Console.ReadLine();
```

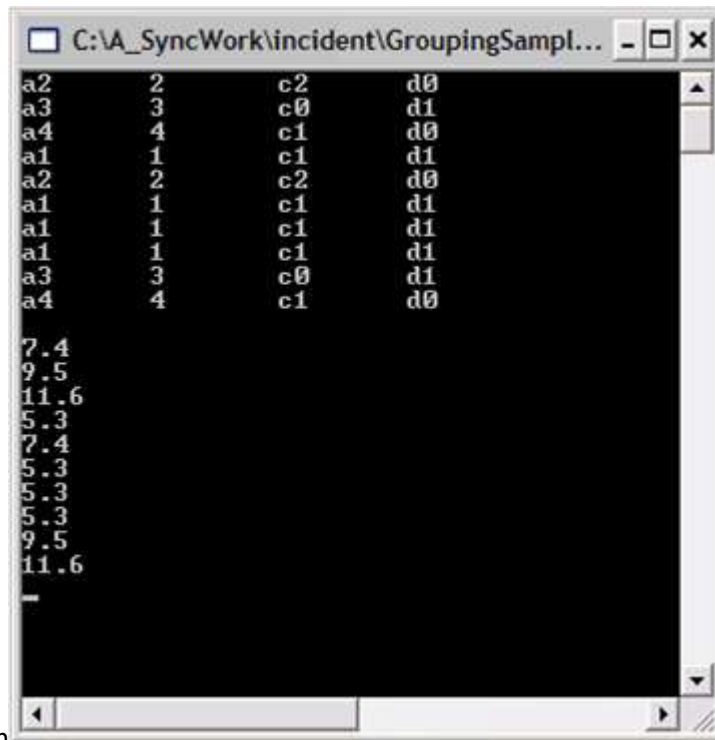
VB.NET

```
groupingEngine.TableDescriptor.RecordFilters.Clear()
rfd = New RecordFilterDescriptor("[D] LIKE 'd1' OR [B] = 2")
groupingEngine.TableDescriptor.RecordFilters.Add(rfd)
' Access the data directly from the Engine.
For Each rec In groupingEngine.Table.FilteredRecords
```

```

Dim obj As MyObject = CType(rec.GetData(), MyObject)
If Not (obj Is Nothing) Then
    Console.WriteLine(obj)
End If
Next rec
' Pause
Console.ReadLine()

```



![[Data-Manipulationimages/Data-Manipulation_img3.png]](Data-

Filtering is applied to the data displayed in the console.

Expressions

You can add new properties to your data object that are algebraic expressions involving other properties of the object.

To add an expression, you need to create an ExpressionFieldDescriptor and add it to the Engine.TableDescriptor.Expression.Fields collection. Here we illustrate this process by adding an expression that computes 2.1 times the value of property B plus 3.2.

1. In the Console Application, comment out all the code that is in the Main method and add this code to create a data object and set it into the GroupingEngine.

C#

```

// Create an arraylist of random MyObjects.
ArrayList list = new ArrayList();
Random r = new Random();
for(int i = 0; i < 10; i++)
{
    list.Add(new MyObject(r.Next(5)));
}

```



```
// Create a Grouping.Engine object.
Engine groupingEngine = new Engine();
// Set its datasource.
groupingEngine.SetSourceList(list);
```

VB.NET

```
' Create an arraylist of random MyObjects.
Dim list As New ArrayList()
Dim r As New Random()
Dim i As Integer
For i = 0 To 10
list.Add(New MyObject(r.Next(5)))
Next
' Create a Grouping.Engine object.
Dim groupingEngine As New Engine()
' Set its datasource.
groupingEngine.SetSourceList(list)
```

2. Now you must add code to list the data, add an expression property and then display the value of the expression. To retrieve the value, you must use the Record.GetValue method by passing it as the name of the expression that you had assigned when it was created.

C#

```
// Display the data before filtering.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
MyObject obj = rec.GetData() as MyObject;
if (obj != null)
{
Console.WriteLine(obj);
}
}
// Pause
Console.ReadLine();
// Add an expression property.
ExpressionFieldDescriptor efd = new ExpressionFieldDescriptor("MultipleOfB",
"2.1 * [B] + 3.2");
groupingEngine.TableDescriptor.ExpressionFields.Add(efd);
// Display the data after adding the field.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
Console.WriteLine(rec.GetValue("MultipleOfB"));
}
// Pause
Console.ReadLine();
```

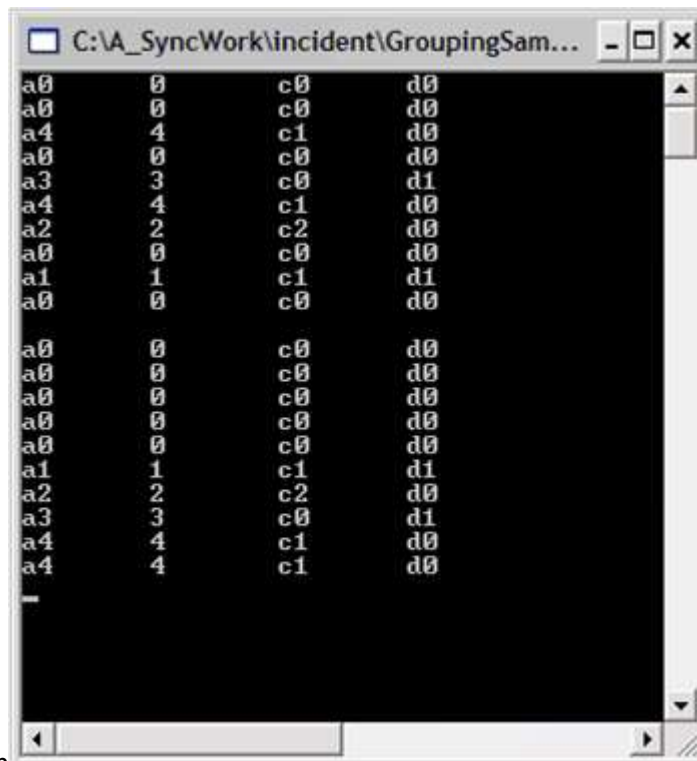
VB.NET

```
' Display the data before filtering.
Dim rec As Record
For Each rec In groupingEngine.Table.FilteredRecords
Dim obj As MyObject = CType(rec.GetData(), MyObject)
If Not (obj Is Nothing) Then
```

```

Console.WriteLine(obj)
End If
Next rec
' Pause
Console.ReadLine()
' Add an expression property.
Dim efd As New ExpressionFieldDescriptor("MultipleOfB", "2.1 * [B] + 3.2")
groupingEngine.TableDescriptor.ExpressionFields.Add(efd)
' Display the data after adding the field.
For Each rec In groupingEngine.Table.FilteredRecords
Console.WriteLine(rec.GetValue("MultipleOfB"))
Next rec

```



![[Data-Manipulation
Data-Manipulationimages/Data-Manipulation_img4.png)](Data-

Sorting

To sort your data, you must add the name of the property that you want to sort to the `Engine.TableDescriptor.SortedColumns` collection.

In the Console Application, comment out all the code that is in the `Main` method and add this code to create a data object, set it into the `GroupingEngine`, display the data, sort it by property `A` and re-display the sorted data.

C#

```

// Create an arraylist of random MyObjects.
ArrayList list = new ArrayList();
Random r = new Random();
for(int i = 0; i < 10; i++)
{
list.Add(new MyObject(r.Next(5)));
}

```

```

}
// Create a Grouping.Engine object.
Engine groupingEngine = new Engine();
// Set its datasource.
groupingEngine.SetSourceList(list);
// Display the data before sorting.
foreach (Record rec in groupingEngine.Table.Records)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}
// Pause
Console.ReadLine();
// Sort column A.
groupingEngine.TableDescriptor.SortedColumns.Add("A");
// Display the data after sorting.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}
// Pause
Console.ReadLine();

```

VB.NET

```

' Create an arraylist of random MyObjects.
Dim list As New ArrayList()
Dim r As New Random()
Dim i As Integer
For i = 0 To 9
    list.Add(New MyObject(r.Next(5)))
Next i
' Create a Grouping.Engine object.
Dim groupingEngine As New Engine()
' Set its datasource.
groupingEngine.SetSourceList(list)
' Display the data before sorting.
Dim rec As Record
For Each rec In groupingEngine.Table.Records
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec
' Pause
Console.ReadLine()
' Sort column A.
groupingEngine.TableDescriptor.SortedColumns.Add("A")
' Display the data after sorting.

```

```

For Each rec In groupingEngine.Table.FilteredRecords
Dim obj As MyObject = CType(rec.GetData(), MyObject)
If Not (obj Is Nothing) Then
    Console.WriteLine(obj)
End If
Next rec
' Pause
Console.ReadLine()

```

![Data-Manipulationimg5](Data-Manipulationimages/Data-Manipulation_img5.png)

Custom Sorting

We used a simplest overload in the previous section for sorting the data. The TableDescriptor.SortedColumns.Add method has three overloads as shown below:

- Add(string propertyName)
- Add(string propertyName, ListSortDirection direction)
- Add(SortColumnDescriptor sdc)

The following code snippets illustrate the syntax for these overloads:

C#

```

public int Add(string propertyName);
public int Add(string propertyName, ListSortDirection direction);
public int Add(SortColumnDescriptor sdc);

```

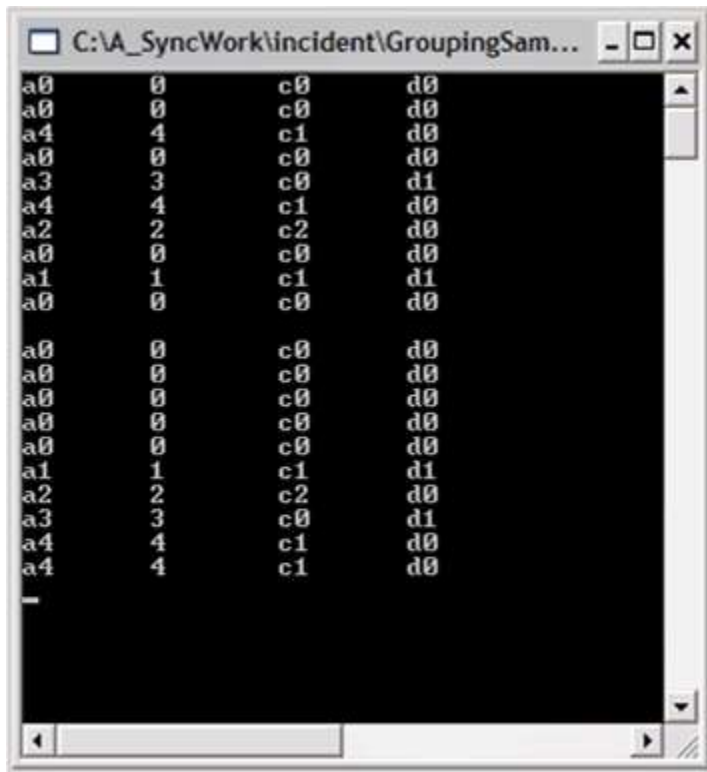
VB.NET

```

Overloads Public Function Add(propertyName As String) As Integer
Overloads Public Function Add(propertyName As String, dir As
ListSortDirection) As Integer
Overloads Public Function Add(sdc As SortColumnDescriptor) As Integer

```

The second overload can be used to specify the sort direction. The SortColumnDescriptor.Comparer is an IComparer property that allows you to specify a custom comparer object.



Note: We are going to use the third

function in this section to perform custom sorting.

The following steps illustrate how to do custom sorting using the `IComparer` property:

1. Here, you must create a class that implements `IComparer` which can accept value such as `ax`, where `x` is a digit which is used to do the comparison. This leads to numerical sorting, ignoring the leading character.
2. Add this code immediately following the end of the `Class1` code.

C#

```
public class AComparer : IComparer
{
    // Implementing IComparer to define the object comparisons.
    public int Compare(object x, object y)
    {
        if (x == null && y == null)
            return 0;
        else if (x == null)
            return -1;
        else if (y == null)
            return 1;
        else
        {
            int sx = 0;
            int sy = 0;
            try
            {
                // Ignoring the leading character to have numerical sorting.
                sx = int.Parse(x.ToString().Substring(1));
            }
        }
    }
}
```

```

sy = int.Parse(y.ToString().Substring(1));
return sy - sx;
}
catch
{
throw new ArgumentException("A must be in the form 'annnn'");
}
}
}
}

```

VB.NET

```

Public Class AComparer
Implements IComparer
' Implementing IComparer to define the object comparisons.
Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer _
Implements IComparer.Compare
If x Is Nothing And y Is Nothing Then
Return 0
Else
If x Is Nothing Then
Return -1
Else
If y Is Nothing Then
Return 1
Else
Dim sx As Integer = 0
Dim sy As Integer = 0
Try
' Ignoring the leading character to have numerical sorting.
sx = Integer.Parse(x.ToString().Substring(1))
sy = Integer.Parse(y.ToString().Substring(1))
Return sy - sx
Catch
End Try
End If
End If
End If
End Function
End Class

```

3.Add this code to the Main method to use this custom comparer to sort column A.

C#

```

// Create an arraylist of random MyObjects.
ArrayList list = new ArrayList();
Random r = new Random();
for(int i = 0; i < 10; i++)
{
list.Add(new MyObject(r.Next(20)));
}
// Create a Grouping.Engine object.
Engine groupingEngine = new Engine();
// Set its datasource.

```

```

groupingEngine.SetSourceList(list);
// Display the data before sorting.
foreach (Record rec in groupingEngine.Table.Records)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}
// Pause
Console.ReadLine();
// Custom sort column A.
// Get an IComparer object to handle the custom sorting.
AComparer comparer = new AComparer();
groupingEngine.TableDescriptor.SortedColumns.Add("A");
groupingEngine.TableDescriptor.SortedColumns["A"].Comparer = comparer;
// Display the data before sorting.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}
// Pause
Console.ReadLine();

```

VB.NET

```

' Create an arraylist of random MyObjects.
Dim list As New ArrayList()
Dim r As New Random()
Dim i As Integer
For i = 0 To 9
    list.Add(New MyObject(r.Next(20)))
Next i
' Create a Grouping.Engine object.
Dim groupingEngine As New Engine()
' Set its datasource.
groupingEngine.SetSourceList(list)
' Display the data before sorting.
Dim rec As Record
For Each rec In groupingEngine.Table.Records
    Dim obj As MyObject = rec.GetData()
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec
' Pause
Console.ReadLine()
' Custom sort column A.
' Get an IComparer object to handle the custom sorting.
Dim comparer As New AComparer()

```

```

groupingEngine.TableDescriptor.SortedColumns.Add("A")
groupingEngine.TableDescriptor.SortedColumns("A").Comparer = comparer
' Display the data before sorting.
Dim rec As Record
For Each rec In groupingEngine.Table.FilteredRecords
Dim obj As MyObject = CType(rec.GetData(), MyObject)
If Not (obj Is Nothing) Then
Console.WriteLine(obj)
End If
Next rec
' Pause
Console.ReadLine()

```

![Data-Manipulationimg7](Data-Manipulationimages/Data-Manipulation_img7.png)

Algebra Supported In Expressions Filters in WPF Grouping

Expressions can be any well-formed algebraic combination of the following:

- Property (column) names enclosed with square brackets []
- Numerical constants and literals
- Algebraic and logical operators

The computations are performed as listed below, with level one operations done first.

- *, /: multiplication, division
- +, -: addition, subtraction
- <, >, =, <=, >=, <>: less than, greater than, equal, less than or equal, greater than or equal, not equal
- match, like, in, between
- or, and, or

Note:

1. Alpha constants used with match and like should be enclosed in apostrophes (').
2. Logical operators return "1", if the logical expression is True and return "0", if the logical expression is False.

Given below is some more information on special logical operators:

- Match>Returns 1 if there is any occurrence of the right-hand argument in the left-hand argument.

Example

[CompanyName] match 'RTR' returns 0 for any record whose CompanyName field does not contain RTR anywhere in the string.

- Like-Checks if the field starts exactly as specified in the right-hand argument.

Example

[CompanyName] like 'RTR' returns 1 for any record whose CompanyName field is exactly 'RTR'. You can use an asterisk as a wildcard. [CompanyName] like 'RTR' returns 1 for any record whose CompanyName field starts with 'RTR'. [CompanyName] like 'RTR' returns 1 for any record whose CompanyName field ends with 'RTR'.

- In-Checks if the field value is any of the values listed in the right-hand operand. The collection of items used as the right-hand should be separated by commas and enclosed within braces({ }).

Example

[code] in {1,10,21}, returns 1 for any record whose code field contains 1 or 10 or 21. [CompanyName] in {RTR,MAS} returns 1 for any record whose CompanyName field is RTR or MAS.

Note that spaces that are significant in the list, i.e. {RTR,MAS} is not the same as {RTR, MAS}.

- Between-Checks if a date field value between the two values is listed in the right-hand operand.

Example

[date] between {2/25/2004, 3/2/2004} returns 1 for any record whose date field is greater than or equal to 2/25/2004 and less than 3/2/2004. To represent the current date, use the token TODAY. To represent DateTime.MinValue, leave the first argument empty. To represent DateTime.MaxValue, leave the second argument empty.

Custom Functions

Essential Grouping lets you add custom functions to your code that can then be used in expressions.

Following are the limitations on the use of custom functions:

- You cannot use custom functions in algebraic expressions.
- They must be used stand-alone in a simple expression that consists of only the function name and its argument list.
- The argument list can be any number of arguments separated by a delimiter.

HierarchyNavigator

WPF Breadcrumb (HierarchyNavigator) Overview

The HierarchyNavigator control in WPF provides a bread-crumb interface, similar to the Windows Explorer address bar in Windows 7, which enables hierarchical navigation.

Features

The features of the HierarchyNavigator control are the following:

- Enables data binding with Business Objects, XML data, and WCF services.
- Enables command binding with MVVM support.
- Helps customize each template part by using Microsoft Expression Blend.
- Provides keyboard navigation support.
- Enables edit mode (with AutoComplete).
- Helps display or remove the progress bar.
- Helps customize templates with HierarchicalDataTemplate.
- Supports multiple skins.

- Displays history.
- Displays ToolTips.

Getting Started with WPF Breadcrumb (HierarchyNavigator)

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 packages in a WPF application in the following link:

[How to install nuget packages](#)

Create a simple application with HierarchyNavigator

You can create a WPF application with the HierarchyNavigator control using the following steps:

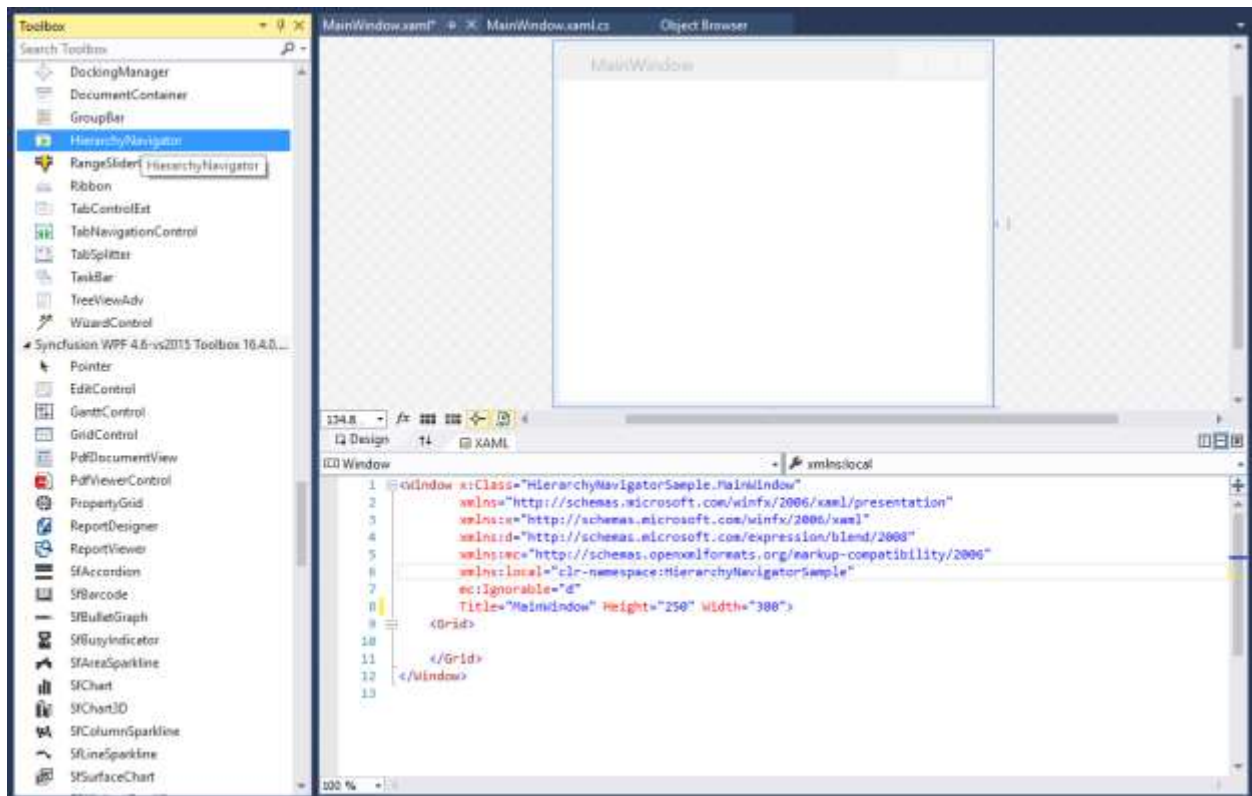
Create a project

Create a new WPF project in Visual Studio to display the HierarchyNavigator with functionalities.

Add control through designer

The HierarchyNavigator control can be added to an application by dragging it from the toolbox and dropping it into a designer view. The following required assembly references will be added automatically to the project:

- 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* *Syncfusion.Shared.WPF*
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the HierarchyNavigator control the 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="HierarchicalNavigatorSample.MainWindow"
Title="HierarchicalNavigator Sample" Height="350" Width="525">
<Grid>
<!--Adding HierarchicalNavigator control -->
<syncfusion:HierarchicalNavigator x:Name="hierarchicalNavigator" Width="100"
Height="100" VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
</Window>
```

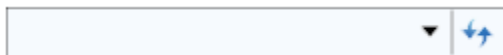
Add control manually in C\#

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 HierarchyNavigator namespace **using Syncfusion.Windows.Tools.Controls;**
3. Create a HierarchyNavigator instance, and add it to the window.

C#

```
using Syncfusion.Windows.Tools;
namespace HierarchicalNavigatorSample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Creating an instance of HierarchicalNavigator control
            HierarchicalNavigator hierarchicalNavigator = new HierarchicalNavigator();
            //Adding HierarchicalNavigator as window content
            this.Content = hierarchicalNavigator;
        }
    }
}
```



Add items using HierarchyNavigatorItem

You can add the items inside the HierarchyNavigator control using [HierarchyNavigatorItem](#).

XML

```
<syncfusion:HierarchyNavigator x:Name="hierarchyNavigatorcontrol1"
VerticalAlignment="Top" Height="30" Width="600">
<syncfusion:HierarchyNavigator.Items>
<syncfusion:HierarchyNavigatorItem Content="Syncfusion">
<syncfusion:HierarchyNavigatorItem.Items>
<syncfusion:HierarchyNavigatorItem Content="User Interface"/>
<syncfusion:HierarchyNavigatorItem Content="Silverlight">
<syncfusion:HierarchyNavigatorItem.Items>
<syncfusion:HierarchyNavigatorItem Content="Tools"/>
</syncfusion:HierarchyNavigatorItem.Items>
</syncfusion:HierarchyNavigatorItem>
</syncfusion:HierarchyNavigatorItem.Items>
</syncfusion:HierarchyNavigatorItem>
</syncfusion:HierarchyNavigator.Items>
</syncfusion:HierarchyNavigator>
```

C#

```
HierarchyNavigator hierarchyNavigator1 = new HierarchyNavigator() { Height =
30 };
HierarchyNavigatorItem hierarchyNavigatorItem1 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem1.Content = "Syncfusion";
HierarchyNavigatorItem hierarchyNavigatorItem11 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem11.Content = "User Interface";
HierarchyNavigatorItem hierarchyNavigatorItem111 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem111.Content = "Silverlight";
HierarchyNavigatorItem hierarchyNavigatorItem112 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem112.Content = "WPF";
HierarchyNavigatorItem hierarchyNavigatorItem113 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem113.Content = "ASP .Net";
HierarchyNavigatorItem hierarchyNavigatorItem114 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem114.Content = "MVC";
hierarchyNavigatorItem11.Items.Add(hierarchyNavigatorItem111);
hierarchyNavigatorItem11.Items.Add(hierarchyNavigatorItem112);
hierarchyNavigatorItem11.Items.Add(hierarchyNavigatorItem113);
hierarchyNavigatorItem11.Items.Add(hierarchyNavigatorItem114);
hierarchyNavigatorItem1.Items.Add(hierarchyNavigatorItem11);
hierarchyNavigator1.Items.Add(hierarchyNavigatorItem1);
this.Content = hierarchyNavigator1;
```

Bind data

HierarchyNavigator accepts any business object collection to be bound to its [ItemsSource](#) property. Refer [Data binding](#) section for more details.

The steps to bind to a Business Object collection are as follows:

1. Create a class named HierarchyItem.

C#

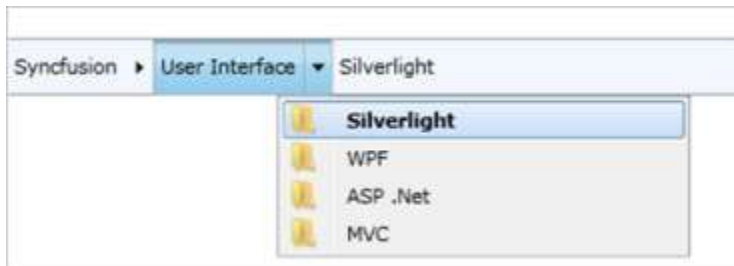
```
public class HierarchyItem
{
    public string ContentString { get; set; }
    public HierarchyItem(string content, params HierarchyItem[] myItems)
    {
        this.ContentString = content;
        itemsObservableCollection = new ObservableCollection<HierarchyItem>();
        foreach (var item in myItems)
        {
            itemsObservableCollection.Add(item);
        }
        HierarchyItems = itemsObservableCollection;
    }
    private ObservableCollection<HierarchyItem> itemsObservableCollection;
    public ObservableCollection<HierarchyItem> HierarchyItems
    {
        get { return itemsObservableCollection; }
        set
        {
            if (itemsObservableCollection != value)
            {
                itemsObservableCollection = value;
            }
        }
    }
}
```

2. Create a collection for ItemsSource to bind HierarchyItem.**C#**

```
public class HierarchicalItemsSource : ObservableCollection<HierarchyItem>
{
    public HierarchicalItemsSource()
    {
        this.Add(new HierarchyItem("Syncfusion",
            new HierarchyItem("User Interface",
                new HierarchyItem("Silverlight"),
                new HierarchyItem("WPF"),
                new HierarchyItem("ASP .Net"),
                new HierarchyItem("MVC")),
            new HierarchyItem("Reporting Edition",
                new HierarchyItem("IO"),
                new HierarchyItem("PDF generator"),
                new HierarchyItem("WPF"))));
    }
}
```

3. In XAML, bind the collections to the ItemsSource property of the HierarchyNavigator control**XML**

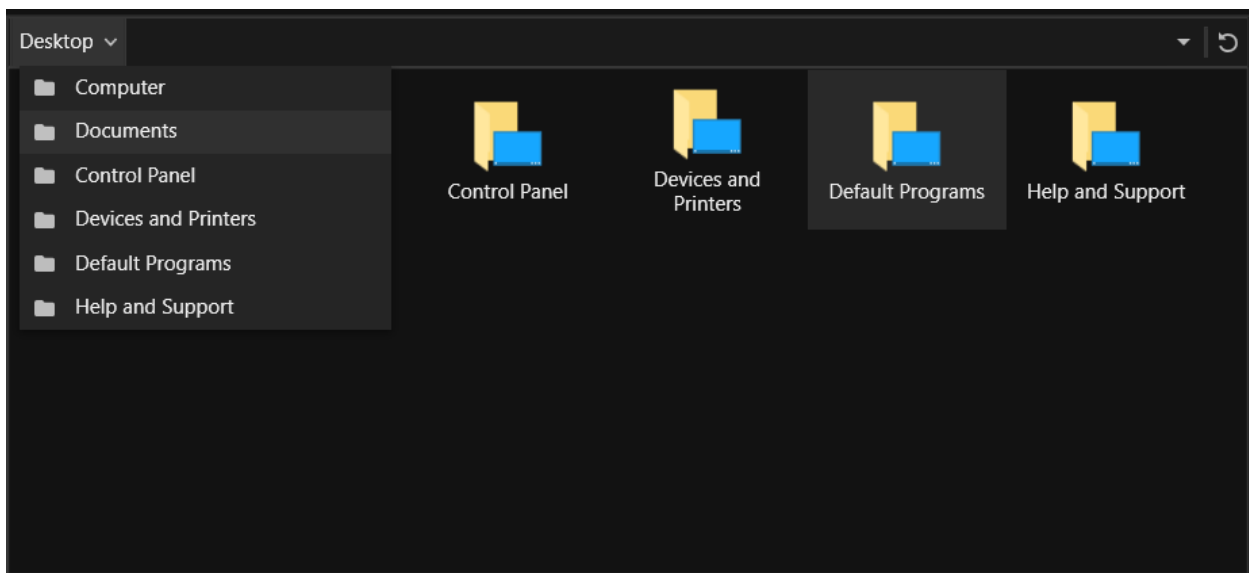
```
<syncfusion:HierarchyNavigator Name="hierarchyNavigator2" Width="250"
Height="25">
<syncfusion:HierarchyNavigator.ItemsSource>
<local:HierarchicalItemsSource />
</syncfusion:HierarchyNavigator.ItemsSource>
<syncfusion:HierarchyNavigator.ItemTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding HierarchyItems}">
<TextBlock Text="{Binding ContentString}" Margin="2,0" />
</HierarchicalDataTemplate>
</syncfusion:HierarchyNavigator.ItemTemplate>
</syncfusion:HierarchyNavigator>
```



Theme

HierarchyNavigator supports various built-in themes. Refer to the below links to apply themes for the HierarchyNavigator,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Populating Data in WPF Breadcrumb (HierarchyNavigator)

HierarchyNavigator Initialization

The steps to add items to the HierarchyNavigator in XAML are as follows:

1. Create a [HierarchyNavigator](#) control.

XML

```
<syncfusion:HierarchyNavigator x:Name="hierarchyNavigatorcontrol1"
VerticalAlignment="Top" Height="30" Width="600">
```

2. Add the [HierarchyNavigatorItem](#) to the [HierarchyNavigator](#) control.

XML

```
<syncfusion:HierarchyNavigator x:Name="hierarchyNavigatorcontrol1"
VerticalAlignment="Top" Height="30" Width="600">
<syncfusion:HierarchyNavigator.Items>
<syncfusion:HierarchyNavigatorItem Content="Syncfusion">
<syncfusion:HierarchyNavigatorItem.Items>
<syncfusion:HierarchyNavigatorItem Content="User Interface"/>
<syncfusion:HierarchyNavigatorItem Content="Silverlight">
<syncfusion:HierarchyNavigatorItem.Items>
<syncfusion:HierarchyNavigatorItem Content="Tools"/>
</syncfusion:HierarchyNavigatorItem.Items>
</syncfusion:HierarchyNavigatorItem>
</syncfusion:HierarchyNavigatorItem.Items>
</syncfusion:HierarchyNavigatorItem>
</syncfusion:HierarchyNavigator.Items>
</syncfusion:HierarchyNavigator>
```

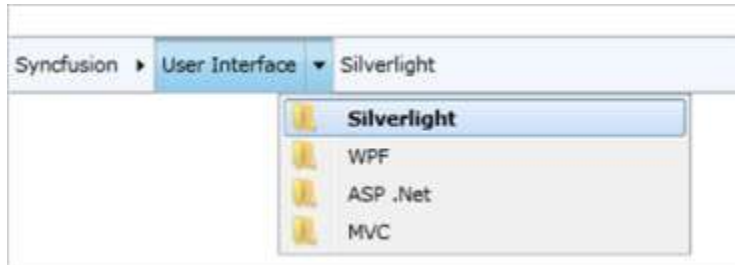
3. The snippet below demonstrates the steps to add items to a [HierarchyNavigator](#) control in code:

C#

```
HierarchyNavigator hierarchyNavigator1 = new HierarchyNavigator() { Height =
30 };
HierarchyNavigatorItem hierarchyNavigatorItem1 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem1.Content = "Syncfusion";
HierarchyNavigatorItem hierarchyNavigatorItem11 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem11.Content = "User Interface";
HierarchyNavigatorItem hierarchyNavigatorItem111 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem111.Content = "Silverlight";
HierarchyNavigatorItem hierarchyNavigatorItem112 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem112.Content = "WPF";
HierarchyNavigatorItem hierarchyNavigatorItem113 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem113.Content = "ASP .Net";
HierarchyNavigatorItem hierarchyNavigatorItem114 = new
HierarchyNavigatorItem();
hierarchyNavigatorItem114.Content = "MVC";
hierarchyNavigatorItem111.Items.Add(hierarchyNavigatorItem111);
hierarchyNavigatorItem111.Items.Add(hierarchyNavigatorItem112);
hierarchyNavigatorItem111.Items.Add(hierarchyNavigatorItem113);
hierarchyNavigatorItem111.Items.Add(hierarchyNavigatorItem114);
```

```
hierarchyNavigatorItem1.Items.Add(hierarchyNavigatorItem11);
hierarchyNavigator1.Items.Add(hierarchyNavigatorItem1);
```

The following figure shows the items added in code displayed on the interface.



Data binding

Data binding is the process of establishing a connection between the application UI and business logic.

Binding to an object

To bind to a Business Object collection, the `ItemsSource` property and `ItemTemplate` should be used in [HierarchyNavigator](#). `HierarchicalDataTemplate` should be used for all item templates.

The steps to bind to a Business Object collection are as follows:

1. Create a class named `HierarchyItem`.

C#

```
public class HierarchyItem
{
    public string ContentString { get; set; }
    public HierarchyItem(string content, params HierarchyItem[] myItems)
    {
        this.ContentString = content;
        itemsObservableCollection = new ObservableCollection<HierarchyItem>();
        foreach (var item in myItems)
        {
            itemsObservableCollection.Add(item);
        }
        HierarchyItems = itemsObservableCollection;
    }
    private ObservableCollection<HierarchyItem> itemsObservableCollection;
    public ObservableCollection<HierarchyItem> HierarchyItems
    {
        get { return itemsObservableCollection; }
        set
        {
            if (itemsObservableCollection != value)
            {
                itemsObservableCollection = value;
            }
        }
    }
}
```


2. Create a collection for `ItemsSource` to bind with.

C#

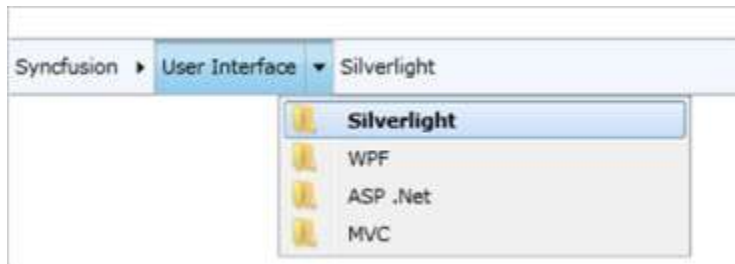
```
public class HierarchicalItemsSource : ObservableCollection<HierarchyItem>
{
    public HierarchicalItemsSource()
    {
        this.Add(new HierarchyItem("Syncfusion",
            new HierarchyItem("User Interface",
                new HierarchyItem("Silverlight"),
                new HierarchyItem("WPF"),
                new HierarchyItem("ASP .Net"),
                new HierarchyItem("MVC")),
            new HierarchyItem("Reporting Edition",
                new HierarchyItem("IO"),
                new HierarchyItem("PDF generator"),
                new HierarchyItem("WPF"))
        ));
    }
}
```

3. In XAML, bind the collections to the `ItemsSource` property of the [HierarchyNavigator](#) control.

XML

```
<syncfusion:HierarchyNavigator Name="hierarchyNavigator2">
  <syncfusion:HierarchyNavigator.ItemsSource>
    <local:HierarchicalItemsSource />
  </syncfusion:HierarchyNavigator.ItemsSource>
  <syncfusion:HierarchyNavigator.ItemTemplate>
    <HierarchicalDataTemplate ItemsSource="{Binding HierarchyItems}">
      <TextBlock Text="{Binding ContentString}" Margin="2,0" />
    </HierarchicalDataTemplate>
  </syncfusion:HierarchyNavigator.ItemTemplate>
</syncfusion:HierarchyNavigator>
```

The following screenshot shows the items added in code displayed on the interface.



Binding XML data

To bind XML data to a [HierarchyNavigator](#) control, convert the XML to a collection, and then bind the collection by using the `ItemsSource` property.

The XML displayed below is used in this example (attached in the sample project named HierarchyItems.xml).

XML

```
<categories>
  <category name="Syncfusion">
  <category name="User Interface">
  <category name="Silverlight">
  <category name="Essential Tools" />
  <category name="Essential Grid" />
  <category name="Essential Schedule" />
  <category name="Essential Ribbon" />
  <category name="Essential Gauge" />
  <category name="Essential Chart" />
</category>
<category name="WPF" />
<category name="MVC" />
<category name="ASP .Net" />
</category>
</category>
</categories>
```

The steps to bind XML data to a [HierarchyNavigator](#) control are as follows:

1. Create a separate class that represents the node in an XML document. In this example, a class named Category is created.

C#

```
public class HierarchyItem
{
  public string ContentStr { get; set; }
  ObservableCollection<HierarchyItem> hierarchyItems = new
  ObservableCollection<HierarchyItem>();
  public ObservableCollection<HierarchyItem> HierarchyItems
  {
    get { return hierarchyItems; }
    set { hierarchyItems = value; }
  }
}
```

2. Convert the XML data to a collection, and then bind the collection to the `ItemsSource` property of [HierarchyNavigator](#).

C#

```
public partial class MainPage : UserControl
{
  public MainPage()
  {
    InitializeComponent();
    CreateXMLDataItemsSource();
  }
}
```

```

}
private void CreateXMLDataItemsSource()
{
    ObservableCollection<HierarchyItem> categories = new
    ObservableCollection<HierarchyItem>();
    XDocument XMLItemSource = XDocument.Load("/HierarchyItems.xml");
    categories = this.GetCategories(XMLItemSource.Element("categories"));
    hierarchyNavigator1.ItemsSource = categories;
}
private ObservableCollection<HierarchyItem> GetCategories(XElement element)
{
    var item = from category in element.Elements("category") select category;
    var itemsObservableCollection = new ObservableCollection<HierarchyItem>();
    foreach (var itm in item)
    {
        var subitm = new HierarchyItem();
        subitm.ContentStr = itm.Attribute("name").Value;
        subitm.HierarchyItems = this.GetCategories(itm);
        itemsObservableCollection.Add(subitm);
    }
    return itemsObservableCollection;
}
}

```

3. The code for the [HierarchyNavigator](#) is shown below. Declare HierarchicalDataTemplate, because the data is in a hierarchical structure. Refer Template Customizing.

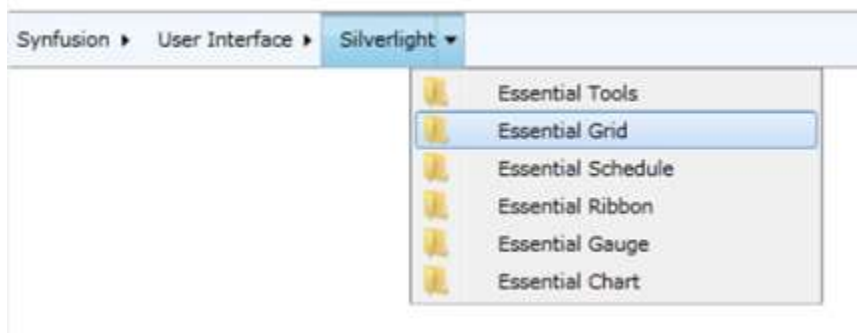
XML

```

<syncfusion:HierarchyNavigator VerticalAlignment="Center"
Name="hierarchyNavigator1" Height="30">
<syncfusion:HierarchyNavigator.ItemTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding HierarchyItems}">
<TextBlock Margin="10,0,0,0" Text="{Binding ContentStr}" Grid.Column="0"/>
</HierarchicalDataTemplate>
</syncfusion:HierarchyNavigator.ItemTemplate>
</syncfusion:HierarchyNavigator>

```

The image displayed below shows the output of the above code—items bound to XML data.

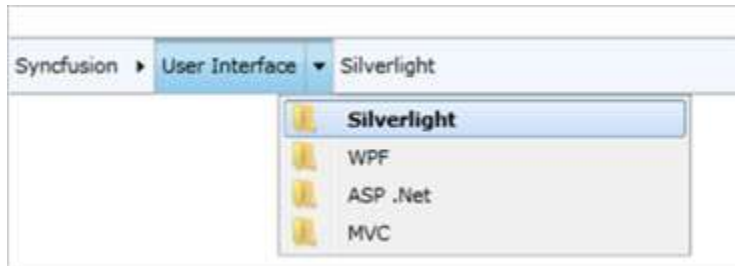


Binding to WCF service

XML data can be bound through WCF services by enabling WCF Services.

The steps to bind XML data through WCF services are as follows:

1. Create an XML and a class object. Refer the XML data-binding class and the XML used in the
2. The following screenshot shows the items added in code displayed on the interface.



3. Binding XML data section.
4. Add an ASP.NET Web application with the sample application, and then add a new WCF Service item to the Web application, to create a WCF service application.
5. Create an Observable Collection from XML data to bind in ItemsSource, as shown below in the service class that has a return type of ObservableCollection.

C#

```
[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
public class Service1
{
    [OperationContract]
    public ObservableCollection<HierarchyItem> CreateXMLDataItems()
    {
        ObservableCollection<HierarchyItem> categories = new
        ObservableCollection<HierarchyItem>();
        XmlDocument XMLItemSource =
        XmlDocument.Load("YourXMLLocation/HierarchyItems.xml");
        categories = this.GetCategories(XMLItemSource.Element("categories"));
        return categories;
    }
    private ObservableCollection<HierarchyItem> GetCategories(XElement element)
    {
        var item = from category in element.Elements("category") select category;
        var itemsObservableCollection = new ObservableCollection<HierarchyItem>();
        foreach (var itm in item)
        {
            var subitm = new HierarchyItem();
            subitm.ContentStr = itm.Attribute("name").Value;
            subitm.HierarchyItems = this.GetCategories(itm);
            itemsObservableCollection.Add(subitm);
        }
        return itemsObservableCollection;
    }
}
```

```

//To connect WCF services with the sample application, use the code snippets
displayed below. Also refer Binding data with WCF Service in the How To
section.
namespace WCFServicesInHierarchy
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
        }
        public class CustomSource
        {
            public CustomSource()
            {
                //This loads WCF Service
                Service1Client client = new Service1Client();
                client.CreateXMLDataItemsCompleted += new
                EventHandler<CreateXMLDataItemsCompletedEventArgs>(client_CreateXMLDataItems
                Completed);
                client.CreateXMLDataItemsAsync();
                this.Categories = new ObservableCollection<HierarchyItem>();
            }
            private void client_CreateXMLDataItemsCompleted(object sender,
            CreateXMLDataItemsCompletedEventArgs e)
            {
                if (e.Error == null && e.Result != null)
                {
                    foreach (HierarchyItem c in e.Result)
                    {
                        this.Categories.Add(c);
                    }
                }
            }
            public ObservableCollection<HierarchyItem> Categories
            {
                get;
                set;
            }
        }
    }
}

```

XML

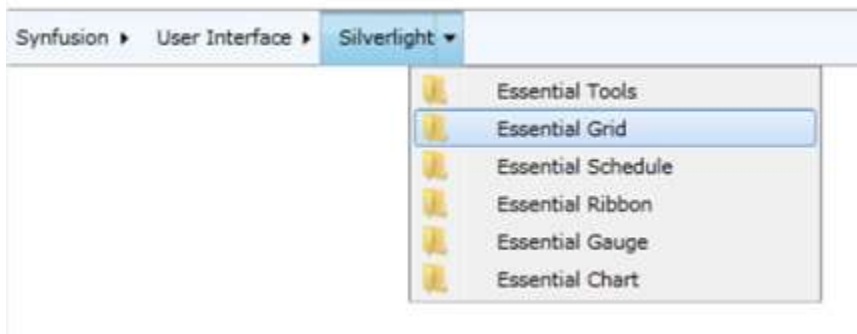
```

<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:local="clr-namespace:WCFServicesInHierarchy"
x:Class="WCFServicesInHierarchy.MainWindow"
x:Name="Window" Title="MainWindow" UseLayoutRounding="True" Width="640" Height="480">
    <Window.DataContext>
        <local:CustomSource/>
    </Window.DataContext>

```

```
<Grid x:Name="LayoutRoot">
  <syncfusion:HierarchyNavigator Name="hierarchyNavigator1" VerticalAlignmen=
    "Center" ItemsSource="{Binding Categories}">
    <syncfusion:HierarchyNavigator.ItemTemplate>
      <HierarchicalDataTemplate ItemsSource="{Binding HierarchyItems}">
        <Border>
          <TextBlock Text="{Binding ContentStr}" Margin="2,0"/>
        </Border>
      </HierarchicalDataTemplate>
    </syncfusion:HierarchyNavigator.ItemTemplate>
  </syncfusion:HierarchyNavigator>
</Grid>
</Window>
```

The image displayed below shows the output of the above code—items bound to XML data.



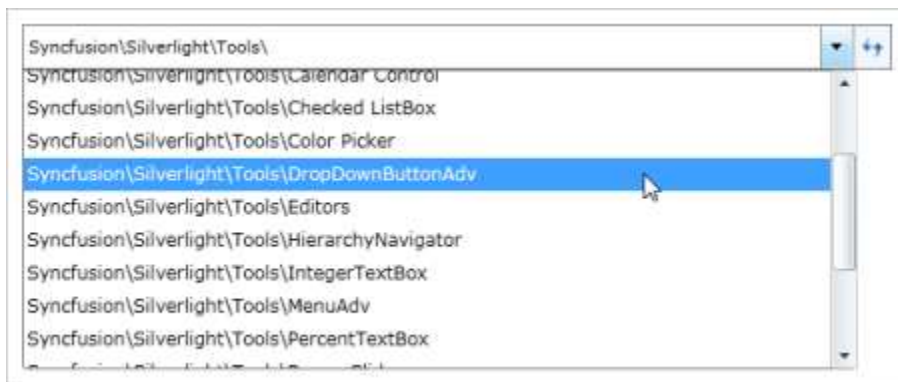
See Also

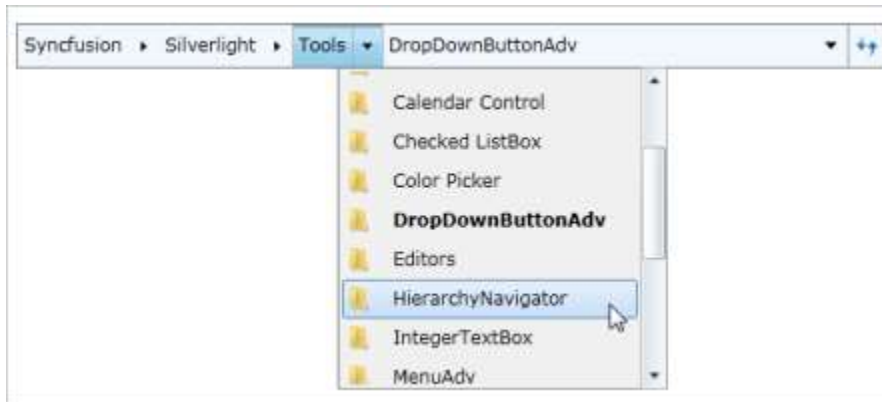
[How to load items in WPF HierarchyNavigator \(Breadcrumb\) control in on-demand?](#)

Mouse Support in WPF Breadcrumb (HierarchyNavigator)

An item can be selected by rolling the mouse over it. The highlight that appears on the rolled over item helps identify the item that is currently selected. You can click next to an item to view a navigation pop-up window.

The mouse wheel can be used to scroll a vertical scroll bar that is enabled in the History or Navigation pop-up control to view the full content.





The mouse can be rolled over a Hierarchy Navigator item to view a pop-up (without clicking the item) even when there is a pop-up displayed for another item.

Keyboard Support in WPF Breadcrumb (HierarchyNavigator)

The HierarchyNavigator control supports keyboard navigation. This allows you to select an item and open a pop-up menu by using the keyboard.

Keyboard functionality table

Key	Functionality
Arrow Keys	Used to navigate to items in the Navigation pop-up window.
Home	Used to select the first item, if the scroll bar is enabled in the Navigation pop-up window.
End	Used to select the last item, if the scroll bar is enabled in the Navigation pop-up window.
Esc	Used to disable the Edit mode, if it is enabled.
Enter	Used to select an item and disable the Edit mode.
Page Up and Page Down	Used to move through pages in the Navigation pop-up window, if the scroll bar is enabled.

Edit Mode in WPF Breadcrumb (HierarchyNavigator)

This feature allows you to easily edit a navigation path by setting the `IsEnableEditMode` property to `true`. The filter support is available in edit mode, which suggests matching nodes based on the path entered in the editor like the Windows Explorer.

To start the edit mode, click `HierarchyNavigatorItemsControl` and enter a navigation path that will display a drop-down list of filtered navigation paths, and then select the expected navigation path.

Note: If you enter an incorrect path or text in the editor, the dropdown suggestion list will be closed.

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="HierarchicalNavigatorSample.MainWindow"
```

```

Title="HierarchicalNavigator Sample" Height="350" Width="525">
<Grid>
<!--Adding HierarchicalNavigator control -->
<syncfusion:HierarchicalNavigator x:Name="hierarchicalNavigator" Width="100"
Height="100" IsEnableEditMode="true" VerticalAlignment="Center"
HorizontalAlignment="Center"/>
</Grid>
</Window>

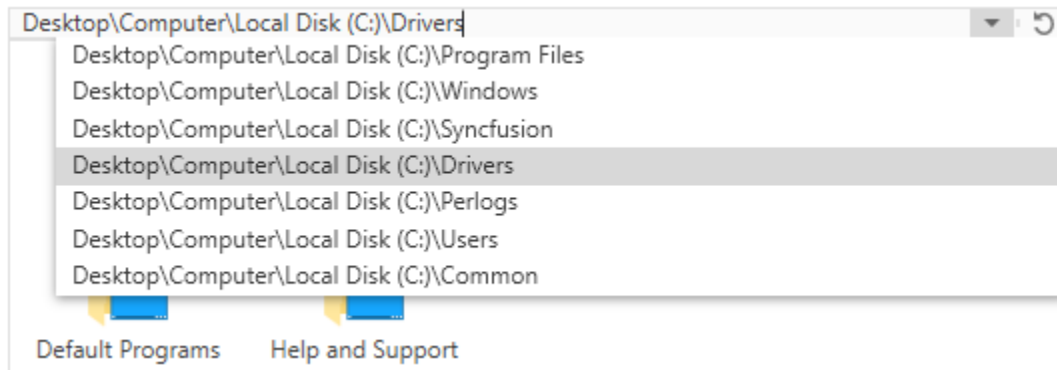
```

C#

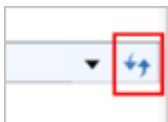
```

using Syncfusion.Windows.Tools;
namespace HierarchicalNavigatorSample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Creating an instance of HierarchicalNavigator control
            HierarchicalNavigator hierarchicalNavigator = new HierarchicalNavigator();
            hierarchyNavigator.IsEnabledEditMode = true;
            //Adding HierarchicalNavigator as window content
            this.Content = hierarchicalNavigator;
        }
    }
}

```

[Refresh Button in WPF Breadcrumb \(HierarchyNavigator\)](#)

The Refresh button enables the `HierarchyNavigatorRefreshButtonClick` event to initiate in the `HierarchyNavigator` control.

**XML**


```
<locals:HierarchyNavigator HierarchyNavigatorRefreshButtonClick="HierarchyNa
vigatorRefreshButtonClick" />
```

C#

```
HierarchyNavigator hierarchyNavigator = new HierarchyNavigator();
<br>
hierarchyNavigator.HierarchyNavigatorRefreshButtonClick += new EventHandler(
HierarchyNavigatorRefreshButtonClick);
private void HierarchyNavigatorRefreshButtonClick(object sender, EventArgs e
)
<br>
{<br>    //Occurs when Refresh Button Click<br>}
```

Template Customizing in WPF Breadcrumb (HierarchyNavigator)

When Business Objects are used for data binding, ItemTemplate should be used to determine how the user interface will be displayed. A HierarchicalDataTemplate can be used as an ItemTemplate specifying the template for each item.

HierarchicalDataTemplate

HierarchicalDataTemplate is used to show data in a hierarchical structure.

The steps to create a HierarchicalDataTemplate are as follows:

1. Create a Business Object with its collections.

C#

```
public class HierarchyItem
{
    public string ContentString { get; set; }
    public HierarchyItem(string content, params HierarchyItem[] myItems)
    {
        this.ContentString = content;
        itemsObservableCollection = new ObservableCollection<HierarchyItem>();
        foreach (var item in myItems)
        {
            itemsObservableCollection.Add(item);
        }
        Items = itemsObservableCollection;
    }
    private ObservableCollection<HierarchyItem> itemsObservableCollection;
    public ObservableCollection<HierarchyItem> Items
    {
        get { return itemsObservableCollection; }
        set
        {
            if (itemsObservableCollection != value)
            {
                itemsObservableCollection = value;
            }
        }
    }
}
```

```
public class HierarchicalItemsSource : ObservableCollection<HierarchyItem>
{
    public HierarchicalItemsSource()
    {
        this.Add(new HierarchyItem("Syncfusion",
            new HierarchyItem("User Interface",
                new HierarchyItem("Silverlight"),
                new HierarchyItem("WPF"),
                new HierarchyItem("MVC")),
            new HierarchyItem("Reporting Edition",
                new HierarchyItem("IO"),
                new HierarchyItem("PDF generator"),
                new HierarchyItem("WPF"))
        ));
    }
}
```

2. Add HierarchicalDataTemplate.
3. Add a template and ItemsSource to bind the object collections.

XML

```
<Window.Resources>
<local:HierarchicalItemsSource x:Key="hierarchicalItemsSource"/>
<HierarchicalDataTemplate x:Key="myHierarchicalTemplate"
    ItemsSource="{Binding Items}">
<TextBlock Text="{Binding ContentString}" Margin="2,0" />
</HierarchicalDataTemplate>
</Window.Resources>
```

4. Add hierarchicalItemsSource with the HierarchyNavigator control ItemsSource property.

XML

```
<syncfusion:HierarchyNavigator Name="hierarchyNavigator1"
    ItemTemplate="{StaticResource myHierarchicalTemplate}"
    ItemsSource="{StaticResource hierarchicalItemsSource}"/>
```



Progress Bar in WPF Breadcrumb (HierarchyNavigator)

The progress bar for the HierarchyNavigator control can be displayed or removed.



There are two methods to display the progress bar:

1. Calling the ShowProgressBar method in HierarchyNavigator, which displays the progress bar for a time span of 500 ms.

C#

```
HierarchyNavigator hierarchyNavigator = new HierarchyNavigator();  
hierarchyNavigator.ShowProgressBar();
```

2. Passing an argument in the method to show a specified time span. The image below shows a time span of 1000 ms that has been passed.

C#

```
HierarchyNavigator hierarchyNavigator = new HierarchyNavigator();  
hierarchyNavigator.ShowProgressBar(new TimeSpan(0, 0, 0, 0, 1000));
```

The progress bar can be canceled by using two methods:

1. Calling the CancelProgressBar method in HierarchyNavigator.

C#

```
HierarchyNavigator hierarchyNavigator = new HierarchyNavigator();  
hierarchyNavigator.CancelProgressBar();
```

2. Passing an argument in the method to cancel the progress bar within a particular span of time. This method helps users cancel the progress bar when preferred. The image displayed below shows a time span of 1000 ms that has been passed.

C#

```
HierarchyNavigator hierarchyNavigator = new HierarchyNavigator();  
hierarchyNavigator.CancelProgressBar(new TimeSpan(0, 0, 0, 0, 1000));
```

Command Binding in WPF Breadcrumb (HierarchyNavigator)

The HierarchyNavigator control enables command binding when a selected item is changed. There are two properties that occur when this happens.

The steps to listen to the command binding are as follows:

1. Create a DelegateCommand class inherited from ICommand, which will be used in the ViewModel sample class.

C#

```
public class DelegateCommand : ICommand  
{  
    private Predicate<object> _canExecute;  
    private Action<object> _method;
```

```

public event EventHandler CanExecuteChanged;
public DelegateCommand(Action<object> method)
: this(method, null)
{
    _method = method;
}
public DelegateCommand(Action<object> method, Predicate<object> canExecute)
{
    _method = method;
    _canExecute = canExecute;
}
public bool CanExecute(object parameter)
{
    if (_canExecute == null)
    {
        return true;
    }
    return _canExecute(parameter);
}
public void Execute(object parameter)
{
    _method.Invoke(parameter);
}
}

```

2. Create the ViewModel sample class, to bind the command in the sample WPF application.

C#

```

public class ViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    private DelegateCommand _myCommand1;
    private string _lastCommand = "Syncfusion";
    public string LastCommand
    {
        get
        {
            return _lastCommand;
        }
        private set
        {
            _lastCommand = value;
            PropertyChanged(this, new PropertyChangedEventArgs("LastCommand"));
        }
    }
    public DelegateCommand SelectedItemCommand
    {
        get
        {
            if (_myCommand1 == null)
            {
                _myCommand1 = new DelegateCommand(MyCommand1Method);
            }
            return _myCommand1;
        }
    }
}

```

```
private void MyCommand1Method(object parameter)
{
    if (parameter as Syncfusion.Windows.Tools.Controls.HierarchyNavigatorItem != null)
    {
        LastCommand = (parameter as Syncfusion.Windows.Tools.Controls.HierarchyNavigatorItem).Content.ToString();
    }
}
```

1. Bind the command in the HierarchyNavigator control.
2. To do this, create a new instance of the ViewModel sample class and set DataContext for the parent StackPanel. This will reflect changes in the children. Whenever the selected item changes, the TextBox Text value will change.

XML

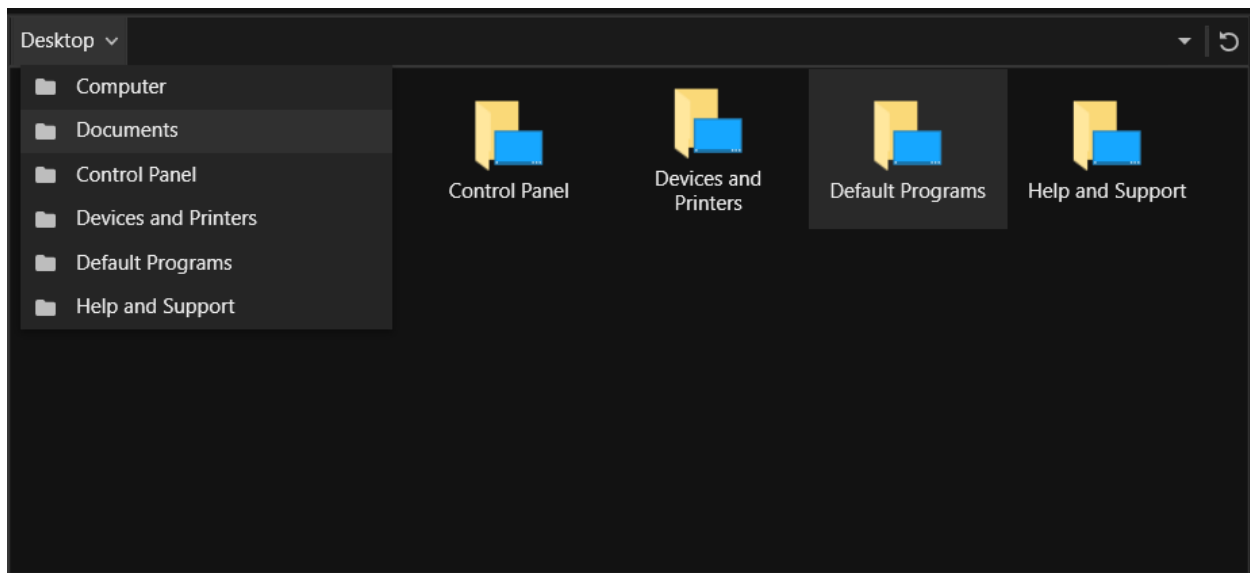
```
<StackPanel Name="CommandBindingStackPanel">
    <StackPanel.DataContext>
        <local:ViewModel />
    </StackPanel.DataContext>
    <syncfusion:HierarchyNavigator Name="hierarchyNavigator1"
        VerticalAlignment="Center"
        Command="{Binding SelectedItemCommand}"
        Items="{StaticResource NavigationSampleData}" />
    <TextBlock Text="Selected Item" VerticalAlignment="Center" FontWeight="Bold" />
    <TextBox Text="{Binding LastCommand}" Height="50" IsReadOnly="True"
        Margin="2" />
</StackPanel>
```

Skins in WPF Breadcrumb (HierarchyNavigator)

Theme

HierarchyNavigator supports various built-in themes. Refer to the below links to apply themes for the HierarchyNavigator,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Tile Control

WPF Tile Control Overview

The WPF hub tile and pulsing tile controls are used to create a UI similar to the tile feature in Windows Desktop and Mobile. It can be arranged in different layouts and provides updates and notifications with various transitions effects.

Key Features

- Hub Tile – Supports tile feature similar to Windows Desktop and Windows Phone. It also supports various transition effects and provides support for the display of primary and secondary content for updates and notifications.
- Pulsing Tile – Supports tile feature similar to music and video tiles in Windows Phone. The control allows the content to be zoomed in/out and translates the content across the x-axis and y-axis.
- Grouping - Support for grouping multiple tiles together.
- Freezing/Unfreezing – Support for freezing and unfreezing the tile.
- Themes - Support for customizing the tile using different themes.

Getting Started with WPF Tile Control

This section gives an overview for working with the Hub Tile and Pulsing Tile controls.

Assembly deployment

Refer [Hub Tile](#) and [Pulsing Tile](#) control dependencies section to get the list of assemblies or [NuGet package](#) needs to be added as reference to use the Hub Tile and Pulsing Tile control in any application.

Creating simple application with Hub Tile and Pulsing Tile

In this walk through, a WPF application that contains Hub Tile and Pulsing Tile control can be created. By the following ways, the controls can be added:

1. [Adding control via Designer](#)
2. [Adding control manually in XAML](#)

3. [Adding control manually in C#](#)

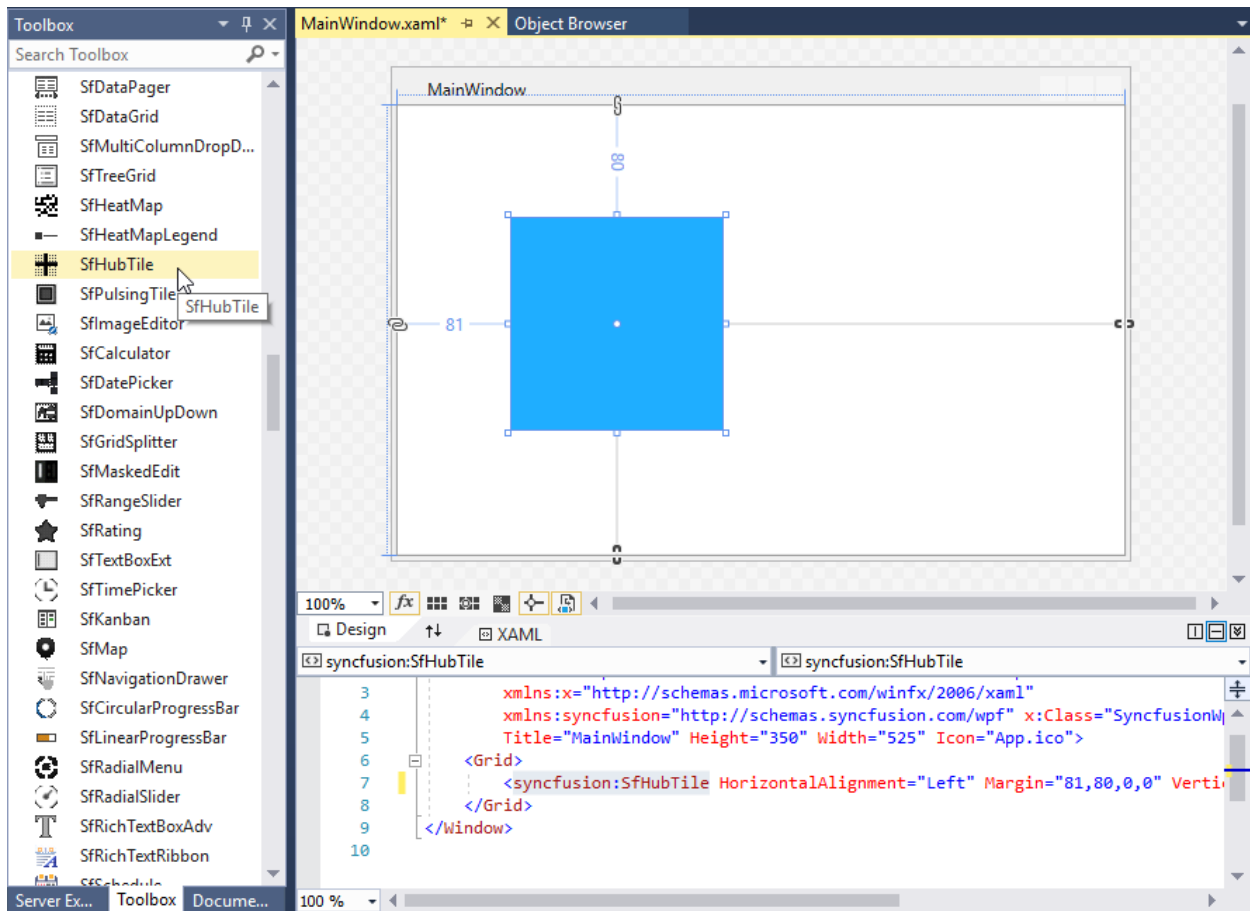
Adding control via Designer

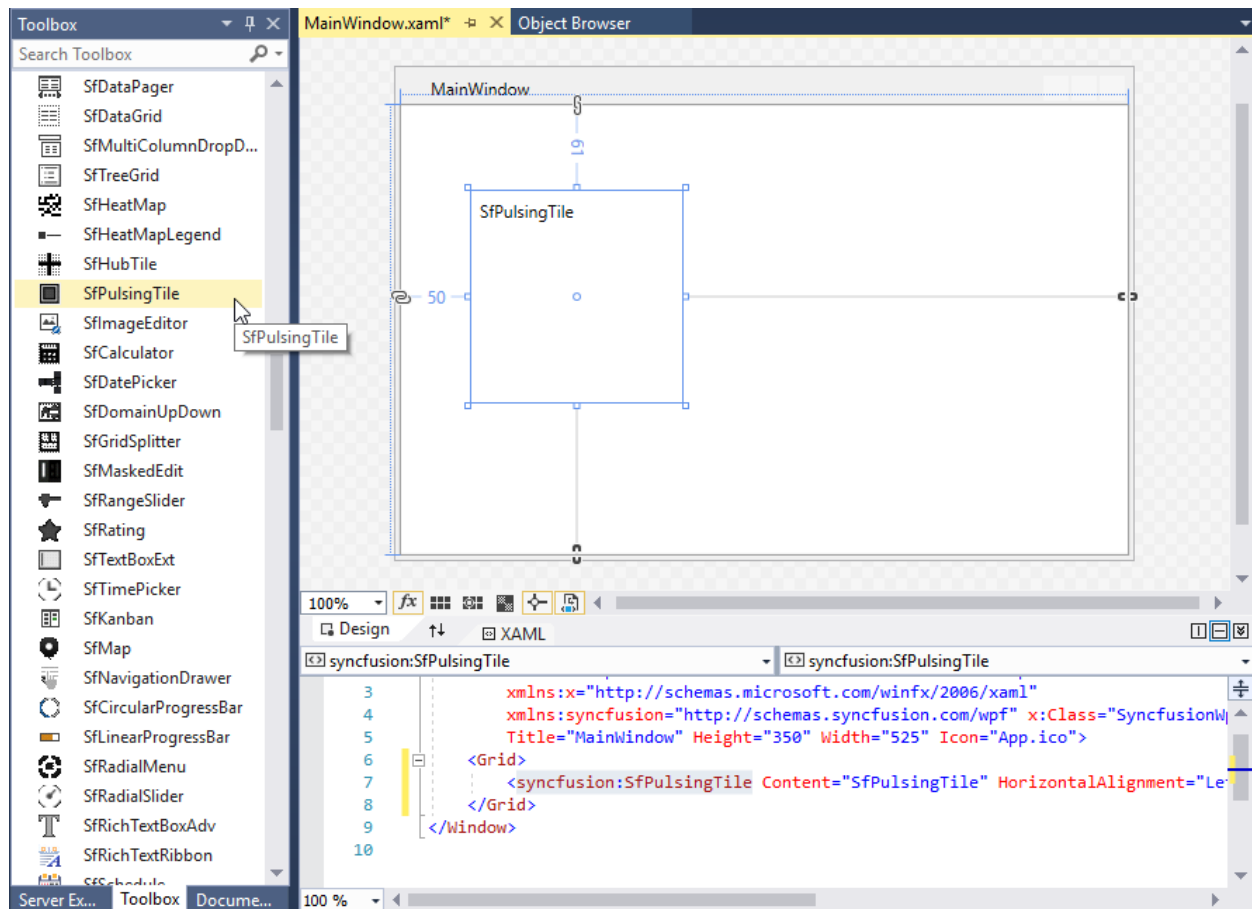
Hub Tile and Pulsing Tile controls can be added to the application by dragging SfHubTile and SfPulsingTile from toolbox and dropping it in designer view. After dropping the controls in designer view, the assemblies such as **Syncfusion.SfHubTile.WPF** and **Syncfusion.SfShared.WPF** gets added into the project automatically. The following code snippets will also be added to XAML.

XML

```
<!--For Hub Tile-->
<syncfusion:SfHubTile HorizontalAlignment="Left" VerticalAlignment="Top"/>
<!--For Pulsing Tile-->
<syncfusion:SfPulsingTile Content="SfPulsingTile" HorizontalAlignment="Left"
VerticalAlignment="Top"/>
```

Note: "syncfusion" in XAML is an auto generated namespace.





Adding control manually in XAML

In order to add control manually in XAML, follow the below steps:

1. Add the below required assembly references to the project.
 - Syncfusion.SfHubTile.WPF
 - Syncfusion.SfShared.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> or the tile control namespace **Syncfusion.Windows.Controls.Notification** in XAML page.
3. Declare **SfHubTile** and **SfPulsingTile** controls 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="WpfApplication1.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid>
<!--Hub Tile-->
<syncfusion:SfHubTile Content="This is a Hub Tile"/>
<!--Pulsing Tile-->
<syncfusion:SfPulsingTile Content="This is a Pulsing Tile"/>
</Grid>
</Window>

```


Adding control manually in C#

In order to add control manually in C#, follow the below steps:

1. Add the below required assembly references to the project.
 - Syncfusion.SfHubTile.WPF
 - Syncfusion.SfShared.WPF
2. Import the `Syncfusion.Windows.Controls.Notification` namespace.
3. Create SfHubTile and SfPulsingTile controls instance and add it to the window.

XML

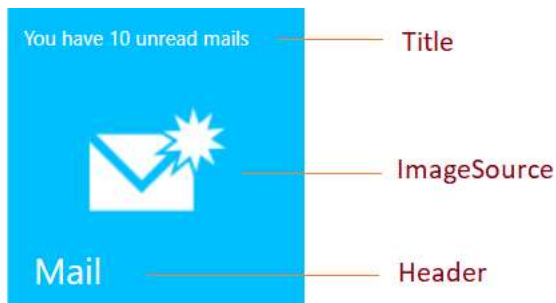
```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
x:Class="WpfApplication1.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid x:Name="grid">
</Grid>
</Window>
```

C#

```
using Syncfusion.Windows.Controls.Notification;
namespace SfHubTileSample
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            // Hub Tile
            SfHubTile hubTile = new SfHubTile();
            grid.Children.Add(hubTile);
            //Pulsing Tile
            SfPulsingTile pulseTile = new SfPulsingTile();
            grid.Children.Add(pulseTile);
        }
    }
}
```

Setting title, header and image in tile

The title, header and image for the tile can be set by using [Title](#), [Header](#) and [ImageSource](#) properties respectively.



Hub Tile



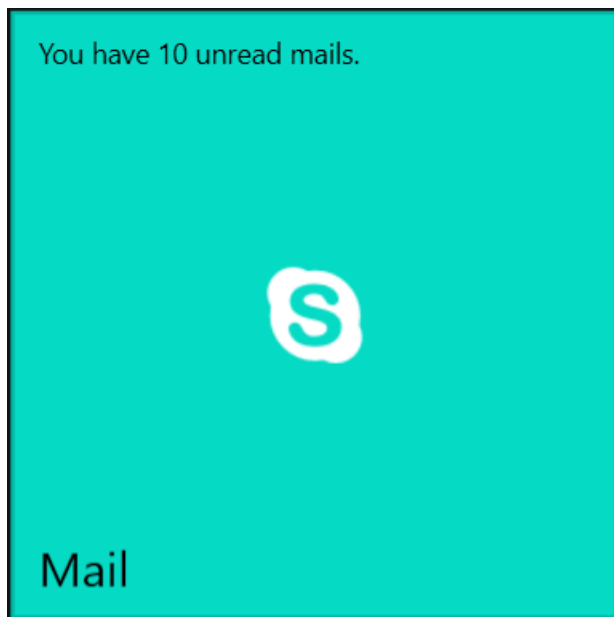
Pulsing Tile

Note: The title will be displayed at the top of the tile. The header will be displayed at the bottom of the tile. The image will be displayed at the center of the tile.

Theme

Hub Tile and Pulsing Tile controls supports various built-in themes. Refer to the below links to apply themes for the Hub Tile and Pulsing Tile,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Hub Tile in WPF Tile Control

The Hub Tile control supports the tile feature similar to the live tile feature of Windows Desktop and Windows Phone. This section explains about the supporting features of Hub Tile control.

Setting header content

Header can act as the name of the tile, that is placed at the bottom explaining its purpose. The content of the header can be an image, a text or a control, etc. The header can be set to the tile by using the [Header](#) property.

XML

```
<Grid x:Name="grid">
  <!--For setting header as text use this code. -->
  <syncfusion:SfHubTile x:Name="hubtile" Header="Mail" Foreground="White"/>
  <!--For setting header as an image use this code.-->
  <syncfusion:SfHubTile x:Name="hubtile">
    <syncfusion:SfHubTile.Header>
      <Image Source="/Assets/syncfusion.png" Stretch="None"/>
    </syncfusion:SfHubTile.Header>
  </syncfusion:SfHubTile>
  <!--For setting header as a control use this code.-->
  <syncfusion:SfHubTile x:Name="hubtile">
    <syncfusion:SfHubTile.Header>
      <TextBlock Text="SYNCFUSION" Foreground="White" FontSize="13"/>
    </syncfusion:SfHubTile.Header>
  </syncfusion:SfHubTile>
</Grid>
```

C#

```
// For setting header as text use this code.
SfHubTile hubTile = new SfHubTile();
hubTile.Header = "Mail";
hubTile.Foreground = Brushes.White;
//For setting header as image use this code.
SfHubTile hubTile = new SfHubTile();
Image image = new Image() {Source = new BitmapImage(new
Uri(@"Assets/syncfusion.png", UriKind.RelativeOrAbsolute)), Stretch =
Stretch.None };
hubTile.Header = image;
//For setting header as control use this code.
SfHubTile hubTile = new SfHubTile();
TextBlock textblock = new TextBlock() {Text = "SYNCFUSION", Foreground =
Brushes.White, FontSize = 13} ;
hubTile.Header = textblock;
grid.Children.Add(hubTile);
```



Setting title content

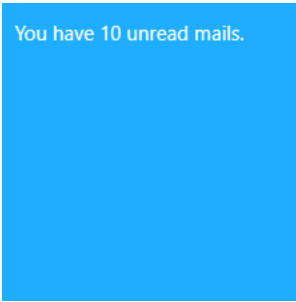
Title can be used to display updates and notifications in a tile. The content of the title can be an image, a text or a control, etc. The title can be set to the tile by using the [Title](#) property.

XML

```
<Grid x:Name="grid">
  <!--For setting title as text use this code.-->
  <syncfusion:SfHubTile x:Name="hubTile" Title="You have 10 unread mails."
  Foreground="White"/>
  <!--For setting title as an image use this code.-->
  <syncfusion:SfHubTile x:Name="hubTile">
    <syncfusion:SfHubTile.Title>
      <Image Source="/Assets/syncfusion.png" Stretch="None"
      HorizontalAlignment="Left"/>
    </syncfusion:SfHubTile.Title>
  </syncfusion:SfHubTile>
  <!--For setting title as a control use this code.-->
  <syncfusion:SfHubTile x:Name="hubTile">
    <syncfusion:SfHubTile.Title>
      <TextBlock Text="SYNCFUSION" Foreground="White" FontSize="13"/>
    </syncfusion:SfHubTile.Title>
  </syncfusion:SfHubTile>
</Grid>
```

C#

```
//Setting title on Hub Tile.
SfHubTile hubTile = new SfHubTile();
hubTile.Title = "You have 10 unread mails.";
hubTile.Foreground = Brushes.White;
//For setting title as image use this code.
SfHubTile hubTile = new SfHubTile();
Image image = new Image(){Source = new BitmapImage(new
Uri(@"Assets/syncfusion.png",UriKind.RelativeOrAbsolute)), Stretch
=Stretch.None,HorizontalAlignment = HorizontalAlignment.Left };
hubTile.Title = image;
// For setting title as control use this code.
SfHubTile hubTile = new SfHubTile();
TextBlock textblock = new TextBlock(){Text = "SYNCFUSION", Foreground =
Brushes.White, FontSize = 13} ;
hubTile.Title = textblock;
grid.Children.Add(hubTile);
```

**Setting image**

The image acts as a pictorial representation of the purpose of tile control. The image can be set to the tile by setting image path to the [ImageSource](#) property.

XML

```
<Grid x:Name="grid">
  <!--SfHubTile-->
  <syncfusion:SfHubTile x:Name="hubTile" ImageSource="/Assets/New Mail.png"/>
</Grid>
```

C#

```
//Setting image for hub tile
SfHubTile hubTile = new SfHubTile();
hubTile.ImageSource = new BitmapImage(new Uri(@"Assets/New
Mail.png",UriKind.RelativeOrAbsolute));
grid.Children.Add(hubTile);
```



Setting secondary content

The [secondary content](#) specifies the content to be displayed when the tile content of the tile changes during each transition, that is, from primary view. Secondary content can be an image, a text or a control.

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:shared="clr-
  namespace:Syncfusion.Windows.Controls;assembly=Syncfusion.SfShared.Wpf"
  x:Class="WpfApplication1.MainWindow"
  Title="MainWindow" Height="350" Width="525">
  <Grid x:Name="grid">
    <!--For setting secondary content as image use this code.-->
    <syncfusion:SfHubTile x:Name="hubTile" ImageSource="/Assets/New Mail.png"
      Title="This is title area." Header="Mail" Foreground="White"
      Interval="00:00:03">
      <syncfusion:SfHubTile.HubTileTransitions>
      <shared:SlideTransition/>
      </syncfusion:SfHubTile.HubTileTransitions>
    <!-- For setting secondary content -->
    <syncfusion:SfHubTile.SecondaryContent>
      <Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
    </syncfusion:SfHubTile.SecondaryContent>
    </syncfusion:SfHubTile>
    <!--For setting secondary content as text use this code.-->
    <syncfusion:SfHubTile x:Name="hubTile" ImageSource="/Assets/New Mail.png"
      Title="This is title area." Width="200" Height="200" Header="Mail"
      Foreground="White" Interval="00:00:03" SecondaryContent="This is the
      secondary content.">
```

```

<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
<!--For setting secondary content as control use this code.-->
<syncfusion:SfHubTile x:Name="hubTile" ImageSource="/Assets/New Mail.png"
Title="This is title area." Header="Mail" Foreground="White"
Interval="00:00:03">
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
<syncfusion:SfHubTile.SecondaryContent>
<TextBlock Text="This is the secondary content of the tile displayed at each
transition." TextWrapping="Wrap" FontSize="15" Foreground="White"/>
</syncfusion:SfHubTile.SecondaryContent>
</syncfusion:SfHubTile>
</Grid>
</Window>

```

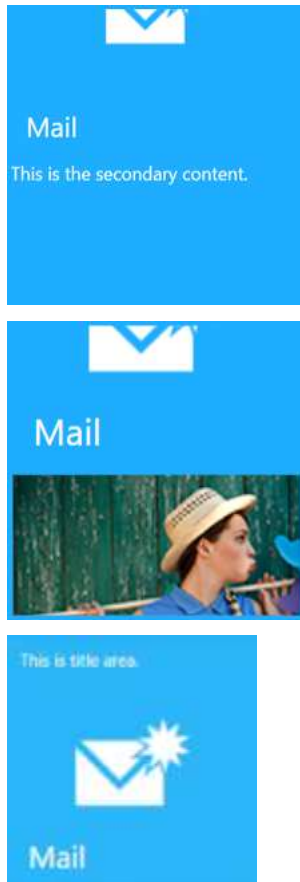
C#

```

SfHubTile hubTile= new SfHubTile();
hubTile.Header = "Mail";
hubTile.Title = "This is title area.";
hubTile.Foreground = Brushes.White;
hubTile.ImageSource = new BitmapImage(new Uri(@"Assets/New Mail.png",
UriKind.RelativeOrAbsolute));
hubTile.HubTileTransitions.Add(new SlideTransition());
//For setting secondary content as an image use this code.
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/HubTile.png", UriKind.RelativeOrAbsolute)) };
hubTile.SecondaryContent = image;
hubTile.Interval = TimeSpan.FromSeconds(3.0);
//For setting secondary content as a control use this code.
SfHubTile hubTile= new SfHubTile();
hubTile.Header = "Mail";
hubTile.Title = "This is title area.";
hubTile.Foreground = Brushes.White;
hubTile.ImageSource = new BitmapImage(new Uri(@"Assets/New Mail.png",
UriKind.RelativeOrAbsolute));
hubTile.HubTileTransitions.Add(new SlideTransition());
TextBlock textblock = new TextBlock() { Text = "This is the secondary
content of the tile displayed at each transition.", TextWrapping =
TextWrapping.Wrap, FontSize = 15, Foreground = Brushes.White };
hubTile.SecondaryContent = textblock;
hubTile.Interval = TimeSpan.FromSeconds(3.0);
//For setting secondary content as text use this code.
SfHubTile hubTile= new SfHubTile();
hubTile.Header = "Mail";
hubTile.Title = "This is title area.";
hubTile.Foreground = Brushes.White;
hubTile.ImageSource = new BitmapImage(new Uri(@"Assets/New Mail.png",
UriKind.RelativeOrAbsolute));
hubTile.HubTileTransitions.Add(new SlideTransition());
hubTile.SecondaryContent = "This is the secondary content.";
hubTile.Interval = TimeSpan.FromSeconds(3.0);

```

```
grid.Children.Add(hubTile);
```



Animation

The tile press animation takes place when the center of the tile is pressed. The tile press animation causes the entire tile to be zoomed in/out at specified interval. The tile press animation can be set by using properties such as [ScaleDepth](#) and [TilePressDuration](#) in the Hub Tile. The [ScaleDepth](#) is used to customize the depth of scaling effect while pressing the center of the tile. The [TilePressDuration](#) is used to determine the time taken for the single tile press animation.

XML

```
<Grid x:Name="grid">
  <syncfusion:SfHubTile x:Name="hubtile" TilePressDuration="00:00:02"
    ScaleDepth="2" Interval="00:00:03" Title="This is title area" Header="Mail"
    Foreground="White" ImageSource="/Assets/New Mail.png">
    <syncfusion:SfHubTile.HubTileTransitions>
      <shared1:SlideTransition/>
    </syncfusion:SfHubTile.HubTileTransitions>
    <syncfusion:SfHubTile.SecondaryContent>
      <Image Source="/Assets/HubTile.png" Stretch="UniformToFill"/>
    </syncfusion:SfHubTile.SecondaryContent>
  </syncfusion:SfHubTile>
</Grid>
```

C#


```
SfHubTile hubTile = new SfHubTile();
hubTile.Header = "Mail";
hubTile.Title = "This is title area";
hubTile.Foreground = Brushes.White;
hubTile.Interval = TimeSpan.FromSeconds(3.0);
hubTile.ImageSource = new BitmapImage(new Uri(@"Assets/New
Mail.png", UriKind.RelativeOrAbsolute));
hubTile.HubTileTransitions.Add(new SlideTransition());
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/HubTile.png", UriKind.RelativeOrAbsolute)) };
hubTile.SecondaryContent = image;
//Setting tile press duration and scale depth
hubTile.TilePressDuration = TimeSpan.FromSeconds(2.0);
hubTile.ScaleDepth = 2;
grid.Children.Add(hubTile);
```

Note: The tile press animation occurs only if the [OverrideDefaultStates](#) property is said to be **false**.

Transitions

The Hub Tile control supports various transitions which causes the tile to change from primary tile content to secondary tile content. The transitions can be set to the Hub Tile by using the [HubTileTransitions](#) property. These transitions happens based on the specified interval set via [Interval](#) property of the Hub Tile. The control provides the following built-in transitions:

1. Slide transition
2. Fade transition

XML

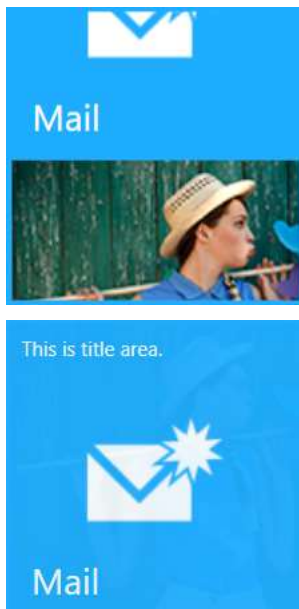
```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:shared="clr-
namespace:Syncfusion.Windows.Controls;assembly=Syncfusion.SfShared.Wpf"
x:Class="WpfApplication1.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid>
<!-- SfHubTile -->
<syncfusion:SfHubTile x:Name="hubTile" ImageSource="/Assets/New Mail.png"
Title="This is title area." Header="Mail" Foreground="White"
Interval="00:00:03">
<syncfusion:SfHubTile.HubTileTransitions>
<!--For SlideTransition use this code.-->
<shared:SlideTransition/>
<!--For FadeTransition use this code.-->
<shared:FadeTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
<!-- For setting secondary content -->
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
</syncfusion:SfHubTile>
</Grid>
</Window>
```

C#

```

SfHubTile hubTile = new SfHubTile();
//Setting header
hubTile.Header = "HubTile";
hubTile.Foreground = Brushes.White;
//Setting title
hubTile.Title = "This is title area.";
//Setting image
hubTile.ImageSource = new BitmapImage(new Uri(@"Assets/New
Mail.png",UriKind.RelativeOrAbsolute));
//For setting secondary content
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/HubTile.png", UriKind.RelativeOrAbsolute)) };
hubTile.SecondaryContent = image;
//For Slide transition use this code.
hubTile.HubTileTransitions.Add(new SlideTransition());
//For Fade transition use this code.
hubTile.HubTileTransitions.Add(new FadeTransition());
//Setting transition interval
hubTile.Interval = TimeSpan.FromSeconds(3.0);
grid.Children.Add(hubTile);

```



Note: **Syncfusion.SfShared.Wpf** assembly should be included in XAML or import using **Syncfusion.Windows.Controls**; namespace at code behind to support built-in transitions.

Notifications on transitions

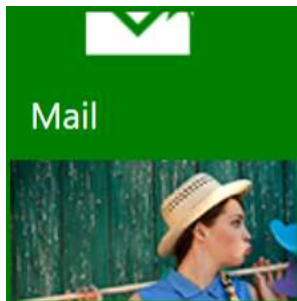
Hub Tile transitions occurs repeatedly at each specified interval. [HubTileTransitionCompleted](#) event rises on each transition completion. Below example demonstrates the working of the event in the Hub Tile.

XML

```
<syncfusion:SfHubTile x:Name="hubTile" Header="Mail" Foreground="White"
Title="This is title area." ImageSource="/Assets/New mail.png"
Interval="00:00:03" >
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill"/>
</syncfusion:SfHubTile.SecondaryContent>
<i:Interaction.Triggers>
<i:EventTrigger EventName="HubTileTransitionCompleted">
<local:HubTileTransitionCompleted />
</i:EventTrigger>
</i:Interaction.Triggers>
</syncfusion:SfHubTile>
```

C#

```
public class HubTileTransitionCompleted : TargetedTriggerAction<SfHubTile>
{
    protected override void Invoke(object parameter)
    {
        var hubTile = this.AssociatedObject as SfHubTile;
        MainWindow window = VisualUtils.FindAncestor(hubTile, typeof(MainWindow)) as
        MainWindow;
        if ((window != null) && (hubTile != null))
        {
            window.hubTile.Background=Brushes.Green;
        }
    }
}
```



Note: View [sample](#) in GitHub. The sample covers topics such as setting header, image, title, secondary content and transitions in Hub Tile control.

Grouping

Several tiles can be grouped using the [GroupName](#) property of hub tile control. The group name will be used when the entire group of tiles needs to be freeze/unfreeze.

XML

```
<Grid x:Name="grid">
<WrapPanel Orientation="Horizontal">
<!-- SfHubTile 1-->
```

```

<syncfusion:SfHubTile x:Name="hubTileOne" GroupName="Applications"
ImageSource="/Assets/New Mail.png" Title="This is title area."
Foreground="White" Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
<!-- SfHubTile 2-->
<syncfusion:SfHubTile x:Name="hubTileTwo" Margin="10"
GroupName="Applications" ImageSource="/Assets/New Mail.png" Title="This is
title area." Foreground="White" Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
<!-- SfHubTile 3-->
<syncfusion:SfHubTile x:Name="hubTileThree" GroupName="Applications"
ImageSource="/Assets/New Mail.png" Title="This is title area."
Foreground="White" Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
</WrapPanel>
</Grid>

```

C#

```

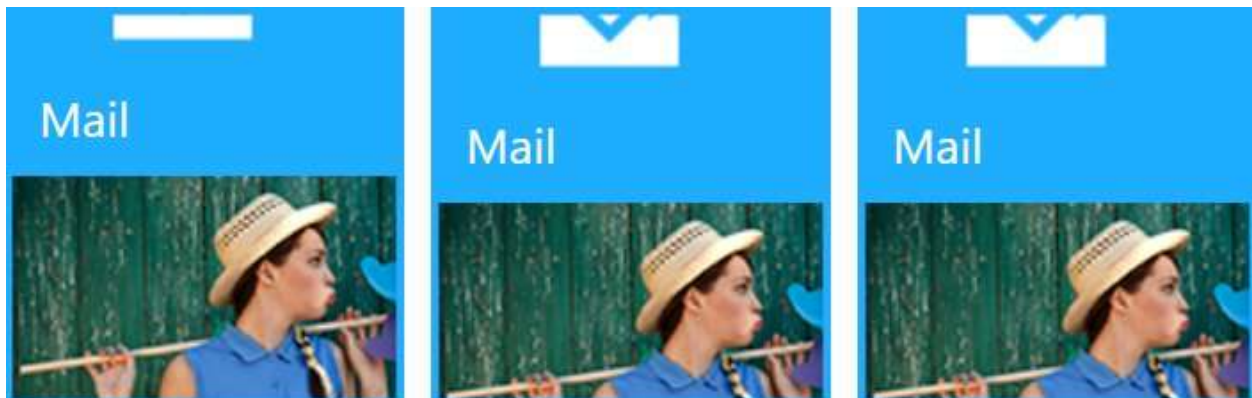
WrapPanel wrapPanel = new WrapPanel();    wrapPanel.Orientation =
Orientation.Horizontal;
grid.Children.Add(wrapPanel);
//SfHubTile 1
SfHubTile hubTileOne= new SfHubTile();
hubTileOne.Header = "Mail";
hubTileOne.Title = "This is title area.";
hubTileOne.Foreground = Brushes.White;
hubTileOne.HubTileTransitions.Add(new SlideTransition());
hubTileOne.Interval = TimeSpan.FromSeconds(3.0);
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/HubTile.png", UriKind.RelativeOrAbsolute)) };
hubTileOne.SecondaryContent = image;
//SfHubTile 2
SfHubTile hubTileTwo= new SfHubTile();
hubTileTwo.Header = "Mail";
hubTileTwo.Title = "This is title area.";
hubTileTwo.Foreground = Brushes.White;
hubTileTwo.HubTileTransitions.Add(new SlideTransition());

```

```

hubTileTwo.Interval = TimeSpan.FromSeconds(3.0);
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/HubTile.png", UriKind.RelativeOrAbsolute)) };
hubTileTwo.SecondaryContent = image;
Thickness margin = hubTileTwo.Margin;
margin.Left = 10;
hubTileTwo.Margin = margin;
//SfHubTile 3
SfHubTile hubTileThree= new SfHubTile();
hubTileThree.Header = "Mail";
hubTileThree.Title = "This is title area.";
hubTileThree.Foreground = Brushes.White;
hubTileThree.HubTileTransitions.Add(new SlideTransition());
hubTileThree.Interval = TimeSpan.FromSeconds(3.0);
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/HubTile.png", UriKind.RelativeOrAbsolute)) };
hubTileThree.SecondaryContent = image;
//Setting group name
hubTileOne.GroupName = "Applications";
hubTileTwo.GroupName = "Applications";
hubTileThree.GroupName = "Applications";
wrapPanel.Children.Add(hubTileOne);
wrapPanel.Children.Add(hubTileTwo);
wrapPanel.Children.Add(hubTileThree);

```



Grouping via DataBinding

Grouping several hub tiles can also be done by populating hub tile into a collection by providing proper binding to the collection. Below example demonstrates how to populate a group of hub tiles inside a listview in MVVM pattern.

- Create a new WPF project and add a **Model** class specifying the elements of the Hub Tile.

C#

```

public class Model
{
    private string header;
    private string imageSource;
    private TimeSpan interval;
    public string Header
    {

```

```

get { return header; }
set { header = value; }
}
public string ImageSource
{
    get { return imageSource; }
    set { imageSource = value; }
}
public TimeSpan Interval
{
    get { return interval; }
    set { interval = value; }
}
}

```

- Create a **ViewModel** class where the collection has been declared and populate the items into it.

C#

```

public class ViewModel
{
    private ObservableCollection<Model> items;
    public ViewModel()
    {
        Items = new ObservableCollection<Model>();
        PopulateItems();
    }
    public ObservableCollection<Model> Items
    {
        get { return items; }
        set { items = value; }
    }
    private void PopulateItems()
    {
        Model hub1 = new Model { Header = "Mail", ImageSource = @"/Assets/New
Mail.png", Interval = TimeSpan.FromSeconds(3.0) };
        Model hub2 = new Model { Header = "Word", ImageSource = @"/Assets/Word.png",
Interval = TimeSpan.FromSeconds(3.0) };
        Model hub3 = new Model { Header = "Paint", ImageSource = @"/Assets/Painting
Brush.png", Interval = TimeSpan.FromSeconds(3.0) };
        Model hub4 = new Model { Header = "NotePad", ImageSource =
@"/Assets/Note.png", Interval = TimeSpan.FromSeconds(3.0) };
        Model hub5 = new Model { Header = "Microsoft Store ", ImageSource =
@"/Assets/Store.png", Interval = TimeSpan.FromSeconds(3.0) };
        Model hub6 = new Model { Header = "Clock", ImageSource =
@"/Assets/Clock.png", Interval = TimeSpan.FromSeconds(3.0) };
        Model hub7 = new Model { Header = "Calculator", ImageSource =
@"/Assets/Calculator.png", Interval = TimeSpan.FromSeconds(3.0) };
        Model hub8 = new Model { Header = "Excel", ImageSource =
@"/Assets/Excel.png", Interval = TimeSpan.FromSeconds(3.0) };
        Model hub9 = new Model { Header = "Microsoft Edge", ImageSource =
@"/Assets/MicroSoft Edge.png", Interval = TimeSpan.FromSeconds(3.0) };
        Items.Add(hub1);
        Items.Add(hub2);
    }
}

```

```

Items.Add(hub3);
Items.Add(hub4);
Items.Add(hub5);
Items.Add(hub6);
Items.Add(hub7);
Items.Add(hub8);
Items.Add(hub9);
}
}

```

- In XAML, bind the collection to the ListView control and use ItemTemplate to populate Hub Tile control into it.

XML

```

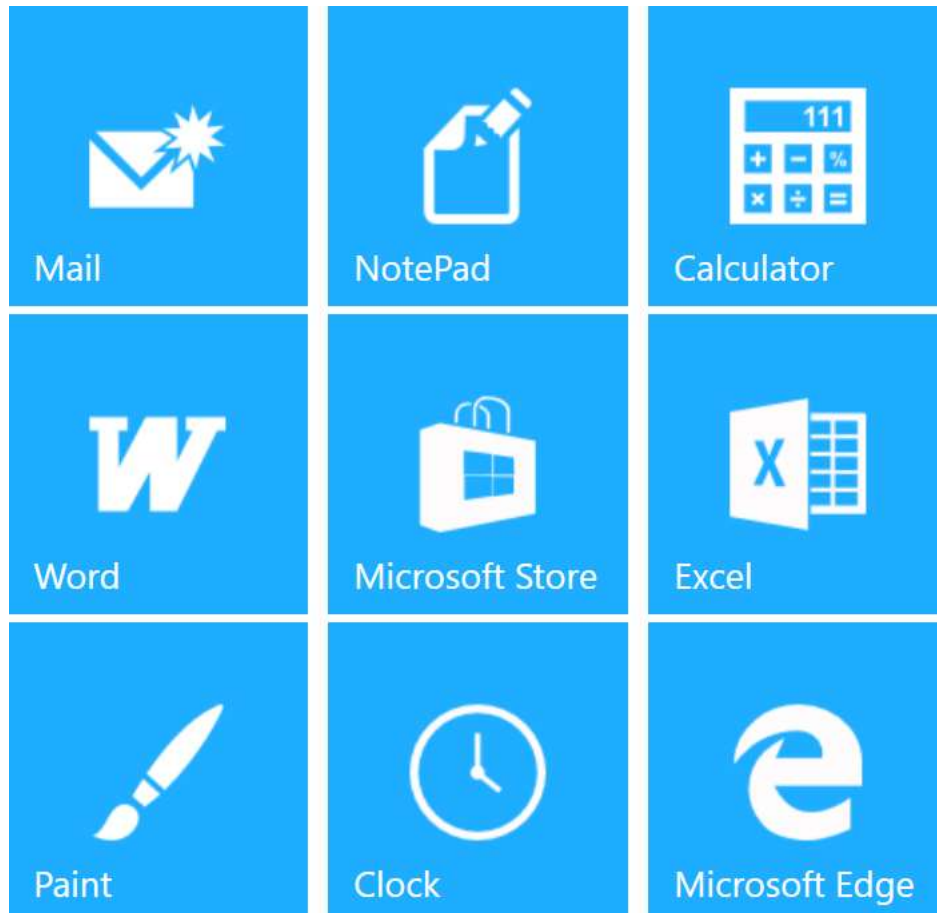
<Window x:Class="HubTile_Data_Binding.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:HubTile_Data_Binding"
xmlns:shared="clr-
namespace:Syncfusion.Windows.Controls;assembly=Syncfusion.SfShared.Wpf"
Title="Hub Tile Data Binding Demo"
ResizeMode="NoResize"
Width="700"
Height="600"
Icon="App.ico"
WindowStartupLocation="CenterScreen">
<Window.Resources>
<local:ImageConverter x:Key="Image" />
</Window.Resources>
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid HorizontalAlignment="Center">
<ListView
x:Name="List"
ItemsSource="{Binding Items}"
ScrollViewer.VerticalScrollBarVisibility="Disabled" Margin="10"
BorderBrush="White" >
<ListView.ItemTemplate>
<DataTemplate>
<syncfusion:SfHubTile
Title="{Binding Title}"
Foreground="White"
Header="{Binding Header}"
ImageSource="{Binding ImageSource, Converter={StaticResource Image}}"
Interval="{Binding Interval}">
</syncfusion:SfHubTile>
</DataTemplate>
</ListView.ItemTemplate>
<ListBox.ItemsPanel>
<ItemsPanelTemplate>
<WrapPanel Margin="10" Orientation="Vertical" />
</ItemsPanelTemplate>

```

```

</ListBox.ItemsPanel>
</ListView>
</Grid>
</Window>

```



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

Freezing/Unfreezing

Freezing provides support to stop animating the tile contents. Unfreezing provides support to keep the tile content animated. By the following two ways freezing/unfreezing can be set to the Hub Tile:

Freezing/unfreezing via property

The tile can be frozen by setting [IsFrozen](#) property to be **true**.

XML

```

<Grid x:Name="grid">
  <!-- SfHubTile -->
  <syncfusion:SfHubTile x:Name="hubTile" Header="Mail" Foreground="White"
    IsFrozen="True" Title="This is title area." ImageSource="/Assets/New
    Mail.png" Interval="00:00:03" HorizontalAlignment="Center"
    VerticalAlignment="Center" >
    <!-- For setting secondary content -->
    <syncfusion:SfHubTile.SecondaryContent>
    <Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>

```



```

</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<!--For SlideTransition-->
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
</Grid>

```

C#

```

SfHubTile hubTile= new SfHubTile();
hubTile.Header = "Mail";
hubTile.Title = "Title area.";
hubTile.Foreground = Brushes.White;
hubTile.HubTileTransitions.Add(new SlideTransition());
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/HubTile.png", UriKind.RelativeOrAbsolute)) };
hubTile.SecondaryContent = image;
hubTile.Interval=TimeSpan.FromSeconds(3.0);
//Setting freeze property
hubTile.IsFrozen = true;
grid.Children.Add(hubTile);

```



The tile can be unfrozen by setting [IsFrozen](#) property to **false**.

XML

```

<Grid x:Name="grid">
<!-- SfHubTile -->
<syncfusion:SfHubTile x:Name="hubTile" Header="Mail" Foreground="White"
IsFrozen="False" Title="This is title area." ImageSource="Assets/New
Mail.png" HorizontalAlignment="Center" VerticalAlignment="Center" >
<!-- For setting secondary content -->
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
</Grid>

```

C#

```

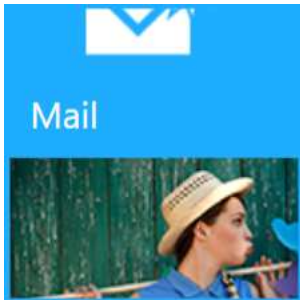
SfHubTile hubTile= new SfHubTile();

```

```

hubTile.Header = "Mail";
hubTile.Title = "This is title area.";
hubTile.Foreground = Brushes.White;
hubTile.HubTileTransitions.Add(new SlideTransition());
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/HubTile.png", UriKind.RelativeOrAbsolute)) };
hubTile.SecondaryContent = image;
hubTile.Interval = TimeSpan.FromSeconds(3.0);
//Setting Unfreeze property
hubTile.IsFrozen = false;
grid.Children.Add(hubTile);

```



Freezing/unfreezing via methods

The [HubTileService](#) class provides helper methods such as [Freeze](#) and [UnFreeze](#) to freeze and unfreeze the animation by passing a Hub Tile instance or [GroupName](#) as an argument.

- Add required **System.Windows.Interactivity** assembly reference in application.
- Import schema for interactivity <http://schemas.microsoft.com/expression/2010/interactivity> in XAML or using **System.Windows.Interactivity** namespace in C#.

Note: The [HubTileService](#) class allows to set the freeze/unfreeze state of the tile after the tiles are loaded.

A single tile or a group of tiles can be frozen by using [Freeze](#) method.

XML

```

<Window x:Class="HubTile_Grouping.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:HubTile_Grouping"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:shared="clr-namespace:Syncfusion.Windows.Controls;assembly=Syncfusion.SfShared.Wpf"
xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<WrapPanel>
<!-- SfHubTile 1-->

```

```

<syncfusion:SfHubTile x:Name="hubTileOne" Foreground="White"
GroupName="Applications" ImageSource="/Assets/New Mail.png" Title="This is
title area." Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
<!-- SfHubTile 2-->
<syncfusion:SfHubTile x:Name="hubTileTwo" Foreground="White"
GroupName="Application" Margin="10" ImageSource="/Assets/New Mail.png"
Title="This is title area." Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
<!--SfHubTile 3-->
<syncfusion:SfHubTile x:Name="hubTileThree"
Foreground="White"GroupName="Applications" ImageSource="/Assets/New
Mail.png" Title="This is title area." Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<local:FreezeTiles/>
</i:EventTrigger>
</i:Interaction.Triggers>
</syncfusion:SfHubTile>
</WrapPanel>
</Grid>
</Window>

```

C#

```

using Syncfusion.Windows.Controls.Notification;
using Syncfusion.Windows.Shared;
using System.Windows.Interactivity;
namespace HubTile_Grouping
{
public class FreezeTiles : TargetedTriggerAction<SfHubTile>
{
protected override void Invoke(object parameter)
{
var hubTile = this.AssociatedObject as SfHubTile;
MainWindow window = VisualUtils.FindAncestor(hubTile, typeof(MainWindow)) as
MainWindow;
if (window != null && hubTile != null)

```

```
{
  //For a single tile
  HubTileService.Freeze(window.hubTileOne);
  //For group of tiles
  HubTileService.Freeze("Applications");
}
}
}
```



Single tile



Group of tiles

A single tile or a group of tiles can be unfrozen by using [UnFreeze](#) method.

XML

```
<Window x:Class="HubTile_Grouping.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:HubTile_Grouping"
  xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
  xmlns:shared="clr-namespace:Syncfusion.Windows.Controls;assembly=Syncfusion.SfShared.Wpf"
  xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
  mc:Ignorable="d"
  Title="MainWindow" Height="450" Width="800">
  <Grid>
    <WrapPanel>
      <!-- SfHubTile 1-->
      <syncfusion:SfHubTile x:Name="hubTileOne" Foreground="White"
        GroupName="Applications" ImageSource="/Assets/New Mail.png" Title="This is
        title area." Interval="00:00:03" Header="Mail">
```

```

<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
<!-- SfHubTile 2-->
<syncfusion:SfHubTile x:Name="hubTileTwo" Foreground="White"
GroupName="Applications" Margin="10" ImageSource="/Assets/New Mail.png"
Title="This is title area." Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
</syncfusion:SfHubTile>
<!--SfHubTile 3-->
<syncfusion:SfHubTile x:Name="hubTileThree" Foreground="White"
GroupName="Applications" ImageSource="/Assets/New Mail.png" Title="This is
title area." Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="UniformToFill" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<local:UnfreezeTiles/>
</i:EventTrigger>
</i:Interaction.Triggers>
</syncfusion:SfHubTile>
</WrapPanel>
</Grid>
</Window>

```

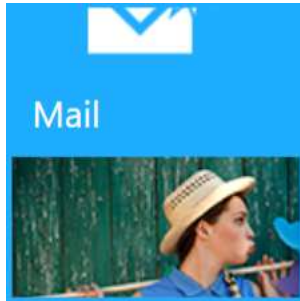
C#

```

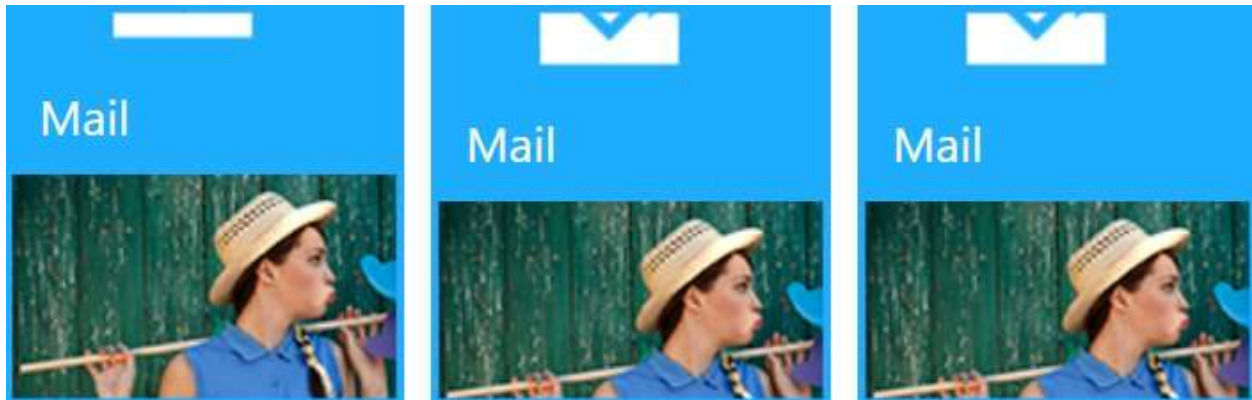
using Syncfusion.Windows.Controls.Notification;
using Syncfusion.Windows.Shared;
using System.Windows.Interactivity;
namespace HubTile_Grouping
{
public class UnfreezeTiles : TargetedTriggerAction<SfHubTile>
{
protected override void Invoke(object parameter)
{
var hubTile = this.AssociatedObject as SfHubTile;
MainWindow window = VisualUtils.FindAncestor(hubTile, typeof(MainWindow)) as
MainWindow;
if (window != null && hubTile != null)
{
//For a single tile use this code
HubTileService.UnFreeze(window.hubTileOne);
}
}
}

```

```
//For group of tiles use this code
HubTileService.UnFreeze ("Applications");
}
}
}
```



Single tile



Group of tiles

Notifications

Once the tile is pressed, it is notified by the click event and the command property of the Hub Tile.

Event

The [Click](#) event rises whenever the tile is pressed.

XML

```
<syncfusion:SfHubTile x:Name="hubTile" Interval="00:00:03" Header="Mail"
Foreground="White" Title="This is title area." ImageSource="/Assets/New
Mail.png">
  <i:Interaction.Triggers>
    <i:EventTrigger EventName="Click">
      <local:ClickEvent/>
    </i:EventTrigger>
  </i:Interaction.Triggers>
</syncfusion:SfHubTile>
```

C#

```
public class ClickEvent : TargetedTriggerAction<SfHubTile>
```

```

{
    protected override void Invoke(object parameter)
    {
        var hubTile = this.AssociatedObject as SfHubTile;
        MainWindow window = VisualUtils.FindAncestor(hubTile, typeof(MainWindow)) as
        MainWindow;
        if ((window != null) && (hubTile != null))
        {
            MessageBox.Show("Hub Tile has been clicked");
        }
    }
}

```

Command binding

Command specifies the operation to be performed when the tile is pressed. [Command](#) and [CommandParameter](#) are used instead of click event in MVVM pattern.

XML

```

<Window x:Class="Hub.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:Hub"
    xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
    xmlns:shared="clr-
    namespace:Syncfusion.Windows.Controls;assembly=Syncfusion.SfShared.Wpf"
    mc:Ignorable="d"
    Title="MainWindow" Height="450" Width="800">
    <Window.DataContext>
    <local:Viewmodel/>
    </Window.DataContext>
    <Grid x:Name="grid">
    <syncfusion:SfHubTile x:Name="hubtile" Command="{Binding HubTileCommand}"
    CommandParameter="{Binding ElementName=hubtile}" Interval="00:00:03"
    Header="Mail" Foreground="White" Title="This is title area."
    ImageSource="/Assets/New Mail.png"/>
    </Grid>
    </Window>

```

C#

```

public class Viewmodel
{
    private ICommand hubTileCommand;
    public ICommand HubTileCommand
    {
        get
        {
            return hubTileCommand ?? (hubTileCommand = new Command(true,
            ()=>MyAction("HubTileCommand")));
        }
    }
    private void MyAction(string parameter)

```

```

{
    if (parameter.Equals("HubTileCommand"))
    {
        string message = string.Format("Hub Tile Command executed");
        MessageBox.Show(message);
    }
}

public class Command : ICommand
{
    private bool _canExecute;
    private Action _execute;
    public Command(bool CanExecute, Action Execute)
    {
        _canExecute = CanExecute;
        _execute = Execute;
    }
    public event EventHandler CanExecuteChanged;
    public bool CanExecute(object parameter)
    {
        return _canExecute;
    }
    public void Execute(object parameter)
    {
        _execute();
    }
}

```

Appearance and styling

Customizing header

Header of the tile can be customized either through [HeaderStyle](#) or [HeaderTemplate](#) as shown below.

[HeaderStyle](#) is used to customize the header of the tile by setting its appropriate properties.

[HeaderTemplate](#) is used to customize the visual appearance of header by adding user-defined template.

XML

```

<syncfusion:SfHubTile x:Name="hubtile" ImageSource="/Assets/New Mail.png"
Title="This is title area." Foreground="White" Header="Mail">
    <!--For setting header style-->
    <syncfusion:SfHubTile.HeaderStyle>
        <Style TargetType="ContentControl">
            <Setter Property="VerticalAlignment" Value="Bottom"/>
            <Setter Property="FontSize" Value="18"/>
            <Setter Property="FontFamily" Value="ALGERIAN"/>
        </Style>
    </syncfusion:SfHubTile.HeaderStyle>
</syncfusion:SfHubTile>

```


**XML**

```
<syncfusion:SfHubTile x:Name="hubTile" ImageSource="/Assets/New Mail.png"
Title="This is a title area." Foreground="White" Header="Mail">
<!--For setting header template-->
<syncfusion:SfHubTile.HeaderTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<Image Source="/Assets/syncfusion.png" HorizontalAlignment="Left"
Stretch="None"/>
<TextBlock Text="SYNCFUSION" Foreground="White" Grid.Column="1"
FontSize="17" />
</Grid>
</DataTemplate>
</syncfusion:SfHubTile.HeaderTemplate>
</syncfusion:SfHubTile>
```

*Customizing title*

The [TitleStyle](#) property is used to customize the title of the tile by setting its appropriate properties.

XML

```
<!-- SfHubTile -->
<syncfusion:SfHubTile x:Name="hubTile" ImageSource="/Assets/New Mail.png"
Title="This is a title area." Header="Mail" Foreground="White">
<!-- For setting title style -->
<syncfusion:SfHubTile.TitleStyle>
<Style TargetType="ContentControl">
<Setter Property="Foreground" Value="White"/>
<Setter Property="FontSize" Value="15"/>
<Setter Property="ContentTemplate">
<Setter.Value>
```

```

<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<Image Source="/Assets/syncfusion.png" HorizontalAlignment="Left"
VerticalAlignment="Top" Stretch="None"/>
<TextBlock Text="SYNCFUSION" FontSize="17" Grid.Column="1" />
</Grid>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:SfHubTile.TitleStyle>
</syncfusion:SfHubTile>

```



Customizing secondary content

By the following two ways the secondary content of the Hub Tile can be customized:

Customizing secondary content via property

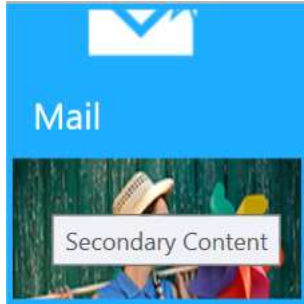
The secondary content of the tile can be customized by using the appropriate [SecondaryContent](#) property like below.

XML

```

<!-- SfHubTile -->
<syncfusion:SfHubTile x:Name="hubTile" Foreground="White"
ImageSource="/Assets/New Mail.png" Title="This is title area."
Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
<syncfusion:SfHubTile.SecondaryContent>
<Image Source="/Assets/HubTile.png" Stretch="Fill" ToolTip="Secondary
Content" Margin="-1"/>
</syncfusion:SfHubTile.SecondaryContent>
</syncfusion:SfHubTile>

```



Customizing secondary content via template

The secondary content of the tile can be customized by using the [SecondaryContentTemplate](#) property like below.

XML

```
<!-- SfHubTile -->
<syncfusion:SfHubTile x:Name="hubTile" ImageSource="/Assets/New Mail.png"
Title="This is title area." Foreground="White" Height="200"
Interval="00:00:03" Header="Mail">
<syncfusion:SfHubTile.HubTileTransitions>
<shared:SlideTransition/>
</syncfusion:SfHubTile.HubTileTransitions>
<!--setting secondary content template-->
<syncfusion:SfHubTile.SecondaryContentTemplate>
<DataTemplate>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Image Source="/Assets/HubTile.png" />
<CheckBox x:Name="checkbox" Grid.Row="1" Foreground="White" Content="Freeze"
IsChecked="{Binding ElementName=hubtile,Path=IsFrozen}" />
<TextBlock Grid.Row="2" Text="This is the secondary content of the tile
displayed at each transition." Foreground="White" FontSize="12"
TextWrapping="Wrap" FontStyle="Italic"/>
</Grid>
</DataTemplate>
</syncfusion:SfHubTile.SecondaryContentTemplate>
</syncfusion:SfHubTile>
```



Pulsing Tile in WPF Tile Control

The Pulsing Tile control (extended from [HubTileBase](#) class) allows to create a tile similar to music and video tiles in Windows Phone. The content zooms in/out randomly with random movement along the X-axis and Y-axis. This section explains about the supporting features of Pulsing Tile control.

Setting header content

Header can act as the name of the tile, that is placed at the bottom explaining its purpose. The content of the header can be an image, a text or a control, etc. The header can be set to the tile by using the [Header](#) property.

XML

```
<Grid x:Name=grid>
  <!--For setting header as text use this code.-->
  <syncfusion:SfPulsingTile x:Name="pulsingTile" Foreground="White"
  Header="Music"/>
  <!--For setting header as an image use this code.-->
  <syncfusion:SfPulsingTile x:Name="pulsingTile" >
    <syncfusion:SfPulsingTile.Header>
      <Image Source="/Assets/syncfusion.png" Stretch="None" />
    </syncfusion:SfPulsingTile.Header>
  </syncfusion:SfPulsingTile>
```

```

<!--For setting header as control use this code.-->
<syncfusion:SfPulsingTile x:Name="pulsingTile" >
<syncfusion:SfPulsingTile.Header>
<TextBlock Text="SYNCFUSION" Foreground="White" FontSize="13" />
</syncfusion:SfPulsingTile.Header>
</syncfusion:SfPulsingTile>
</Grid>

```

C#

```

//For setting header on as text use this code.
SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.Header = "Music";
pulsingTile.Foreground = Brushes.White;
//For setting header as image use this code.
SfPulsingTile pulsingTile = new SfPulsingTile();
Image image = new Image(){Source = new BitmapImage(new
Uri(@"Assets/syncfusion.png",UriKind.RelativeOrAbsolute))};
pulsingTile.Header = image;
//For setting header as control use this code.
SfPulsingTile pulsingTile = new SfPulsingTile();
TextBlock textblock = new TextBlock(){Text = "SYNCFUSION", Foreground =
Brushes.White, FontSize = 13} ;
pulsingTile.Header = textblock;
grid.Children.Add(pulsingTile);

```



Setting title content

Title can be used to display updates and notifications in a tile. The content can be an image, a text or a control, etc. The title can be set to the tile by using the [Title](#) property.

XML

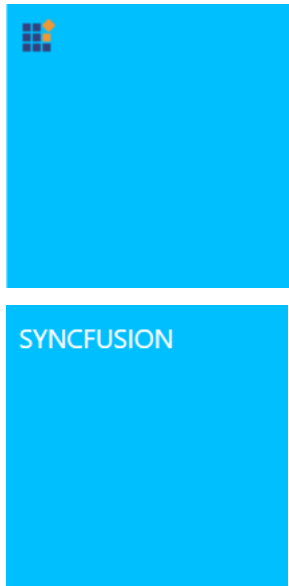
```
<Grid x:Name="grid">
  <!--For setting title as text use this code.-->
  <syncfusion:SfPulsingTile x:Name="pulsingTile" Title="Now Playing - Song
  Name" Foreground="White"/>
  <!--For setting title as an image use this code.-->
  <syncfusion:SfPulsingTile x:Name="pulsingTile">
    <syncfusion:SfPulsingTile.Title>
      <Image Source="/Assets/syncfusion.png" Stretch="None"
      HorizontalAlignment="Left"/>
    </syncfusion:SfPulsingTile.Title>
  </syncfusion:SfPulsingTile>
  <!--For setting title as a control use this code.-->
  <syncfusion:SfPulsingTile x:Name="pulsingTile">
    <syncfusion:SfPulsingTile.Title>
      <TextBlock Text="SYNCFUSION" Foreground="White" />
    </syncfusion:SfPulsingTile.Title>
  </syncfusion:SfPulsingTile>
</Grid>
```

C#

```
//Setting title for Pulsing Tile.
SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.Title = "Now Playing - Song Name";
pulsingTile.Foreground = Brushes.White;
//For setting title as image use this code.
SfPulsingTile pulsingTile = new SfPulsingTile();
Image image = new Image() {Source = new BitmapImage(new
Uri(@"Assets/syncfusion.png", UriKind.RelativeOrAbsolute)), Stretch=Stretch.N
one, HorizontalAlignment=HorizontalAlignment.Left};
pulsingTile.Title = image;
// For setting title as control use this code.
SfPulsingTile pulsingTile = new SfPulsingTile();
Textblock textblock = new TextBlock() {Text = "SYNCFUSION", Foreground =
Brushes.White, FontSize = 13} ;
pulsingTile.Title = textblock;
grid.Children.Add(pulsingTile);
```



Now Playing - Song Name



Setting image

The image acts as a pictorial representation of the purpose of tile control. The image can be set to the tile by setting image path to the [ImageSource](#) property.

XML

```
<Grid x:Name="grid">
  <!--SfPulsingTile-->
  <syncfusion:SfPulsingTile x:Name="pulsingtile"
    ImageSource="/Assets/PulsingTile.jpg"/>
</Grid>
```

C#

```
//Setting image for Pulsing Tile
SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.ImageSource = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg", UriKind.RelativeOrAbsolute));
grid.Children.Add(pulsingTile);
```



Animations

Scaling animation

Scaling animation causes the content of the tile to zoom in/out and it is achieved by setting the [PulseScale](#) property. The property specifies the translation range in the x-axis and y-axis while scaling the content.

XML

```
<!-- SfPulsingTile-->
<Grid x:Name="grid">
<syncfusion:SfPulsingTile x:Name="pulsingTile" Width="200" Height="200"
PulseScale="2" Header="Music" Title="This is title area."
Foreground="White">
<Image Source="/Assets/PulsingTile.jpg" Stretch="None"
VerticalAlignment="Center" HorizontalAlignment="Center" />
</syncfusion:SfPulsingTile>
</Grid>
```

C#

```
SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.Header = "Music";
pulsingTile.Title = "This is title area.";
pulsingTile.Foreground = Brushes.White;
Image image = new Image() {Source = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg", UriKind.RelativeOrAbsolute))};
pulsingTile.Content = image;
// Setting pulse scale
pulsingTile.PulseScale = 2;
grid.Children.Add(pulsingTile);
```

The time required to complete a single scaling animation is achieved by setting the [PulseDuration](#) property.

XML

```
<Grid x:Name="grid">
<!-- SfPulsingTile -->
<syncfusion:SfPulsingTile x:Name="pulsingTile" PulseDuration="00:00:03"
Header="Music" Title="This is title area." Foreground="White" >
<Image Source="/Assets/PulsingTile.jpg" Stretch="None"
VerticalAlignment="Center" HorizontalAlignment="Center" />
</syncfusion:SfPulsingTile>
</Grid>
```

C#

```
SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.Header = "Music";
pulsingTile.Title = "This is title area.";
pulsingTile.Foreground = Brushes.White;
Image image = new Image() {Source = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg", UriKind.RelativeOrAbsolute))};
pulsingTile.Content = image;
//Setting pulse duration
```



```
pulsingTile.PulseDuration = TimeSpan.FromSeconds(3.0);
grid.Children.Add(pulsingTile);
```

Before applying pulse scale to the Pulsing Tile:



After applying pulse scale to the Pulsing Tile :



Press animation

The tile press animation takes place when the center of the tile is pressed. The tile press animation causes the entire tile to be zoomed in/out at specified interval. The tile press animation can be set by using properties such as [ScaleDepth](#) and [TilePressDuration](#). The [ScaleDepth](#) is used to customize the depth of scaling effect while pressing the center of the tile. The [TilePressDuration](#) is used to determine the time taken for the single tile press animation.

XML

```
<Grid x:Name="grid">
  <syncfusion:SfPulsingTile Header="Music" Title="This is title area."
    Foreground="White" PulseScale="3" PulseDuration="00:00:03"
    TilePressDuration="00:00:03" ScaleDepth="2">
    <Image Source="/Assets/PulsingTile.jpg" Stretch="None"/>
  </syncfusion:SfPulsingTile>
</Grid>
```

C#

```
SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.Header = "Music";
pulsingTile.Title = "This is title area.";
pulsingTile.Foreground = Brushes.White;
pulsingTile.PulseScale = 3;
pulsingTile.PulseDuration = TimeSpan.FromSeconds(3.0);
//setting tile press duration and scale depth
pulsingTile.TilePressDuration = TimeSpan.FromSeconds(3.0);
pulsingTile.ScaleDepth = 2;
grid.Children.Add(pulsingTile);
```

Note: The tile press animation occurs only if the [OverrideDefaultStates](#) property is said to be **false**.

Translations

Horizontal translation

Horizontal translation allows the content of the tile to move from left to right along x-axis. The Pulsing Tile provides support for horizontal translation by using [RadiusX](#) property. The property specifies the translation range of the content along the x-axis.

XML

```
<Grid x:Name="grid">
  <!-- SfPulsingTile -->
  <syncfusion:SfPulsingTile x:Name="pulsingTile" RadiusX="100" Header="Music"
    Title="This is title area." Foreground="White">
    <Image Source="/Assets/PulsingTile.jpg" VerticalAlignment="Center"
      HorizontalAlignment="Center" />
  </syncfusion:SfPulsingTile>
</Grid>
```

C#

```
SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.Header = "Music";
pulsingTile.Title = "This is title area.";
pulsingTile.Foreground = Brushes.White;
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg", UriKind.RelativeOrAbsolute)) };
pulsingTile.Content = image;
//Setting radiusX
pulsingTile.RadiusX = 100;
grid.Children.Add(pulsingTile);
```



Vertical translation

Vertical translation allows the content of the tile to move up and down along the y-axis. The Pulsing Tile provides support for vertical translation by using [RadiusY](#) property. The property specifies the translation range of the content along the y-axis.

XML

```
<Grid x:Name="grid">
  <!-- SfPulsingTile -->
  <syncfusion:SfPulsingTile x:Name="pulsingTile" RadiusY="100" Header="Music"
    Title="This is title area." Foreground="White">
    <Image Source="/Assets/PulsingTile.jpg" VerticalAlignment="Center"
      HorizontalAlignment="Center" />
  </syncfusion:SfPulsingTile>
</Grid>
```

C#

```
SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.Header = "Music";
pulsingTile.Title = "This is title area.";
pulsingTile.Foreground = Brushes.White;
Image image = new Image() { Source = new BitmapImage(new
  Uri(@"Assets/PulsingTile.jpg", UriKind.RelativeOrAbsolute)) };
pulsingTile.Content = image;
//Setting radiusY
pulsingTile.RadiusY = 100;
grid.Children.Add(pulsingTile);
```

*Duration of horizontal and vertical translation*

The time taken for translating the content along the x-axis and y-axis is achieved by setting the [TranslateDuration](#) property.

XML

```
<Grid x:Name="grid">
  <syncfusion:SfPulsingTile x:Name="pulsingTile" RadiusX=100 Title="This is
    title area." Foreground="White" RadiusY=100 TranslateDuration="00:00:03"
    Header="Music">
    <Image Source="/Assets/PulsingTile.jpg" Stretch="UniformToFill"
      HorizontalAlignment="Center" VerticalAlignment="Center"/>
  </syncfusion:SfPulsingTile>
</Grid>
```

C#

```
SfPulsingTile pulsingTile = new SfPulsingTile();
```

```
pulsingTile.Header = "Music";
pulsingTile.Title = "This is title area.";
pulsingTile.Foreground = Brushes.White;
Image image = new Image() {Source = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg", UriKind.RelativeOrAbsolute))};
pulsingTile.Content = image;
//Setting translate duration
pulsingTile.TranslateDuration = TimeSpan.FromSeconds(3.0);
grid.Children.Add(pulsingTile);
```

Note: View [sample](#) in GitHub. The sample covers topics such as setting header, image, title, animations and translations in Pulsing Tile control.

Grouping

Several tiles can be grouped using the [GroupName](#) property of pulsing tile control. The group name will be used when the entire group of tiles needs to be freeze/unfreeze.

XML

```
<Grid x:Name="grid">
  <WrapPanel Orientation="Horizontal">
    <!-- SfPulsingTile 1-->
    <syncfusion:SfPulsingTile x:Name="pulsingTileOne" Title="This is title
area." Foreground="White" GroupName="Applications" Header="Music"
PulseScale="3" PulseDuration="00:00:03" >
      <Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
    </syncfusion:SfPulsingTile>
    <!-- SfPulsingTile 2-->
    <syncfusion:SfPulsingTile x:Name="pulsingTileTwo" Title="This is title
area." Foreground="White" GroupName="Applications" Margin="10"
PulseScale="3" PulseDuration="00:00:03" Header="Music">
      <Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
    </syncfusion:SfPulsingTile>
    <!-- SfPulsingTile 3-->
    <syncfusion:SfPulsingTile x:Name="pulsingTileThree" Title="This is title
area." Foreground="White" GroupName="Applications" PulseScale="3"
PulseDuration="00:00:03" Header="Music">
      <Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
    </syncfusion:SfPulsingTile>
  </WrapPanel>
</Grid>
```

C#

```
WrapPanel wrapPanel = new WrapPanel();
wrapPanel.Orientation = Orientation.Horizontal;
grid.Children.Add(wrapPanel);
//SfPulsingTile 1
SfPulsingTile pulsingTileOne = new SfPulsingTile();
pulsingTileOne.Header = "Music";
pulsingTileOne.Title = "This is title area.";
pulsingTileOne.Foreground = Brushes.White;
```

```

Image image = new Image(){Source = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg",UriKind.RelativeOrAbsolute))};
pulsingTileOne.Content = image;
pulsingTileOne.PulseScale = 3;
pulsingTileOne.PulseDuration = TimeSpan.FromSeconds(3.0);
//SfPulsingTile 2
SfPulsingTile pulsingTileTwo = new SfPulsingTile();
pulsingTileTwo.Header = "Music";
pulsingTileTwo.Title = "This is title area.";
pulsingTileTwo.Foreground = Brushes.White;
Image image = new Image(){Source = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg",UriKind.RelativeOrAbsolute))};
pulsingTileTwo.Content = image;
pulsingTileTwo.PulseScale = 3;
pulsingTileTwo.PulseDuration = TimeSpan.FromSeconds(3.0);
Thickness margin = pulsingTileTwo.Margin;
margin.Left = 10;
pulsingTileTwo.Margin = margin;
//SfPulsingTile 3
SfPulsingTile pulsingTileThree = new SfPulsingTile();
pulsingTileThree.Header = "Music";
pulsingTileThree.Title = "This is title area.";
pulsingTileThree.Foreground = Brushes.White;
Image image = new Image(){Source = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg",UriKind.RelativeOrAbsolute))};
pulsingTileThree.Content = image;
pulsingTileThree.PulseScale = 3;
pulsingTileThree.PulseDuration = TimeSpan.FromSeconds(3.0);
//Setting group name
pulsingTileOne.GroupName = "Applications";
pulsingTileTwo.GroupName = "Applications";
pulsingTileThree.GroupName = "Applications";
wrapPanel.Children.Add(pulsingTileOne);
wrapPanel.Children.Add(pulsingTileTwo);
wrapPanel.Children.Add(pulsingTileThree);

```



Freezing/Unfreezing

Freezing provides support to stop animating the tile contents. Unfreezing provides support to keep the tile content animated. By the following two ways freezing/unfreezing can be set to the Pulsing Tile:

Freezing/unfreezing via property

The tile can be frozen by setting [IsFrozen](#) property to be **true**.

XML

```
<Grid x:Name="grid">
```

```

<!-- SfPulsingTile -->
<syncfusion:SfPulsingTile x:Name="pulsingTile" Header="Music"
IsFrozen="True" Foreground="White" Title="This is title area."
PulseScale="3" PulseDuration="00:00:03" >
<Image Source="/Assets/PulsingTile.jpg"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
</syncfusion:SfPulsingTile>
</Grid>

```

C#

```

SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.Header = "Music";
pulsingTile.Title = "This is title area.";
pulsingTile.Foreground = Brushes.White;
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg", UriKind.RelativeOrAbsolute)) };
pulsingTile.Content = image;
pulsingTile.PulseScale = 3;
//Setting freeze property
pulsingTile.IsFrozen = true;
grid.Children.Add(pulsingTile);

```



The tile can be unfrozen by setting [IsFrozen](#) property to **false**.

XML

```

<Grid x:Name="grid">
<!-- SfPulsingTile -->
<syncfusion:SfPulsingTile x:Name="pulsingTile" Header="Music"
Foreground="white" IsFrozen="False" Title="This is title area."
PulseScale="3" PulseDuration="00:00:03" >
<Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</syncfusion:SfPulsingTile>
</Grid>

```

C#

```

SfPulsingTile pulsingTile = new SfPulsingTile();
pulsingTile.Header = "Music";
pulsingTile.Title = "This is title area.";
pulsingTile.Foreground = Brushes.White;
Image image = new Image() { Source = new BitmapImage(new
Uri(@"Assets/PulsingTile.jpg", UriKind.RelativeOrAbsolute)) };

```

```
pulsingTile.Content = image;
pulsingTile.PulseScale = 3;
//Setting unfreeze property
pulsingTile.IsFrozen = false;
grid.Children.Add(pulsingTile);
```



Freezing/unfreezing via methods

The [HubTileService](#) class provides helper methods such as [Freeze](#) and [UnFreeze](#) to freeze and unfreeze the animation by passing a Pulsing Tile instance or [GroupName](#) as an argument.

- Add required **System.Windows.Interactivity** assembly reference in application.
- Import schema for interactivity <http://schemas.microsoft.com/expression/2010/interactivity> in XAML or using **System.Windows.Interactivity** namespace in C#.

Note: The [HubTileService](#) class allows to set the freeze/unfreeze state of the tile after the tiles are loaded.

A single tile or a group of tiles can be frozen by using [Freeze](#) method.

XML

```
<Window x:Class="PulsingTile_Grouping.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:PulsingTile_Grouping"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:shared="clr-namespace:Syncfusion.Windows.Controls;assembly=Syncfusion.SfShared.Wpf"
xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<WrapPanel>
<!--SfPulsingTile 1-->
<syncfusion:SfPulsingTile x:Name="pulsingTileOne" Foreground="White"
GroupName="Applications" Header="Music" PulseScale="3" Title="This is title
area." PulseDuration="00:00:03" >
<Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</syncfusion:SfPulsingTile>
<!-- SfPulsingTile 2-->
```

```

<syncfusion:SfPulsingTile x:Name="pulsingTileTwo" Foreground="White"
GroupName="Applications" Margin="10" PulseScale="3" Title="This is title
area." PulseDuration="00:00:03" Header="Music">
<Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</syncfusion:SfPulsingTile>
<!-- SfPulsingTile 3-->
<syncfusion:SfPulsingTile x:Name="pulsingTileThree" Foreground="White"
GroupName="Applications" PulseScale="3" PulseDuration="00:00:03"
Title="This is title area." Header="Music">
<Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<local:FreezeTiles/>
</i:EventTriggers>
</i:Interaction.Trigger>
</syncfusion:SfPulsingTile>
</WrapPanel>
</Grid>
</Window>

```

C#

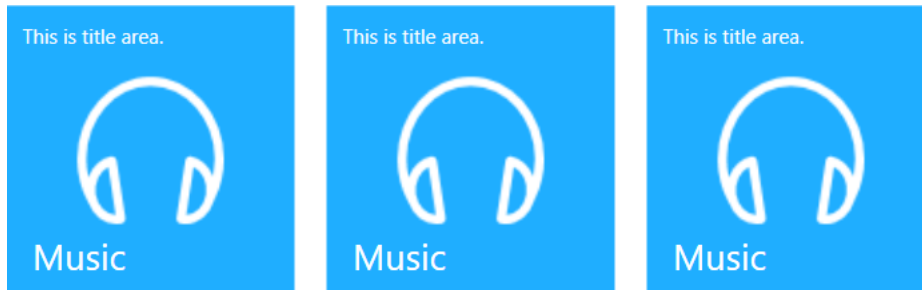
```

using Syncfusion.Windows.Controls.Notification;
using Syncfusion.Windows.Shared;
using System.Windows.Interactivity;
namespace PulsingTile_Grouping
{
    public class FreezeTiles : TargetedTriggerAction<SfPulsingTile>
    {
        protected override void Invoke(object parameter)
        {
            var pulsingTile = this.AssociatedObject as SfPulsingTile;
            MainWindow window = VisualUtils.FindAncestor(pulsingTile,
                typeof(MainWindow)) as MainWindow;
            if (window != null && pulsingTile != null)
            {
                //For a single tile use this code.
                HubTileService.Freeze(window.pulsingTileOne);
                //For Group of Tiles use this code.
                HubTileService.Freeze("Applications");
            }
        }
    }
}

```




Single tile



Group of tiles

A single tile or a group of tiles can be unfrozen by using [UnFreeze](#) method.

XML

```
<Window x:Class="PulsingTile_Grouping.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:PulsingTile_Grouping"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:shared="clr-
namespace:Syncfusion.Windows.Controls;assembly=Syncfusion.SfShared.Wpf"
xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<WrapPanel>
<!--SfPulsingTile1-->
<syncfusion:SfPulsingTile x:Name="pulsingTileOne" GroupName="Applications"
Header="Music" Foreground="White" PulseScale="3" Title="This is title area."
PulseDuration="00:00:03" >
<Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</syncfusion:SfPulsingTile>
<!-- SfPulsingTile 2-->
<syncfusion:SfPulsingTile x:Name="pulsingTileTwo" GroupName="Applications"
Margin="10" Foreground="White" PulseScale="3" Title="This is title area."
PulseDuration="00:00:03" Header="Music">
<Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</syncfusion:SfPulsingTile>
<!-- SfPulsingTile 3-->
```

```
<syncfusion:SfPulsingTile x:Name="pulsingTileThree" GroupName="Applications"
Foreground="White" PulseScale="3" PulseDuration="00:00:03" Title="This is
title area." Header="Music">
<Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<local:UnfreezeTiles/>
</i:EventTrigger>
</i:Interaction.Triggers>
</syncfusion:SfPulsingTile>
</WrapPanel>
</Grid>
</Window>
```

C#

```
using Syncfusion.Windows.Controls.Notification;
using Syncfusion.Windows.Shared;
using System.Windows.Interactivity;
namespace PulsingTile_Grouping
{
    public class UnfreezeTiles : TargetedTriggerAction<SfPulsingTile>
    {
        protected override void Invoke(object parameter)
        {
            var pulsingTile = this.AssociatedObject as SfPulsingTile;
            MainWindow window = VisualUtils.FindAncestor(pulsingTile,
                typeof(MainWindow)) as MainWindow;
            if (window != null && pulsingTile != null)
            {
                //For a single tile use this code.
                HubTileService.UnFreeze(window.pulsingTileOne);
                //For group of tiles use this code.
                HubTileService.UnFreeze("Applications");
            }
        }
    }
}
```



Single tile



Group of tiles

Notifications

Once the tile is pressed, it is notified by the click event and the command property of the Pulsing Tile.

Event

The [Click](#) event rises whenever the tile is pressed.

XML

```
<syncfusion:SfPulsingTile Header="Music" Foreground="White" Title="This is
title area." PulseScale="3" PulseDuration="00:00:03" >
<Image Source="/Assets/PulsingTile.jpg"/>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Click">
<local:ClickEvent />
</i:EventTrigger>
</i:Interaction.Triggers>
</syncfusion:SfPulsingTile>
```

C#

```
public class ClickEvent : TargetedTriggerAction<SfPulsingTile>
{
    protected override void Invoke(object parameter)
    {
        var pulsingTile = this.AssociatedObject as SfPulsingTile;
        MainWindow window = VisualUtils.FindAncestor(pulsingTile,
        typeof(MainWindow)) as MainWindow;
        if ((window != null) && (pulsingTile != null))
        {
            MessageBox.Show("Pulsing Tile has been Clicked");
        }
    }
}
```

Command binding

Command specifies the operation to be performed when the tile is pressed. [Command](#) and [CommandParameter](#) are used instead of click event in MVVM pattern.

XML

```
<Window x:Class="Pulsingtile.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:Pulsingtile"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:shared="clr-
namespace:Syncfusion.Windows.Controls;assembly=Syncfusion.SfShared.Wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:Viewmodel/>
</Window.DataContext>
<Grid x:Name="grid">
<syncfusion:SfPulsingTile Header="Music" Title="This is title area."
PulseScale="3" PulseDuration="00:00:03" Command="{Binding
PulsingTileCommand}" CommandParameter="{Binding ElementName=pulsingtile}">
<Image Source="/Assets/PulsingTile.jpg" Stretch="None"/>
</syncfusion:SfPulsingTile>
</Grid>
</Window>

```

C#

```

public class Viewmodel
{
private ICommand pulsingTileCommand;
public ICommand PulsingTileCommand
{
get
{
return pulsingTileCommand ?? (pulsingTileCommand = new Command(true,
()=>MyAction("PulsingTileCommand")));
}
}
private void MyAction(string parameter)
{
if (parameter.Equals("PulsingTileCommand"))
{
string message = string.Format("Pulsing Tile Command executed");
MessageBox.Show(message);
}
}
public class Command : ICommand
{
private bool _canExecute;
private Action _execute;
public Command(bool CanExecute, Action Execute)
{
_canExecute = CanExecute;
_execute = Execute;
}
public event EventHandler CanExecuteChanged;
public bool CanExecute(object parameter)
{
return _canExecute;
}
public void Execute(object parameter)

```

```
{
    _execute();
}
```

Appearance and styling

Customizing header

Header of the tile can be customized either through [HeaderStyle](#) or [HeaderTemplate](#) as shown below.

[HeaderStyle](#) is used to customize the header of the tile by setting its appropriate properties.

[HeaderTemplate](#) is used to customize the visual appearance of the header by adding user-defined template.

XML

```
<syncfusion:SfPulsingTile x:Name="pulsingtile" Foreground="White"
Title="This is title area." Header="Music">
<Image Source="/Assets/PulsingTile.jpg" Margin="-1"/>
<!--For setting header style-->
<syncfusion:SfPulsingTile.HeaderStyle>
<Style TargetType="ContentControl">
<Setter Property="VerticalAlignment" Value="Bottom"/>
<Setter Property="FontSize" Value="17"/>
<Setter Property="FontFamily" Value="ALGERIAN"/>
</Style>
</syncfusion:SfPulsingTile.HeaderStyle>
</syncfusion:SfPulsingTile>
```



XML

```
<syncfusion:SfPulsingTile x:Name="pulsingTile" Foreground="White"
Title="This is a title area." Header="Music">
<Image Source="/Assets/PulsingTile.png" Stretch="None"/>
<!--For setting header template-->
<syncfusion:SfPulsingTile.HeaderTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<Image Source="/Assets/syncfusion.png" HorizontalAlignment="Left"
Stretch="None"/>
<TextBlock Text="SYNCFUSION" Foreground="White" Grid.Column="1"
FontSize="17" />
</DataTemplate>
</syncfusion:SfPulsingTile.HeaderTemplate>
</syncfusion:SfPulsingTile>
```

```

</Grid>
</DataTemplate>
</syncfusion:SfPulsingTile.HeaderTemplate>
</syncfusion:SfPulsingTile>

```



Customizing title

The [TitleStyle](#) property is used to customize the title of the tile by setting its appropriate properties.

XML

```

<!--SfPulsingTile-->
<syncfusion:SfPulsingTile x:Name="pulsingtile" Foreground="White"
Title="Title" Header="Music">
<Image Source="/Assets/PulsingTile.jpg" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
<!-- For setting title style -->
<syncfusion:SfPulsingTile.TitleStyle>
<Style TargetType="ContentControl">
<Setter Property="Foreground" Value="White"/>
<Setter Property="FontSize" Value="17"/>
<Setter Property="ContentTemplate">
<Setter.Value>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<Image Source="/Assets/syncfusion.png" HorizontalAlignment="Left"
VerticalAlignment="Top" Stretch="None"/>
<TextBlock Text="SYNCFUSION" Grid.Column="1" />
</Grid>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:SfPulsingTile.TitleStyle>
</syncfusion:SfPulsingTile>

```



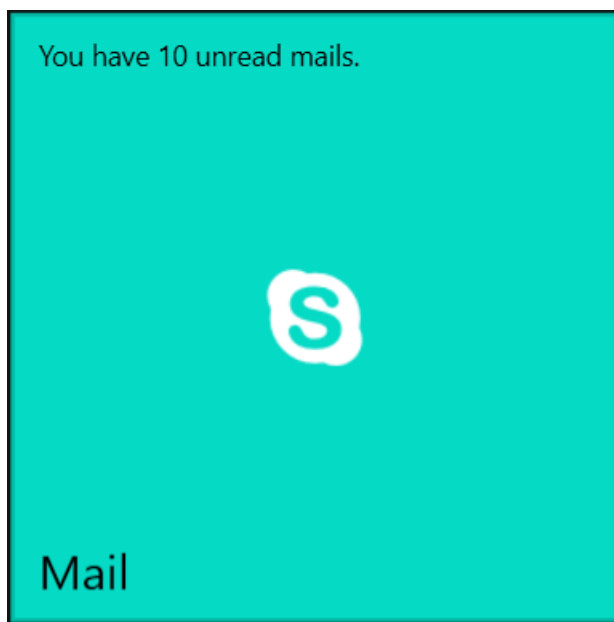
Themes in WPF Tile Control

Hub Tile and Pulsing Tile supports various themes which can be applied using [SfSkinManager](#) and also provided support to create custom theme using [Theme Studio](#).

Theme

Hub Tile and Pulsing Tile controls supports various built-in themes. Refer to the below links to apply themes for the Hub Tile and Pulsing Tile,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



IntegerTextBox

WPF IntegerTextBox Overview

The [IntegerTextBox](#) control restricts text box input to only integer values with support for data binding, Watermark, Null Value, and culture support. It provides many customization options to improve its appearance and to suit your applications.

Control structure





Features

The core features of the `IntegerTextBox` are as follows:

- Supports upto Int64 data type.
- Provides the ability to control the range of input values by using the `MinValue` and `MaxValue` properties.
- Provides different foreground brushes for positive and negative numbers.
- Provides data binding support.
- Provides built-in Visual Styles and themes.
- Provides Watermark support.
- Provides Number Format support.
- Provides culture support.

Getting started with WPF IntegerTextBox

This section explains how to create a WPF `IntegerTextBox` 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.

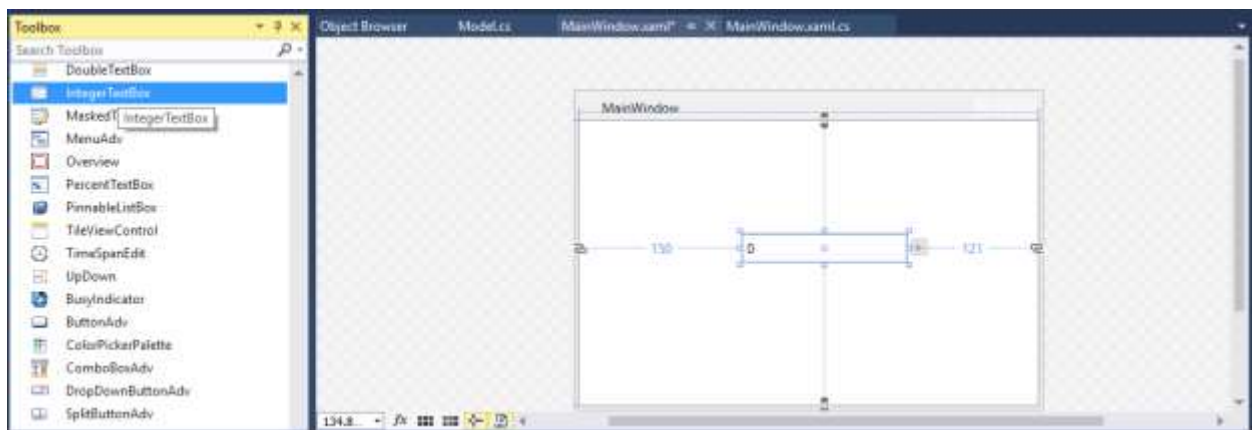
You can find more details about installing the NuGet packages in a WPF application in the following link:

[How to install nuget packages](#)

Adding WPF IntegerTextBox via designer

You can add the `IntegerTextBox` 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 IntegerTextBox via XAML

To add the `IntegerTextBox` 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 `IntegerTextBox` 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="IntegerTextBoxSample.MainWindow"
Title="IntegerTextBox Sample" Height="350" Width="525">
<Grid>
<!--Adding IntegerTextBox control -->
<syncfusion:IntegerTextBox x:Name="integerTextBox" Width="100" Height="20"
VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
</Window>
```

Adding WPF IntegerTextBox via C\#

To add the IntegerTextBox 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.

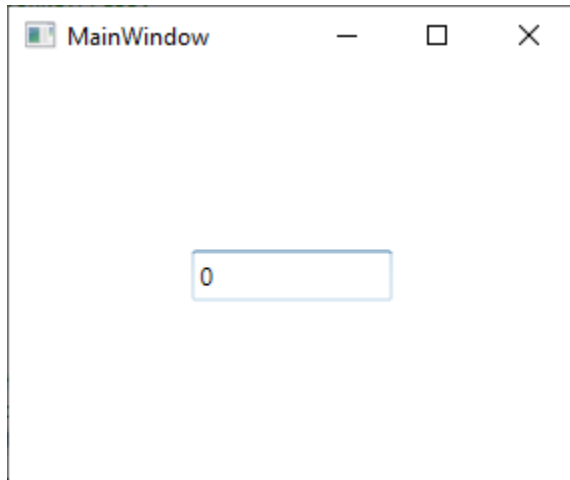
C#

```
using Syncfusion.Windows.Shared;
```

4. Create an instance of IntegerTextBox and add it to the window.

C#

```
//Creating an instance of IntegerTextBox control
IntegerTextBox integerTextBox = new IntegerTextBox();
// Setting height and width to IntegerTextBox
integerTextBox.Height = 25;
integerTextBox.Width = 100;
//Adding IntegerTextBox as window content
this.Content = integerTextBox;
```



Setting Value

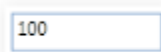
The value of the `IntegerTextBox` can be set by using the [Value](#) property.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Width="100" Height="23" Value="100"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();  
integerTextBox.Width = 100;  
integerTextBox.Height = 23;  
integerTextBox.Value = 100;
```



Note: Do not use the [Text](#) property to set the value for the `IntegerTextBox`. 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 `IntegerTextBox` using the `Value` Property.

The following code snippets illustrate the value binding from one `IntegerTextBox` to another.

XML


```
<StackPanel>  
<syncfusion:IntegerTextBox x:Name="integerTextBox1" Height="25" Width="100" Value="{Binding MyValue, UpdateSourceTrigger=PropertyChanged}"/>  
<syncfusion:IntegerTextBox x:Name="integerTextBox2" Width="100" Height="25" Value="{Binding MyValue, UpdateSourceTrigger=PropertyChanged}" />  
</StackPanel>
```

C#

```

class ViewModel : NotificationObject
{
    private int myValue;
    public int MyValue
    {
        get
        {
            return myValue;
        }
        set
        {
            myValue = value;
            RaisePropertyChanged("MyValue");
        }
    }
}

```




Value Changed Notification

The `IntegerTextBox` 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:IntegerTextBox ValueChanged="IntegerTextBox_ValueChanged"/>

```

C#

```

IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.ValueChanged += new
PropertyChangedCallback(IntegerTextBox_ValueChanged);

```

You can handle the event as follows:

C#

```

private void IntegerTextBox_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 **IntegerTextBox** 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:IntegerTextBox x:Name="integerTextBox" Width="100" Height="25" Value="100" MaxValue="999" MinValue="-999"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();  
integerTextBox.Width = 100;  
integerTextBox.Height = 25;  
//Setting minimum value  
integerTextBox.MinValue = -999;  
//Setting maximum value  
integerTextBox.MaxValue = 999;  
integerTextBox.Value = 100;
```

Step Interval to increase or decrease the value

The **IntegerTextBox** 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:IntegerTextBox x:Name="integerTextBox" Width="150" Height="25" Value="8" IsScrollingOnCircle="True" ScrollInterval="4"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();  
integerTextBox.Width = 150;  
integerTextBox.Height = 25;  
integerTextBox.MinValue = 0;  
integerTextBox.MaxValue = 100;  
integerTextBox.Value = 8;  
integerTextBox.IsScrollingOnCircle = true;  
integerTextBox.ScrollInterval = 4;
```

Formatting the value

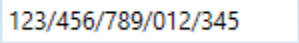
You can customize the number format by either setting the [NumberFormat](#) property or the [NumberGroupSeparator](#) and the [NumberGroupSizes](#) property of **IntegerTextBox**.

XML

```
<!--Number Format -->
<syncfusion:IntegerTextBox x:Name="integerTextBox" Height="25" Width="150"
Culture="en-US" Value="123456789012345" NumberGroupSeparator="/" />
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 150;
integerTextBox.Height = 25;
integerTextBox.Value = 123456789012345;
integerTextBox.Culture = new System.Globalization.CultureInfo("en-US");
integerTextBox.NumberFormat = new System.Globalization.NumberFormatInfo()
{
    NumberGroupSeparator = "/"
};
```


Setting the Culture


The **IntegerTextBox** provides support for globalization by using the **Culture** property. The **Culture** is used to format the group separator of the **IntegerTextBox** value based on the respective culture.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Height="25" Width="100"
Culture="en-US" Value="1234567" />
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
integerTextBox.Height = 25;
integerTextBox.Value = 1234567;
integerTextBox.Culture = new System.Globalization.CultureInfo("en-US");
```



Note: When you use both **NumberFormat** and **Culture**, the **NumberFormat** will have a higher priority.

Theme

IntegerTextBox supports various built-in themes. Refer to the below links to apply themes for the IntegerTextBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Changing Integer Value in WPF IntegerTextBox

The [IntegerTextBox](#) allows the user to change the value using the [Value](#) property.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Height="25"
Width="150" Value="10"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 150;
integerTextBox.Height = 25;
integerTextBox.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 [IntegerTextBox](#) value.

The following code snippets illustrate the value binding from one [IntegerTextBox](#) to another.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox1" Value="{Binding
MyValue,UpdateSourceTrigger=PropertyChanged}" Height="25" Width="100"/>
<syncfusion:IntegerTextBox x:Name="integerTextBox2" Value="{Binding
MyValue,UpdateSourceTrigger=PropertyChanged}" Width="100" Height="25" />
```

ViewModel.cs

C#

```
class ViewModel : NotificationObject
{
    private int myValue;
    public int MyValue
    {
        get
        {
            return myValue;
        }
        set
        {
            myValue = value;
            RaisePropertyChanged("MyValue");
        }
    }
}
```




Change integer value by pasting the clipboard's text

By default, **IntegerTextBox** 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 and copied value contains number decimal separator	it won't perform pasting operation.

XML

```
<syncfusion:IntegerTextBox PasteMode="Advanced"
Value="12345"
Name="integerTextBox"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.PasteMode = PasteMode.Advanced;
integerTextBox.Value = 12345;
```

Pasting "5"



Show UpDown Button

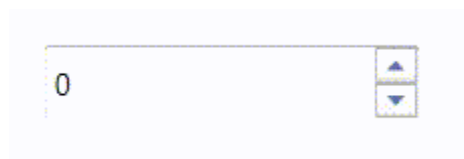
You can increment or decrement the integer value of `IntegerTextBox` by setting the `ShowSpinButton` property value as `true`. Click `UpButton` to increment or `DownButton` to decrement the integer value. The default value of `ShowSpinButton` property is `false`.

XML

```
<syncfusion:IntegerTextBox Height="30" Width="150" ShowSpinButton="True" />
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.ShowSpinButton = true;
```



Value Changed Event

The `IntegerTextBox` 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:IntegerTextBox ValueChanged="IntegerTextBox_ValueChanged"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.ValueChanged += new
PropertyChangedCallback(IntegerTextBox_ValueChanged);
```

You can handle the event as follows:

C#

```
private void IntegerTextBox_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 `IntegerTextBox` 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 integer value. It will display only on setting the `UseNullOption` property is set to `true`.

NullValue = Null**XML**

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Height="25"
Width="100" UseNullOption="True" NullValue="{x:Null}"/>
```

C#

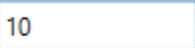
```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
integerTextBox.Height = 25;
integerTextBox.NullValue = null;
integerTextBox.UseNullOption = true;
```

**NullValue = 10****XML**

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Height="25"
Width="100" UseNullOption="True" NullValue="10"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
integerTextBox.Height = 25;
integerTextBox.NullValue = 10;
integerTextBox.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 [IntegerTextBox](#) allows you to set the desired brush as a foreground for WaterMarkText using [WaterMarkTextForeground](#) property. The default color of [WaterMarkTextForeground](#) is Black.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Width="100"
Height="25" UseNullOption="True" WatermarkText="Type here"
WatermarkTextIsVisible="True" WatermarkTextForeground="Red"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
```

```
integerTextBox.Height = 25;
integerTextBox.UseNullOption = true;
integerTextBox.WatermarkText = "Type Here";
integerTextBox.WatermarkTextIsVisible = true;
integerTextBox.WatermarkTextForeground = Brushes.Red;
```

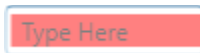


Setting Watermark Template

You can customize the Visual appearance of the `WatermarkText` by using the [WatermarkTemplate](#) property.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Width="100" Height="25"
WatermarkText="Type Here" CornerRadius="3"
WatermarkTextIsVisible="True" WatermarkOpacity="0.5"
UseNullOption="True">
  <syncfusion:IntegerTextBox.WatermarkTemplate >
    <DataTemplate>
      <Border Background="Red">
        <TextBlock Text="{Binding}" VerticalAlignment="Center" Margin="5,0,0,0"/>
      </Border>
    </DataTemplate>
  </syncfusion:IntegerTextBox.WatermarkTemplate>
</syncfusion:IntegerTextBox>
```



Note: The `UseNullOption` property must be enabled if you want to see `NullValue` or `WaterMarkText` in `IntegerTextBox` control.

Note: If both `NullValue` and `WaterMarkText` are specified, you will only see `NullValue` but not `WaterMarkText`.

Step Interval in WPF IntegerTextBox

The [IntegerTextBox](#) 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 `IntegerTextBox` 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

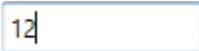
The `IntegerTextBox` control allows you to increase or decrease the `Value` of `IntegerTextBox` based on the `ScrollInterval` by pressing the up arrow and down arrow keys on the keyboard.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Width="150"
Height="25" Value ="10" ScrollInterval="2"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 150;
integerTextBox.Height = 25;
integerTextBox.Value = 10;
integerTextBox.ScrollInterval = 2;
```



Change Value on Mouse Wheel

The `IntegerTextBox` 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:IntegerTextBox x:Name="integerTextBox" Width="150" Height="25"
Value ="34"
IsScrollingOnCircle="True" ScrollInterval="3"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 150;
integerTextBox.Height = 25;
integerTextBox.Value = 34;
integerTextBox.IsScrollingOnCircle = true;
integerTextBox.ScrollInterval = 3;
```



Change Value on Click and Drag

The `IntegerTextBox` allows you to increase or decrease the value based on the `ScrollInterval` by clicking and dragging the mouse when the `EnableExtendedScrolling` property is `true`. `IntegerTextBox` 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:IntegerTextBox x:Name="integerTextBox" Width="120" Height="25"
Value ="10"
ScrollInterval="5" EnableExtendedScrolling="True"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 120;
integerTextBox.Height = 25;
integerTextBox.Value = 10;
integerTextBox.ScrollInterval = 5;
integerTextBox.EnableExtendedScrolling = true;
```



Allow or restrict selection on focus

IntegerTextBox 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.

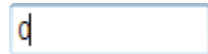
XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox"
TextSelectionOnFocus="False"/>
```

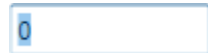
C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.TextSelectionOnFocus = true;
```

TextSelectionOnFocus = "False"



TextSelectionOnFocus = "True"



Restriction or Validation in WPF IntegerTextBox

This section explains how to validate or restrict the [IntegerTextBox](#) control value.

Restrict the value within minimum and maximum value

The [Value](#) of the [IntegerTextBox](#) 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 minimum and maximum limits by using the [MinValue](#) and [MaxValue](#) properties.

You can choose when to validate the maximum and minimum limits while changing the values by using the [MinValidation](#) and [MaxValidation](#) properties.

- **OnKeyPress** — When [MaxValidation](#) or [MinValidation](#) properties value is **OnKeyPress**, the value in the IntegerTextBox will be validated shortly after pressing a key. It is not possible to provide any invalid input at all and the value does not exceed the maximum and minimum limits.
- **OnLostFocus** - When [MaxValidation](#) or [MinValidation](#) properties is **OnLostFocus**, the value in the IntegerTextBox is validated only when the IntegerTextBox loses the focus. After validation, when the value of the IntegerTextBox 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 either 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` or 200 will be retained if `MaxValueOnExceedMaxDigit` is `false`.

- [MinValueOnExceedMinDigit](#) - When you give input less than specified minimum limit, `MinValueOnExceedMinDigit` property will 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 change the value to 20. `Value` will be changed to 200 when `MinValueOnExceedMinDigit` is `true` or when `MinValueOnExceedMinDigit` is `false`, Old value 205 will be retained.

Note: `MaxValueOnExceedMinDigit` and `MinValueOnExceedMinDigit` properties will be enabled only when the `MaxValidation` and `MinValidation` is set to `OnKeyPress`.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Width="150"
MaxValue="100" MinValue="10"
MinValueOnExceedMinDigit="True" MaxValueOnExceedMaxDigit="True"
MinValidation="OnKeyPress" MaxValidation="OnLostFocus"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
integerTextBox.Height = 25;
integerTextBox.MinValue = 10;
integerTextBox.MaxValue = 100;
integerTextBox.MinValidation = MinValidation.OnKeyPress;
integerTextBox.MaxValidation = MaxValidation.OnLostFocus;
integerTextBox.MinValueOnExceedMinDigit = true;
integerTextBox.MaxValueOnExceedMaxDigit = true;
```

When `MinValidation` value is `OnKeyPress`, you cannot enter value less than the `MinValue`. If try to enter a value less than the `MinValue`, then the `MinValue` will be set to the `Value` property because `MinValueOnExceedMinDigit` is set to `true`.

10

`MaxValidation` is set to `OnLostFocus`, so the `MaxValidation` will be performed only in the lost focus.

100

Read only mode

The [IntegerTextBox](#) doesn't allow the user input on application runtime when `IsReadOnly` property is `true`. The user can still select text and display the cursor on the `IntegerTextBox` by setting the [IsReadOnlyCaretVisible](#) property to `true`.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" IsReadOnly="True"
Value="78" IsReadOnlyCaretVisible="True"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();  
integerTextBox.Value = 78;  
integerTextBox.IsReadOnly = true;  
integerTextBox.IsReadOnlyCaretVisible = true;
```



Customize the behavior for invalid value

You can customize how the [IntegerTextBox](#) behaves when entered value is not equal to the value of [ValidationValue](#) property, using [InvalidValueBehavior](#) property. It can be customized by below values,

- **DisplayErrorMessage** - Shows a MessageBox with message "String validation failed" after focus is lost from IntegerTextBox.
- **None** - Validation will not occurs.
- **ResetValue** - Resets the entered value to 0 after focus is lost.

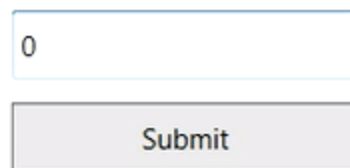
Note: By default ValidationValue property value is String.Empty.

XML

```
<syncfusion:DoubleTextBox Width="120" Height="30"  
InvalidValueBehavior="DisplayErrorMessage"  
ValidationValue="1222"  
VerticalAlignment="Center"  
HorizontalAlignment="Center" />
```

C#

```
DoubleTextBox doubleTextBox1 = new DoubleTextBox()  
{  
    Height = 30,  
    Width = 120,  
    InvalidValueBehavior = InvalidInputBehavior.DisplayErrorMessage,  
    ValidationValue = "1222",  
    HorizontalAlignment = HorizontalAlignment.Center,  
    VerticalAlignment = VerticalAlignment.Center  
};
```



Culture and Formatting in WPF IntegerTextBox

Value of `IntegerTextBox` can be formatted in following ways:

- Culture
- NumberFormatInfo
- Dedicated properties (NumberGroupSeparator, NumberGroupSizes)

Culture based formatting

The `IntegerTextBox` provides support for globalization by using the `Culture` property. The `Culture` property is used to format the number group size and group separator of the `IntegerTextBox` value based on the respective culture.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Height="25" Width="150"
Culture="bs-Latn" Value="1234567"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 150;
integerTextBox.Height = 25;
integerTextBox.Value = 1234567;
//Setting Latin culture for integer textbox.
integerTextBox.Culture = new System.Globalization.CultureInfo("bs-Latn");
```

By default the US culture uses “,” as the `NumberGroupSeparator` where as the Latin culture uses “.” as the `NumberGroupSeparator`.

Default Culture



Latin Culture



NumberFormatInfo based formatting

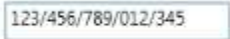
The number formatting of `IntegerTextBox` can be customized by setting `NumberFormat` property.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Height="25" Width="150"
Culture="en-US"
Value="123456789012345" GroupSeparatorEnabled="True" >
<syncfusion:IntegerTextBox.NumberFormat >
<numberformat:NumberFormatInfo NumberGroupSeparator="/" />
</syncfusion:IntegerTextBox.NumberFormat>
</syncfusion:IntegerTextBox>
```

C#

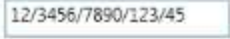
```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 150;
integerTextBox.Height = 25;
integerTextBox.GroupSeparatorEnabled = true;
integerTextBox.Value = 123456789012345;
integerTextBox.Culture = new System.Globalization.CultureInfo("en-US");
integerTextBox.NumberFormat = new System.Globalization.NumberFormatInfo()
{
    NumberGroupSeparator = "/"
};
```



The following code illustrate how to set `NumberGroupSizes` by using the `NumberFormat` property.

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 150;
integerTextBox.Height = 25;
integerTextBox.Value = 123456789012345;
integerTextBox.GroupSeparatorEnabled = true;
integerTextBox.NumberFormat = new NumberFormatInfo()
{
    NumberGroupSeparator = "/",
    NumberGroupSizes = new int[] { 2, 3, 4 }
};
```

Formatting with dedicated properties

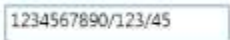
The number formatting of `IntegerTextBox` can also be customized by setting the [NumberGroupSeparator](#) property and the [NumberGroupSizes](#) property.

You can show the group separator by enable the [GroupSeperatorEnabled](#) property to `true`.

The following code illustrate how to format using the `NumberGroupSeparator`, `NumberGroupSizes` property of the `IntegerTextBox`.

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 150;
integerTextBox.Height = 25;
integerTextBox.Value = 123456789012345;
integerTextBox.NumberGroupSeparator = "/";
integerTextBox.GroupSeperatorEnabled = true;
integerTextBox.NumberGroupSizes = new Int32Collection() { 2, 3, 0 };
```



Note: When you use both the `NumberFormat` and the dedicated properties (`NumberGroupSeparator` and `NumberGroupSizes`) to format the value of `IntegerTextBox`, the `NumberGroupSeparator` and `NumberGroupSizes` properties have higher priority.

Note: When you use both `NumberFormat` and `Culture`, the `NumberFormat` will have a higher priority.

Appearance in WPF IntegerTextBox

This section deals with the appearance of `IntegerTextBox` control and contains the following topics.

Setting the Foreground

The `IntegerTextBox` control [Foreground](#) can be modified based on the value of the control. The following are the foreground for `IntegerTextBox` control.

Foreground for Positive Value

We can change a positive color for the value of `IntegerTextBox` by setting the [PositiveForeground](#) property and it will be applied when the [Value](#) is positive. The default color of `PositiveForeground` is Black.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Value="10" Width="100"
Height="25" PositiveForeground="Blue" />
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
integerTextBox.Height = 25;
integerTextBox.Value = 10;
```

```
integerTextBox.PositiveForeground = Brushes.Blue;
```



Foreground for Negative Value

We can change a negative color for the value of IntegerTextBox 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:IntegerTextBox x:Name="integerTextBox" Value="-10" Width="100"
Height="25"
NegativeForeground="SpringGreen" ApplyNegativeForeground="True" />
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
integerTextBox.Height = 25;
integerTextBox.Value = -10;
integerTextBox.ApplyNegativeForeground = true;
integerTextBox.NegativeForeground = Brushes.SpringGreen;
```



Foreground for Zero Value

We can change a zero color for the value of IntegerTextBox 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:IntegerTextBox x:Name="integerTextBox" Value="0" Width="100"
Height="25"
ApplyZeroColor="True" ZeroColor="DarkGoldenrod"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
integerTextBox.Height = 25;
integerTextBox.Value = 0;
integerTextBox.ApplyZeroColor = true;
integerTextBox.ZeroColor = Brushes.DarkGoldenrod;
```



Setting the Background

IntegerTextBox 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:IntegerTextBox x:Name="integerTextBox" Width="100"
Height="25" Value ="80" Background="Cyan"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
integerTextBox.Height = 25;
integerTextBox.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 **IntegerTextBox**, use [CornerRadius](#) property. The default value of **CornerRadius** property is 1.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Width="100" Height="25"
CornerRadius="5"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
integerTextBox.Height = 25;
integerTextBox.CornerRadius = new CornerRadius(5);
```



Apply Background for Selection

IntegerTextBox allows different brushes to highlight the selected text by setting the [SelectionBrush](#) and [SelectionOpacity](#) properties. The **SelectionOpacity** property specifies the opacity of the **SelectionBrush**.

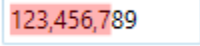
XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Width="100" Height="25"
SelectionBrush="Red" SelectionOpacity="0.5"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.Width = 100;
```

```
integerTextBox.Height = 25;  
integerTextBox.SelectionBrush = Brushes.Red;  
integerTextBox.SelectionOpacity = 0.3;
```



Align Value

IntegerTextBox 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:IntegerTextBox x:Name="integerTextBox" Width="100" Height="25"  
TextAlignment="Center"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();  
integerTextBox.Width = 100;  
integerTextBox.Height = 25;  
integerTextBox.TextAlignment = TextAlignment.Center;
```



Setting ToolTip

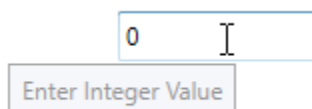
IntegerTextBox provides support for ToolTip to display certain information when the mouse hovers on the IntegerTextBox. You can customize the tooltip information by setting the [ToolTip](#) property.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" Width="100" Height="25"  
ToolTip="Enter Integer Value"/>
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();  
integerTextBox.Width = 100;  
integerTextBox.Height = 25;  
integerTextBox.ToolTip = "Enter Integer Value";
```



Theme

IntegerTextBox supports various built-in themes. Refer to the below links to apply themes for the IntegerTextBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Range Adorner in WPF IntegerTextBox

Value of [IntegerTextBox](#) can be visually indicated like a progress bar using range-adorner feature, this feature is disabled by default. You can show the adorner over IntegerTextBox 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:IntegerTextBox x:Name="integerTextBox" Value="63" MinValue="0"
MaxValue="100" EnableRangeAdorner="True" />
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.MinValue = 0;
integerTextBox.MaxValue = 100;
integerTextBox.Value = 63;
integerTextBox.EnableRangeAdorner = true;
```



Changing background of range-adorner

You can change the background color of the range adorner using [RangeAdornerBackground](#) property.

XML

```
<syncfusion:IntegerTextBox x:Name="integerTextBox" MinValue="0"
MaxValue="100" Value="57" EnableRangeAdorner="True"
RangeAdornerBackground="LightGreen" />
```

C#

```
IntegerTextBox integerTextBox = new IntegerTextBox();
integerTextBox.MinValue = 0;
integerTextBox.MaxValue = 100;
integerTextBox.Value = 57;
integerTextBox.EnableRangeAdorner = true;
integerTextBox.RangeAdornerBackground = Brushes.LightGreen;
```



SfImageEditor

WPF ImageEditor (SfImageEditor) Overview

SfImageEditor

The image editor control is a very handy tool that is used to edit an image by annotating with text, pen and built-in shapes. It allows you to crop, rotate, and flip an image. The image editor control has a built-in toolbar, which helps in performing editing operations.



Key features

- Built-in toolbar
- Flip
- Crop
- Rotate
- Adding shapes such as rectangle, circle, and arrow
- Annotating with text, pen (i.e. free hand drawing)
- Saving the edited image
- Zooming and panning
- Undo and redo
- Reset

Getting Started with SfImageEditor

This section explains the steps required to load an image to the image editor control. It has a built-in toolbar that helps in performing various editing operations such as flip, crop, rotate, save, annotating with shapes, text, path (i.e. free hand drawing), zoom, and pan.

Adding ImageEditor reference

Refer to this [document](#) to learn how to add Syncfusion controls in Visual Studio projects through various ways. Refer to this [document](#) to learn about the assemblies required for adding ImageEditor to your project.

Initialize ImageEditor

Import the Image editor namespace as demonstrated in the following code snippet.

XML

```
xmlns:editor="clr-namespace:Syncfusion.UI.Xaml.ImageEditor;assembly=Syncfusion.SfImageEditor.WPF"
```

C#

```
using Syncfusion.UI.Xaml.ImageEditor;
```

You can either use the below schemas or the above mentioned namespace to refer the ImageEditor control in xaml.

XML

```
xmlns:editor="http://schemas.syncfusion.com/wpf"
```

Then, initialize the image editor as demonstrated in the following code snippet.

XML

```
<editor:SfImageEditor>  
</editor:SfImageEditor>
```

C#

```
SfImageEditor editor = new SfImageEditor();
```

Loading image in ImageEditor

Image can be loaded in the following two ways:

- Using image source
- Using stream

You can load the [ImageSource](#) as demonstrated in the following code snippet.

XML

```
<editor:SfImageEditor x:Name="editor"  
ImageSource="Assets/Buldingimage.jpeg">
```

```
</editor:SfImageEditor>
```

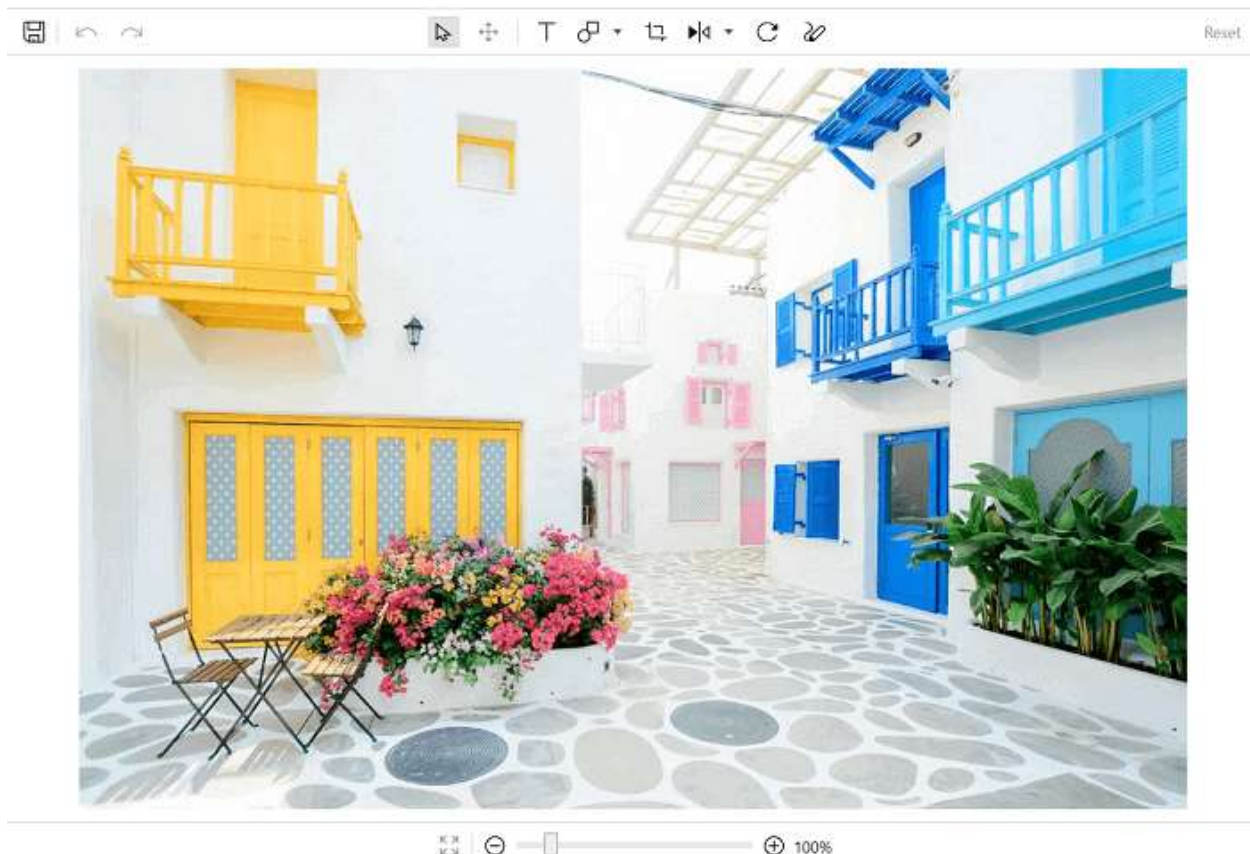
C#

```
BitmapImage image = new BitmapImage();  
image.BeginInit();  
image.UriSource = new Uri(@"Assets/Buldingimage.jpeg", UriKind.Relative);  
image.EndInit();  
editor.ImageSource = image;
```

You can load the image as stream using the [Image](#) property as in the following code snippet.

C#

```
OpenFileDialog openFileDialog = new OpenFileDialog();  
openFileDialog.Filter = "Image  
Files (*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|All files (*.*)|*.* ";  
if (openFileDialog.ShowDialog() == true)  
{  
    var stream = openFileDialog.OpenFile();  
    editor.Image = stream;  
}
```



Theme

Image editor control supports various built-in themes. Refer to the below links to apply themes for the Image editor control,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



See also

[How to retrieve a edited image as a stream in an image editor](#)

Text in SfImageEditor

You can annotate an image by adding the desired text to it. This can be done in the following two ways:

1. Adding text using built-in toolbar
2. Adding text programmatically

Toolbar

To add text, click the Text icon (T) in the toolbar. Now, the text will be added at the center of the image along with the selection handle, which helps in resizing. By default, the added text will be in pan mode, so you can position the text anywhere by dragging it. By clicking the text, it will change into typing mode, and you can edit the text directly. Click outside to exit from typing mode.

Customization

The following properties of the added text can be customized:

- Font family
- Font size
- Font color
- Alignment (center, left, and right)
- Text effects (bold, italic, and underline)

Upon selecting the Text icon in the toolbar, a sub toolbar will be generated below the main toolbar for providing the text customizing options.

N Text needs to be selected to apply the customization from the sub toolbar.

Font family

Font family icon lists out the predefined system font families. You can select the required font family.

Font size

You can select the size from the listed-out font sizes.

Font color

Color picker will be opened by clicking this icon. You can pick the required color from it.

Text effects

You can make the text bold, italic, or underline by clicking the corresponding icons in the toolbar. By default, no effects will be applied.

Alignment

Text can be aligned to the left, center, or right by clicking the corresponding icon. By default, text will be aligned at the left.

Adding text programmatically

You can add text to an image using the AddText method programmatically. This method requires the following parameters:

- Text – Specifies the content you need to add on the image.
- TextSettings - Customizes the text.

C#

```
editor.AddText("Hello", new TextSettings());
```

Text settings

Using the following properties in TextSettings, text can be customized.

- [FontFamily](#) - Changes the font family of the text.
- [FontSize](#) - Changes the font size of the text.
- [Color](#) - Modifies the font color of the text.
- [TextAlignment](#) - Text alignment such as left, top, and center can be applied.
- [TextEffects](#) - Text effects such as bold, italic, and underline can be applied.
- [Opacity](#) - Modifies the opacity of the text.
- [Angle](#) - Rotates the text to the specified angle.
- [Bounds](#) - Rect used to position the text.

N Values of bounds will be in percentage. For example (25, 25, 25, 25) will take the position of 25 percent from the left and top.

C#

```
TextSettings textSettings = new TextSettings();
textSettings.FontFamily = new FontFamily("Century Schoolbook");
textSettings.FontSize = 30;
textSettings.Color = new SolidColorBrush(Colors.Red);
textSettings.TextEffects = TextEffects.Bold | TextEffects.Italic;
textSettings.Bounds = new Rect(50, 10, 50, 15);
```

```
editor.AddText("Good morning", textSettings);  
textSettings = new TextSettings();  
textSettings.FontFamily = new FontFamily("Bell MT");  
textSettings.FontSize = 22;  
textSettings.Color = new SolidColorBrush(Colors.DarkGreen);  
textSettings.TextEffects = TextEffects.Italic;  
textSettings.Bounds = new Rect(50, 23, 30, 15);  
textSettings.TextAlignment = TextAlignment.Center;  
editor.AddText("The happiness of your \nlife depend upon the \nquality of  
your thoughts.", textSettings);
```



See also

[How to load the image annotated with shapes and text on the image editor](#)

Shapes in SfImageEditor

You can annotate an image by adding regular shapes such as circle, rectangle, and arrow. Also image editor provides support to draw path (i.e. free hand sketching). Shapes can be added in the following two ways:

- Using Toolbar
- Programmatically

Adding shapes using toolbar

To add a shape, click the AddShape icon in the toolbar. This will list out the default shapes, and you can select the desired shape from it. By default, selected shape will be added at the center. You can select

and drag the shape to position at the desired location. Upon selecting the shape, handles will be enabled. Handles help in resizing the shapes. Click outside to disable the selection.

Free hand sketch (Path)

For free hand sketching on the image, select the pen tool in the toolbar, if needed also select the required Stroke and StrokeWidth of the pen from the secondary toolbar. This will let you to draw on the image and you can annotate it by your own shapes, signature or any form of irregular shapes.

Note: After selecting the pen tool click and drag on the image to draw.

Customization

The following properties of the added shape can be customized:

- [Fill](#)
- [Stroke](#)
- [StrokeWidth](#)

By selecting the shape icon in the toolbar, a sub toolbar will be generated below the main toolbar to provide the customizing options for the shapes.

Note: Shapes need to be selected to apply customization from the sub toolbar.

In case of free hand sketching, to apply customization you must select it from the sub toolbar before drawing on the image. You cannot change the customization of the drawn path.

Fill

Color picker will be opened on selecting Fill icon in the secondary toolbar. You can select the desired color from color picker to fill the selected shape. By default, fill is in transparent. This is property is not applicable for free hand sketching.

Stroke

On selecting the required color from this color picker in the secondary toolbar, you can either update the stroke of the selected shape or can draw a new path with that. By default, stroke will be in Red.

Stroke width

You can select the desired stroke width from the listed-out sizes either to update the stroke width of the selected shape or to draw a new path.



Delete

To delete the added shape, select the shape, and then use delete key from the keyboard.

Adding shapes programmatically

Shapes can be added into an image using the AddShape method. This method takes the following two parameters:

- [ShapeType](#) - You can choose the required shape type. The available shape types are rectangle, circle, and arrow.
- [PenSettings](#) – Customizes the added shapes.

C#

```
editor.AddShape(ShapeType.Rectangle, new PenSettings());
```

Use the below code to enable free hand sketching programmatically.

C#

```
editor.AddShape(ShapeType.Path, new PenSettings());
```

Delete

To delete the selected shape, use the Delete method as follows.

C#

```
editor.Delete();
```

Pen settings

The added shapes can be customized using the following properties in pen settings:

- [Fill](#) - Fills the selected shape with this color.

- [Stroke](#) – Applies this stroke to the selected shape.
- [Opacity](#) – Applies opacity to both stroke and fill of the shape.
- [StrokeWidth](#) – Applies the specified width to the stroke of a shape.
- [Bounds](#) – Rect used to position the shape.
- [IsResizable](#) - To control the resizing of the shape.

Note: Values of the bounds rect will be in percentage. For example (25,25,25,25) will take the position of 25 percent from the left and top.

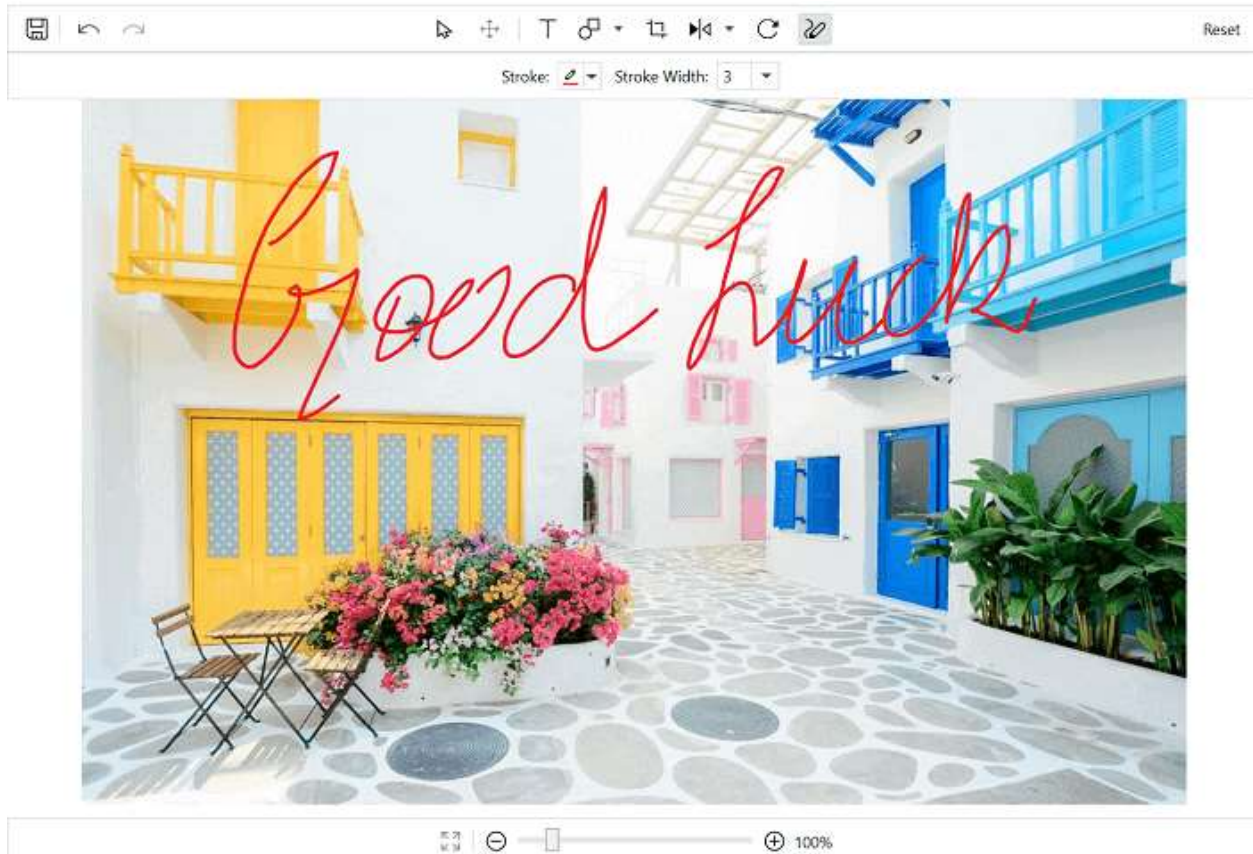
C#

```
PenSettings penSettings = new PenSettings();
penSettings.Fill = new SolidColorBrush(Colors.AliceBlue);
penSettings.Opacity = 0.5f;
penSettings.Stroke = new SolidColorBrush(Colors.Red);
penSettings.StrokeWidth = 3;
penSettings.Bounds = new Rect(10, 30, 20, 20);
editor.AddShape(ShapeType.Rectangle, penSettings);
penSettings = new PenSettings();
penSettings.Fill = new SolidColorBrush(Colors.AliceBlue);
penSettings.Opacity = 0.5f;
penSettings.Stroke = new SolidColorBrush(Colors.Red);
penSettings.StrokeWidth = 3;
penSettings.Bounds = new Rect(40, 30, 20, 25);
editor.AddShape(ShapeType.Circle, penSettings);
```



C#


```
PenSettings penSettings = new PenSettings();  
penSettings.Opacity = 0.7f;  
penSettings.Stroke = new SolidColorBrush(Colors.Red);  
penSettings.StrokeWidth = 3;  
editor.AddShape(Syncfusion.UI.Xaml.ImageEditor.Enums.ShapeType.Path,  
penSettings);
```



Shape resizing

By default, both shapes and text are resizable. You can also control the resizable using the `ResizableElements` property in image editor. The following code enables the resizing functionality to both shapes and text.

C#

```
editor.ResizableElements = ImageEditorResizableElements.Shapes;
```

Events

Shapes and Text supports the following two events:

- [ItemSelected](#)
- [ItemUnselected](#)

ItemSelected

This event occurs when an item (shape/text) is selected. `ItemSelectedEventArgs` will be the parameter. In the `Settings` property, you can get either `TextSettings` or `PenSettings` based on the selected item.

You can make changes on the settings such as stroke etc.

C#

```
private void Editor_ItemSelected(object sender, ItemSelectedEventArgs args)
{
}
```

ItemUnSelected

This event occurs when an item (shape/text) is unfocused. `ItemUnselectedEventArgs` will be the parameter. In the `Settings` property, you can get either `TextSettings` or `PenSettings` based on the selected item.

You can make changes on the settings such as stroke etc. This is applied after the item has been unselected.

C#

```
private void Editor_ItemUnselected(object sender, ItemUnselectedEventArgs args)
{
}
```

See also

[How to load the image annotated with shapes and text on the image editor](#)

Transformation in WPF ImageEditor (SfImageEditor) control

The image editor control provides the following two transformations:

- Flip
- Rotate

Flip

Images can be flipped either in horizontal or vertical direction. By default, images will be flipped in horizontal direction.

Using toolbar

To flip an image, click the Flip icon in the toolbar. Popup will be displayed prompting for whether horizontal flip or vertical flip. Select the required flip direction to flip the image.

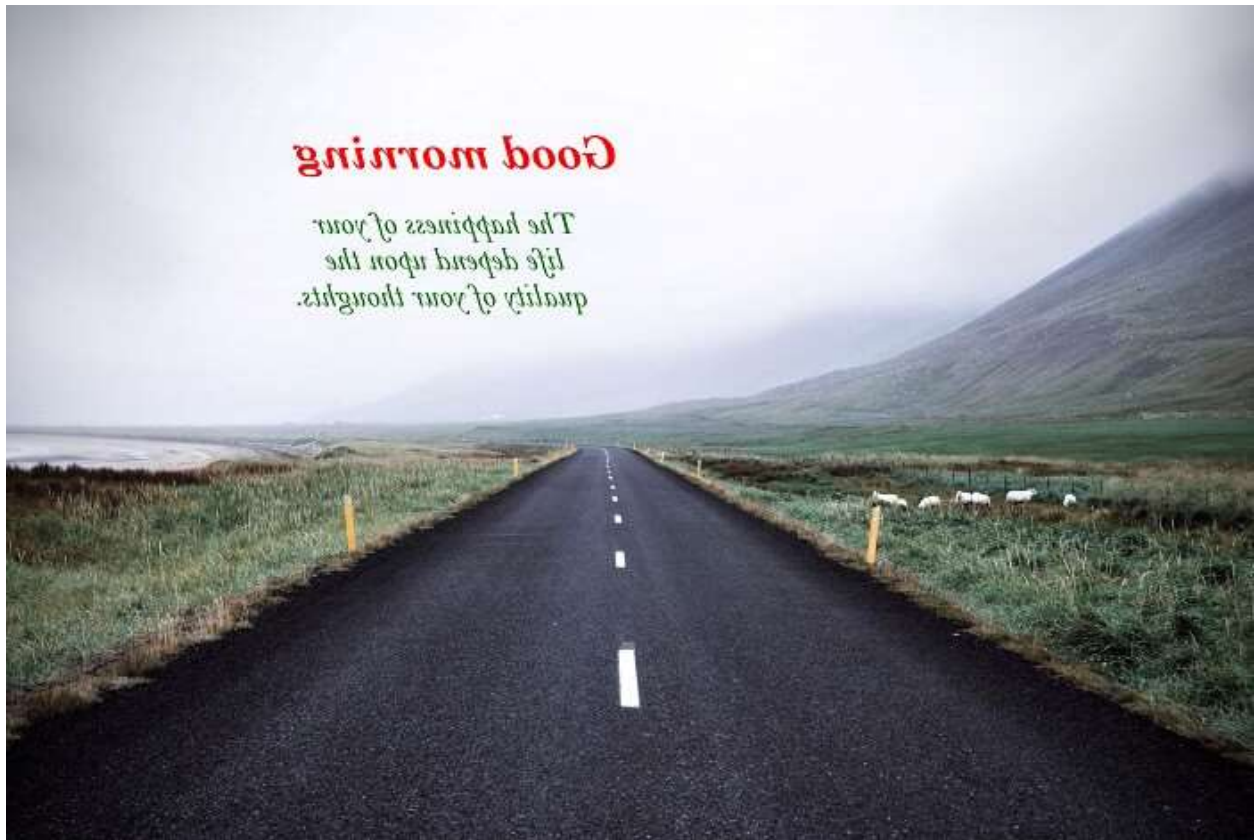
Using code

Programmatically, you can flip an image using the Flip method. This method takes the [FlipDirection](#) as the parameter to specify whether it is a horizontal flip or vertical flip.

C#

```
editor.Flip(FlipDirection.Horizontal);
```


The following screenshot depicts the horizontal flip.



Vertical flip of the image.



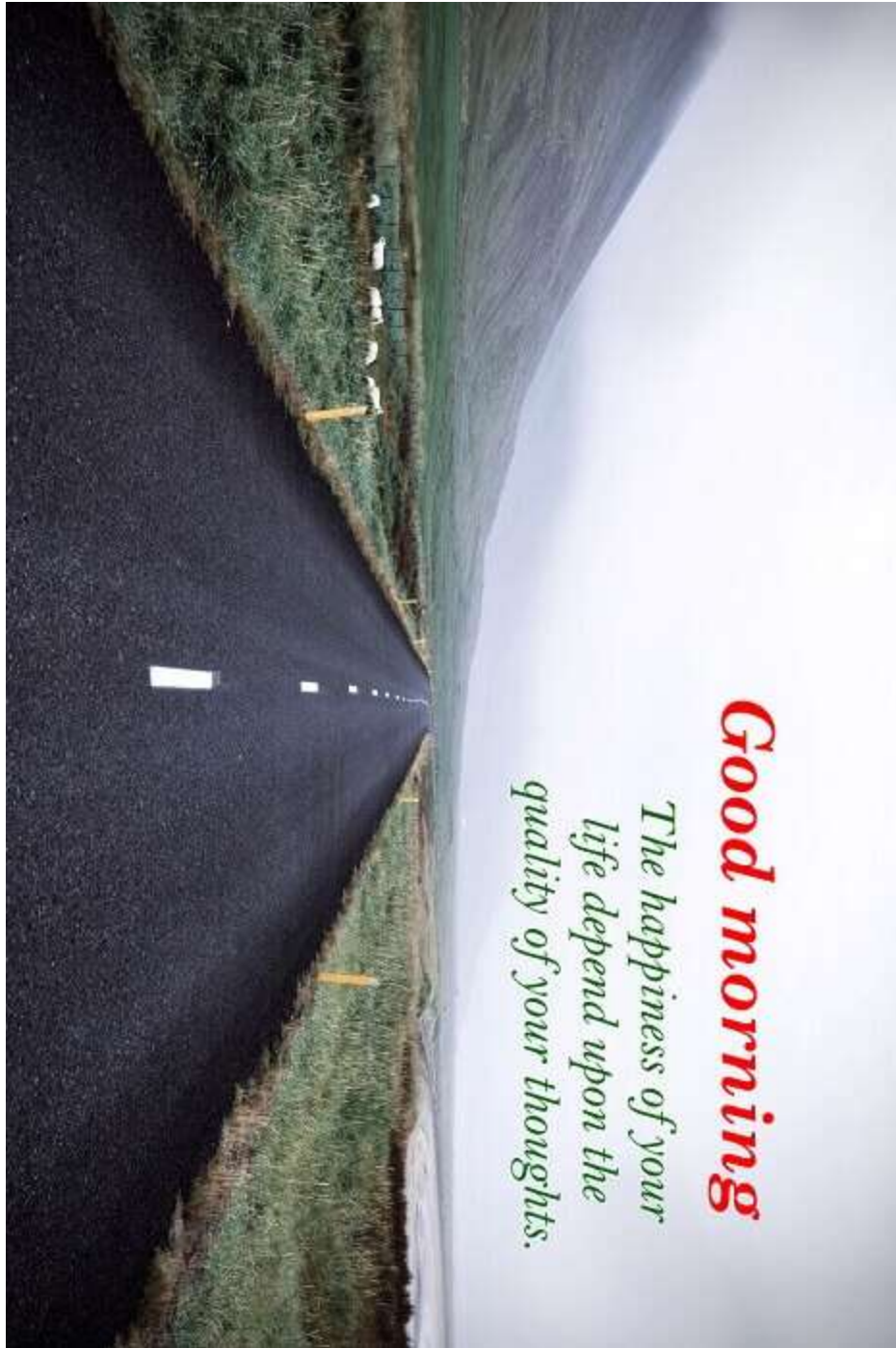
Rotate

Using toolbar

To rotate an image, click the rotate icon in the toolbar. This rotates the image to 90 degrees from the current state. By continuous clicking, angle will be increased since it is rotated from the current state.

Using code

Use the rotate method to rotate an image to 90 degrees from the current state.



Crop support in SfImageEditor

An image can be cropped using toolbar and programmatically.

Toolbar cropping

To crop an image, click the **Crop** icon in the toolbar. Cropping handles will be added on the image. You can resize the handles to crop the required portion.

By clicking the **Crop** icon, sub toolbar will be displayed below the main toolbar. After selecting the cropping portion, click **OK** in the sub toolbar to crop that selected portion.

The sub toolbar contains the following the icons:

- Width - Specifies the width of the cropping area.
- Height - Specifies the height of the cropping area.
- OK - Crops the selected portion when clicking OK.
- Cancel - Removes the cropping handles when clicking Cancel.

Note: On specifying width and height, cropping area is located from the left-top corner.

You can also perform continuous cropping on an image.

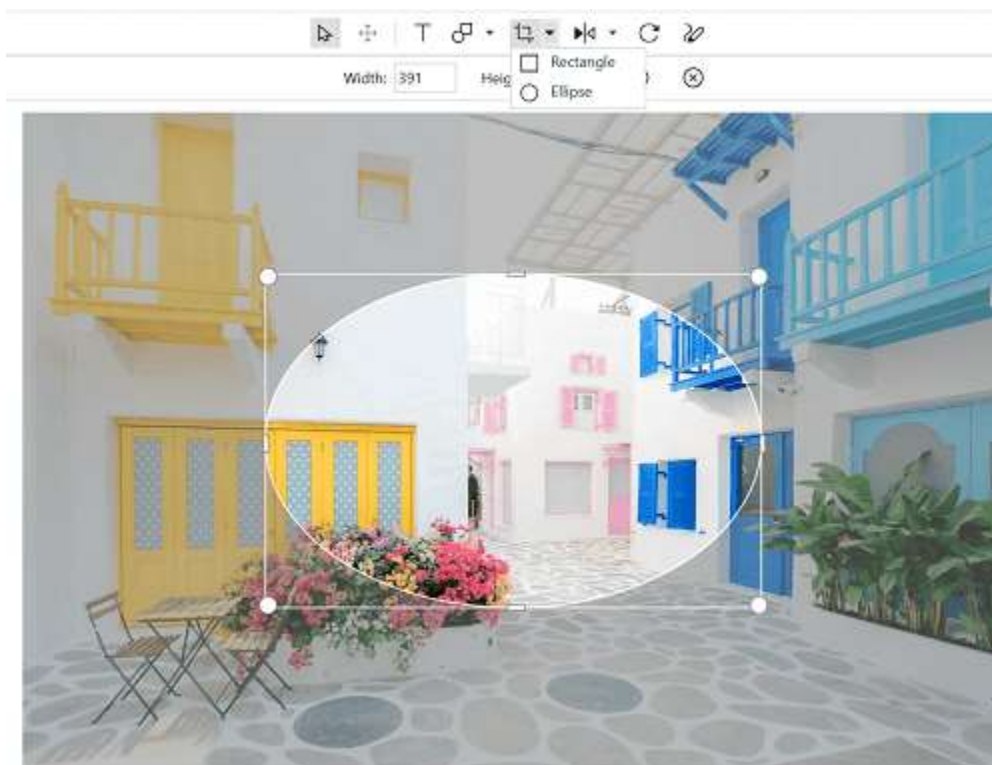
The following screenshot illustrates the area selected for cropping.



The following screenshot illustrates the image taken after performing cropping on the selected area.



An image can be cropped in rectangle and circle format. By default, rectangle format is enabled in the toolbar and you can also enable the circle format by using the drop-down button in crop icon.



Programmatic cropping

Cropping can be done programmatically using the following two methods in image editor:

- [ToggleCropping](#) - Selects the cropping area.
- [Crop](#) - Crops the selected area in an image.

Toggle cropping

[ToggleCropping](#) method selects the cropping area based on the specified parameters.

Crop area selection

The following method selects the full image area for cropping, and you can resize the image as needed.

C#

```
editor.ToggleCropping();
```

The following method gets the ratio as the parameter to select the cropping area.

C#

```
editor.ToggleCropping(3, 1);
```



The following method gets the rect in terms of percentage to select the cropping area.

C#

```
Rect rect = new Rect(20, 20, 50, 50);
```

```
editor.ToggleCropping(rect);
```



Circle cropping

Specify the [ToggleCropping](#) optional parameter is true as shown in the following code sample to enable the circle or elliptical format.

C#

```
editor.ToggleCropping(new Rect(25, 25, 50, 50), true);
```




Circle cropping with a ratio

To crop an image in a circle or an ellipse with a specific ratio, use [ToggleCropping](#) with a ratio argument and an optional parameter of true, which specifies whether the cropping panel should be added in an elliptical or rectangle shape. The default value is false.

C#

```
editor.ToggleCropping(1, 1, true);
```



Crop

After selecting the crop area, use the [crop](#) method in the image editor to crop the selected portion as demonstrated in the following method.

C#


```
editor.Crop(new Rect(0, 0, 0, 0))
```

Manual cropping

To manually select and crop the location, use the same [Crop](#) method, but specify the portion to be cropped in terms of rect as in the following code.

C#

```
editor.Crop(new Rect(0, 0, 0, 0))
```



See also

[How to crop an image based on the ratio in the image editor](#)

Save and Reset functionality in Image Editor

Image can be saved along with the changes. An image can be saved in the following two ways:

- Saving an image using the toolbar.
- Saving an image programmatically.

[Saving an image using the toolbar](#)

To save an image, click the save icon in the toolbar. You can choose the desired location from the save file dialog to save the edited image.

[Saving an image programmatically](#)

An image can be saved programmatically using the Save method in image editor as follows. The save method takes the following three parameters:

- ImageFormat - Specifies the format in which image has to be saved.
- Size - Saves the image in the specified size.
- FilePath - Specifies the location in which image has to be saved.

C#

```
editor.Save();
```

[Image size specification](#)

You can save an image with the required size by specifying the size parameter in the save method as follows.

C#

```
editor.Save(null, new Size(750, 500), null);
```

[Image location](#)

Image can be saved at the specified location as demonstrated in the following code.

C#

```
editor.Save(null, default(Size), @"E:\Images\");
```

[Image format](#)

You can specify the format, in which image has to be saved as the parameter in Save method as follows.

C#

```
editor.Save("png", default(Size), null);
```

[Reset](#)

To reset the changes made, click the Reset icon in the toolbar. To programmatically reset, use the following code.

C#

```
editor.Reset();
```

Events

Saving an events

Image editor has the following two events:

- [ImageSaving](#)
- [ImageSaved](#)

Image saving

This event occurs before the image is saved to the destination location. [ImageSavingEventArgs](#) is the parameter. This argument contains the following two properties:

- **Cancel** - Cancels the saving functionality.
- **Stream** - Stream of the image that is going to be saved.
- **FileName** - You can save the image in the specified name using the **ImageSaving** event.

Hence, you can control the saving using the **Cancel** property, and you can also access the stream as needed.

The following code cancels the default saving and saves the stream in the specified location.

XML

```
<editor:SfImageEditor x:Name="editor" ImageSaving="editor_ImageSaving"
ImageSource="Assets\Buldingimage.jpeg">
</editor:SfImageEditor>
```

C#

```
private void Editor_ImageSaving(object sender, ImageSavingEventArgs args)
{
    args.Cancel = true;
    FileStream stream = new FileStream(@"E:\Images\Resized.jpg",
    FileMode.Create);
    args.Stream.CopyTo(stream);
    stream.Seek(0, 0);
    args.FileName = "SavedImage";
}
```

Image saved

This event occurs after the image has been saved to the destination location. The [ImageSavedEventArgs](#) will be the parameter. This parameter contains the [Location](#) property, which specifies the location in which image is saved.

C#

```
private void Editor_ImageSaved(object sender, ImageSavedEventArgs args)
{
    string filePath = args.Location;
}
```

Reset events

The Reset functionality has the following two events:

- [BeginReset](#)
- [EndReset](#)

Begin reset

This event occurs before resetting the changes. Hence, you can control the reset operation using the Cancel property in the [BeginResetEventArgs](#).

C#

```
private void Editor_BeginReset(object sender, BeginResetEventArgs e)
{
    e.Cancel = true;
}
```

End reset

This event occurs after the reset function has been completed.

C#

```
private void Editor_EndReset(object sender, EndResetEventArgs e)
{
}
```

See also

[How to capture and save signature using the image editor](#)

Undo and Redo Supports

The undo and redo supports help in performing the edited image transition from current state to previous state or from current state to next state.

All the editing operation related to the shapes will be stored, hence you can easily go to the previous state or next state. This can be done using toolbar or programmatically.

Note: Currently, the undo and redo support only for shapes and text.

The undo and redo supports perform the following operations:

- Add or delete shapes/text
- Change positions
- Fill/Stroke changes

Undo

In the left side of the toolbar, you can find the undo and redo icons. Both these icons will be in disabled state by default. These icons will be enabled only when the changes are done in the images.

Undo clears the last performed change on the image. By performing undo continuously, you can reach the initial state (i.e., before changes related to shapes and text were done).

Programmatically, undo can be performed using the undo method as demonstrated in following code snippet.

C#

```
editor.Undo();
```

Redo

The redo icon is enabled only when an undo operation is performed. Redo will bring the changes excluded using the undo operation.

Programmatically, redo can be performed using the undo method as demonstrated in the following code snippet.

C#

```
editor.Redo();
```

Zooming and Panning in WPF ImageEditor (SfImageEditor) control

Zooming

Toolbar

Images can be zoomed in or zoomed out for better viewing and this can be enabled using the [EnableZooming](#). In the footer toolbar, you can find the slider, which helps in increasing the zoom level of an image.

Zoom level ranges from 50 to 400 percents. You can move the slider to increase the zoom level. Also, there will be Increase and Decrease icons on both sides of the slider. These icons help in increasing of the level gradually.

At a time, this Increase/Decrease icon can increase/decrease the level upto 10 percent. To reset the zoom level, click the ResetZoom icon, which is placed at the left of the DecreaseZoom icon.

Mouse wheel

You can also zoom an image using the mouse wheel. Based on the mouse wheel, delta images will be zoomed from the cursor position.



Panning

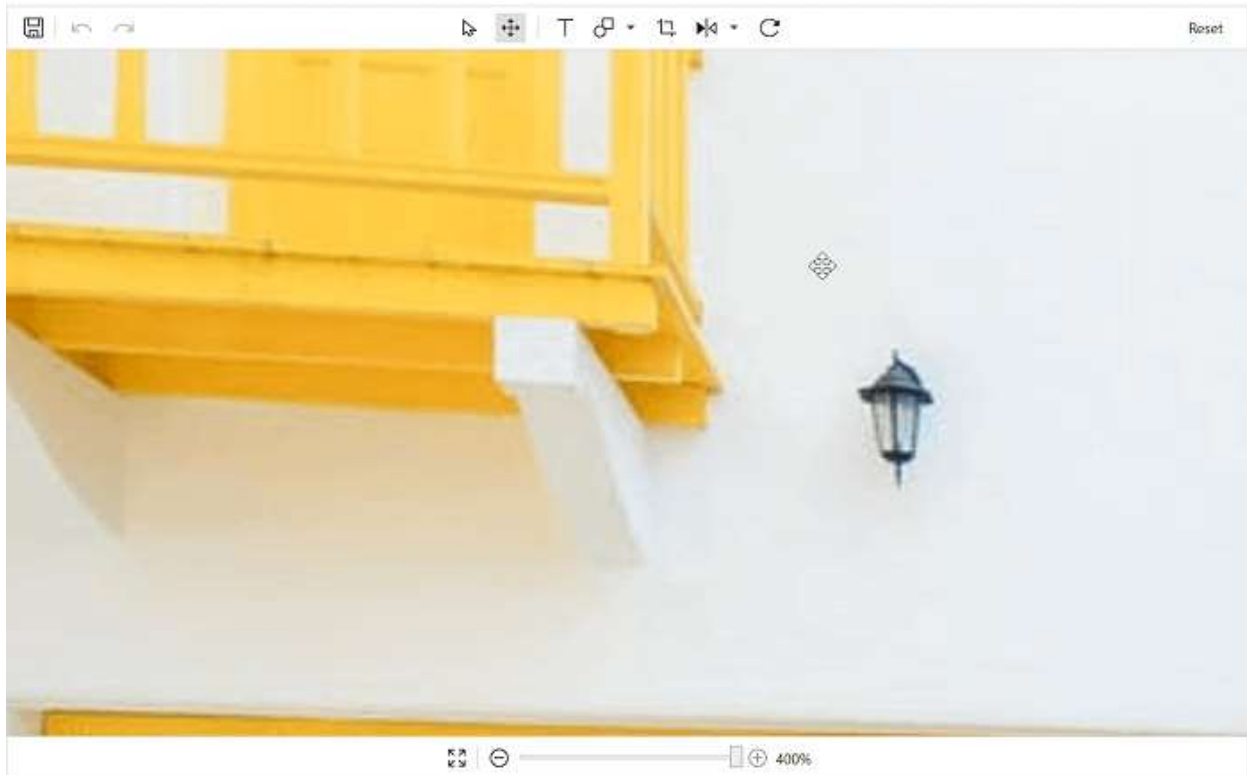
With toolbar

Zoomed image can be panned to view the hidden portion. To enable pan, click the pan icon in the top toolbar. This enables the panning operation on the image. When panning is enabled, shapes or text added in the image cannot be resized or repositioned. To resize the shape, enable the Select icon in the toolbar; it will disable the pan operation.

Select and Pan operations work like toggle functions.

Without toolbar (Programmatically)

Panning can be enabled using the `EnablePanning` property rather than the toolbar icons. When panning is enabled, shapes and tests cannot be selected. By default, the property is set to false.



Toolbar Customization

Customization

Visibility

Toolbar can be made visible or hidden using the `IsToolbarVisibility` property in the [ToolbarSettings](#).

XML

```
<editor:SfImageEditor.ToolbarSettings>
<editor:ToolbarSettings IsToolbarVisibility="True" />
</editor:SfImageEditor.ToolbarSettings>
```

C#

```
editor.ToolbarSettings.IsToolbarVisibility = true;
```

Add a item

You can add additional items to the toolbar and can perform your own operation. To add an additional item, specify the toolbar item, and add it in the `ToolbarItems` collection as demonstrated in following code snippet. You can specify your own template using the `IconTemplate` property in `ToolbarItem`.

XML

```
<Grid.Resources>
<DataTemplate x:Key="template">
<TextBlock Text="New Item"></TextBlock>
</DataTemplate>
</Grid.Resources>
```

C#

```
editor.ToolbarSettings.ToolbarItems.Add(new ToolbarItem() { IconTemplate =  
grid.Resources["template"] as DataTemplate });
```

Customization

You can change the [Background](#) and [BorderColor](#) of the toolbar. Also, you can change the height of the main toolbar using the [HeaderToolbarHeight](#) property, and the height of the sub toolbar can be changed using the [SubItemToolbarHeight](#) property, and the footer toolbar height can be changed using the [FooterToolbarHeight](#).

This can be done as in the following code snippet.

XML

```
<editor:SfImageEditor.ToolbarSettings>  
<editor:ToolBarSettings IsToolBarVisibility="True" HeaderToolbarHeight="36"  
FooterToolbarHeight="36" Background="#DEDEDE" BorderColor="Black"  
</editor:SfImageEditor.ToolbarSettings>
```

C#

```
editor.ToolbarSettings.Background= (SolidColorBrush) new  
BrushConverter().ConvertFromString("#DEDEDE");  
editor.ToolbarSettings.BorderColor = new SolidColorBrush(Colors.Black);  
editor.ToolbarSettings.HeaderToolbarHeight = 36;  
editor.ToolbarSettings.FooterToolbarHeight = 36;  
editor.ToolbarSettings.IsToolBarVisibility = true;
```




Events

ToolbarItemSelected

This event occurs when an item in the toolbar is selected. `ToolbarItemSelectedEventArgs` is the parameter. You can control the selected item operation by setting the `Cancel` property to `true`. You can also get the information about the `ToolbarItem`.

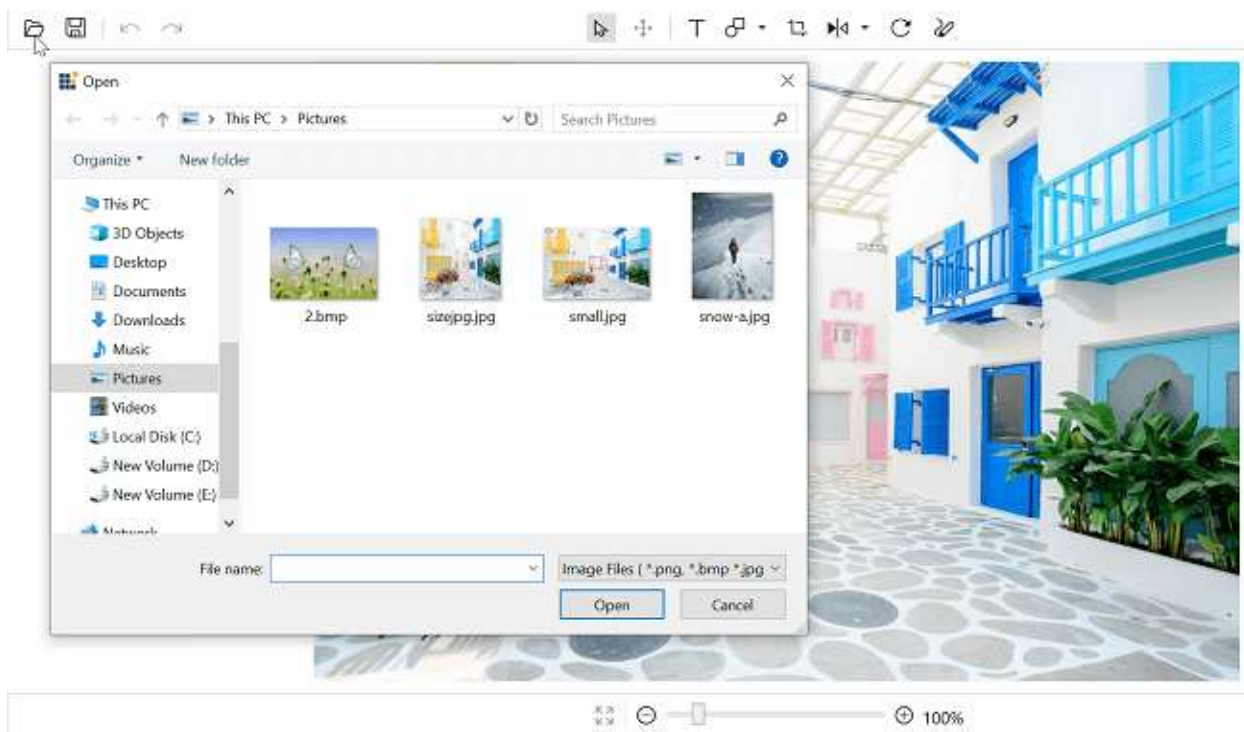
C#

```
private void ToolbarSettings_ToolbarItemSelected(object sender,
ToolbarItemSelectedEventArgs e)
{
    e.Cancel = true;
}
```



Image picker support

You can browse images in a local folder and load them in the Image Editor using the toolbar item browse icon.



Commands

Invoke commands from the custom toolbar to customize toolbar items of the image editor. Must set the **CommandTarget** while using the Command.

Command	Description
BrowseImage	Browses the local folder to pick and load the image to an image editor.
Save	Saves the edited image.
Undo	Reverses the last performed action.
Redo	Restores the actions carried out by Undo.
Reset	Clears all the editing done on the image and brings to an initial state.
ResetZoom	Resets the zooming applied to the image.
IncreaseZoom	Zoom In the image by increasing the zoom level from its current state.
DecreaseZoom	Zoom Out the image by decreasing the zoom level from its current state.
Select	Used to select the items such as Shapes, Text, Custom view added on the image.
Pan	Used to Pan the image when it is in zoomed state.

This can be done as in the below code snippet.

XML

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="200" />
  </Grid.ColumnDefinitions>
  <editor:SfImageEditor Grid.Column="0" x:Name="imageEditor"
    ImageSource="Assets\RoadView.jpeg">
    <editor:SfImageEditor.ToolbarSettings>
      <editor:ToolbarSettings IsToolbarVisiblity="False"></editor:ToolbarSettings>
    </editor:SfImageEditor.ToolbarSettings>
  </editor:SfImageEditor>
  <Button Grid.Column="1" HorizontalAlignment="Center"
    VerticalAlignment="Center" Background="White"
    Width="Auto" CommandTarget="{Binding ElementName=imageEditor}"
    Content="Save" Command="{x:Static
      editor:ImageEditorCommands.Save}"></Button>
</Grid>
```



Z ordering in WPF ImageEditor (SfImageEditor)

The image editor allows you to change the position of shapes that are arranged in the editor. This can be achieved using the following methods:

1. BringToFront
2. SendToBack
3. BringForward
4. SendBackward

BringToFront

This method brings the selected shape to the front of all the shapes added in the image.

C#

```
editor.BringToFront();
```

BringForward

This method brings the selected shape to one step forward from its current position.

C#

```
editor.BringForward();
```

SendToBack

This method pushes the selected shape to the back of all the shapes added.

C#

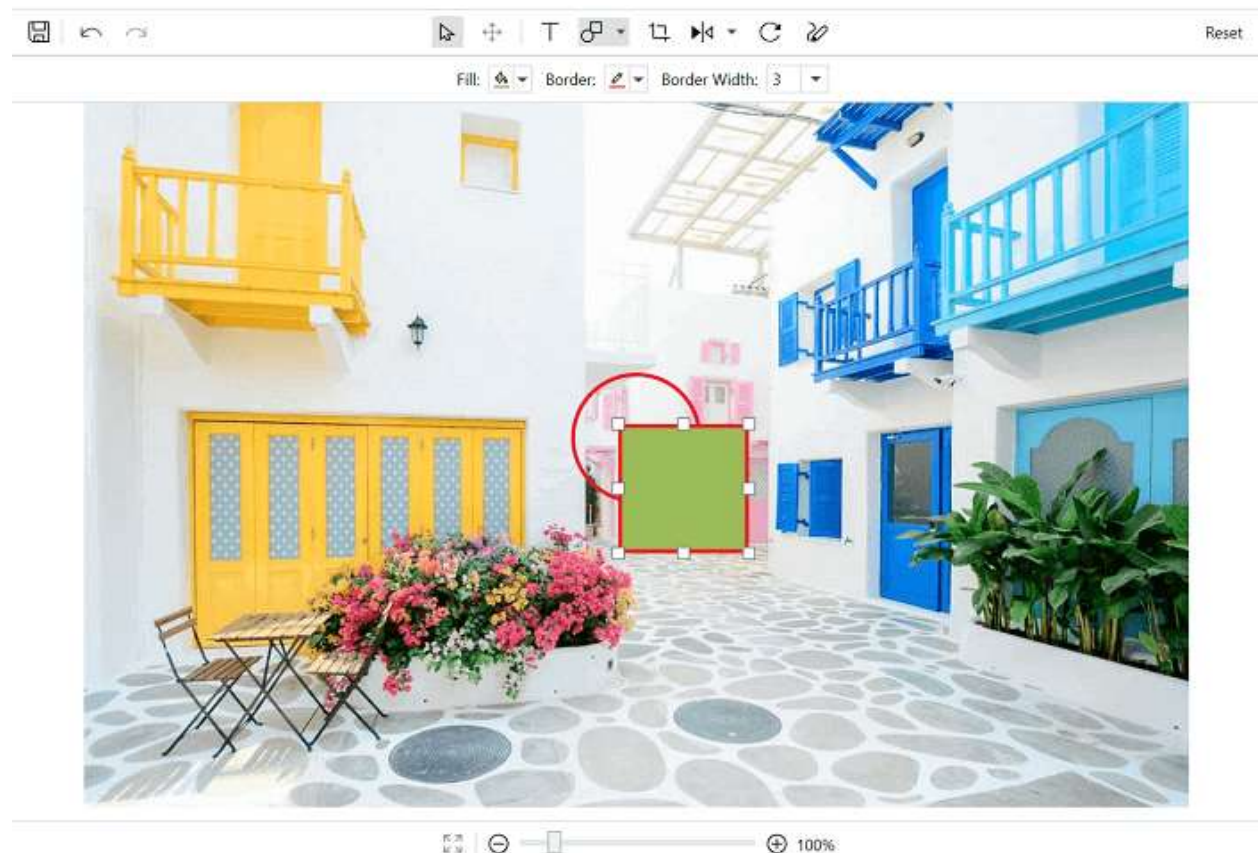

```
editor.SendToBack();
```

SendBackward

This method pushes the selected shape one step back from its current position.

C#

```
editor.SendBackward();
```



Localization in WPF ImageEditor (SfImageEditor)

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the SfImageEditor by adding resource file. Application culture can be changed by setting `CurrentUICulture` before `InitializeComponent()` method.

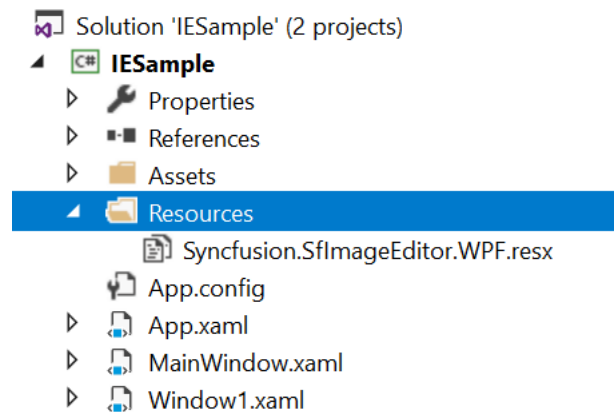
Use the below code to change the application culture to French.

C#

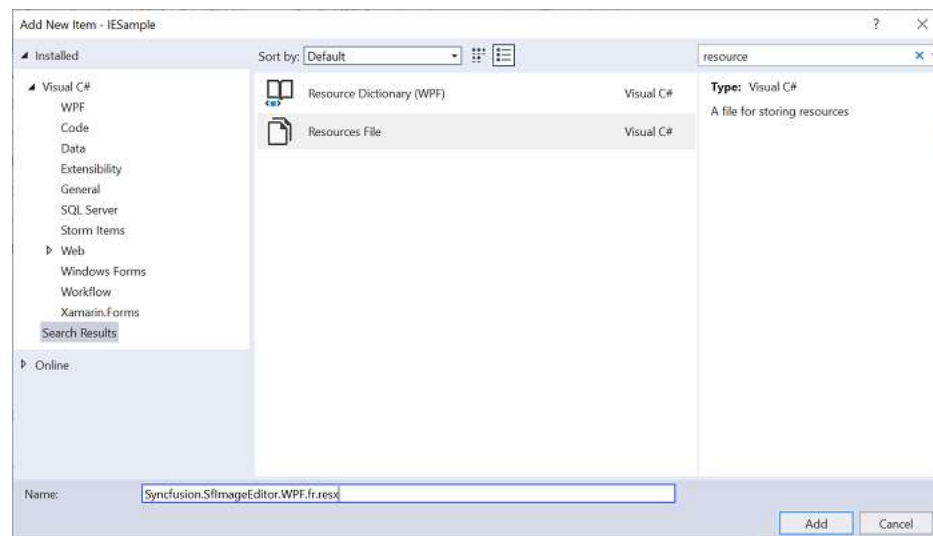
```
public MainWindow()
{
    Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("fr-FR");
    InitializeComponent();
}
```

To localize SfImageEditor based on `CurrentUICulture` using resource files, follow the below steps.

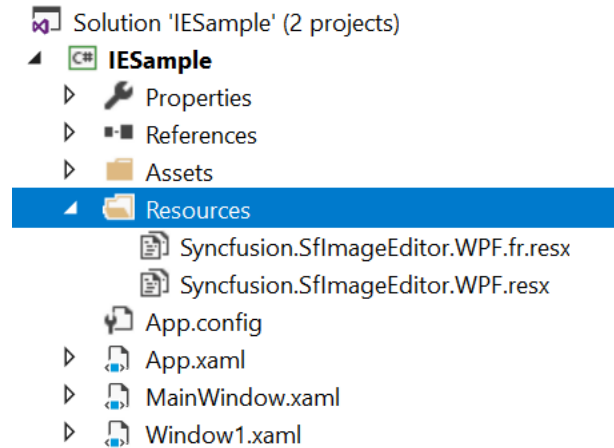
1. Create new folder and name it as **Resources** in your application. 2. Add the default resource file of SfImageEditor into **Resources** folder. You can download the Syncfusion.SfImageEditor.WPF.resx [here](#).



3. Right-click on the Resources folder, select Add and thenNewItem. 4. In Add New Item wizard, select the Resource File option and name it as Syncfusion.SfImageEditor.WPF..resx. For example, you have to give name as Syncfusion.SfImageEditor.WPF.fr.resx for French culture.



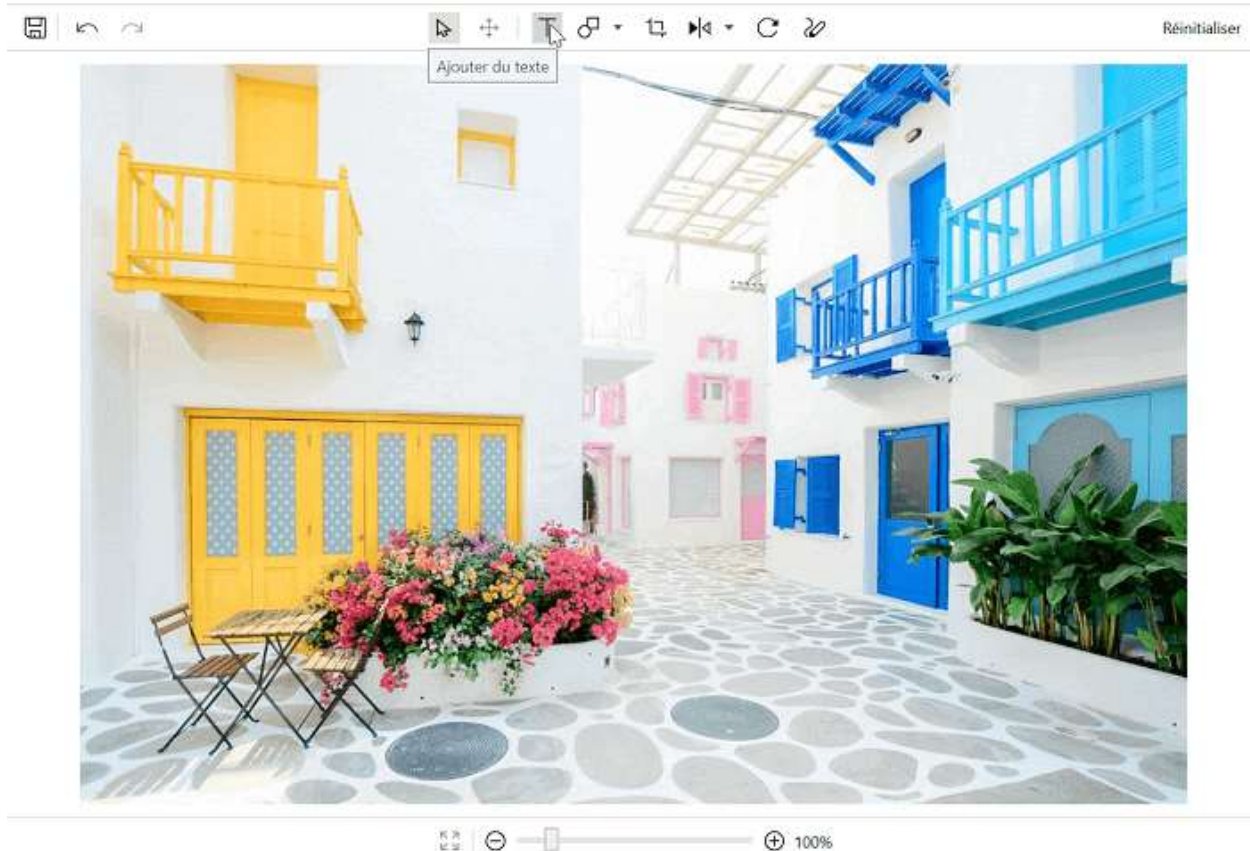
5. Now, select Add option to add the resource file in the Resources folder.



6. Add the Name/Value pair in Resource Designer of Syncfusion.SfImageEditor.WPF.fr.resx file and change its corresponding value to the corresponding culture.

Name	Value	Comment
AddShape	Add Shape	
AddText	Add Text	
AlignLeft	Align Left	
AlignRight	Align Right	
Arrow	Arrow	
Automatic	Automatic	
Bold	Bold	
Border	Border	
BorderWidth	Border Width	
Cancel	Cancel	
Center	Center	
Circle	Circle	
Crop	Crop	
CropHeight	Height. Click Enter to change crop height.	
CropWidth	Width. Click Enter to change crop width	
DefaultText	Enter Text	
Fill	Fill	
Flip	Flip	
FlipHorizontal	Flip Horizontal	
FlipVertical	Flip Vertical	
Font	Font	

Below image depicts the application translated into French culture.



CustomView in SfImageEditor

This feature allows you to add a custom view in the Image Editor and provide the different customization options.

Add a custom view on image editor

You can add any custom shapes or views to an image using the `AddCustomView` method in the Image Editor control. To add a custom view, specify the view and its desired `CustomViewSettings` as shown in the following code snippet.

XML

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="100"/>
    <RowDefinition Height="*/>
  </Grid.RowDefinitions>
  <Button x:Name="but" Click="Button_Click" Content="CustomView" />
  <editor:SfImageEditor Grid.Row="1" x:Name="editor"
    ImageSource="Assets\RoadView.jpeg" >
  </editor:SfImageEditor>
</Grid>
```

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
```



```
BitmapImage bitmapImage = new BitmapImage();  
bitmapImage.BeginInit();  
bitmapImage.UriSource = new Uri(@"Assets/adventure.jpg", UriKind.Relative);  
bitmapImage.EndInit();  
Image image = new Image() { Height = 100, Width = 100, Source=bitmapImage}  
;  
editor.AddCustomView(image, new CustomViewSettings());  
}
```

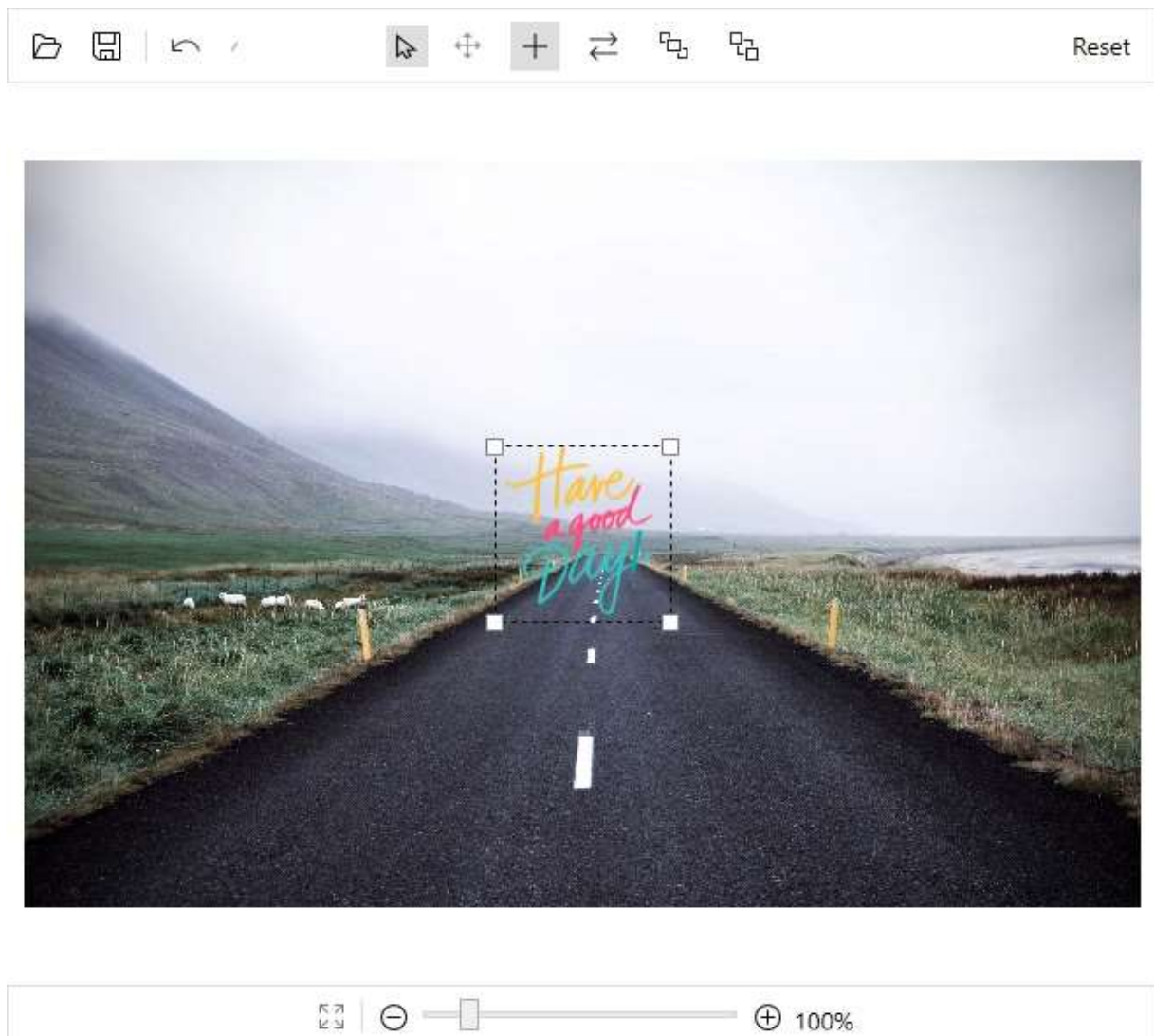
Customize the custom view

The CustomViewSettings is defined to set the values for **CanMaintainAspectRatio**, **Bounds**, and **Angle**.

- **CanMaintainAspectRatio** property is used to decide, whether the aspect ratio value needs to be maintained when resizing the custom view.
- **Bounds** property is used to set the bounds of the custom view. Using this property, you can position the custom view wherever you want on the image. In percentage, the value should fall between 0 and 100.
- **Angle** property is used to set the angle of the custom view. Using this property, you can rotate the custom view at desired angle.
- **IsResizable** property is indicating, whether to resize the custom view or not. Default value of IsResizable is true.
- **IsRotatable** property is indicating, whether to rotate the custom view or not. Default value of IsRotatable is false.

C#

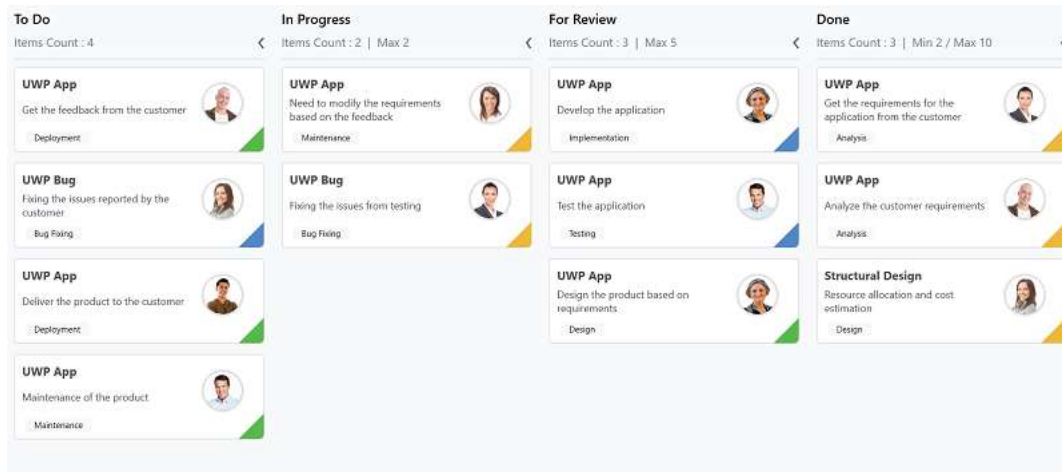
```
CustomViewSettings customViewSettings = new CustomViewSettings()  
{  
    CanMaintainAspectRatio = false,  
    Bounds = new Rect(0, 0, 100, 100),  
    IsResizable = false,  
    IsRotatable=true,  
    Angle = 45  
};
```



SfKanban

[WPF Kanban \(SfKanban\) Overview](#)

The Kanban control is an efficient way to visualize a workflow at each stage of completion. Kanban helps to define elegant planning and clear visualization of work progression. SfKanban also provide many features that are used to monitor the progressing tasks in software development cycle.



Key features

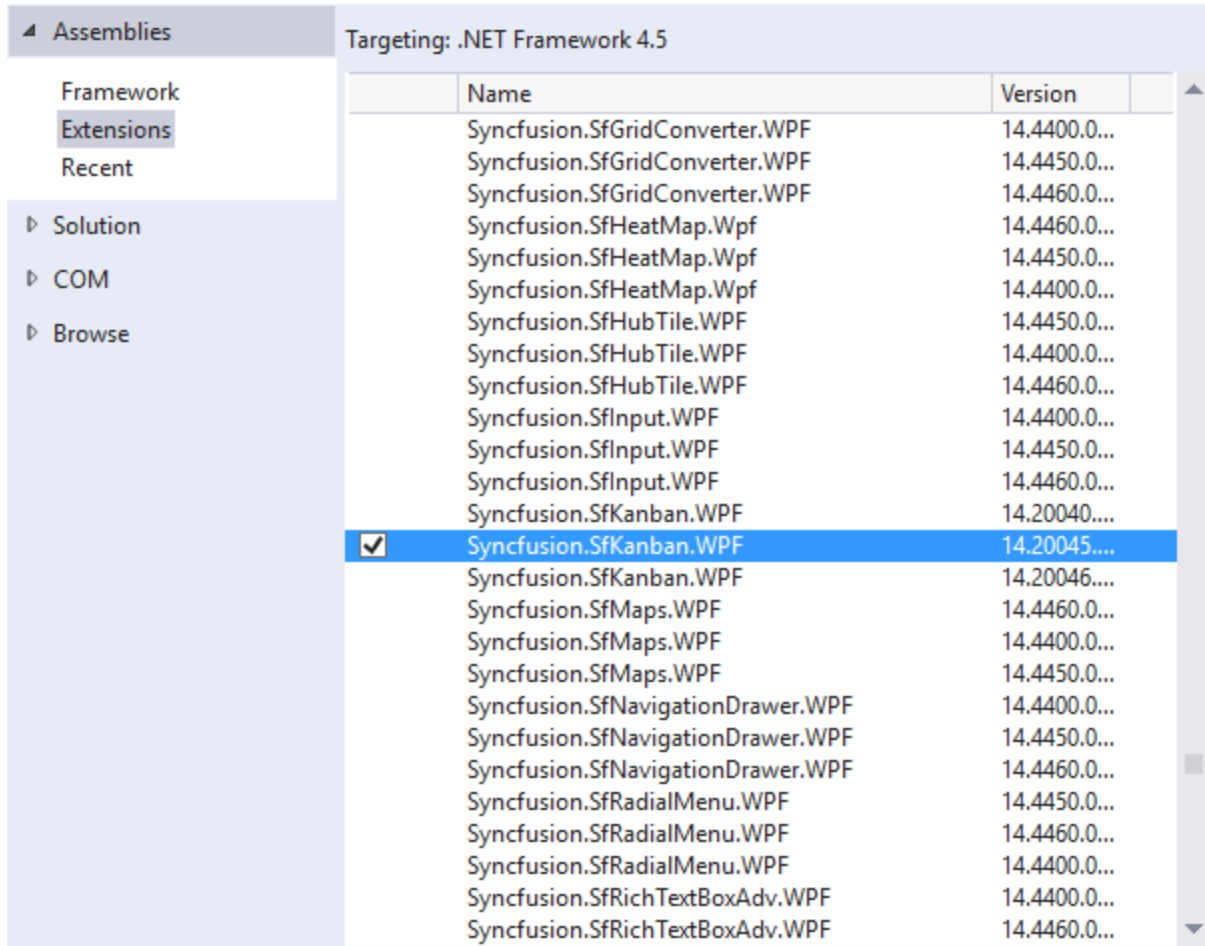
- **WorkFlows** – It allows user to control the transitions between tasks from one category (status) to another.
- **Customization** – Kanban cards and place holders can be customized.
- **WIP Limit** – Visualize Work-In Progress limit.
- **Indicator color customization** – Indicator color for Kanban cards can be customized.
- **Dragging events** – Dragging events occurred while rearranging or repositioning Kanban cards.

Getting Started with WPF Kanban(SfKanban)

The following section provides an assistance to create a simple Kanban application and to configure it.

Adding assembly reference

1. Open the Add Reference window from your project
2. Choose Assemblies -> Extensions -> Syncfusion.SfKanban.WPF



Note: This window differs for the Visual Basic project.

Create a simple Kanban

In this walk through, you will create a new application that contains the SfKanban which includes the below topics.

- Adding SfKanban
- Create data model
- Binding data
- Defining columns
- Working with Workflows
- Work In-Progress Limit

Adding SfKanban

1. Add the required assembly references to the project as discussed in the Reference Essential Studio Components in your Solution section.
2. Add the "Syncfusion.UI.Xaml.Kanban" namespace to the application as shown below.

XML

```
xmlns:syncfusion="clr-namespace:Syncfusion.UI.Xaml.Kanban;assembly=Syncfusion.SfKanban.WPF"
```

C#

```
using Syncfusion.UI.Xaml.Kanban;
```

3. Create an instance of SfKanban control.

XML

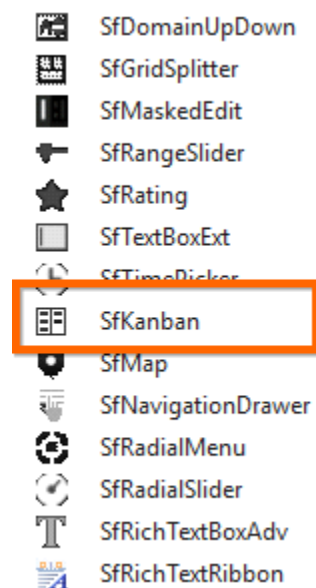
```
<syncfusion:SfKanban>  
</syncfusion:SfKanban>
```

C#

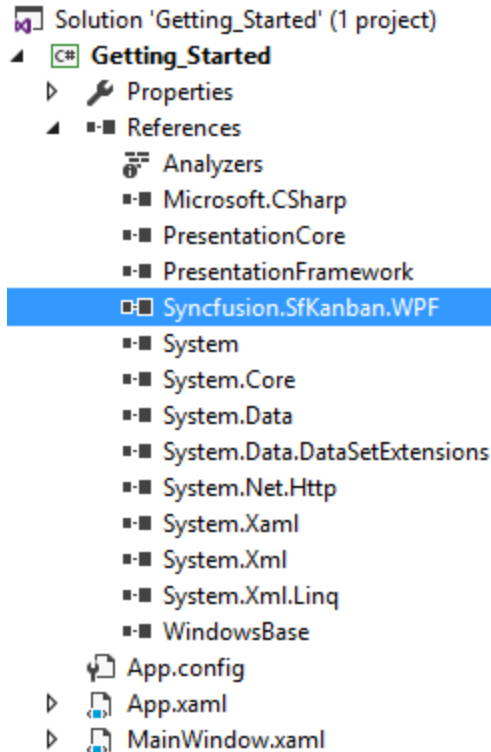
```
SfKanban kanban = new SfKanban();
```

Adding SfKanban from toolbox

Drag and drop the Kanban control from the toolbox to your application.



Now the Syncfusion.SfKanban.WPF reference is added to the application references and the xmlns namespace code is generated in MainWindow.xaml as below.



```
<Window x:Class="Getting_Started.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:Getting_Started"
        xmlns:syncfusion="clr-namespace:Syncfusion.UI.Xaml.Kanban;assembly=Syncfusion.SfKanban.WPF"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <syncfusion:SfKanban>

        </syncfusion:SfKanban>
    </Grid>
</Window>
```

Create data model

You need to create a collection of KanbanModel objects for populating SfKanban.

C#

```
public class TaskDetails
{
    public TaskDetails()
    {
        Tasks = new ObservableCollection<KanbanModel>();
        Tasks.Add(new KanbanModel()
        {
            Title = "Universal App",
            ID = "27654",
            Description = "Incorporate feedback into functional specifications",
```

```

Category = "Open",
ColorKey = "Low",
Tags = new string[] { "Deployment" },
ImageUrl = new Uri("/images/icon.jpg", UriKind.RelativeOrAbsolute )
});
Tasks.Add(new KanbanModel()
{
    Title = "Universal App",
    ID = "29477",
    Description = "Design functional specifications",
    Category = "In Progress",
    ColorKey = "Normal",
    Tags = new string[] { "Design" },
    ImageUrl = new Uri("/images/icon.jpg", UriKind.RelativeOrAbsolute )
});
Tasks.Add(new KanbanModel()
{
    Title = "Universal App",
    ID = "25678",
    Description = "Review preliminary software specifications",
    Category = "Done",
    ColorKey = "Low",
    Tags = new string[] { "Analysis" },
    ImageUrl = new Uri("/images/icon.jpg", UriKind.RelativeOrAbsolute )
});
Tasks.Add(new KanbanModel()
{
    Title = "Universal App",
    ID = "6593",
    Description = "Draft preliminary software specifications",
    Category = "Review",
    ColorKey = "High",
    Tags = new string[] { "Analysis" },
    ImageUrl = new Uri("/images/icon.jpg", UriKind.RelativeOrAbsolute )
});
}
public ObservableCollection<KanbanModel> Tasks { get; set; }
}

```

Binding data

In order to bind the data source of the SfKanban, set ItemsSource property as shown below.

XML

```
<syncfusion:SfKanban ItemsSource="{Binding Tasks}" />
```

C#

```

SfKanban kanban = new SfKanban()
{
    ItemsSource = new TaskDetails().Tasks
};

```

Defining columns

By default, we need to define the columns manually by adding the KanbanColumn object to the [Columns](#) collection property in SfKanban.

ItemsSource which was bound to the Kanban will be added to the respective columns using ColumnMappingPath property in SfKanban and Categories property in KanbanColumn.

We need to set the required property name to ColumnMappingPath which will be essential to add the data to the respective columns.

In this example, the data whose Category property's value is set as Open will be added to the 'To Do' Column and other data will be added to the respective columns.

The following code example illustrates how this can be done.

XML

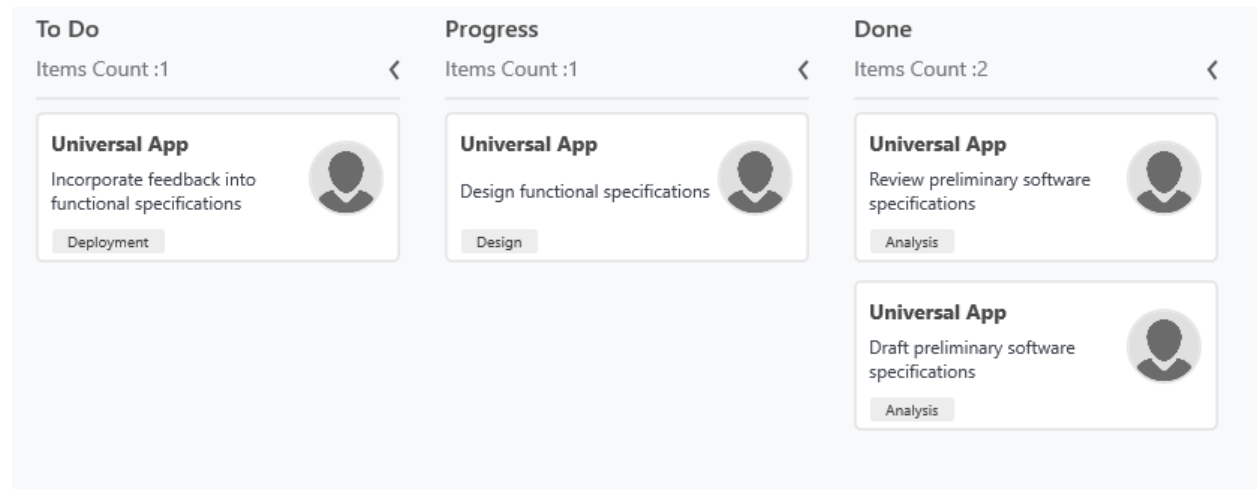
```
<syncfusion:SfKanban MinColumnWidth="150"
ColumnMappingPath="Category"
ItemsSource="{Binding Tasks}"
AutoGenerateColumns="False">
<syncfusion:KanbanColumn Categories="Open" Title="To
Do"></syncfusion:KanbanColumn>
<syncfusion:KanbanColumn Categories="In Progress"
Title="Progress"></syncfusion:KanbanColumn>
<syncfusion:KanbanColumn Categories="Review, Done"
Title="Done"></syncfusion:KanbanColumn>
</syncfusion:SfKanban>
```

C#

```
SfKanban kanban = new SfKanban()
{
    AutoGenerateColumns = false,
    ItemsSource = new TaskDetails().Tasks
};
kanban.Columns.Add(new KanbanColumn()
{
    Categories = "Open",
    Title = "To Do",
    MinimumLimit = 1,
    MaximumLimit = 2,
});
kanban.Columns.Add(new KanbanColumn()
{
    Categories = "In Progress",
    Title = "Progress",
    MinimumLimit = 1,
    MaximumLimit = 2
});
kanban.Columns.Add(new KanbanColumn()
{
    Categories = "Review, Done",
    Title = "Done",
    MinimumLimit = 1,
    MaximumLimit = 2
});
```



```
grid.Children.Add(kanban);
```



You can also set [AutoGenerateColumns](#) property to true in which you don't need to define the columns as mentioned in the above example. This will create columns depending on the ColumnMappingPath property for all the distinct values in ItemsSource.

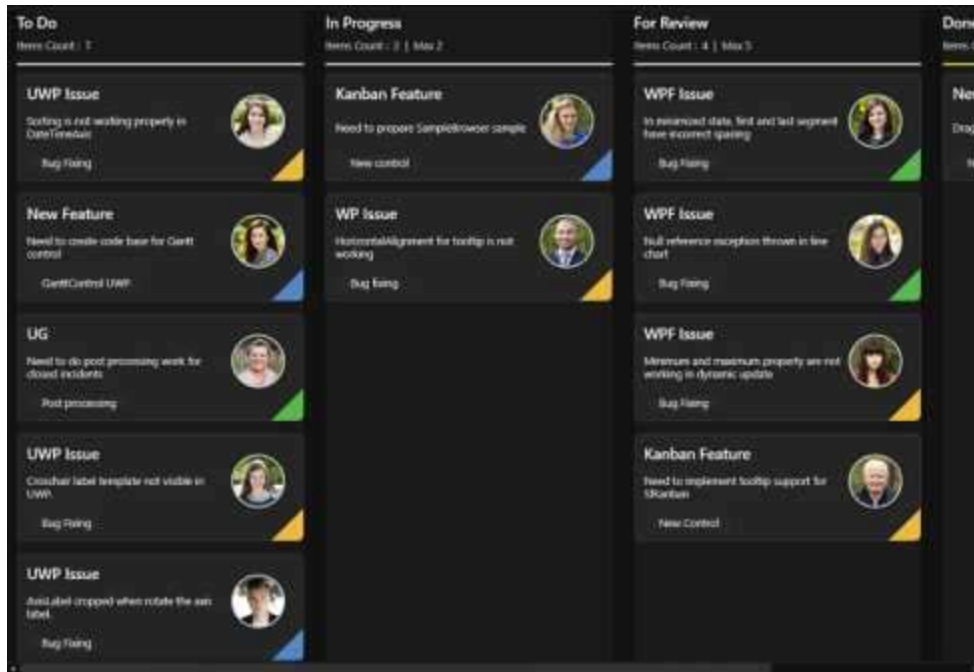
You can find the complete getting started sample from this [link](#).

Note: When the columns are auto-generated, you can handle the ColumnsGenerated event to customize the columns.

Theme

Kanban control supports various built-in themes. Refer to the below links to apply themes for the Kanban control,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Column in WPF Kanban (SfKanban) control

Customizing column size

By default, the columns are sized smartly to arrange the default elements of the cards with better readability. You can also define the minimum and maximum widths for the columns in [SfKanban](#) using the [SfKanban.MinColumnWidth](#) and [SfKanban.MaxColumnWidth](#) properties, respectively.

XML

```
<kanban:SfKanban MinColumnWidth="300" MaxColumnWidth="340"/>
```

C#

```
kanban.MinColumnWidth = 300;
kanban.MaxColumnWidth = 340;
```

You can also define the exact column width using the [SfKanban.ColumnWidth](#) property.

XML

```
<kanban:SfKanban ColumnWidth="250"/>
```

C#

```
kanban.ColumnWidth = 250;
```

Categorizing columns

The cards are categorized and added into the respective columns using the value of the [KanbanModel.Category](#) property if the [ItemsSource](#) is the collection of KanbanModel. But, if the [ItemsSource](#) is populated with custom objects, the property from the custom object used to categorize the card should be defined explicitly using the [ColumnMappingPath](#) property.

XML

```
<kanban:SfKanban ColumnMappingPath="Group"/>
```

C#

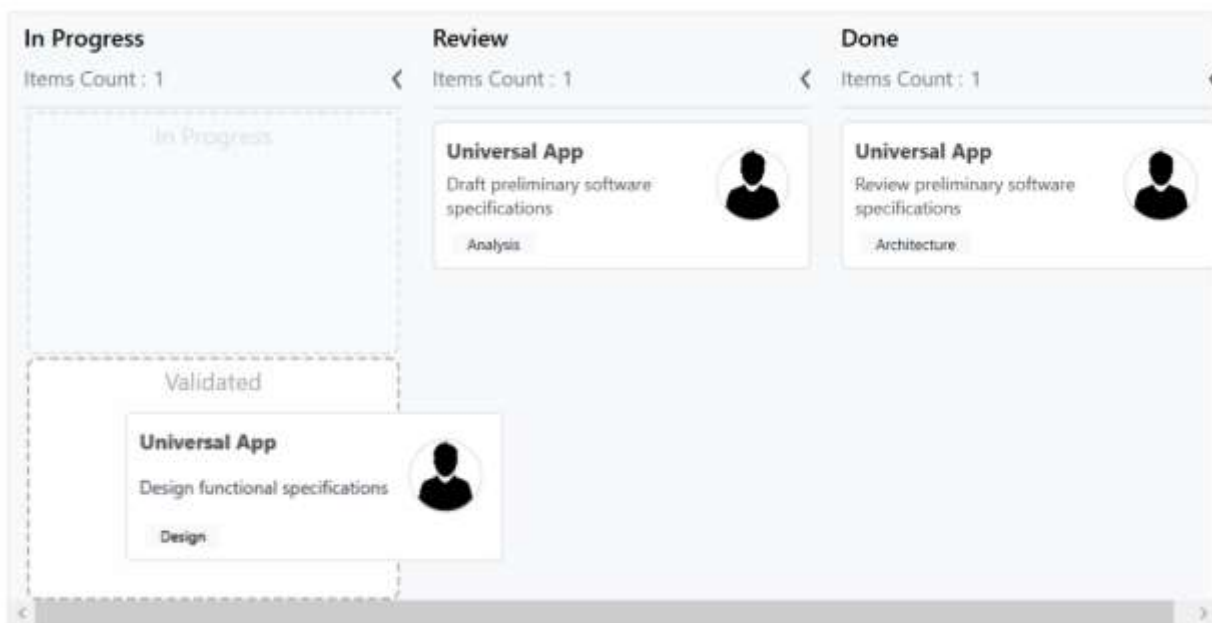
```
kanban.ColumnMappingPath = "Group";
```

Populate the column with cards from different categories

More than one category can be mapped to a column by assigning multiple values to the '[Categories](#)' collection of [KanbanColumn](#). For example, you can map the "In Progress" and "Validated" types under the "In progress" column.

C#

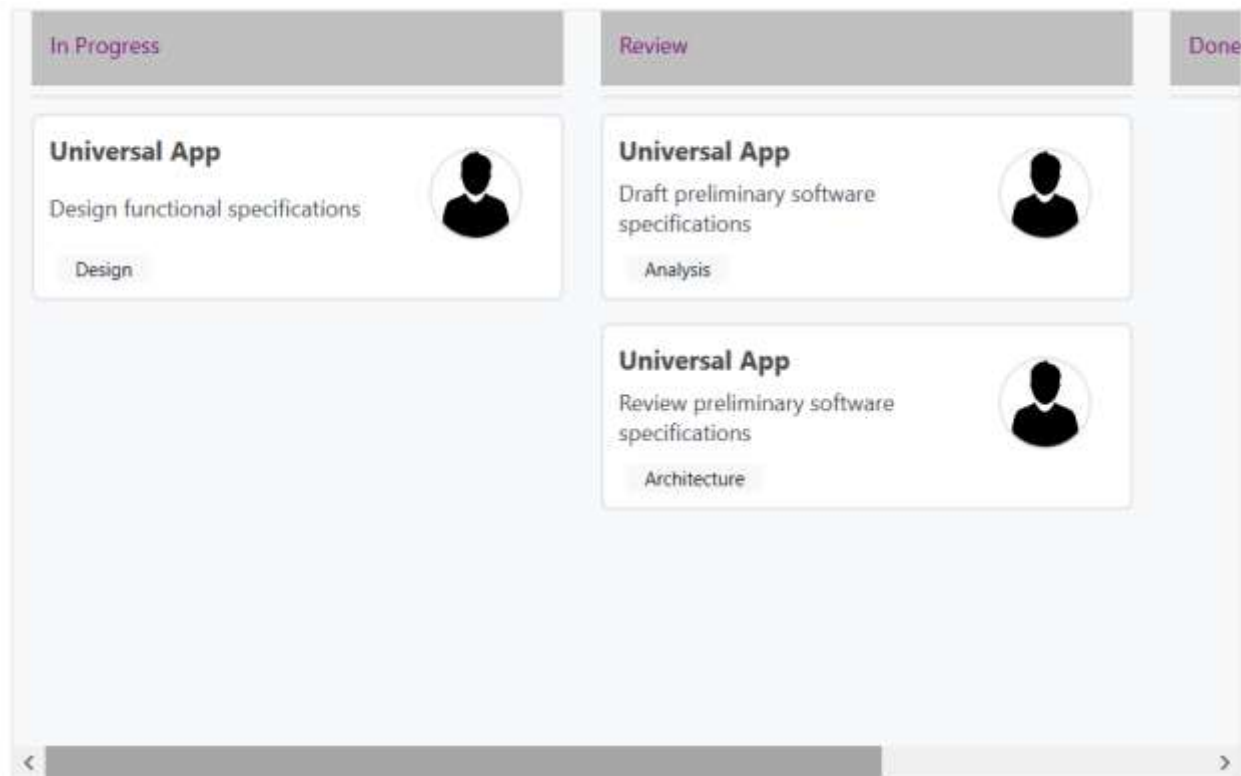
```
progressColumn.Categories = new List<object>() { "In Progress", "Validated" };;
```

**Headers**

[Header](#) shows the [Title](#) category, items count and the min and max informations of a column. The UI of the header can be replaced entirely using the [SfKanban.ColumnHeaderTemplate](#) property. The following code sample and screenshot illustrates this.

XML

```
<kanban:SfKanban.ColumnHeaderTemplate>
  <DataTemplate>
    <StackPanel Width="300" Height="40" Background="Silver">
      <TextBlock Margin="10" Text="{Binding Header}" Foreground="Purple"
        HorizontalAlignment="Left"/>
    </StackPanel>
  </DataTemplate>
</kanban:SfKanban.ColumnHeaderTemplate>
```



Column Tags

The [Tags](#) property customizes the header of a kanban column. The following properties of the tags are used to customize the column header:

- [CardCount](#) - Gets or sets the count of Cards available in column.
- [Maximum](#) - Gets or sets a value that indicates cards collection's maximum limit of KanbanColumn.
- [Minimum](#) - Gets or sets a value that indicates cards collection's minimum limit of KanbanColumn.
- [IsExpanded](#) - Gets or sets a value that indicates whether the KanbanColumn is in expanded or not.
- [Header](#) - Gets or sets a object which indicates KanbanColumn header.

Expand/Collapse Column

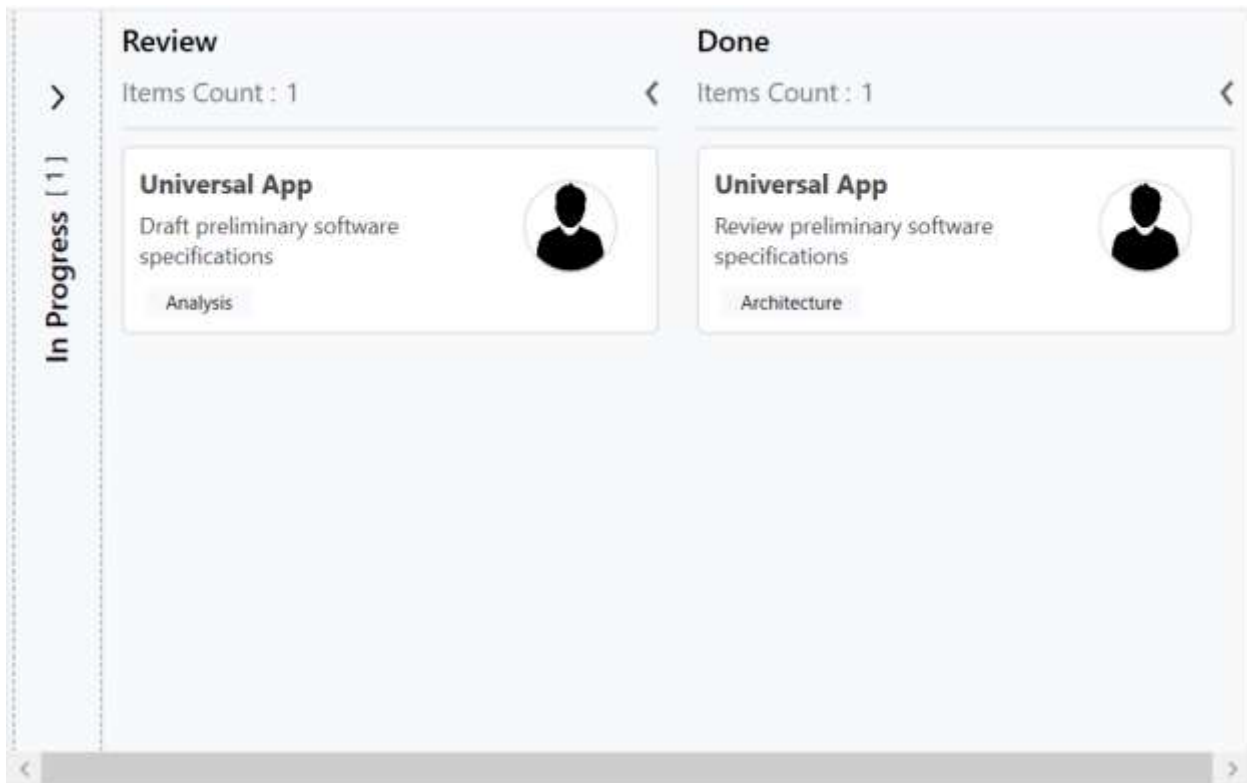
The Columns can be expanded/collapsed by tapping the toggle button, which is placed at top-right corner of the Kanban header. The [KanbanColumn.IsExpanded](#) property is used to programmatically expand/collapse the Kanban column.

XML

```
<kanban:KanbanColumn Categories="In Progress,Validated" Title="In Progress"
IsExpanded="False"></kanban:KanbanColumn>
```

C#

```
KanbanColumn kanbanColumn = new KanbanColumn();
kanbanColumn.IsExpanded = false;
```



Enable/Disable Drag & Drop

You can enable or disable the drag-and-drop operation of the cards for a particular column using the [KanbanColumn.AllowDrag](#) property.

XML

```
<syncfusion:KanbanColumn AllowDrag="false"></syncfusion:KanbanColumn>
```

C#

```
KanbanColumn kanbanColumn = new KanbanColumn();
kanbanColumn.AllowDrag = false;
```

Cards in WPF Kanban (SfKanban) control

The default elements of a card can be customized using the below properties of [KanbanModel](#).

- [Title](#) - Used to set the title of a card.
- [ImageUrl](#) - Used to set the image URL of a card. The image will be displayed at right side in default card template.
- [Category](#) - Used to set the category of a card. Based on the category the [Cards](#) will be added to the respective columns.
- [Description](#) - Used to set the description text of a card.

- [ColorKey](#) - Used to specify the indicator [ColorKey](#). The [Color](#) value of the corresponding [Key](#) should be added in [IndicatorColorPalette](#) collection of [SfKanban](#).
- [Tags](#) - Used to specify the tags of a card. The tags will be displayed at bottom in default card template.
- [ID](#) - Used to set the ID of a card.

C#

```
new KanbanModel ()
{
    Title = "Universal App",
    ID = "27654",
    Description = "Incorporate feedback into functional specifications",
    Category = "Open",
    ColorKey = "Low",
    Tags = new string[] { "Deployment" },
    ImageURL = new Uri("/images/icon.jpg", UriKind.RelativeOrAbsolute )
};
```

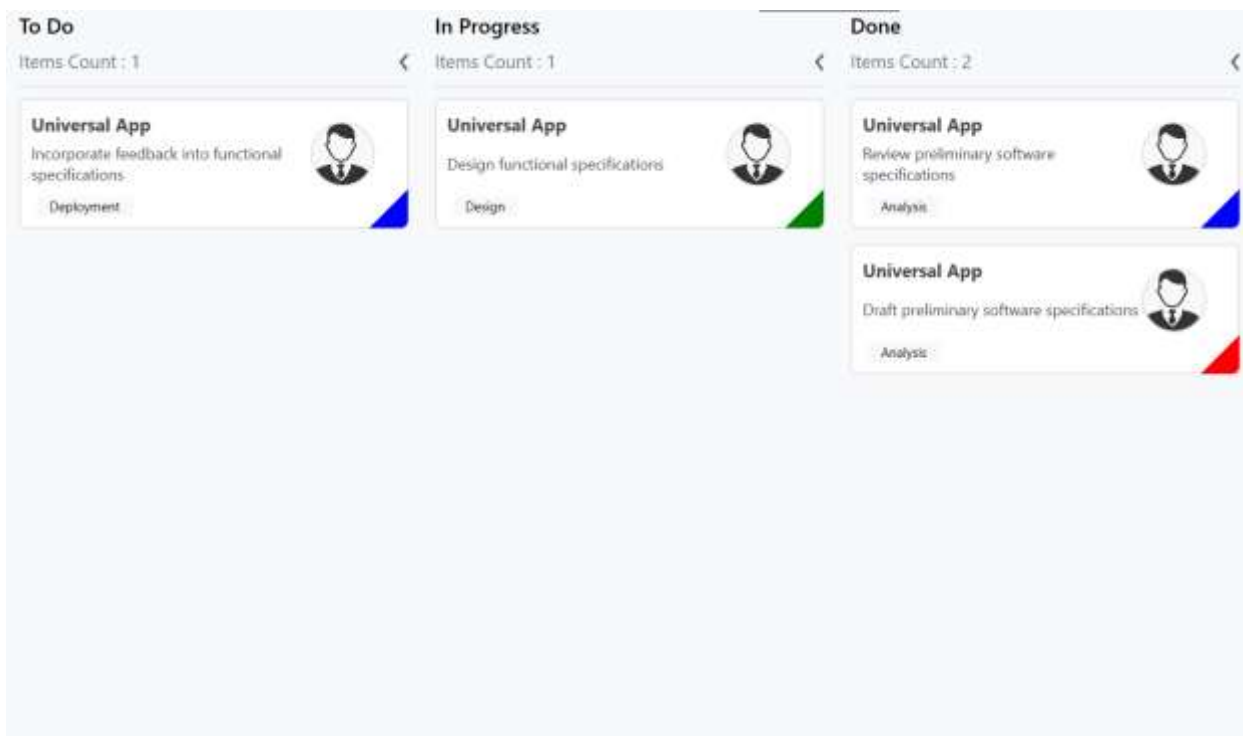
Following code snippet is used to define the colors for each key.

XML

```
<kanban:SfKanban.IndicatorColorPalette>
<kanban:ColorMapping Key="Low" Color="Blue"/>
<kanban:ColorMapping Key="Normal" Color="Green" />
<kanban:ColorMapping Key="High" Color="Red" />
</kanban:SfKanban.IndicatorColorPalette>
```

C#

```
IndicatorColorPalette indicatorColorPalette = new IndicatorColorPalette();
indicatorColorPalette.Add(new ColorMapping() { Key = "Low", Color =
Colors.Blue });
indicatorColorPalette.Add(new ColorMapping() { Key = "High", Color =
Colors.Red });
indicatorColorPalette.Add(new ColorMapping() { Key = "Normal", Color =
Colors.Green });
sfKanban.IndicatorColorPalette = indicatorColorPalette;
```



Customizing kanban cards

The [CardStyle](#) property customizes the kanban cards. The following properties of [CardStyle](#) are used to customize its appearance:

- [Background](#) - Changes the background color of a card.
- [BorderBrush](#) - Changes the border brush of a card.
- [BorderThickness](#) - Changes the border thickness of a card.
- [CornerRadius](#) - Adds rounded corners to a card.
- [IconVisibility](#) - Changes the icon visibility of a card.
- [IndicatorVisibility](#) - Changes the indicator visibility of a card.
- [TagVisibility](#) - Changes the tag panel visibility of a card.
- [TitleColor](#) - Changes the header color of a kanban card item.
- [TitleFontSize](#) - Changes the font size of a card title.
- [TitleHorizontalAlignment](#) - Changes the horizontal alignment of a card title.
- [FontSize](#) - Changes the font size of a card description.
- [Foreground](#) - Changes the foreground color of a card description.
- [TagBackground](#) - Changes the tag's background color.
- [TagForeground](#) - Changes the tag's foreground color.

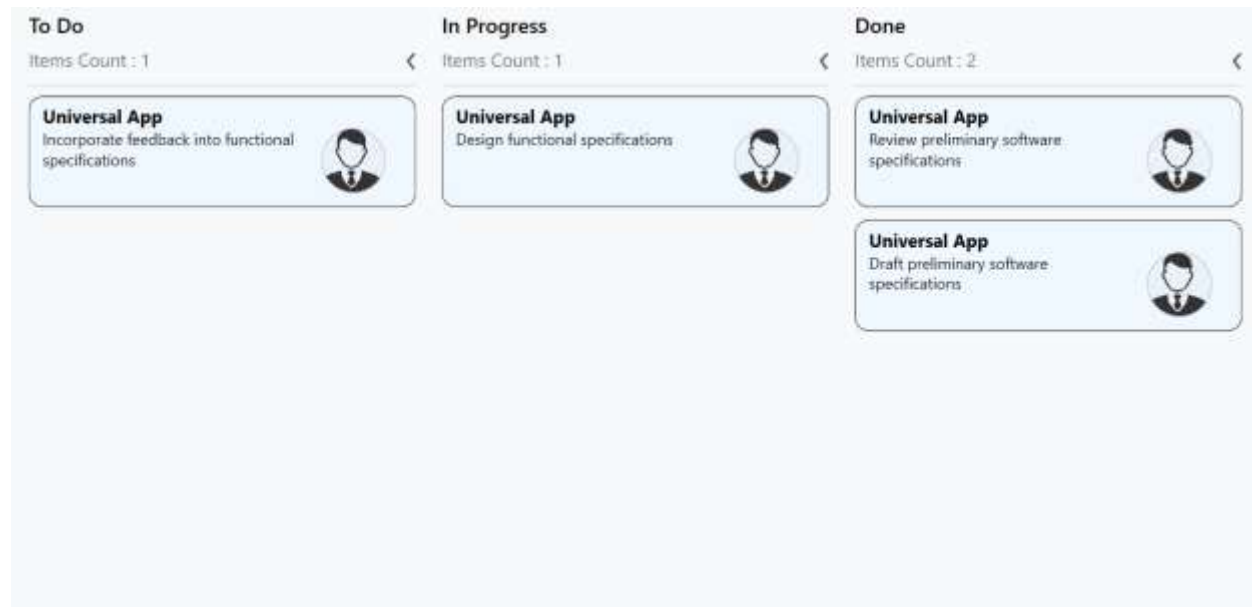
Template

You can replace the entire card template with your own design using [SfKanban.CardTemplate](#) property. The following code snippet and screenshot illustrates this.

XML

```
<kanban:SfKanban.CardTemplate>
  <DataTemplate>
    <Border BorderBrush="Black"
```

```
BorderThickness="0.75"
CornerRadius="10"
Background="AliceBlue"
Margin="0,5,0,5">
<Grid Margin="10,5,5,10">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="7*" />
<ColumnDefinition Width="3*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<TextBlock Text="{Binding Path=Title}"
FontWeight="Bold"
FontSize="16" />
<TextBlock Grid.Row="1"
FontSize="14"
HorizontalAlignment="Left"
Text="{Binding Description}"
TextWrapping="WrapWholeWords" />
<Border Grid.Row="1"
Grid.Column="1"
Height="50"
CornerRadius="50"
Width="50"
BorderBrush="Silver"
BorderThickness=".75">
<Border.Background>
<ImageBrush ImageSource="{Binding ImageURL}" />
</Border.Background>
</Border>
</Grid>
</Border>
</DataTemplate>
</kanban:SfKanban.CardTemplate>
```

Events in WPF Kanban (SfKanban) control

CardTapped

This event is triggered when you tap on any card. The argument contains the following information.

- [SelectedColumn](#) - Used to get the column of the selected card.
- [SelectedCard](#) - Used to get the selected card.
- [SelectedCardIndex](#) - Used to get the index of the card in a column.
- [SelectedColumnIndex](#) - Used to get the index of dragging card's column.
- [SelectedRowIndex](#) - Used to get the index of dragging card's row.

Command

The `CardTappedCommand` property is used to associate a command with an instance of SfKanban. This property is most often set with MVVM pattern to bind callbacks back into the ViewModel.

CommandParameter

The `CardTappedCommandParameter` property is used to set the parameter reference, based on which the event argument is shown.

NOTE

The default value of the `CardTappedCommandParameter` is `null`.

CardDoubleTapped

The [CardDoubleTapped](#) event is triggered when you double tap on any card. The argument contains the following information:

- [SelectedCard](#) - Used to get the selected card.
- [SelectedCardIndex](#) - Used to get the index of dragging card in a column.
- [SelectedColumn](#) - Used to get the column of the selected card.

CardDragStart

This event is triggered when you start to drag a card. The argument contains the following information.

- [IsCancel](#) - Used to cancel the drag action.
- [SelectedCard](#) - Used to get the underlying model of the card.
- [KeepCard](#) - Determines whether to keep the dragged card in the source location itself, until it is dropped in a new location. When it is true, the preview of the card will be created for dragging.
- [SelectedColumn](#) - Used to get the source column of card.
- [SelectedCardIndex](#) - Used to get the index of the card in source column.
- [SelectedColumnIndex](#) - Used to get the index of dragging card's column.
- [SelectedRowIndex](#) - Used to get the index of dragging card's row.

CardDragEnd

This event is triggered when whenever dragging is canceled. The argument contains the following information.

- [IsCancel](#) - Used to cancel the drag action.
- [SelectedCard](#) - Used to get the underlying model of the card.
- [SelectedColumn](#) - Used to get the source column of the card.
- [SelectedCardIndex](#) - Used to get the index of the card in source column.
- [TargetKey](#) - Used to get the category of the column where the card is going to be dropped.
- [TargetColumn](#) - Used to get the current column which is the drop target for the card.
- [TargetCardIndex](#) - Used to get the index of the card in target column.
- [SelectedColumnIndex](#) - Used to get the index of dragging card's column.
- [SelectedRowIndex](#) - Used to get the index of dragging card's row.
- [TargetRowIndex](#) - Used to get the target row index where the card is going to be inserted.
- [TargetColumnIndex](#) - Used to get the target column index where the card is going to be inserted.

CardDragEnter

This event is triggered when a card enters into a column while dragging. The argument contains the following information.

- [IsCancel](#) - Used to cancel the drag action.
- [SelectedCard](#) - Used to get the underlying model of the card.
- [SelectedColumn](#) - Used to get the source column of the card.
- [SelectedCardIndex](#) - Used to get the index of the card in source column.
- [CurrentColumn](#) - Used to get the column upon which the card enters.
- [CurrentIndex](#) - Used to get the index of the card in current column.
- [SelectedColumnIndex](#) - Used to get the index of dragging card's column.
- [SelectedRowIndex](#) - Used to get the index of dragging card's row.
- [CurrentRowIndex](#) - Used to get the current index of the card's row.
- [CurrentColumnIndex](#) - Used to get the current index of the card's column.

CardDragLeave

This event is triggered when a card leaves a column while dragging. The argument contains the following information.

- [SelectedCard](#) - Used to get the underlying model of the card.
- [SelectedColumn](#) - Used to get the source column of the card.
- [SelectedCardIndex](#) - Used to get the index of the card in source column.

- [LeftColumn](#) - Used to get the column from which the card leaves.
- [PreviousCardIndex](#) - used to get the index of the card left.
- [SelectedColumnIndex](#) - Used to get the index of dragging card's column.
- [SelectedRowIndex](#) - Used to get the index of dragging card's row.
- [PreviousRowIndex](#) - Used to get the previous card's row index while drag enter into next column.
- [PreviousColumnIndex](#) - Used to get the previous card's column index while drag enter into next column.

CardDragOver

This event is triggered when a card is dragged to a new index within a column. The argument contains the following information.

- [IsCancel](#) - Used to cancel the drag action.
- [SelectedCard](#) - Used to get the underlying model of the card.
- [SelectedColumn](#) - Used to get the source column of the card.
- [SelectedCardIndex](#) - Used to get the index of the card in source column.
- [CurrentColumn](#) - Used to get the current column which is the drop target for the card.
- [CurrentIndex](#) - Used to get the new index of the card in current column.
- [SelectedColumnIndex](#) - Used to get the index of dragging card's column.
- [SelectedRowIndex](#) - Used to get the index of dragging card's row.
- [CurrentRowIndex](#) - Used to get the current index of the card's row.
- [CurrentColumnIndex](#) - Used to get the current index of the card's column.

ColumnsGenerated

This event will be fired after the columns are generated automatically. You can access the auto-generated columns using `SfKanban.ActualColumns` property.

ColumnGenerated

This event is triggered when a column generated.

- [Columns](#) - used to get the generated columns.
- [IsCancel](#) - used to cancel the generated column added to the SfKanban.
- [CurrentColumn](#) - used to get the current generated column.

Placeholder in WPF Kanban (SfKanban) control

The placeholder is to denote a card's new position in the [KanbanColumn](#). It will appear while dragging a card over the column.

Placeholder style

[PlaceholderStyle](#) property is used to customize the placeholder. Following properties are used to customize its appearance.

- [Fill](#) - This property is used to change the background color of the placeholder.
- [Stroke](#) - This property is used to change the border color of the placeholder.
- [StrokeThickness](#) - This property is used to change the border width of the placeholder.
- [StrokeDashArray](#) - This property is used to change the dashes of the placeholder border.
- [FontSize](#) - This is used to change the text size of the placeholder.

- [Foreground](#) - This property is used to change the text color of the placeholder.
- [RadiusX](#) - This property is used to change the x-radius of the placeholder.
- [RadiusY](#) - This property is used to change the y-radius of the placeholder.
- [TextHorizontalAlignment](#) - This property is used to change the horizontal alignment of the placeholder text.
- [TextVerticalAlignment](#) - This property is used to change the vertical alignment of the placeholder text.

Following properties are used to customize the selected category when you have more than one category in a column.

- [SelectedBackground](#) - This property is used to change the background color of the selected placeholder.
- [SelectedStroke](#) - This property is used to change the border color of the selected placeholder.
- [SelectedStrokeThickness](#) - This property is used to change the border width of the selected placeholder.
- [SelectedStrokeDashArray](#) - This property is used to change the dashes of the selected placeholder.
- [SelectedFontSize](#) - This is used to change the font size of the text in selected placeholder.
- [SelectedForeground](#) - This property is used to change the color of the text in selected placeholder.

The following code example describes the above behavior.

XML

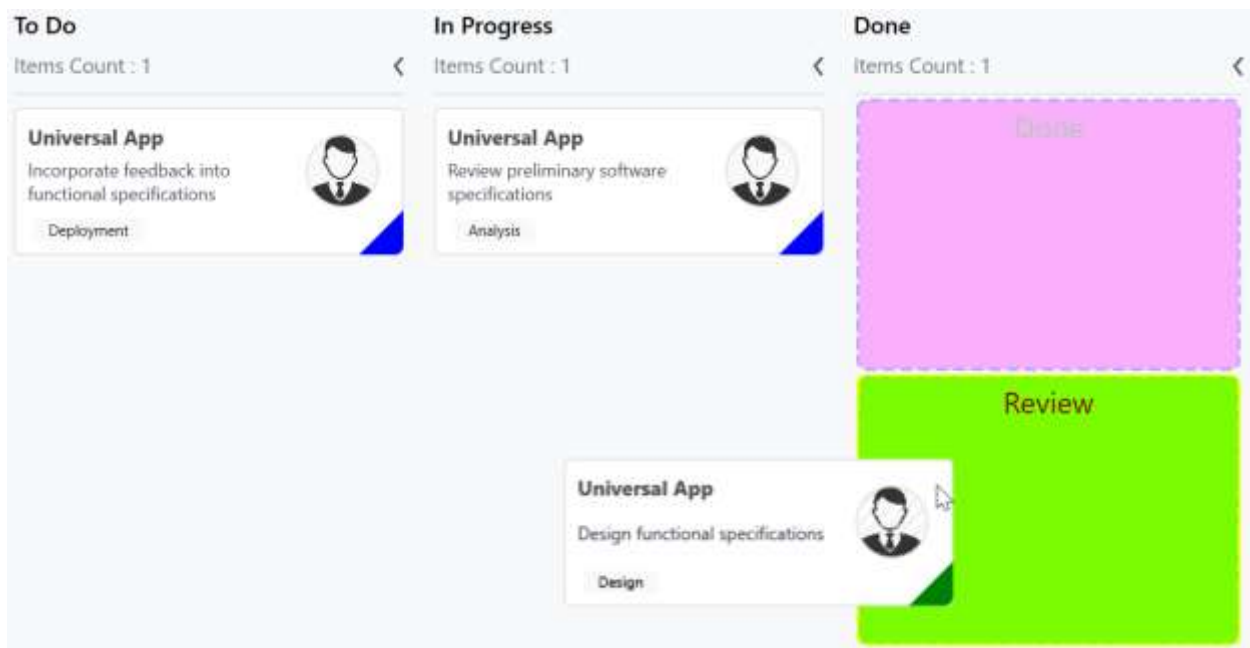
```
<kanban:PlaceholderStyle FontSize="20"
Foreground="DarkGreen"
Fill="Fuchsia"
Stroke="Blue"
StrokeThickness="2"
SelectedFontSize="20"
SelectedForeground="Maroon"
SelectedStroke="Yellow"
SelectedStrokeThickness="2"
SelectedBackground="LawnGreen">
</kanban:PlaceholderStyle>
```

C#

```
PlaceholderStyle style = new PlaceholderStyle();
style.FontSize = 20;
style.Foreground = new SolidColorBrush(Colors.DarkGreen);
style.Fill = new SolidColorBrush(Colors.Fuchsia);
style.Stroke = new SolidColorBrush(Colors.Blue);
style.StrokeThickness = 2;
style.StrokeDashArray = new DoubleCollection() {1, 1};
style.SelectedFontSize = 20;
style.SelectedForeground = new SolidColorBrush(Colors.Maroon);
style.SelectedStroke = new SolidColorBrush(Colors.Yellow);
style.SelectedBackground = new SolidColorBrush(Colors.LawnGreen);
style.SelectedStrokeThickness = 2;
```

```
style.SelectedStrokeDashArray = new DoubleCollection() {2, 1};
sfKanban.PlaceholderStyle = style;
```

The following output demonstrates the above code example.



Note: The UI of the placeholder can be replaced entirely using [PlaceholderTemplate](#) property.

Workflow configuration

A Kanban [Workflows](#) is a set of Category and AllowedTransitions that an item moves through its life cycle and typically represents processes within your organization.

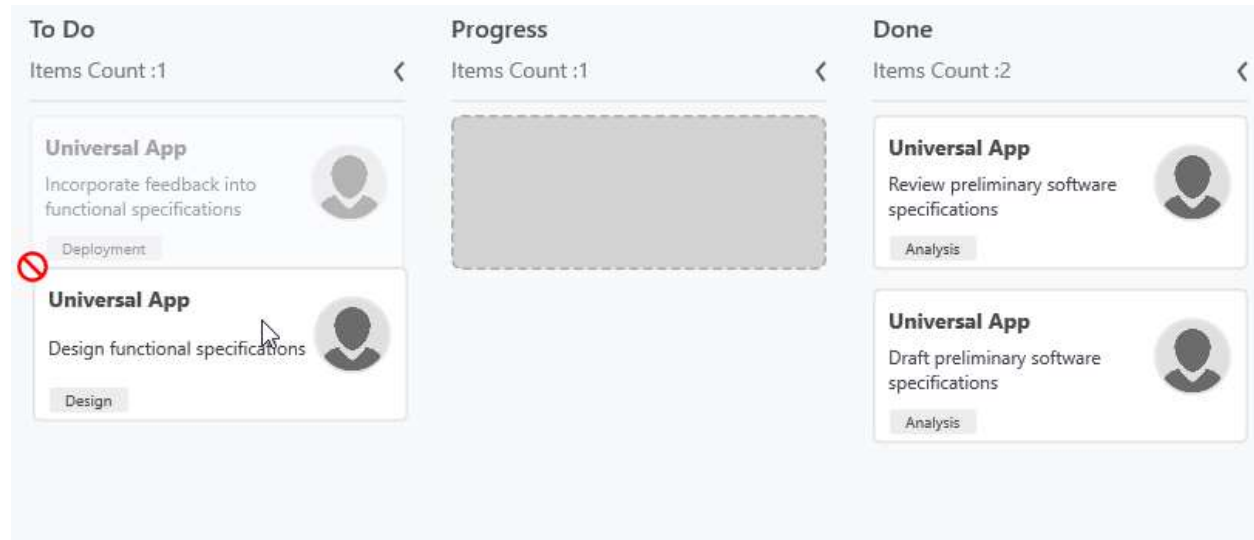
- [Category](#) – It represents a state of an item at a particular point in a specific workflow.
- [AllowedTransitions](#) – It is a list of categories to where the card can be moved from the current category.

XML

```
<syncfusion:SfKanban.Workflows>
<syncfusion:KanbanWorkflow Category="Open">
<syncfusion:KanbanWorkflow.AllowedTransitions>
<x:String>In Progress</x:String>
</syncfusion:KanbanWorkflow.AllowedTransitions>
</syncfusion:KanbanWorkflow>
<syncfusion:KanbanWorkflow Category="In Progress">
<syncfusion:KanbanWorkflow.AllowedTransitions>
<x:String>Review</x:String>
<x:String>Done</x:String>
</syncfusion:KanbanWorkflow.AllowedTransitions>
</syncfusion:KanbanWorkflow>
</syncfusion:SfKanban.Workflows>
```

C#

```
WorkflowCollection workflows = new WorkflowCollection();
workflows.Add(new KanbanWorkflow()
{
    Category = "Open",
    AllowedTransitions = new List<object>() {"In Progress"}
});
workflows.Add(new KanbanWorkflow()
{
    Category = "In Progress",
    AllowedTransitions = new List<object>() {"Review", "Done"}
});
Kanban.Workflows = workflows;
```



Work In-Progress limit

[MinimumLimit](#) and [MaximumLimit](#) properties are used to limit the minimum and maximum items in the Kanban column. However, this will not restrict moving the items from one column to another column. But the violation of the limit can be indicated by changing the [ValidationColor](#) of the error bar.

Following properties of [ErrorBarSettings](#) are used to customize the error bar.

- [Color](#) – used to set the default color of the error bar.
- [MinValidationColor](#) – used to set the color of the error bar when the items count is lesser than [MinimumLimit](#).
- [MaxValidationColor](#) – used to set the color of the error bar when the items count is greater than [MaximumLimit](#).
- [Height](#) - used to provide height for error bar.

XML

```
<syncfusion:KanbanColumn x:Name="column1" Categories="Review, Done"
    Title="Done"
    MinimumLimit="1"
    MaximumLimit="2">
    <syncfusion:KanbanColumn.ErrorBarSettings>
    <syncfusion:ErrorBarSettings Color="Gray" MaxValidationColor="Red"
        MinValidationColor="Green">
```

```

</syncfusion:ErrorBarSettings>
</syncfusion:KanbanColumn.ErrorBarSettings>
</syncfusion:KanbanColumn>

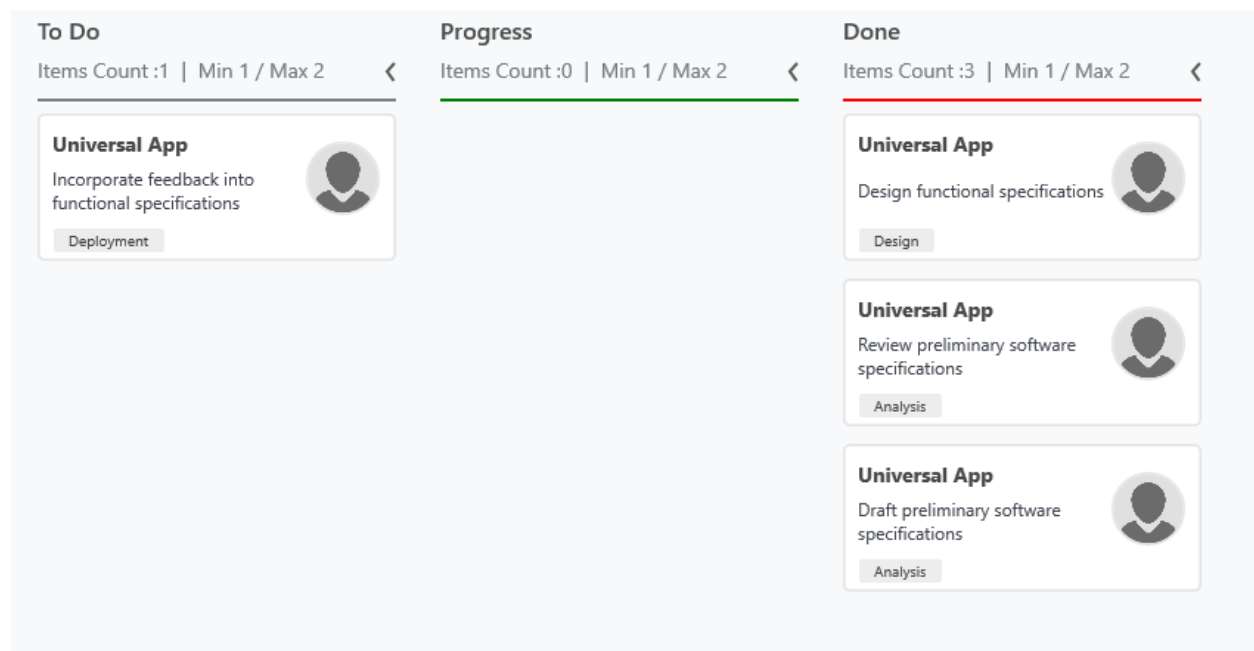
```

C#

```

column1.MinimumLimit = 1;
column1.MaximumLimit = 2;
column1.ErrorBarSettings = new ErrorBarSettings()
{
    Color = new SolidColorBrush(Colors.Gray),
    MinValidationColor = new SolidColorBrush(Colors.Green),
    MaxValidationColor = new SolidColorBrush(Colors.Red)
};

```



Swim lanes in WPF Kanban (SfKanban) control

Swim lanes are horizontal categorizations; they allow you to categorize your current workflow by different projects, teams, users, or whatever you need.

By default, it will be categorized based on the [Assignee](#) values in the [KanbanModel](#) class. You can also define category by mapping [SwimlaneKey](#) to appropriate property name in the defined data model.

The following code example demonstrates how to group the underlying data collection based on [SwimlaneKey](#).

XML

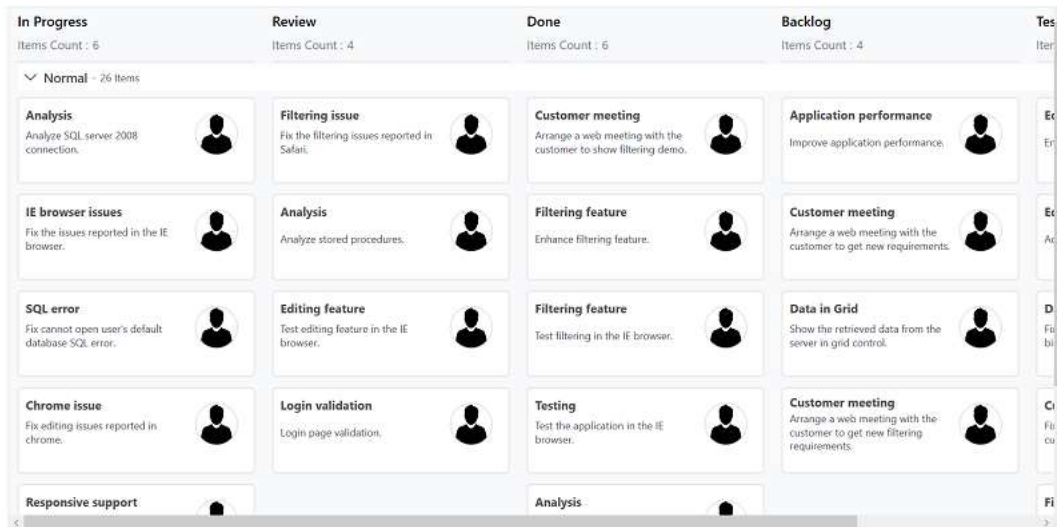
```

<kanban:SfKanban SwimlaneKey="ColorKey" x:Name="Kanban"
ItemsSource="{Binding Tasks}">
</kanban:SfKanban>

```

C#

```
kanban.SwimlaneKey = "ColorKey";
```



Note: If no value is assigned to the [SwimlaneKey](#) mapped property in a task, then it will be grouped under the Unassigned swim lane.

Customization

SfKanban provides support to customize the header, which is displayed before the swim lane group using [SwimlaneHeaderTemplate](#). The following code example demonstrates how to customize the swim lane.

XML

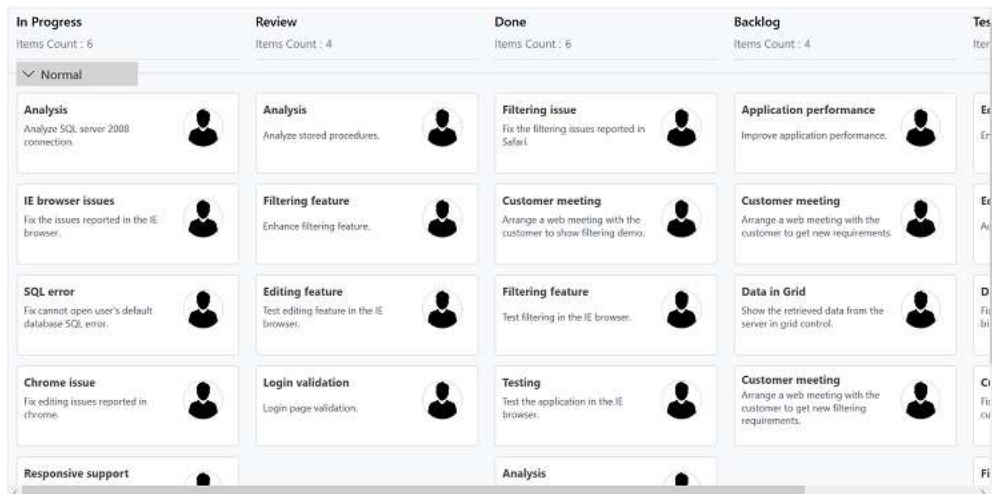
```
<kanban:SfKanban x:Name="Kanban" ItemsSource="{Binding Tasks}">
  <kanban:SfKanban.SwimlaneHeaderTemplate>
    <DataTemplate>
      <Grid>
        <Border BorderBrush="LightGray" BorderThickness="1" Width="{Binding
          ElementName=kanban, Path=ActualWidth}" Height="1">
        </Border>
        <Border BorderBrush="Black" CornerRadius="5,5,5,5" Width="150"
          Margin="10,2,10,0" HorizontalAlignment="Left" >
          <StackPanel Background="LightGray" x:Name="SwimlaneHeaderPanel"
            Orientation="Horizontal">
            <Grid x:Name="CollapsedIcon" Background="Transparent"
              Height="30" Width="30">
            <Path x:Name="ExpandedPath" IsHitTestVisible="False"
              Data="M30.587915,0L31.995998,1.4199842 15.949964,17.351 0,1.4979873
              1.4099131,0.078979151 15.949964,14.53102z"
              Stretch="Uniform" Fill="#FF000000" Width="14" Height="14" Margin="0,0,0,0"
              RenderTransformOrigin="0.5,0.5">
              <Path.RenderTransform>
                <TransformGroup>
                  <TransformGroup.Children>
                    <RotateTransform Angle="0" />
                    <ScaleTransform ScaleX="1" ScaleY="1" />
                  </TransformGroup.Children>
                </TransformGroup>
              </Path>
            </Grid>
          </StackPanel>
        </Border>
      </Grid>
    </DataTemplate>
  </kanban:SfKanban.SwimlaneHeaderTemplate>
</kanban:SfKanban>
```



```

</Path.RenderTransform>
</Path>
<Path x:Name="CollapsedPath" Visibility="Collapsed" IsHitTestVisible="False"
Data="M1.4200482,0L17.351001,16.046996 1.4980513,31.996001
0.078979631,30.585997 14.531046,16.046019 0,1.4089964z"
Stretch="Uniform" Fill="#FF000000" Width="14" Height="14" Margin="0,0,0,0"
RenderTransformOrigin="0.5,0.5">
<Path.RenderTransform>
<TransformGroup>
<TransformGroup.Children>
<RotateTransform Angle="0" />
<ScaleTransform ScaleX="1" ScaleY="1" />
</TransformGroup.Children>
</TransformGroup>
</Path.RenderTransform>
</Path>
</Grid>
<TextBlock FontWeight="Medium" IsHitTestVisible="False" FontSize="15"
FontStretch="Expanded" TextWrapping="NoWrap"
VerticalAlignment="Center" Text="{Binding Title}" />
</StackPanel>
</Border>
</Grid>
</DataTemplate>
</kanban:SfKanban.SwimlaneHeaderTemplate>
</kanban:SfKanban>

```



SfLinearGauge

WPF LinearGauge (SfLinearGauge) Overview

The [LinearGauge](#) displays a range of values graphically along the linear scale, which is considered as the linear form of the linear gauge. It measures the values of the scale and presents in the horizontal sliding, vertical sliding, or meter.

The [LinearGauge](#) control is used to visualize numerical values of a scale in linear manner. By using the [LinearGauge](#) control, you can render thermometer.

Key features

Scale

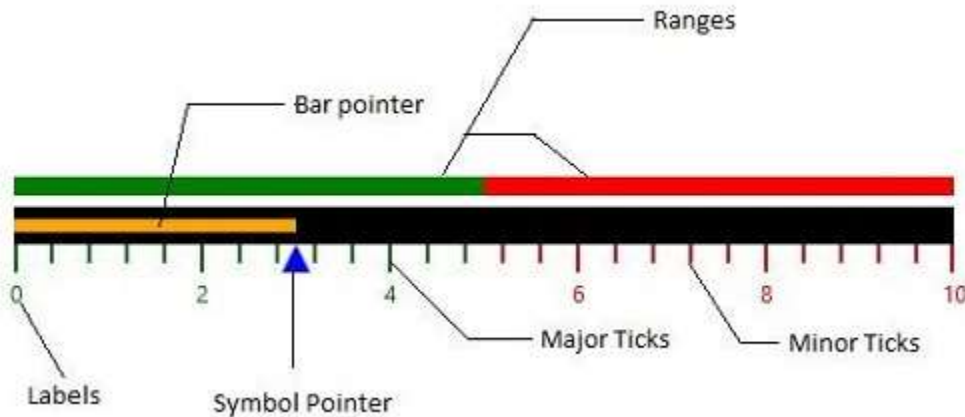
The [Scale](#) support adding a scale to linear gauge using horizontal and vertical orientations.

Ranges

Highlights the desired [Ranges](#) of values in the gauge scale.

Pointers

The [Pointers](#) supports adding multiple pointers (bar and symbol) to the linear scale.



Getting Started with SfLinearGauge

This section explains the steps required to configure the [SfLinearGauge](#) control in a real-time scenario and provides a walk-through on its customization features.

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. 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 linear 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 [SfLinearGauge](#) namespace to your respective Window as in the following.

XML

```
xmlns:gauge="clr-
namespace:Syncfusion.UI.Xaml.Gauges;assembly=Syncfusion.SfGauge.Wpf"
```

C#

```
using Syncfusion.UI.Xaml.Gauges;
```

You can initialize an empty [SfLinearGauge](#) control.

XML

```
<gauge:SfLinearGauge/>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
this.Content = sfLinearGauge;
```

Configuring scale

Scales is a collection of [LinearScale](#) which is used to indicate the numeric values. Scale bar, ticks, labels, ranges, and pointers are the sub elements of a scale.

The [Minimum](#) and [Maximum](#) properties allow you to set the scale range.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0"
      LabelStroke="#424242"
      MajorTickStroke="Gray"
      MajorTickSize="15"
      MajorTickStrokeThickness="1"
      MinorTickStroke="Gray"
      MinorTickSize="7"
      MinorTickStrokeThickness="1"
      MinorTicksPerInterval="3"
      ScaleBarLength="300"
      ScaleBarSize="10">
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.LabelStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#424242");
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MajorTickSize = 15;
linearScale.MajorTickStrokeThickness = 1;
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickSize = 7;
```

```
linearScale.MinorTickStrokeThickness = 1;
linearScale.MinorTicksPerInterval = 3;
linearScale.ScaleBarStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#E0E0E0");
linearScale.ScaleBarLength = 300;
linearScale.ScaleBarSize = 10;
sfLinearGauge.MainScale = linearScale;
this.Content = sfLinearGauge;
```

Adding a symbol pointer

SymbolPointer is a shape that can be placed to mark the pointer value in gauge.

XML

```
<gauge:LinearScale.Pointers>
<gauge:LinearPointer PointerType="SymbolPointer"
Value="60"
SymbolPointerHeight="10"
SymbolPointerWidth="10"
SymbolPointerPosition="Below"
SymbolPointerStroke="#757575" />
</gauge:LinearScale.Pointers>
```

C#

```
LinearPointer linearPointer = new LinearPointer();
linearPointer.PointerType = LinearPointerType.SymbolPointer;
linearPointer.Value = 60;
linearPointer.SymbolPointerHeight = 10;
linearPointer.SymbolPointerWidth = 10;
linearPointer.SymbolPointerPosition = LinearSymbolPointersPosition.Below;
linearPointer.SymbolPointerStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#757575");
linearScale.Pointers.Add(linearPointer);
```

Adding a bar pointer

BarPointer is used to mark the scale values. This starts at the beginning of gauge and ends at the pointer value.

XML

```
<gauge:LinearScale.Pointers>
<gauge:LinearPointer PointerType="BarPointer"
Value="50"
BarPointerStroke="#757575"
BarPointerStrokeThickness="10" />
</gauge:LinearScale.Pointers>
```

C#

```
LinearPointer linearPointer1 = new LinearPointer();
linearPointer1.PointerType = LinearPointerType.BarPointer;
linearPointer1.Value = 50;
```

```
linearPointer1.BarPointerStroke = (SolidColorBrush) new
BrushConverter().ConvertFrom("#757575");
linearPointer1.BarPointerStrokeThickness = 10;
linearScale.Pointers.Add(linearPointer1);
```

Adding ranges

You can categorize the scale values using the start and end values properties in [LinearRange](#). You can add multiple ranges for a scale using the [Ranges](#) property.

XML

```
<gauge:LinearScale.Ranges>
<gauge:LinearRange StartValue="0"
EndValue="40"
RangeStroke="#27BEB7"
StartWidth="10"
EndWidth="10"
RangeOffset="0.4" />
<gauge:LinearRange StartValue="40"
EndValue="100"
RangeStroke="LightCyan"
RangeOffset="0.4"
StartWidth="10"
EndWidth="10" />
</gauge:LinearScale.Ranges>
```

C#

```
//Adding Range
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 40;
linearRange.RangeStroke = (SolidColorBrush) new
BrushConverter().ConvertFrom("#27BEB7");
linearRange.StartWidth = 10;
linearRange.EndWidth = 10;
linearRange.RangeOffset = 0.4;
linearScale.Ranges.Add(linearRange);
LinearRange linearRange1 = new LinearRange();
linearRange1.StartValue = 40;
linearRange1.EndValue = 100;
linearRange1.RangeStroke = new SolidColorBrush(Colors.LightCyan);
linearRange1.RangeOffset = 0.4;
linearRange1.StartWidth = 10;
linearRange1.EndWidth = 10;
linearScale.Ranges.Add(linearRange1);
```

The following code example is the complete code of the previous configurations.

XML

```
<gauge:SfLinearGauge>
<gauge:SfLinearGauge.MainScale>
<gauge:LinearScale LabelStroke="#424242"
MajorTickStroke="Gray"
```

```

MajorTickSize="15"
MajorTickStrokeThickness="1"
MinorTickStroke="Gray"
MinorTickSize="7"
MinorTickStrokeThickness="1"
MinorTicksPerInterval="3"
ScaleBarStroke="#E0E0E0"
ScaleBarLength="300"
ScaleBarSize="10">
<gauge:LinearScale.Pointers>
<gauge:LinearPointer PointerType="SymbolPointer"
Value="60"
SymbolPointerHeight="10"
SymbolPointerWidth="10"
SymbolPointerPosition="Below"
SymbolPointerStroke="#757575" />
<gauge:LinearPointer PointerType="BarPointer"
Value="50"
BarPointerStroke="#757575"
BarPointerStrokeThickness="10" />
</gauge:LinearScale.Pointers>
<gauge:LinearScale.Ranges>
<gauge:LinearRange StartValue="0"
EndValue="40"
RangeStroke="#27BEB7"
StartWidth="10"
EndWidth="10"
RangeOffset="0.4" />
<gauge:LinearRange StartValue="40"
EndValue="100"
RangeStroke="LightCyan"
RangeOffset="0.4"
StartWidth="10"
EndWidth="10" />
</gauge:LinearScale.Ranges>
</gauge:LinearScale>
</gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```

C#

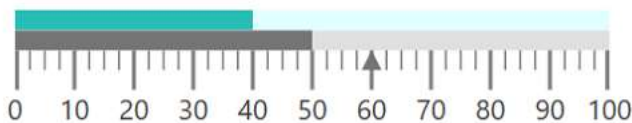
```

SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.LabelStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#424242");
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MajorTickSize = 15;
linearScale.MajorTickStrokeThickness = 1;
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickSize = 7;
linearScale.MinorTickStrokeThickness = 1;
linearScale.MinorTicksPerInterval = 3;
linearScale.ScaleBarStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#E0E0E0");
linearScale.ScaleBarLength = 300;
linearScale.ScaleBarSize = 10;

```

```
//Adding symbol pointer
LinearPointer linearPointer = new LinearPointer();
linearPointer.PointerType = LinearPointerType.SymbolPointer;
linearPointer.Value = 60;
linearPointer.SymbolPointerHeight = 10;
linearPointer.SymbolPointerWidth = 10;
linearPointer.SymbolPointerPosition = LinearSymbolPointersPosition.Below;
linearPointer.SymbolPointerStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#757575");
linearScale.Pointers.Add(linearPointer);
////Adding bar pointer
LinearPointer linearPointer1 = new LinearPointer();
linearPointer1.PointerType = LinearPointerType.BarPointer;
linearPointer1.Value = 50;
linearPointer1.BarPointerStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#757575");
linearPointer1.BarPointerStrokeThickness = 10;
linearScale.Pointers.Add(linearPointer1);
//Adding Range
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 40;
linearRange.RangeStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#27BEB7");
linearRange.StartWidth = 10;
linearRange.EndWidth = 10;
linearRange.RangeOffset = 0.4;
linearScale.Ranges.Add(linearRange);
LinearRange linearRange1 = new LinearRange();
linearRange1.StartValue = 40;
linearRange1.EndValue = 100;
linearRange1.RangeStroke = new SolidColorBrush(Colors.LightCyan);
linearRange1.RangeOffset = 0.4;
linearRange1.StartWidth = 10;
linearRange1.EndWidth = 10;
linearScale.Ranges.Add(linearRange1);
sfLinearGauge.MainScale = linearScale;
this.Content = sfLinearGauge;
```

The following screenshot illustrates the result of the previous codes.

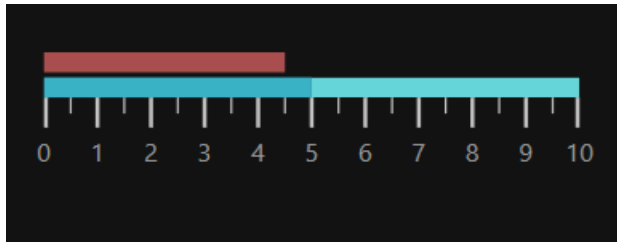


You can find the complete getting started sample from this [link](#).

Theme

LinearGauge supports various built-in themes. Refer to the below links to apply themes for the LinearGauge,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



See also

[How to apply themes for SfLinearGauge](#)

Scale in SfLinearGauge

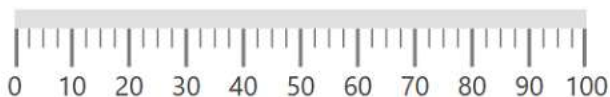
The [MainScale](#) is a linear scale integrates ticks, labels, ranges, and pointers to customize the basic look and feel of the linear gauge.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3"/>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;
```



Setting minimum and maximum values for a scale

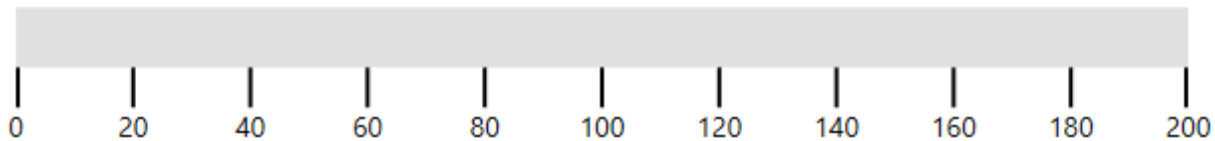
To change the minimum and maximum values of a linear scale, use the [Minimum](#) and [Maximum](#) properties as shown in the following code snippet.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale Minimum="0" Maximum="200" ScaleBarStroke="#E0E0E0"/>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```


C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.Minimum = 0;
linearScale.Maximum = 200;
sfLinearGauge.MainScale = linearScale;
```



Setting interval for a scale

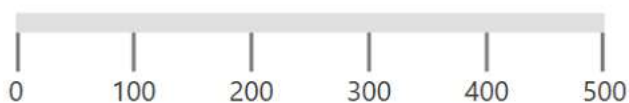
The [Interval](#) property allows to set intervals for scale. The default value of the [Interval](#) property is auto interval. Auto interval defines the count of the scale labels as 3 for 100 pixels.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale Minimum="0" Maximum="500" Interval="100"
      ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242" ScaleBarSize="10"/>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.Minimum = 0;
linearScale.Maximum = 500;
linearScale.Interval = 100;
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
sfLinearGauge.MainScale = linearScale;
```



Scale customization

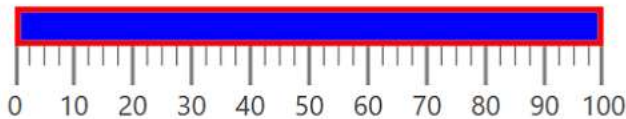
You can customize the color, length, size, and position of the `LinearScale` using the [ScaleBarStroke](#), [ScaleBarBorderThickness](#), and [ScaleBarBorderBrush](#) properties, respectively.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarBorderBrush="Red" ScaleBarStroke="Blue"
      MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242" ScaleBarBorderThickness="3"
      ScaleBarSize="20" MinorTicksPerInterval="3" />
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Colors.Blue);
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 20;
linearScale.ScaleBarBorderThickness = new Thickness(3);
linearScale.ScaleBarBorderBrush = new SolidColorBrush(Colors.Red);
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;
```



Size customization

Size of the scale can be customized using the following two properties.

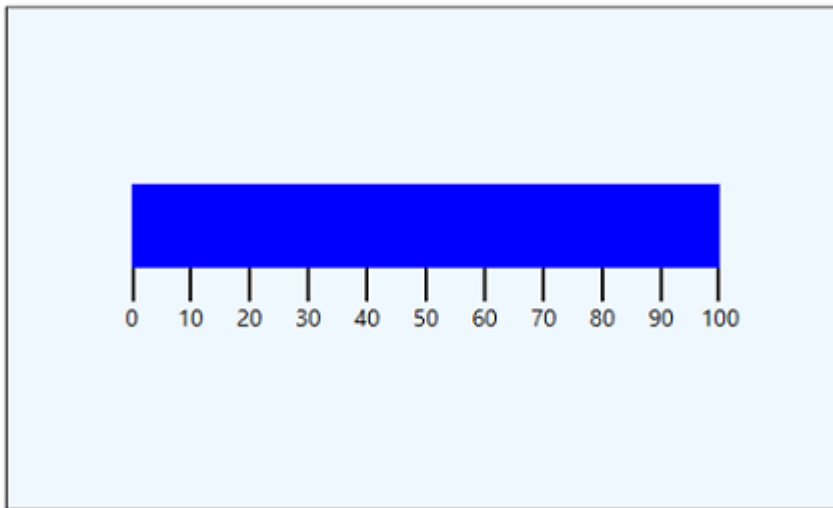
- [ScaleBarSize](#) - Customizes the size of the scale bar (i.e height in case of landscape orientation).
- [ScaleBarLength](#) - Customizes the length of the scale bar (i.e width in case of landscape orientation).

XML

```
<Gauges:SfLinearGauge Background="AliceBlue" BorderBrush="Black"
  BorderThickness="1"
  Height="300" Width="500">
  <Gauges:SfLinearGauge.MainScale>
    <Gauges:LinearScale ScaleBarStroke="Blue" ScaleBarSize="50"
      ScaleBarLength="350">
    </Gauges:LinearScale>
  </Gauges:SfLinearGauge.MainScale>
</Gauges:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
sfLinearGauge.Background = new SolidColorBrush(Colors.AliceBlue);
sfLinearGauge.BorderBrush = new SolidColorBrush(Colors.Black);
sfLinearGauge.BorderThickness = new Thickness(1);
sfLinearGauge.Height = 300;
sfLinearGauge.Width=500;
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Colors.Blue);
linearScale.ScaleBarSize = 50;
linearScale.ScaleBarLength = 350;
sfLinearGauge.MainScale = linearScale;
grid.Children.Add(sfLinearGauge);
```



Setting scale direction

You can set the scale position to its forward and backward using the [ScaleDirection](#) property.

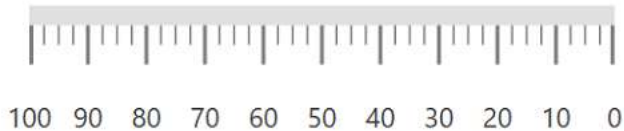
XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleDirection="Backward" ScaleBarStroke="#E0E0E0"
      MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242" ScaleBarSize="10"
      MinorTicksPerInterval="3"/>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleDirection = LinearScaleDirection.Backward;
```

```
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;
```



Setting position for a scale

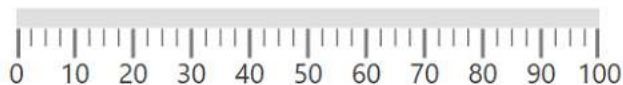
You can set the scale position using the [ScaleBarPositionFactor](#) property. First, set the [ElementsPositionMode](#) to custom, and then set [ScaleBarPositionFactor](#).

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ElementsPositionMode="Custom"
      ScaleBarPositionFactor="0.5" TickPositionFactor="0.443"
      ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      MinorTickSize="9" MajorTickSize="15"
      ScaleBarSize="10" MinorTicksPerInterval="3" />
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ElementsPositionMode = LinearScalePositionModes.Custom;
linearScale.ScaleBarPositionFactor = 0.5;
linearScale.TickPositionFactor = 0.443;
linearScale.MajorTickSize = 9;
linearScale.MinorTickSize = 15;
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;
```



Ticks support in WPF Linear Gauge (SfLinearGauge) with customization

Ticks are used to identify the gauge's data value by marking the gauge scale in regular increments.

Tick customization

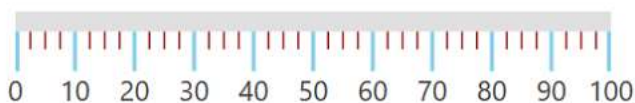
By setting the [MajorTickStroke](#) and [MinorTickStroke](#) properties, the stroke of the major ticks and minor ticks can be customized. Using the [MajorTickStrokeThickness](#) and [MinorTickStrokeThickness](#), the stroke thickness of the major ticks and minor ticks can be customized. The size of the major ticks and minor ticks can be modified using the [MajorTickSize](#) and [MinorTickSize](#) properties.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale MajorTickSize="20" MinorTickSize="9"
      ScaleBarStroke="#E0E0E0" MajorTickStroke="SkyBlue"
      MinorTickStroke="Brown" LabelStroke="#424242"
      MinorTickStrokeThickness="1" MajorTickStrokeThickness="2"
      ScaleBarSize="10" MinorTicksPerInterval="3" />
    </gauge:SfLinearGauge.MainScale>
  </gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.SkyBlue);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Brown);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.MinorTickStrokeThickness = 1;
linearScale.MajorTickStrokeThickness = 2;
linearScale.MajorTickSize = 20;
linearScale.MinorTickSize = 9;
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;
```



Setting position for tick

The ticks in the scale can be placed above, below, or in between the scale by choosing one of the following options available in the [TickPosition](#) property:

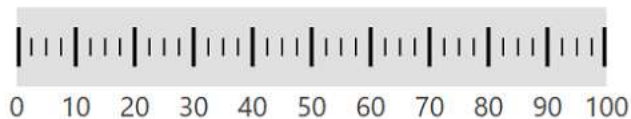
1. Above
2. Below (Default)
3. Cross

XML

```
<gauge:SfLinearGauge>
<gauge:SfLinearGauge.MainScale>
<gauge:LinearScale TickPosition="Cross" MajorTickSize="20" MinorTickSize="9"
ScaleBarStroke="#E0E0E0" MajorTickStroke="Black" MinorTickStroke="Black"
LabelStroke="#424242"
ScaleBarSize="40" MinorTicksPerInterval="3" />
</gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Black);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Black);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.MajorTickSize = 20;
linearScale.MinorTickSize = 9;
linearScale.TickPosition = LinearTicksPosition.Cross;
linearScale.ScaleBarSize = 40;
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;
```



Setting minor ticks per interval

The [Interval](#) property is used to calculate the tick counts for a scale. Like ticks, minor ticks can also be calculated by using the [MinorTicksPerInterval](#) property.

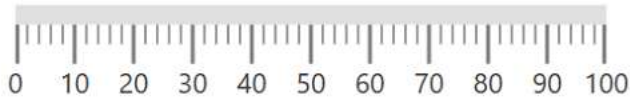
XML

```
<gauge:SfLinearGauge>
<gauge:SfLinearGauge.MainScale>
<gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
MinorTickStroke="Gray" LabelStroke="#424242"
ScaleBarSize="10" MinorTicksPerInterval="4" />
</gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
```

```
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.ScaleBarLength = 300;
linearScale.MinorTicksPerInterval = 4;
sfLinearGauge.MainScale = linearScale;
```



Labels support in WPF Linear Gauge (SfLinearGauge)

Labels of the linear scale provide a numeric value to the major ticks that will be specified according to the range of the scale.

Label color customization

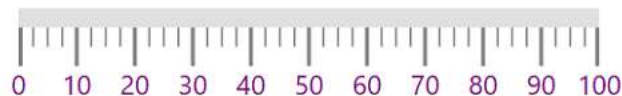
The foreground of the label is customized by setting the [LabelStroke](#) of the linear scale.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="Purple"
      ScaleBarSize="10" MinorTicksPerInterval="3" />
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Colors.Purple);
linearScale.ScaleBarSize = 10;
linearScale.ScaleBarLength = 300;
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;
```



Label font customization

The label font can be customized using the [LabelSize](#), [FontFamily](#), and [FontStyle](#) properties. The labels can be positioned far away from the ticks using the [LabelOffset](#) property.

XML

```

<gauge:SfLinearGauge>
<gauge:SfLinearGauge.MainScale>
<gauge:LinearScale FontFamily="Monotype Corsiva" FontSize="15"
FontStyle="Italic" LabelOffset="15"
ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray" MinorTickStroke="Gray"
LabelStroke="#424242" LabelSize="20"
ScaleBarSize="10" MinorTicksPerInterval="3">
</gauge:LinearScale>
</gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

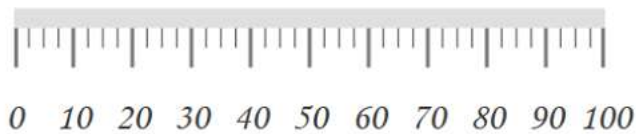
```

C#

```

SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.LabelSize = 20;
linearScale.LabelOffset = 15;
linearScale.FontFamily = new FontFamily("Monotype Corsiva");
linearScale.FontStyle = FontStyles.Italic;
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;

```

**Setting position for labels**

The labels in the scale can be placed above or below the linear scale by choosing the following options available in the [LabelPosition](#) property. The default value of [LabelPosition](#) property is below.

1. Above
2. Below (Default)

XML

```

<gauge:SfLinearGauge>
<gauge:SfLinearGauge.MainScale>
<gauge:LinearScale LabelPosition="Above" ScaleBarStroke="#E0E0E0"
MajorTickStroke="Gray"
MinorTickStroke="Gray" LabelStroke="#424242" ScaleBarSize="10"
MinorTicksPerInterval="3" />
</gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```

C#

```

SfLinearGauge sfLinearGauge = new SfLinearGauge();

```



```

LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.LabelPosition = LinearLabelsPosition.Above;
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;

```

0 10 20 30 40 50 60 70 80 90 100



Setting postfix and prefix for labels

You can postfix and prefix values to the scale labels using the [LabelPostfix](#) and [LabelPrefix](#) properties, respectively.

Setting label postfix

The [LabelPostfix](#) property allows to postfix the values to scale labels.

XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale Minimum="0" Maximum="50" LabelPostfix="%" Interval="10"
      ScaleBarStroke="#E0E0E0"
      MajorTickStroke="Gray" MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="0" />
    </gauge:SfLinearGauge.MainScale>
  </gauge:SfLinearGauge>

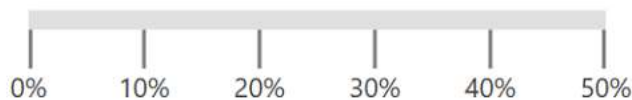
```

C#

```

SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.Minimum = 0;
linearScale.Maximum = 50;
linearScale.Interval = 10;
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.LabelPostfix = "%";
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 0;
sfLinearGauge.MainScale = linearScale;

```



Setting label prefix

The `LabelPrefix` property allows to prefix the values to scale labels.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale Minimum="0" Maximum="50" LabelPrefix="$" Interval="10"
      ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="0">
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.Minimum = 0;
linearScale.Maximum = 50;
linearScale.Interval = 10;
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.LabelPrefix = "$";
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 0;
sfLinearGauge.MainScale = linearScale;
```



Labels visibility

Labels visibility can be customized using the `LabelVisibility` property of linear scale.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale TickPosition="Cross" LabelVisibility="Collapsed"
      MajorTickSize="20" MinorTickSize="9"
      ScaleBarStroke="#E0E0E0" MajorTickStroke="Black"
      MinorTickStroke="Black" LabelStroke="#424242"
      ScaleBarSize="40" MinorTicksPerInterval="3">
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

```

</gauge:LinearScale>
</gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```

C#

```

SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.LabelVisibility = Visibility.Collapsed;
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Black);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Black);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.MajorTickSize = 20;
linearScale.MinorTickSize = 9;
linearScale.TickPosition = LinearTicksPosition.Cross;
linearScale.ScaleBarSize = 40;
linearScale.MinorTicksPerInterval = 3;
sfLinearGauge.MainScale = linearScale;

```



Ranges support in WPF Linear Gauge (SfLinearGauge)

Range is a visual element, which begins and ends at specified values within a scale. You can add any number of ranges to a scale using the array of range objects.

Setting start and end values for range

The start and end values of ranges are set using the [StartValue](#) and [EndValue](#) properties.

XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3">
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartValue="0" EndValue="60"
          RangeStroke="#27BEB7" StartWidth="10" EndWidth="10" RangeOffset="0.4" />
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```

C#

```

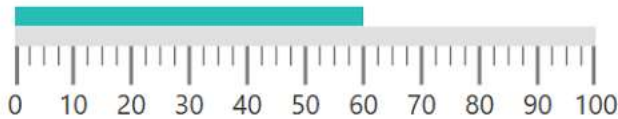
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));

```

```

linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 60;
linearRange.RangeStroke = new SolidColorBrush(Color.FromRgb(39, 190, 183));
linearRange.StartWidth = 10;
linearRange.EndWidth = 10;
linearRange.RangeOffset = 0.4;
linearScale.Ranges.Add(linearRange);
sfLinearGauge.MainScale = linearScale;

```



Range customization

The UI element of a range is customized by changing the [RangeStroke](#) of the linear range, and the appearance of linear range is customized by setting the [StartWidth](#) and [EndWidth](#) properties. By setting the [RangeOpacity](#) of LinearRange, the opacity of the range can be modified.

XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3" ScaleBarLength="300">
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartValue="0" EndValue="50" RangeStroke="#F95C85"
          StartWidth="5"
          EndWidth="20" RangeOpacity="1"/>
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```

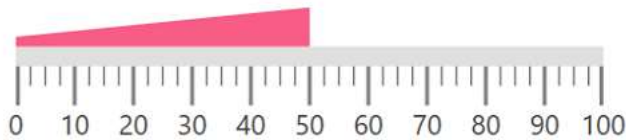
C#

```

SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;

```

```
linearRange.EndValue = 50;
linearRange.RangeOpacity = 1;
linearRange.RangeStroke = new SolidColorBrush(Color.FromRgb(249, 92, 133));
linearRange.StartWidth = 5;
linearRange.EndWidth = 20;
linearScale.Ranges.Add(linearRange);
sfLinearGauge.MainScale = linearScale;
```



Binding range stroke to ticks and labels

You can bind the range's stroke to tick lines and labels within its range by setting the [BindWithRangeStrokeToLabels](#). Stroke of the labels can be set related to stroke of the specified ranges. Similarly, by setting the [BindWithRangeStrokeToTicks](#), stroke of the ticks can be set related to stroke of the specified ranges.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale BindRangeStrokeToLabels="True"
      BindRangeStrokeToTicks="True"
      ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3">
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartValue="0" EndValue="35" StartWidth="15"
          EndWidth="15"
          RangeOffset="5" RangeStroke="Green"/>
        <gauge:LinearRange StartValue="35" EndValue="65"
          StartWidth="15" EndWidth="15" RangeOffset="5" RangeStroke="Yellow"/>
        <gauge:LinearRange StartValue="65" EndValue="100" StartWidth="15"
          EndWidth="15"
          RangeOffset="5" RangeStroke="Red"/>
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

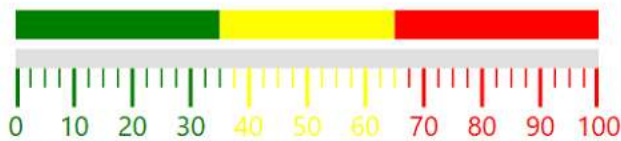
C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
```

```

linearScale.BindRangeStrokeToLabels = true;
linearScale.BindRangeStrokeToTicks = true;
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 35;
linearRange.StartWidth = 15;
linearRange.EndWidth = 15;
linearRange.RangeOffset = 5;
linearRange.RangeStroke = new SolidColorBrush(Colors.Green);
linearScale.Ranges.Add(linearRange);
LinearRange linearRange1 = new LinearRange();
linearRange1.StartValue = 35;
linearRange1.EndValue = 65;
linearRange1.StartWidth = 15;
linearRange1.EndWidth = 15;
linearRange1.RangeOffset = 5;
linearRange1.RangeStroke = new SolidColorBrush(Colors.Yellow);
linearScale.Ranges.Add(linearRange1);
LinearRange linearRange2 = new LinearRange();
linearRange2.StartValue = 65;
linearRange2.EndValue = 100;
linearRange2.StartWidth = 15;
linearRange2.EndWidth = 15;
linearRange2.RangeOffset = 5;
linearRange2.RangeStroke = new SolidColorBrush(Colors.Red);
linearScale.Ranges.Add(linearRange2);
sfLinearGauge.MainScale = linearScale;

```



Setting range position

The range can be placed above or below the scale by one of the following ways:

Setting range offset for linear range

Using the [RangeOffset](#) property, the linear range can be positioned with respect to the linear scale.

XML

```

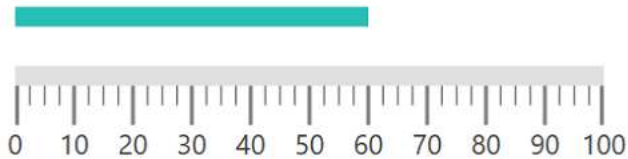
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale RangePosition="Below"
      ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray" MinorTickStroke="Gray"
      LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3">
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartValue="0" EndValue="60" RangeStroke="#27BEB7"
          RangeOffset="-40" StartWidth="10"
          EndWidth="10"/>
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>

```

```
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.RangePosition = LinearRangesPosition.Below;
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 60;
linearRange.RangeOffset = -40;
linearRange.RangeStroke = new SolidColorBrush(Color.FromRgb(39, 190, 183));
linearRange.StartWidth = 10;
linearRange.EndWidth = 10;
linearScale.Ranges.Add(linearRange);
sfLinearGauge.MainScale = linearScale;
```

*Setting range position in linear scale*

The range can be placed above or below the scale by choosing one of the following options available in the [RangePosition](#) property:

1. Above(Default)
2. Below

XML

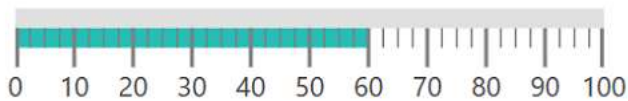
```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale RangePosition="Below"
      ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray" MinorTickStroke="Gray"
      LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3" ScaleBarLength="300">
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartValue="0" EndValue="60"
          RangeStroke="#27BEB7" StartWidth="10" EndWidth="10" />
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```

SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.RangePosition = LinearRangesPosition.Below;
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 60;
linearRange.RangeStroke = new SolidColorBrush(Color.FromRgb(39, 190, 183));
linearRange.StartWidth = 10;
linearRange.EndWidth = 10;
linearScale.Ranges.Add(linearRange);
sfLinearGauge.MainScale = linearScale;

```



Adding multiple ranges

You can add “n” number of ranges to a scale using the `LinearRange` property of range as shown in the following code.

XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3" ScaleBarLength="300">
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartValue="0" EndValue="35" StartWidth="25"
          EndWidth="10"
          RangeOffset="5" RangeOpacity="1" RangeStroke="Green"/>
        <gauge:LinearRange StartValue="65" EndValue="100" StartWidth="10"
          EndWidth="25"
          RangeOffset="5" RangeOpacity="1" RangeStroke="Red"/>
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```

C#

```

SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);

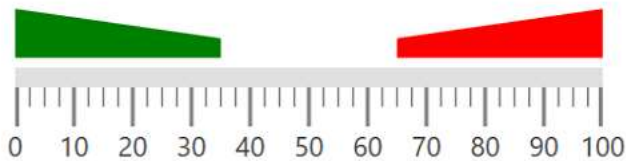
```



```

linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 35;
linearRange.StartWidth = 25;
linearRange.EndWidth = 10;
linearRange.RangeOffset = 5;
linearRange.RangeOpacity = 1;
linearRange.RangeStroke = new SolidColorBrush(Colors.Green);
linearScale.Ranges.Add(linearRange);
LinearRange linearRange1 = new LinearRange();
linearRange1.StartValue = 65;
linearRange1.EndValue = 100;
linearRange1.StartWidth = 10;
linearRange1.EndWidth = 25;
linearRange1.RangeOffset = 5;
linearRange1.RangeOpacity = 1;
linearRange1.RangeStroke = new SolidColorBrush(Colors.Red);
linearScale.Ranges.Add(linearRange1);
sfLinearGauge.MainScale = linearScale;

```



Pointers support in WPF Linear Gauge (SfLinearGauge)

The [LinearGauge](#) provides support to mark values using the [BarPointer](#) and [SymbolPointer](#).

Bar pointer

[BarPointer](#) is used to mark scale values. It starts at the beginning of gauge and ends at the pointer value. You can add bar pointer using the [PointerType](#) property.

XML

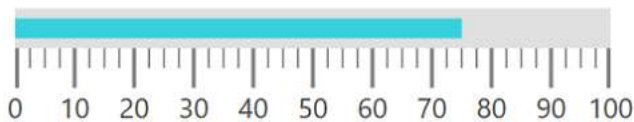
```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242" ScaleBarSize="20"
      MinorTicksPerInterval="3">
      <gauge:LinearScale.Pointers>
        <gauge:LinearPointer PointerType="BarPointer" Value="75"
          BarPointerStroke="#36D1DC"
          BarPointerStrokeThickness="10"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 20;
linearScale.MinorTicksPerInterval = 3;
LinearPointer linearPointer1 = new LinearPointer();
linearPointer1.PointerType = LinearPointerType.BarPointer;
linearPointer1.Value = 75;
linearPointer1.BarPointerStroke = new SolidColorBrush(Color.FromRgb(54, 209, 220));
linearPointer1.BarPointerStrokeThickness = 10;
linearScale.Pointers.Add(linearPointer1);
sfLinearGauge.MainScale = linearScale;
```

*Bar pointer customization*

The UI of **Bar pointer** is customized using the [BarPointerStroke](#) and [BarPointerStrokeThickness](#) properties.

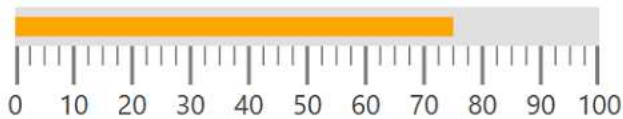
XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="20" MinorTicksPerInterval="3">
      <gauge:LinearScale.Pointers>
        <gauge:LinearPointer PointerType="BarPointer" Value="75"
          BarPointerStroke="Orange"
          BarPointerStrokeThickness="10"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 20;
```

```
linearScale.MinorTicksPerInterval = 3;
LinearPointer linearPointer1 = new LinearPointer();
linearPointer1.PointerType = LinearPointerType.BarPointer;
linearPointer1.Value = 75;
linearPointer1.BarPointerStroke = new SolidColorBrush(Colors.Orange);
linearPointer1.BarPointerStrokeThickness = 10;
linearScale.Pointers.Add(linearPointer1);
sfLinearGauge.MainScale = linearScale;
```



Symbol pointer in linear scale

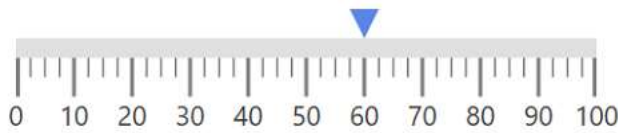
In **SymbolPointer**, the value is pointed by a symbol on the scale.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3">
      <gauge:LinearScale.Pointers>
        <gauge:LinearPointer PointerType="SymbolPointer" Value="60"
          SymbolPointerHeight="15" SymbolPointerWidth="15"
          SymbolPointerPosition="Above" SymbolPointerStroke="#5B86E5"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
LinearPointer linearPointer = new LinearPointer();
linearPointer.PointerType = LinearPointerType.SymbolPointer;
linearPointer.Value = 60;
linearPointer.SymbolPointerHeight = 15;
linearPointer.SymbolPointerWidth = 15;
linearPointer.SymbolPointerPosition = LinearSymbolPointersPosition.Above;
linearPointer.SymbolPointerStroke = new SolidColorBrush(Color.FromRgb(91, 134, 229));
linearScale.Pointers.Add(linearPointer);
sfLinearGauge.MainScale = linearScale;
```



Symbol pointer customization

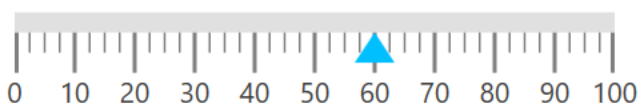
You can modify the size of symbol pointer by changing the [SymbolPointerHeight](#) and [SymbolPointerWidth](#) properties. The stroke of symbol pointer is changed using the [SymbolPointerStroke](#) property.

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3">
      <gauge:LinearScale.Pointers>
        <gauge:LinearPointer PointerType="SymbolPointer" Value="60"
          SymbolPointerHeight="15" SymbolPointerWidth="20"
          SymbolPointerStroke="DeepSkyBlue"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
LinearPointer linearPointer = new LinearPointer();
linearPointer.PointerType = LinearPointerType.SymbolPointer;
linearPointer.Value = 60;
linearPointer.SymbolPointerHeight = 15;
linearPointer.SymbolPointerWidth = 20;
linearPointer.SymbolPointerStroke = new SolidColorBrush(Colors.DeepSkyBlue);
linearScale.Pointers.Add(linearPointer);
sfLinearGauge.MainScale = linearScale;
```



Positioning symbol pointer

SymbolPointer in the scale can be placed above, below, or in between the scale by choosing the following options available in the [SymbolPointerPosition](#) property:

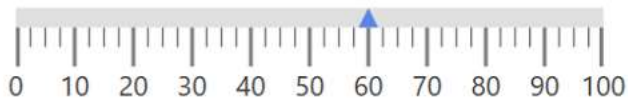
1. Above
2. Below (Default)
3. Cross

XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3" >
      <gauge:LinearScale.Pointers>
        <gauge:LinearPointer PointerType="SymbolPointer" Value="60"
          SymbolPointerPosition="Cross" SymbolPointerHeight="10"
          SymbolPointerWidth="10"
          SymbolPointerStroke="#5B86E5"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>
```

C#

```
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 10;
linearScale.MinorTicksPerInterval = 3;
LinearPointer linearPointer = new LinearPointer();
linearPointer.PointerType = LinearPointerType.SymbolPointer;
linearPointer.Value = 60;
linearPointer.SymbolPointerPosition = LinearSymbolPointersPosition.Cross;
linearPointer.SymbolPointerHeight = 10;
linearPointer.SymbolPointerWidth = 10;
linearPointer.SymbolPointerStroke = new SolidColorBrush(Color.FromRgb(91,
134, 229));
linearScale.Pointers.Add(linearPointer);
sfLinearGauge.MainScale = linearScale;
```



Change symbol pointer shapes

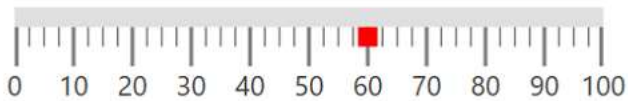
The [SymbolPointerStyle](#) property is used to select symbol pointer style.

XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3">
      <gauge:LinearScale.Pointers>
        <gauge:LinearPointer PointerType="SymbolPointer" Value="60"
          SymbolPointerStyle="Custom">
          <gauge:LinearPointer.SymbolPointerTemplate>
            <DataTemplate>
              <Rectangle Width="10" Height="10" Stroke="Red" StrokeThickness="10">
            </Rectangle>
            </DataTemplate>
          </gauge:LinearPointer.SymbolPointerTemplate>
        </gauge:LinearPointer>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```



Adding multiple pointers

In addition to the default pointer, you can add "n" number of [Pointers](#) to a linear scale using the [Pointers](#) property.

XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="20" MinorTicksPerInterval="3">
      <gauge:LinearScale.Pointers>
        <gauge:LinearPointer PointerType="SymbolPointer" Value="60"
          SymbolPointerHeight="15" SymbolPointerWidth="15"
          SymbolPointerPosition="Above" SymbolPointerStroke="#5B86E5"/>
        <gauge:LinearPointer PointerType="BarPointer" Value="75"
          BarPointerStroke="#36D1DC" BarPointerStrokeThickness="10"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```

C#

```

SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224,
224));
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);

```

```

linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));
linearScale.ScaleBarSize = 20;
linearScale.MinorTicksPerInterval = 3;
LinearPointer linearPointer1 = new LinearPointer();
linearPointer1.PointerType = LinearPointerType.BarPointer;
linearPointer1.Value = 75;
linearPointer1.BarPointerStroke = new SolidColorBrush(Color.FromRgb(54, 209, 220));
linearPointer1.BarPointerStrokeThickness = 10;
linearScale.Pointers.Add(linearPointer1);
LinearPointer linearPointer = new LinearPointer();
linearPointer.PointerType = LinearPointerType.SymbolPointer;
linearPointer.Value = 60;
linearPointer.SymbolPointerHeight = 15;
linearPointer.SymbolPointerWidth = 15;
linearPointer.SymbolPointerPosition = LinearSymbolPointersPosition.Above;
linearPointer.SymbolPointerStroke = new SolidColorBrush(Color.FromRgb(91, 134, 229));
linearScale.Pointers.Add(linearPointer);
sfLinearGauge.MainScale = linearScale;

```



Setting animation for pointer

The [EnableAnimation](#) is a Boolean property, which is used to enable or disable the animation of the [Pointers](#) in linear gauge and the animation speed can be controlled with [AnimationDuration](#) property.

XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="20" MinorTicksPerInterval="3">
      <gauge:LinearScale.Pointers>
        <gauge:LinearPointer EnableAnimation="True" AnimationDuration="500"
          PointerType="SymbolPointer" Value="60"
          SymbolPointerHeight="15" SymbolPointerWidth="15"
          SymbolPointerPosition="Above" SymbolPointerStroke="#5B86E5"/>
        <gauge:LinearPointer EnableAnimation="True" AnimationDuration="1000"
          PointerType="BarPointer" Value="75" BarPointerStroke="#36D1DC"
          BarPointerStrokeThickness="10"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

```

C#

```

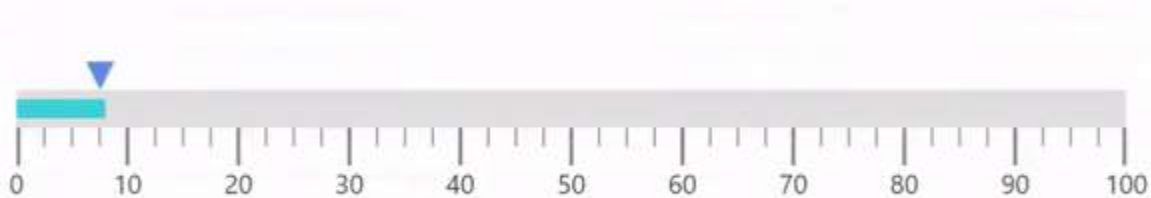
SfLinearGauge sfLinearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale()

```

```

{
    ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224)),
    MajorTickStroke = new SolidColorBrush(Colors.Gray),
    MinorTickStroke = new SolidColorBrush(Colors.Gray),
    LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66)),
    ScaleBarSize = 20,
    MinorTicksPerInterval = 3,
};
LinearPointer linearPointer1 = new LinearPointer()
{
    PointerType = LinearPointerType.BarPointer,
    Value = 75,
    BarPointerStroke = new SolidColorBrush(Color.FromRgb(54, 209, 220)),
    BarPointerStrokeThickness = 10,
    EnableAnimation = true,
    AnimationDuration = 1000
};
linearScale.Pointers.Add(linearPointer1);
LinearPointer linearPointer = new LinearPointer()
{
    PointerType = LinearPointerType.SymbolPointer,
    Value = 60,
    SymbolPointerHeight = 15,
    SymbolPointerWidth = 15,
    SymbolPointerPosition = LinearSymbolPointersPosition.Above,
    SymbolPointerStroke = new SolidColorBrush(Color.FromRgb(91, 134, 229)),
    EnableAnimation = true,
    AnimationDuration = 500
};
linearScale.Pointers.Add(linearPointer);
sfLinearGauge.MainScale = linearScale;
this.Content = sfLinearGauge;

```



Orientation customization in WPF Linear Gauge (SfLinearGauge)

The **LinearGauge** control supports horizontal and vertical orientations. By default, the **LinearGauge** is rendered with horizontal orientation. You can change the orientation using the [Orientation](#) property.

XML

```

<gauge:SfLinearGauge Orientation="Vertical">
  <gauge:SfLinearGauge.MainScale>
    <gauge:LinearScale ScaleBarStroke="#E0E0E0" MajorTickStroke="Gray"
      MinorTickStroke="Gray" LabelStroke="#424242"
      ScaleBarSize="10" MinorTicksPerInterval="3">
    </gauge:LinearScale>
  </gauge:SfLinearGauge.MainScale>
</gauge:SfLinearGauge>

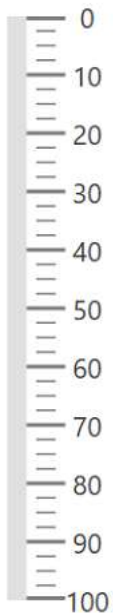
```



```
</gauge:SfLinearGauge.MainScale>  
</gauge:SfLinearGauge>
```

C#

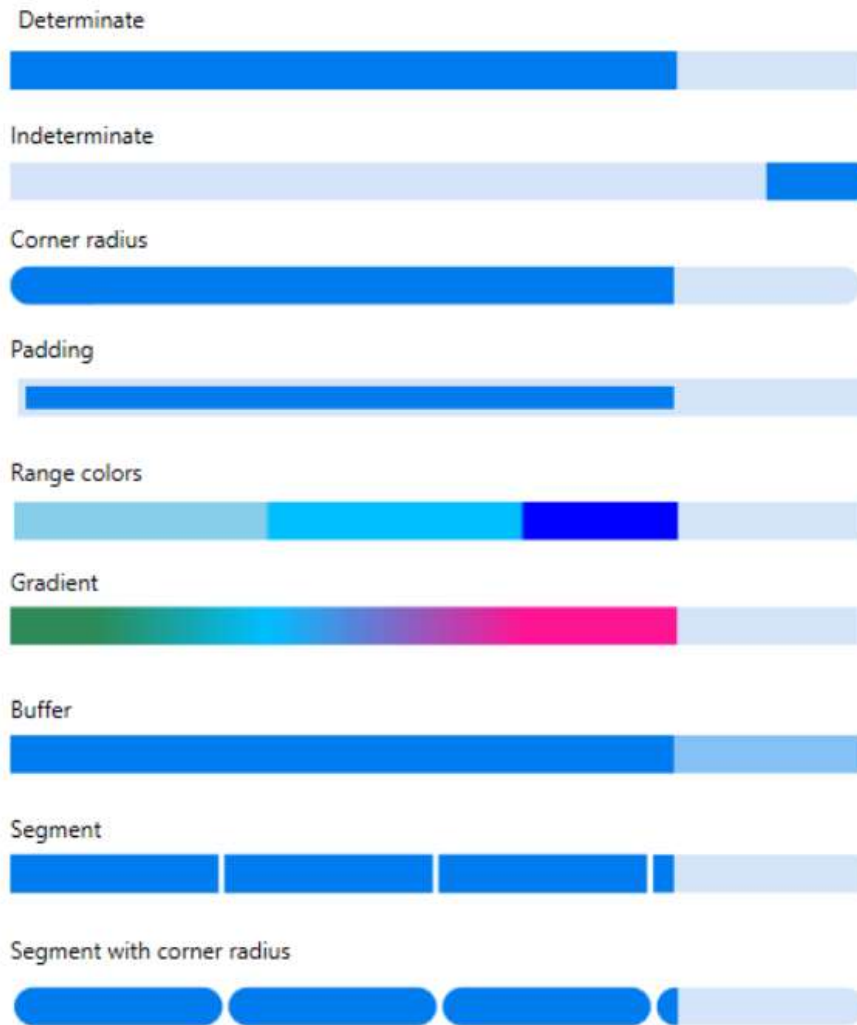
```
SfLinearGauge sfLinearGauge = new SfLinearGauge();  
sfLinearGauge.Orientation = System.Windows.Controls.Orientation.Vertical;  
LinearScale linearScale = new LinearScale();  
linearScale.ScaleBarStroke = new SolidColorBrush(Color.FromRgb(224, 224, 224));  
linearScale.MajorTickStroke = new SolidColorBrush(Colors.Gray);  
linearScale.MinorTickStroke = new SolidColorBrush(Colors.Gray);  
linearScale.LabelStroke = new SolidColorBrush(Color.FromRgb(66, 66, 66));  
linearScale.ScaleBarSize = 10;  
linearScale.MinorTicksPerInterval = 3;  
sfLinearGauge.MainScale = linearScale;
```



SfLinearProgressBar

WPF Linear ProgressBar Overview

The SfLinearProgressBar control for WPF provides a customizable visual to indicate the progress of an operation and let users know the remaining time for completion.



Key features

- **Determinate and indeterminate:** Determinate shows specific quantity of progress that occurred and indeterminate shows a redundant animations of linear progress.
- **Corner radius:** It customizes to frame rounded edges in the Linear ProgressBar.
- **Padding:** Padding generates space between track bar and progress bar in Linear ProgressBar.
- **Ranges:** Specifies the start position and end position to visualize multiple ranges with different colors that are mapped to each range.
- **Gradient:** Gradient shows change in intensity of the colors during the linear progress.
- **Segments:** Segment splits the ProgressBar into multiple segments and indicates the progress.
- **Segment with corner radius:** Segment with corner radius splits the Linear ProgressBar into multiple segments with the rounded edges at the corner.

Creating a simple application with Linear ProgressBar

You can create a WPF application with the SfLinearProgressBar control using the following steps:

Assembly deployment

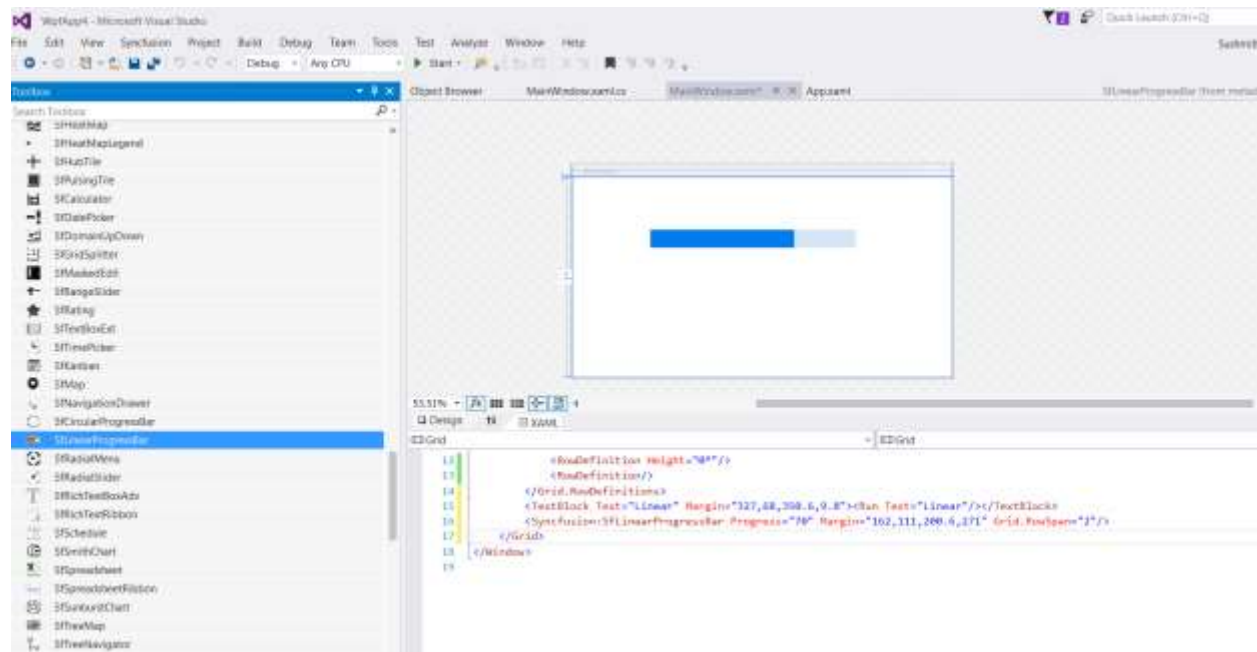
Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

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 SfLinearProgressBar 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 reference to the project: * Syncfusion.SfProgressBar.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the SfLinearProgressBar 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: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">
```

```
<TextBlock Text="Linear"/>
<Syncfusion:SfLinearProgressBar Progress="70" "/>
</Grid>
</Window>
```

Adding control through code behind

To add control manually through code behind, follow the given steps:

1. Add the following required assembly reference to the project: * Syncfusion.SfProgressBar.WPF 2. Import the Linear ProgressBar namespace

using Syncfusion.UI.Xaml.ProgressBar;

3. Create an Linear 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();
            SfLinearProgressBar linear = new SfLinearProgressBar();
            linear.Progress = 70;
            grid.Children.Add(linear);
        }
    }
}
```

Theme

SfLinearProgressBar supports various built-in themes. Refer to the below links to apply themes for the SfLinearProgressBar,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Different states in Linear ProgressBar

States help to visualize the progress of a task in different modes. You can configure states of the linear 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.



Indeterminate

By enabling the [IsIndeterminate](#) property, the state of the progressbar can be changed to indeterminate when the progress cannot be estimated or is not being calculated.

XML



C#

```
SfLinearProgressBar linear = new SfLinearProgressBar { Progress = 70,
IsIndeterminate=true};
linear.Width = 500;
linear.Height = 20;
grid.Children.Add(linear);
```



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 primary and secondary tasks progress simultaneously. The [SecondaryProgress](#) property can be set to visualize secondary progress and separate color for the secondary progress can be set by [SecondaryProgressColor](#) property.

XML



C#

```
SfLinearProgressBar linear = new SfLinearProgressBar { Progress = 70,
SecondaryProgress=90};
linear.Width = 500;
linear.Height = 20;
grid.Children.Add(linear);
```



Segments in Linear ProgressBar

Divides the progressbar into multiple segments using the API.

Segment

To visualize the progress of multiple sequential tasks, split the progressbar into multiple segments by setting the [SegmentCount](#).

XML

```
<Syncfusion:SfLinearProgressBar Progress="70" SegmentCount="4" />
```

C#

```
SfLinearProgressBar linear = new SfLinearProgressBar();
linear.Progress = 70;
linear.SegmentCount = 5;
linear.Width = 500;
linear.Height = 20;
grid.Children.Add(linear);
```



Segment with corner radius

Corner radius helps to generate rounded edges for the progressbar. This can be achieved through the [IndicatorCornerRadius](#) property.

XML

```
<Syncfusion:SfLinearProgressBar Progress="70" SegmentCount="4"
IndicatorCornerRadius="10"/>
```

C#

```
SfLinearProgressBar linear = new SfLinearProgressBar();
linear.Progress = 70;
linear.SegmentCount = 5;
linear.IndicatorCornerRadius = 10;
linear.Width = 500;
linear.Height = 20;
grid.Children.Add(linear);
```



Appearance in Linear ProgressBar

You can highly customize the appearance of the linear progressbar in the following ways.

Corner radius

The [IndicatorCornerRadius](#) property is used to frame rounded edges in the linear progressbar as demonstrated in the following code sample.

XML

```
<Syncfusion:SfLinearProgressBar Progress="70" IndicatorCornerRadius="10"/>
```

C#

```
SfLinearProgressBar linear = new SfLinearProgressBar { Progress = 70,
IndicatorCornerRadius=10};
linear.Width = 500;
linear.Height = 20;
grid.Children.Add(linear);
```

**Padding**

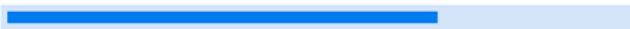
Padding helps to generate space between track bar and progressbar in the linear progressbar. The [IndicatorPadding](#) property is used to set the padding value.

XML

```
<Syncfusion:SfLinearProgressBar Progress="70" IndicatorPadding="5"/>
```

C#

```
SfLinearProgressBar linear = new SfLinearProgressBar();
linear.IndicatorPadding = new Thickness(5);
linear.Progress = 70;
linear.Width = 500;
linear.Height = 20;
grid.Children.Add(linear);
```

**Color Customization**

You can customize the color of the linear 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:SfLinearProgressBar Progress="70" ProgressColor="LawnGreen"
TrackColor="DarkOliveGreen"/>
```

C#

```
SfLinearProgressBar linear = new SfLinearProgressBar();
linear.Progress = 70;
linear.ProgressColor = new SolidColorBrush(Colors.LawnGreen);
```

```
linear.TrackColor = new SolidColorBrush(Colors.DarkOliveGreen);
grid.Children.Add(linear);
```



Range Colors

You can visualize 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](#).

- [Color](#): Represents the color to the specified range.
- [Start](#): Represents the start range of the color.
- [End](#): Represents the end range of the color.

You can visualize the multiple ranges with different solid colors that are mapped to each range to enhance the readability of progress.

XML

```
<Syncfusion:SfLinearProgressBar Progress="80">
  <Syncfusion:SfLinearProgressBar.RangeColors>
    <Syncfusion:RangeColorCollection>
      <Syncfusion:RangeColor Color="BlanchedAlmond" Start="10" End="30"/>
      <Syncfusion:RangeColor Color="Coral" Start="30" End="60"/>
      <Syncfusion:RangeColor Color="Crimson" Start="60" End="100"/>
    </Syncfusion:RangeColorCollection>
  </Syncfusion:SfLinearProgressBar.RangeColors>
</Syncfusion:SfLinearProgressBar>
```

C#

```
SfLinearProgressBar linear = new SfLinearProgressBar();
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 });
linear.RangeColors = rangeColors;
linear.Progress = 80;
linear.Width = 500;
linear.Height = 20;
grid.Children.Add(linear);
```



Gradient

Gradient shows change in intensity of the colors during the progress. [IsGradient](#) property in [RangeColor](#) class to get the gradient effect in the colors applied to the progressbar.

XML

```
<Syncfusion:SfLinearProgressBar Progress="80">
  <Syncfusion:SfLinearProgressBar.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:SfLinearProgressBar.RangeColors>
</Syncfusion:SfLinearProgressBar>
```

C#

```
SfLinearProgressBar linear = new SfLinearProgressBar();
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 });
linear.RangeColors = rangeColors;
linear.Progress = 80;
linear.Width = 500;
linear.Height = 20;
grid.Children.Add(linear);
```



AnimationDuration

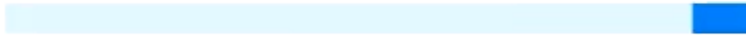
You can customize the duration for completing one animation cycle and it applies when the [IsIndeterminate](#) is true. The default value is 3000ms.

XML

```
<Grid Name="grid">
  <Syncfusion:SfLinearProgressBar
    Width="250"
    Height="4"
    AnimationDuration="00:00:01"
    IsIndeterminate="True"
    Progress="50" />
</Grid>
```

C#

```
SfLinearProgressBar linear = new SfLinearProgressBar();
linear.Progress = 50;
linear.IsIndeterminate = true;
linear.Width = 250;
linear.Height = 4;
linear.AnimationDuration = new TimeSpan(0, 0, 1);
grid.Children.Add(linear);
```



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:SfLinearProgressBar
    Width="250"
    Height="4"
    IndicatorCornerRadius="10"
    IsIndeterminate="True"
    Progress="50">
    <Syncfusion:SfLinearProgressBar.AnimationEasing>
      <BounceEase
        Bounces="20"
        Bounciness="5"
        EasingMode="EaseOut" />
    </Syncfusion:SfLinearProgressBar.AnimationEasing>
  </Syncfusion:SfLinearProgressBar>
</Grid>
```

C#

```
SfLinearProgressBar linear = new SfLinearProgressBar();
linear.Progress = 50;
linear.IsIndeterminate = true;
linear.Width = 250;
linear.Height = 4;
BounceEase bounceEase = new BounceEase();
bounceEase.Bounces = 20;
bounceEase.Bounciness = 5;
bounceEase.EasingMode = EasingMode.EaseOut;
linear.AnimationEasing = bounceEase;
grid.Children.Add(linear);
```



SfMap

WPF Maps (SfMap) Overview

A map is a graphical representation of geographical data. It is used to represent the statistical data of a particular geographical area on Earth. By using pan and zoom, the maps can be navigated. Maps supports enhanced data visualization with bubbles and labels. Bubbles visualize the data that is bound to the map.

Use case scenario

The Maps control can be used in following cases:

1. To visualize weather data.
2. To visualize geographical statistics information.
3. To visualize the density or availability of resources in an area.
4. To visualize political information.
5. To visualize the layout of a building.

Key Concepts of Map

A map contains a set of elements, including shapes, bubbles, annotations, and data items, that is maintained in layers. ShapeFileLayer is one of the layers that can be used to generate map shapes and bind business objects with them. Bubbles and MapItems enhance the data visualization capabilities of the map with data binding.

Tree map-like support provides rich UI for shapes and bubbles. Annotation and CustomDataBinding items shows additional information on the map.

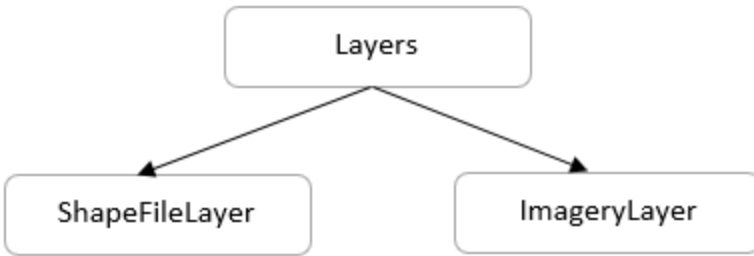
Maps also provide multilayer support, which can be added as a sublayer, and shapes such as polygons, polylines, and points can be added to them.

Options like zooming, panning, and map selection extend the interactivity of the map.



Structure of Map

A map is maintained through layers. The shape file layer is one of the layers that consist of vector shapes, bubbles and data visual items. The imagery layer is used to visualize satellite, aerial, street map, or other imagery tiles without using shapefiles.



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Getting Started with WPF Maps (SfMap)

This section explains the steps required to configure the [SfMap](#) and add basic elements to it using various APIs.

Adding SfMap reference

You can add SfMap reference using one of the following methods:

Method 1: Adding SfMap reference from nuget.org

Syncfusion WPF components are available in [nuget.org](#). To add SfMap to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.SfMaps.WPF](#), and then install it.

![Adding SfMap reference from NuGet](Getting-Started_images/Adding SfMaps reference.png)

Method 2: Adding SfMap reference from toolbox

You can drag the SfMap control from the toolbox and drop it to the designer. It will automatically reference the required assemblies and add the namespace to the page.

Method 3: Adding SfMap 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 map to your project.

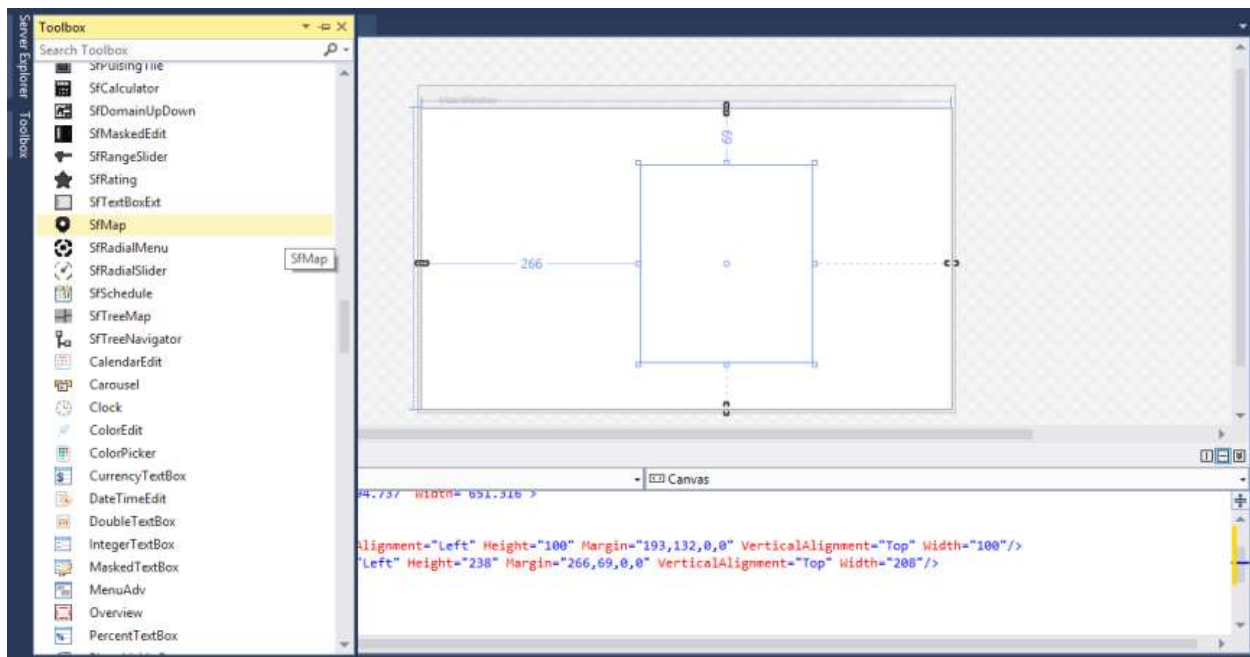
Create SfMap control

There are three possible ways to create a simple SfMap control.

Through Visual Studio

To create the SfMap control through Visual Studio, drag SfMap from Toolbox and drop it to the designer.

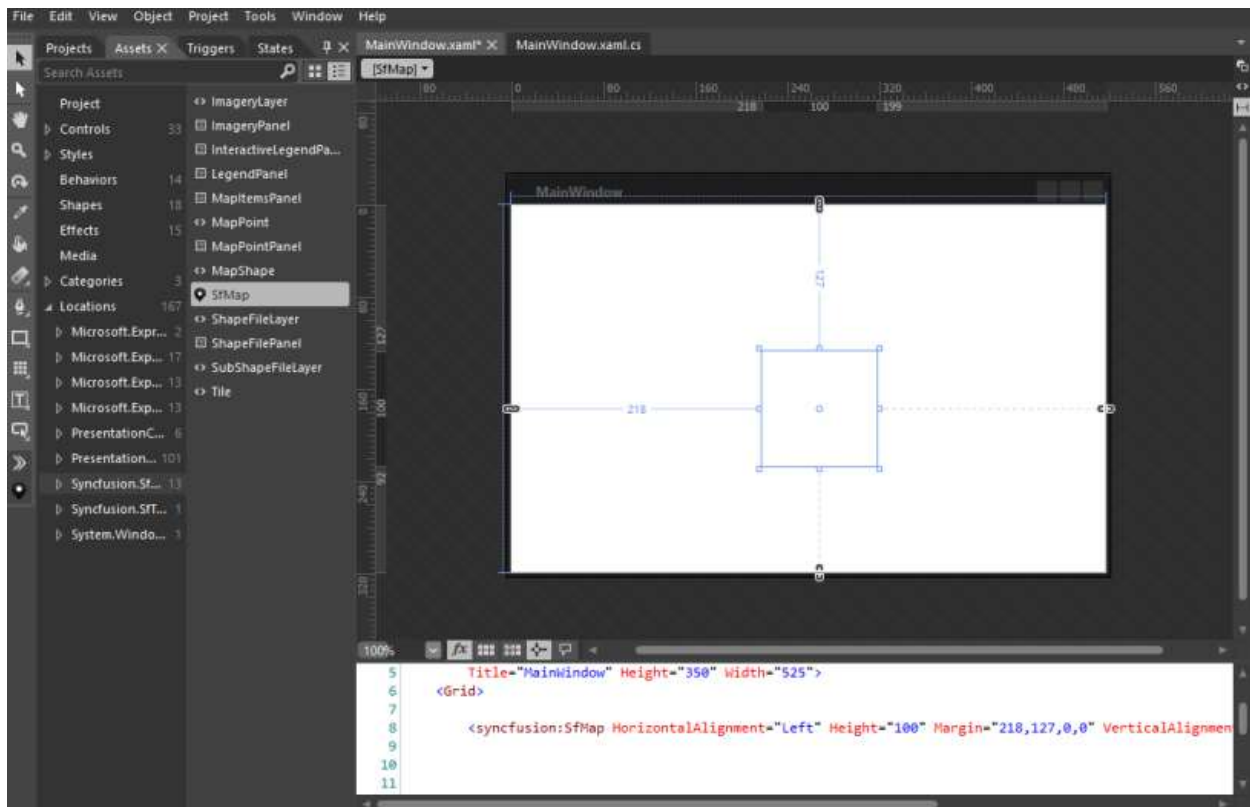
It generates the following the SfMap control.



Through Expression Blend

The SfMap control can also be created and configured by using Expression Blend. Follow these steps to do so.

1. Create a WPF project in Expression Blend and reference the following assemblies.
2. Syncfusion.SfMaps.Wpf
3. Syncfusion.Shared.Wpf
4. Search for SfMap in the Toolbox.
5. Drag SfMap to the designer. It generates the SfMap control with one child element.



Through XAML and C#

Adding namespace

Add the following namespace.

XML

```
xmlns:syncfusion="clr-namespace:Syncfusion.UI.Xaml.Maps;assembly=Syncfusion.SfMaps.WPF"
```

C#

```
using Syncfusion.UI.Xaml.Maps;
```

Initializing maps

You can create the SfMap control programmatically through XAML and C#. In the following code example.

XML

```
<syncfusion:SfMap>
</syncfusion:SfMap>
```

C#

```
SfMap syncMap = new SfMap();
```

Adding layers

The maps control is maintained through [Layers](#). It can be either ShapeFileLayer or ImageryLayer. The following example will show to add ShapeFileLayer on map.

XML

```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer></syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap maps = new SfMap();
ShapeFileLayer shapeLayer = new ShapeFileLayer();
syncMap.Layers.Add(shapeLayer);
```

Configure the SfMap Control

Read and Load the Shapes

The Maps control supports reading and loading shape files. A shape file is a set of files that are stored in a non-topological geometry and the attribute information for the spatial features and records in a data set. Spatial features and records are stored as shapes that consist of set vector coordinates. A computer program can read the content of the shape files and parse them as vector elements. The Maps control also reads and parses the spatial information of a shape file into the graphical elements.

As mentioned earlier, a shape file can be a set of files or a single file. Generally, the shape file contains the following files:

Main file (.shp)

dBase file (.dbf)

All files must adhere to the 8.3 naming conventions. The Main file and dBase file must have the same prefix, i.e., they must have the same file name. This naming convention allows you to identify specific geographical information.

The main file (.shp) contains a fixed-length file header followed by the variable-length records. Each variable-length record is made up of a fixed-length record header followed by the variable-length record contents.

The dBase file (.dbf) contains any desired feature attributes or attributes keys, where other tables can be joined. Its format is a standard .dbf file used by many table-based applications in Windows™ and DOS. Any set of fields can be present in the table.

For more information about the 8.3 naming convention, shape files and their descriptions, visit the following link:

<http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>

Maps read the main file and create the map shapes. The associated .dbf file contents are then incorporated with the shapes created from the main files.

Attach the Shape file with Map

To read the shape file using Map, the shape file's main file and .dbf file need to be added as an embedded resource in the application project. Then, the main file's path has to be given in the Uri file of the shape file layer.

About the Uri property

Uri is the string type property that retrieves the location of the shape file that is added as an embedded resource.

Structure of Uri property

The Uri property contains the following information:

Namespace

Subfolder names

ShapeFilename.shp

XML

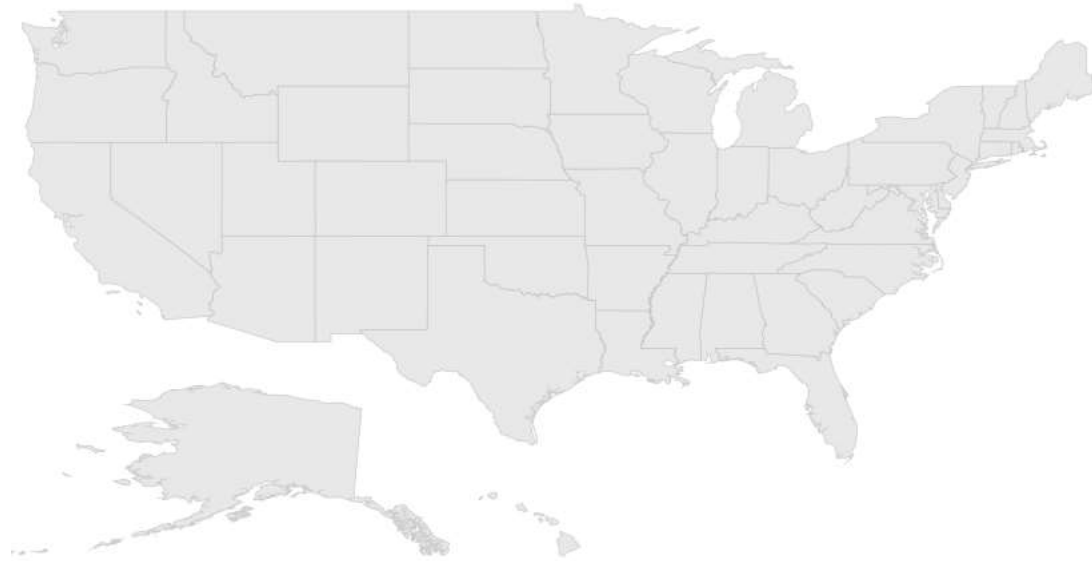
```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="GettingStarted.ShapeFiles.usa_state.shp">
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap maps = new SfMap();
ShapeFileLayer shapeLayer = new ShapeFileLayer();
shape.Uri = "GettingStarted.ShapeFiles.usa_state.shp";
maps.Layers.Add(shapeLayer);
this.Content = maps;
```

In the above code sample, GettingStarted is the namespace, ShapeFile is the subfolder name, and usa_state.shp is the name of the shape file.

The above code example results in the following output



Data Binding in Map

Data can be bound to the shape file layer using the [ItemsSource](#), [ShapeIDPath](#), and [ShapeIDTableField](#) properties.

The [Populate data](#) section gives the detailed explanation of data binding.

XML

```
<Grid>
<Grid.DataContext>
<local:ViewModel />
</Grid.DataContext>
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer ItemsSource="{Binding ElectionResults}"
ShapeIDPath="State"
ShapeIDTableField="STATE_NAME"
Uri="GettingStarted.ShapeFiles.usa_states.shp">
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeValuePath="Electors" />
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>
```

C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        ViewModel viewModel = new ViewModel();
        this.DataContext = viewModel;
        SfMap maps = new SfMap();
        ShapeFileLayer shapeLayer = new ShapeFileLayer();
```

```

shapeLayer.Uri = "GettingStarted.ShapeFiles.usa_state.shp";
shapeLayer.ItemsSource = ViewModel.ElectionResults;
shapeLayer.ShapeIDTableField = "STATE_NAME";
shapeLayer.ShapeIDPath = "State";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeValuePath = "Electors";
shapeLayer.ShapeSettings = shapeSetting;
maps.Layers.Add(shapeLayer);
this.Content = maps;
}
}

```

Adding markers

Markers are used to identify the shapes. They can be added to the shape file layers as demonstrated in the following code sample. Markers can be customized using the customization properties in the shape file layer.

The detailed explanation of marker and its customization are provided under [Markers](#) section.

XML

```

<Grid>
<Grid.DataContext>
<local:ViewModel />
</Grid.DataContext>
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="GettingStarted.ShapeFiles.usa_state.shp"
Markers="{Binding Models}">
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>

```

C#

```

ViewModel viewModel = new ViewModel();
this.DataContext = viewModel;
SfMap maps = new SfMap();
ShapeFileLayer shapeLayer = new ShapeFileLayer();
shapeLayer.Uri = "GettingStarted.ShapeFiles.usa_state.shp";
shapeLayer.Markers = viewModel.Models;
maps.Layers.Add(shapeLayer);
this.Content = maps;

```

The following code sample shows how to define a marker class and add markers.

C#

```

public class Model
{
public string Label { get; set; }
public string Longitude { get; set; }
public string Latitude { get; set; }
}

```

```

public class ViewModel
{
    public ObservableCollection<Model> Models { get; set; }
    public ViewModel()
    {
        this.Models = new ObservableCollection<Model>
        {
            new Model() { Label = "California", Latitude = "37", Longitude = "-120" }
        };
    }
}

```

Color mapping

The color mapping support allows you customize the shape colors based on the underlying value of shape received from the bound data. Both the range color mapping and equal color mapping are supported in maps.

The detailed explanation of color mapping is provided in [Shapes Color Customization](#) section.

XML

```

<syncfusion:ShapeSetting.FillSetting>
<syncfusion:ShapeFillSetting AutoFillColors="False">
<syncfusion:ShapeFillSetting.ColorMappings>
<syncfusion:EqualsColorMapping Value="Romney"
Color="#D84444" />
<syncfusion:EqualsColorMapping Value="Obama"
Color="#316DB5" />
</syncfusion:ShapeFillSetting.ColorMappings>
</syncfusion:ShapeFillSetting>
</syncfusion:ShapeSetting.FillSetting>

```

C#

```

// ShapeFillSetting
ShapeFillSetting shapeFillSetting = new ShapeFillSetting
{
    AutoFillColors = false
}
// EqualColorMapping
EqualsColorMapping romeColorMapping = new EqualsColorMapping
{
    Color = (Color)new ColorConverter().ConvertFrom("#D84444"),
    LegendLabel = "Romney",
    Value = "Romney"
}
EqualsColorMapping obamaColorMapping = new EqualsColorMapping
{
    Color = (Color)new ColorConverter().ConvertFrom("#316DB5"),
    LegendLabel = "Obama",
    Value = "Obama"
}
// Adding EqualColorMapping to ColorMappings.
shapeFillSetting.ColorMappings.Add(romeColorMapping);
shapeFillSetting.ColorMappings.Add(obamaColorMapping)

```

```
shapeSetting.FillSetting = shapeFillSetting;
shapeLayer.ShapeSettings = shapeSetting;
```

Adding legends

Legends interpret what the map displays. They can be added to the shape file layer as demonstrated in the following code sample. Legends will be displayed based on the data bound to the layer, and color mapping plays a major role in enabling legends.

The detailed explanation of legend is provided under [Legend](#) section.

XML

```
<syncfusion:ShapeFileLayer x:Name="shapeLayer"
LegendColumnSplit="2"
LegendPositionX="950"
LegendPositionY="580"
LegendVisibility="Visible"
LegendIcon="Rectangle">
</syncfusion:ShapeFileLayer>
```

C#

```
// Adding legend
shapeLayer.LegendColumnSplit = 2;
shapeLayer.LegendPositionX = 950;
shapeLayer.LegendPositionY = 580;
shapeLayer.LegendVisibility = Visibility.Visible;
shapeLayer.LegendIcon = LegendIcons.Rectangle;
```

The following code sample gives you the complete code for map with markers and legends.

XML

```
<Grid>
<Grid.DataContext>
<local:ViewModel />
</Grid.DataContext>
<syncfusion:SfMap x:Name="map"
EnableZoom="False"
EnablePan="False">
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer x:Name="shapeLayer"
Background="White"
EnableSelection="True"
Uri="GettingStarted.ShapeFiles.usa_state.shp"
ItemsSource="{Binding ElectionResults}"
MapItemsVisibility="Hidden"
ShapeIDPath="State"
ShapeIDTableField="STATE_NAME"
Markers="{Binding Models}"
MarkerIconFill="LimeGreen"
MarkerLabelForeground="White"
LegendColumnSplit="2"
LegendPositionX="370"
LegendPositionY="240"
```

```

LegendVisibility="Visible"
LegendIcon="Rectangle">
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeFill="#E5E5E5"
ShapeStroke="White"
ShapeStrokeThickness="0.5"
SelectedShapeColor="#CEBF93"
ShapeValuePath="Electors"
ShapeColorValuePath="Candidate">
<syncfusion:ShapeSetting.FillSetting>
<syncfusion:ShapeFillSetting AutoFillColors="False">
<syncfusion:ShapeFillSetting.ColorMappings>
<syncfusion:EqualsColorMapping Value="Romney"
Color="#D84444" />
<syncfusion:EqualsColorMapping Value="Obama"
Color="#316DB5" />
</syncfusion:ShapeFillSetting.ColorMappings>
</syncfusion:ShapeFillSetting>
</syncfusion:ShapeSetting.FillSetting>
</syncfusion:ShapeSetting>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>

```

C#

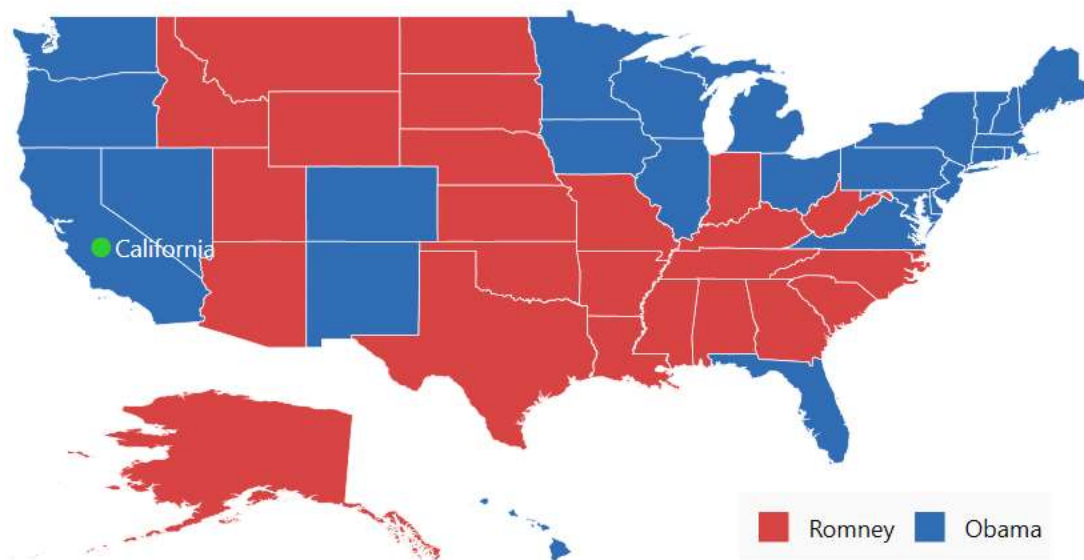
```

// Setting DataContext.
ViewModel viewModel = new ViewModel();
this.DataContext = viewModel;
// SfMap.
SfMap maps = new SfMap();
// ShapeFileLayer.
ShapeFileLayer shapeLayer = new ShapeFileLayer();
shapeLayer.Background = new SolidColorBrush(Colors.White);
shapeLayer.EnableSelection = true;
shapeLayer.Uri = "GettingStarted.ShapeFiles.usa_state.shp";
shapeLayer.ItemsSource = viewModel.ElectionResults;
shapeLayer.MapItemsVisibility = Visibility.Hidden;
shapeLayer.ShapeIDPath = "State";
shapeLayer.ShapeIDTableField = "STATE_NAME";
shapeLayer.Markers = viewModel.Models;
shapeLayer.MarkerIconFill = new SolidColorBrush(Colors.LimeGreen);
shapeLayer.MarkerLabelForeground = new SolidColorBrush(Colors.White);
// Adding legend
shapeLayer.LegendColumnSplit = 2;
shapeLayer.LegendPositionX = 950;
shapeLayer.LegendPositionY = 580;
shapeLayer.LegendVisibility = Visibility.Visible;
shapeLayer.LegendIcon = LegendIcons.Rectangle;
// ShapeSetting
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeFill = (SolidColorBrush)new
BrushConverter().ConvertFrom("#E5E5E5");
shapeSetting.ShapeStroke = new SolidColorBrush(Colors.White);

```

```
shapeSetting.ShapeStrokeThickness = 0.5;
shapeSetting.SelectedShapeColor = (SolidColorBrush)new
BrushConverter().ConvertFrom("#CEBF93");
shapeSetting.ShapeValuePath = "Electors";
shapeSetting.ShapeColorValuePath = "Candidate";
// ShapeFillSetting
ShapeFillSetting shapeFillSetting = new ShapeFillSetting
{
    AutoFillColors = false
};
// EqualColorMapping
EqualsColorMapping romneyColorMapping = new EqualsColorMapping
{
    Color = (Color)new ColorConverter().ConvertFrom("#D84444"),
    LegendLabel = "Romney",
    Value = "Romney"
};
EqualsColorMapping obamaColorMapping = new EqualsColorMapping
{
    Color = (Color)new ColorConverter().ConvertFrom("#316DB5"),
    LegendLabel = "Obama",
    Value = "Obama"
};
// Adding EqualColorMapping to ColorMappings.
shapeFillSetting.ColorMappings.Add(romneyColorMapping);
shapeFillSetting.ColorMappings.Add(obamaColorMapping);
shapeSetting.FillSetting = shapeFillSetting;
shapeLayer.ShapeSettings = shapeSetting;
// Adding ShapeFileLayer into Map.
maps.Layers.Add(shapeLayer);
maps.EnableZoom = false;
maps.EnablePan = false;
this.Content = maps;
```

The following screenshot illustrates the result of the above code sample.

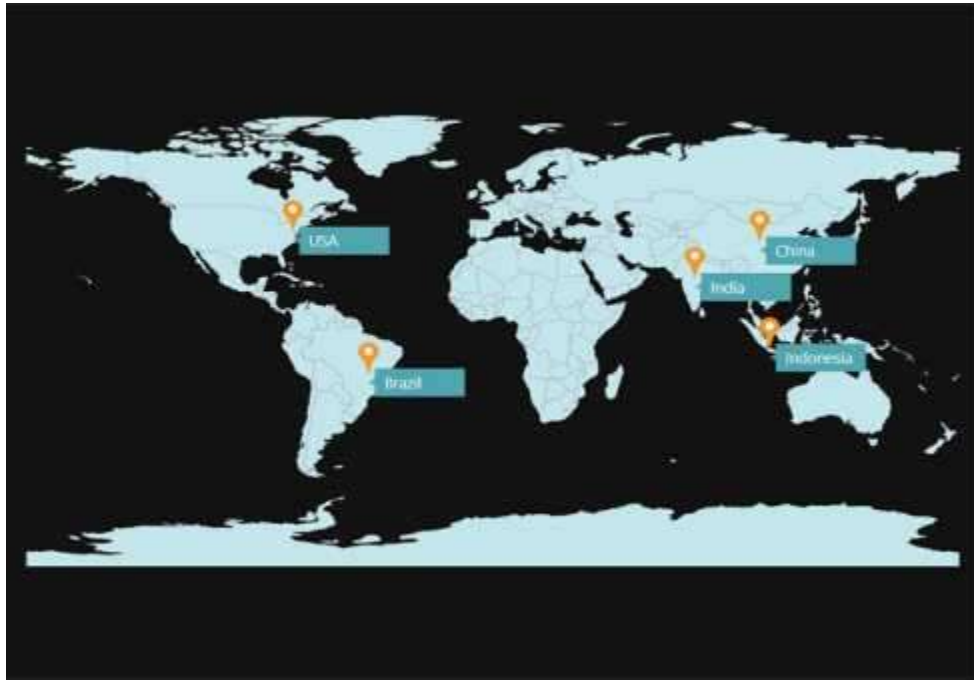


You can download the complete [Getting started](#) sample.

Theme

SfMap supports various built-in themes. Refer to the below links to apply themes for the SfMap,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

See also

[How to show google map in WPF SfMap](#)

[How to view bing map using WPF SfMap](#)

[How to render custom data source in SfMap](#)

[How to customize the markers in maps](#)

[How to drilldown map layers](#)

Populate Data in WPF Maps (SfMap)

This section explains how to populate shape data for providing data input to the maps control and usage of the [ItemsSource](#) property.

Data binding

The maps control supports data binding using the [ItemsSource](#) property in the shape layers.

The following properties in shape layers are used for binding data in the maps control:

- ItemsSource
- ShapeIDPath

- [ShapeIDTableField](#)

ItemsSource

This is the basic property that exposes data binding for Maps. [ItemsSource](#) is the property that accepts the collection type values. For example, the ItemsSource property accepts the ObservableCollections, Lists, and LinqResult values.

ShapeIDPath

[ShapeIDPath](#) is the string type property used to refer to the ID of a shape from the ItemsSource. The ItemsSource property must have a property with name of the ShapeIDPath. The ShapeIDPath and the ShapeIDTableField properties are related to each other (refer to ShapeIDTableField for more details).

ShapeIDTableField

The [ShapeIDTableField](#) property is similar to the ShapeIDPath. It is a string type property that refers to the column name in the dbf file to identify the shape. When values of the ShapeIDPath property in the ItemsSource and the value of ShapeIDTableField in the .dbf file match, then the associated object from the ItemsSource is bound to the corresponding shape.

ShapeValuePath

[ShapeValuePath](#) is a string type property used to define the object bound to the shapes of the map. The ShapeValuePath must be the name of any property that is defined in the ItemsSource items. The ShapeIDPath, ShapeIDTableField, and ShapeValue paths are dependent upon one another. Without specifying the ShapeIDPath and ShapeIDTableField, ShapeValuePath has no effect. When ShapeIDPath and ShapeIDTableField are properly set as mentioned in the Data Binding section, the ShapeValuePath has an impact on the map.

Role of DBF file in Data Binding

The .dbf file that is included in the main shape file, is required to work with data binding. The .dbf file contains the information about the shapes in the main shape file. Each record in the .dbf file is associated with the each shape in the main file. Shapes in the main file and records in the .dbf file are organized in the same sequence. Therefore, the Nth shape in the main file is associated with Nth record in the .dbf file. A record of the .dbf file can contain the name of the shape or population data or some other statistical data of a geographic shape.

XML

```
<Grid>
<Grid.DataContext>
<local:ViewModel />
</Grid.DataContext>
<syncfusion:SfMap x:Name="map">
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer x:Name="shapeLayer"
Background="White"
Uri="GettingStarted.ShapeFiles.usa_state.shp"
ItemsSource="{Binding ElectionResults}"
ShapeIDPath="State"
ShapeIDTableField="STATE_NAME">
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>
```


C#

```
// Setting DataContext.
ViewModel viewModel = new ViewModel();
this.DataContext = viewModel;
// SfMap.
SfMap maps = new SfMap();
// ShapeFileLayer.
ShapeFileLayer shapeLayer = new ShapeFileLayer();
shapeLayer.Background = new SolidColorBrush(Colors.White);
shapeLayer.Uri = "GettingStarted.ShapeFiles.usa_state.shp";
shapeLayer.ItemsSource = viewModel.ElectionResults;
shapeLayer.ShapeIDPath = "State";
shapeLayer.ShapeIDTableField = "STATE_NAME";
// Adding ShapeFileLayer into Map.
maps.Layers.Add(shapeLayer);
this.Content = maps;
```

The following code sample demonstrates referring the United States election data as items source.

C#

```
public class ElectionData
{
    private string state;
    public string State
    {
        get { return state; }
        set { state = value; }
    }
    public string candidate;
    public string Candidate
    {
        get { return candidate; }
        set { candidate = value; }
    }
    private double electors;
    public double Electors
    {
        get { return electors; }
        set { electors = value; }
    }
}

public class ViewModel
{
    private ObservableCollection<ElectionData> electionResults;
    public ObservableCollection<ElectionData> ElectionResults
    {
        get { return electionResults; }
        set { electionResults = value; }
    }
    public ViewModel()
    {
        ElectionResults = new ObservableCollection<ElectionData>
        {
            new ElectionData { State = "Alabama", Candidate = "Romney", Electors = 9 },
            new ElectionData { State = "Alaska", Candidate = "Romney", Electors = 3 },
        }
    }
}
```

```

new ElectionData { State = "Arizona", Candidate = "Romney", Electors = 11 },
new ElectionData { State = "Arkansas", Candidate = "Romney", Electors = 6 },
new ElectionData { State = "California", Candidate = "Obama", Electors = 55
},
new ElectionData { State = "Colorado", Candidate = "Obama", Electors = 9 },
new ElectionData { State = "Connecticut", Candidate = "Obama", Electors = 7
},
new ElectionData { State = "Delaware", Candidate = "Obama", Electors = 3 },
new ElectionData { State = "District of Columbia", Candidate = "Obama",
Electors = 3 },
new ElectionData { State = "Florida", Candidate = "Obama", Electors = 29 },
new ElectionData { State = "Georgia", Candidate = "Romney", Electors = 16 },
new ElectionData { State = "Hawaii", Candidate = "Obama", Electors = 4 },
new ElectionData { State = "Idaho", Candidate = "Romney", Electors = 4 },
new ElectionData { State = "Illinois", Candidate = "Obama", Electors = 20 },
new ElectionData { State = "Indiana", Candidate = "Romney", Electors = 11 },
new ElectionData { State = "Iowa", Candidate = "Obama", Electors = 6 },
new ElectionData { State = "Kansas", Candidate = "Romney", Electors = 6 },
new ElectionData { State = "Kentucky", Candidate = "Romney", Electors = 8 },
new ElectionData { State = "Louisiana", Candidate = "Romney", Electors = 8
},
new ElectionData { State = "Maine", Candidate = "Obama", Electors = 4 },
new ElectionData { State = "Maryland", Candidate = "Obama", Electors = 10 },
new ElectionData { State = "Massachusetts", Candidate = "Obama", Electors =
11 },
new ElectionData { State = "Michigan", Candidate = "Obama", Electors = 16 },
new ElectionData { State = "Minnesota", Candidate = "Obama", Electors = 10
},
new ElectionData { State = "Mississippi", Candidate = "Romney", Electors = 6
},
new ElectionData { State = "Missouri", Candidate = "Romney", Electors = 10
},
new ElectionData { State = "Montana", Candidate = "Romney", Electors = 3 },
new ElectionData { State = "Nebraska", Candidate = "Romney", Electors = 5 },
new ElectionData { State = "Nevada", Candidate = "Obama", Electors = 6 },
new ElectionData { State = "New Hampshire", Candidate = "Obama", Electors =
4 },
new ElectionData { State = "New Jersey", Candidate = "Obama", Electors = 14
},
new ElectionData { State = "New Mexico", Candidate = "Obama", Electors = 5
},
new ElectionData { State = "New York", Candidate = "Obama", Electors = 29 },
new ElectionData { State = "North Carolina", Candidate = "Romney", Electors
= 15 },
new ElectionData { State = "North Dakota", Candidate = "Romney", Electors =
3 },
new ElectionData { State = "Ohio", Candidate = "Obama", Electors = 18 },
new ElectionData { State = "Oklahoma", Candidate = "Romney", Electors = 7 },
new ElectionData { State = "Oregon", Candidate = "Obama", Electors = 7 },
new ElectionData { State = "Pennsylvania", Candidate = "Obama", Electors =
20 },
new ElectionData { State = "Rhode Island", Candidate = "Obama", Electors = 4
},
new ElectionData { State = "South Carolina", Candidate = "Romney", Electors
= 9 },
new ElectionData { State = "South Dakota", Candidate = "Romney", Electors =
3 },

```

```

new ElectionData { State = "Tennessee", Candidate = "Romney", Electors = 11
},
new ElectionData { State = "Texas", Candidate = "Romney", Electors = 38 },
new ElectionData { State = "Utah", Candidate = "Romney", Electors = 6 },
new ElectionData { State = "Vermont", Candidate = "Obama", Electors = 3 },
new ElectionData { State = "Virginia", Candidate = "Obama", Electors = 13 },
new ElectionData { State = "Washington", Candidate = "Obama", Electors = 12
},
new ElectionData { State = "West Virginia", Candidate = "Romney", Electors =
5 },
new ElectionData { State = "Wisconsin", Candidate = "Obama", Electors = 10
},
new ElectionData { State = "Wyoming", Candidate = "Romney", Electors = 3 }
};
}
}

```

Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Custom Data Binding in WPF Maps (SfMap)

A map can be bound with custom objects. For custom data binding, a .dbf file is not required. In Data Binding, an object is bound to a shape. In custom data binding, an object is bound to a point based on latitude and longitude values. [CustomDataSource](#) is the API exposed in the custom data source. It is an IEnumerable type API. Each item in the CustomDataSource should have latitude and longitude properties with each mentioned name. Stick to the following rules for custom data binding:

Rule for Specifying the Latitude

Latitude should specify its decimal value along with the first letter of direction. Since latitude is related to the North and South directions on a map, the values should end with N or S. For example: 10.12245N or 23.4566S.

Rule for Specifying the Longitude

The rule for longitude is similar to that of the latitude, apart from the directional value. Since longitude is related to the East and West directions on a map, the values should end with E or W. For example: 34.345E or 56.345W.

To show the custom data on the map, the CustomDataSourceTemplate must be specified.

CustomDataSourceTemplate is a DataTemplate type API used to expose the template for custom data.

XML

```

<Syncfusion:SfMap Name="Weather_Report" >
<Syncfusion:SfMap.Layers>
<Syncfusion:ShapeFileLayer Background="#FFF2E5A2" CustomDataSource="{Binding
Models}"
EnableSelection="True" Uri="MapApp.world1.shp"
ShapeIDPath="NAME" ShapeIDTableField="NAME">
<Syncfusion:ShapeFileLayer.ShapeSettings>
<Syncfusion:ShapeSetting ShapeValuePath="Latitude"
SelectedShapeColor="#FFD96666"
ColorPalette="CustomPalette" ShapeStrokeThickness="0">
<Syncfusion:ShapeSetting.FillSetting>
<Syncfusion:ShapeFillSetting AutoFillColors="True"/>
</Syncfusion:ShapeSetting.FillSetting>

```

```

<Syncfusion:ShapeSetting.CustomColors>
<Syncfusion:MapColorPalette FillBrush="#FFD6C787"/>
</Syncfusion:ShapeSetting.CustomColors>
</Syncfusion:ShapeSetting>
</Syncfusion:ShapeFileLayer.ShapeSettings>
<Syncfusion:ShapeFileLayer.CustomDataSourceTemplate>
<DataTemplate>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="1*" />
<RowDefinition Height="9*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="1*" />
<ColumnDefinition Width="9*" />
</Grid.ColumnDefinitions>
<Ellipse Width="12" Height="12" Grid.Column="0" Grid.Row="0"
Fill="#FF474747"/>
<Grid Grid.Column="1" Grid.Row="1">
<Grid.ColumnDefinitions>
<ColumnDefinition/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<!-- left column-->
<Grid Grid.Column="0" Width="80" Height="80" Background="#FFFF4411"
DataContext="{Binding Data}">
<StackPanel Orientation="Vertical">
<Grid HorizontalAlignment="Center" VerticalAlignment="Center"
Margin="3,3,3,-3">
<StackPanel Orientation="Horizontal">
<TextBlock FontSize="40" FontFamily="Segoe UI"
FontWeight="Thin" Text="{Binding CurrentTemperature}"
HorizontalAlignment="Center" VerticalAlignment="Center"
Foreground="White"/>
<TextBlock Text="°C" FontSize="13" Foreground="White"
Padding="0,4,0,0" HorizontalAlignment="Left"
VerticalAlignment="Top"/>
</StackPanel>
</Grid>
<StackPanel Orientation="Horizontal" Margin="0,0,0,35">
<Grid Margin="19,0,0,5" HorizontalAlignment="Center"
VerticalAlignment="Center">
<StackPanel Orientation="Horizontal">
<TextBlock FontSize="12" FontFamily="Segoe UI"
Text="{Binding AverageHighTemperature}"
HorizontalAlignment="Center" VerticalAlignment="Center"
Foreground="White"/>
<TextBlock Text="°C" FontSize="8" Foreground="White"
HorizontalAlignment="Left" VerticalAlignment="Top"/>
</StackPanel>
</Grid>
<TextBlock Margin="0,0,0,5" FontFamily="Segoe UI" Text="/"
HorizontalAlignment="Center" VerticalAlignment="Center"
Foreground="White" FontSize="16"/>
<Grid Margin="0,0,0,5" HorizontalAlignment="Center"
VerticalAlignment="Center">
<StackPanel Orientation="Horizontal">

```

```

<TextBlock FontSize="12" FontFamily="Segoe UI"
Text="{Binding AverageLowTemperature}"
HorizontalAlignment="Left" VerticalAlignment="Top"
Foreground="White"/>
<TextBlock Text="°C" FontFamily="Segoe UI" FontSize="7"
Foreground="White" HorizontalAlignment="Left"
VerticalAlignment="Top"/>
</StackPanel>
</Grid>
</StackPanel>
</StackPanel>
</Grid>
<Grid Grid.Column="1" Height="80" DataContext="{Binding Data}"
Background="Wheat"
Width="80">
<StackPanel Orientation="Vertical">
<TextBlock Padding="10,0,0,0" TextWrapping="Wrap" Margin="0,6,0,0"
FontFamily="Segoe UI" Text="{Binding City}" FontSize="12"
HorizontalAlignment="Left" VerticalAlignment="Center"
Foreground="Black" FontWeight="Medium"/>
<TextBlock Margin="10,3,0,0" TextWrapping="Wrap" Foreground="Black"
FontFamily="Segoe UI" FontWeight="Thin"
Text="{Binding WeatherDescription}" HorizontalAlignment="Left"
FontSize="16" VerticalAlignment="Center"/>
</StackPanel>
</Grid>
<Grid Grid.Column="1">
</Grid>
</Grid>
</Grid>
</DataTemplate>
</Syncfusion:ShapeFileLayer.CustomDataSourceTemplate>
</Syncfusion:ShapeFileLayer>
</Syncfusion:SfMap.Layers>
</Syncfusion:SfMap >

```

C#

```

public class Weather
{
    public int CurrentTemperature { get; set; }
    public int AverageHighTemperature { get; set; }
    public int AverageLowTemperature { get; set; }
    public string Country { get; set; }
    public string Continent { get; set; }
    public string City { get; set; }
    public string WeatherDescription { get; set; }
    public int Humidity { get; set; }
    public string Longitude { get; set; }
    public string Latitude { get; set; }
}

public class ViewModel
{
    public List<Weather> Models { get; set; }
    public ViewModel()
    {
    }
}

```

```

this.Models = new List<Weather>();
this.Models = ViewModel.GetWeatherData();
}
public static List<Weather> GetWeatherData()
{
List<Weather> weatherCollection = new List<Weather>();
weatherCollection.Add(new Weather() { Humidity = 86, CurrentTemperature =
44, AverageHighTemperature = 63, AverageLowTemperature = 46, City =
"Chicago", Continent = "North America", Country = "United States",
WeatherDescription = "Partly Cloudy", Latitude = "41.8500N", Longitude =
"87.6500W" });
weatherCollection.Add(new Weather() { Humidity = 94, CurrentTemperature =
77, AverageHighTemperature = 89, AverageLowTemperature = 75, City =
"Chennai", Continent = "Asia", Country = "India", WeatherDescription =
"Rainy", Latitude = "12.5810N", Longitude = "76.0740E" });
weatherCollection.Add(new Weather() { Humidity = 60, CurrentTemperature =
70, AverageHighTemperature = 70, AverageLowTemperature = 57, City = "Tokyo",
Continent = "Asia", Country = "Japan", WeatherDescription = "Partly Cloudy",
Latitude = "35.6833N", Longitude = "139.7667E" });
weatherCollection.Add(new Weather() { Humidity = 72, CurrentTemperature =
55, AverageHighTemperature = 47, AverageLowTemperature = 38, City =
"Moscow", Continent = "Asia", Country = "Russia", WeatherDescription =
"Clear", Latitude = "55.7517N", Longitude = "37.6178E" });
weatherCollection.Add(new Weather() { Humidity = 70, CurrentTemperature =
53, AverageHighTemperature = 69, AverageLowTemperature = 54, City = "Cape
Town", Continent = "Africa", Country = "South Africa", WeatherDescription =
"Partly Cloudy", Latitude = "33.9767S", Longitude = "18.4244E" });
weatherCollection.Add(new Weather() { Humidity = 77, CurrentTemperature =
64, AverageHighTemperature = 69, AverageLowTemperature = 56, City =
"Anchorage", Continent = "North America", Country = "United States",
WeatherDescription = "Mostly Cloudy", Latitude = "61.1919N", Longitude =
"149.7621W" });
weatherCollection.Add(new Weather() { Humidity = 55, CurrentTemperature =
91, AverageHighTemperature = 95, AverageLowTemperature = 74, City =
"Panama", Continent = "South America", Country = "Republic Of Panama",
WeatherDescription = "Fair", Latitude = "8.7515N", Longitude = "79.8772W"
});
weatherCollection.Add(new Weather() { Humidity = 88, CurrentTemperature =
61, AverageHighTemperature = 76, AverageLowTemperature = 59, City = "Sao
Paulo", Continent = "South America", Country = "Brazil", WeatherDescription =
"Fair", Latitude = "23.5000S", Longitude = "46.6167W" });
weatherCollection.Add(new Weather() { Humidity = 83, CurrentTemperature =
70, AverageHighTemperature = 85, AverageLowTemperature = 72, City = "Cairo",
Continent = "Africa", Country = "Egypt", WeatherDescription = "Mostly
Cloudy", Latitude = "31.2262E", Longitude = "30.0566N" });
weatherCollection.Add(new Weather() { Humidity = 78, CurrentTemperature =
70, AverageHighTemperature = 85, AverageLowTemperature = 72, City =
"Melbourne", Continent = "Oceania", Country = "Australia",
WeatherDescription = "Cloudy", Latitude = "35.0833S", Longitude =
"142.0667E" });
return weatherCollection;
}
}

```



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

See Also

[How to render CustomDataSource in SfMap?](#)

Layers in WPF Maps (SfMap)

The [WPF Maps](#) control is maintained through [Layers](#), a map can accommodate one or more layers.

The maps control consists the following two layers:

- Imagery layer
- Shape file layer

Imagery layer

The [MapsProvider](#) section explains about the imagery layer.

Shape file layer

Using shape file layer, custom shape files can be rendered and the shapes can be customized.

Shape settings

This section defines how to customize the shapes in a map.

You can customize a shape's fill, stroke, and stroke thickness using the [ShapeFill](#), [ShapeStroke](#), [ShapeStrokeThickness](#) properties.

Refer to the following code sample for customizing shapes.

XML

```
<maps:SfMap>
  <maps:SfMap.Layers >
    <maps:ShapeFileLayer Uri="GettingStarted.ShapeFiles.usa_state.shp" >
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeFill="LightBlue" ShapeStroke="Black"
          ShapeStrokeThickness="1" >
```

```

</maps:ShapeSetting>
</maps:ShapeFileLayer.ShapeSettings>
</maps:ShapeFileLayer>
</maps:SfMap.Layers>
</maps:SfMap>

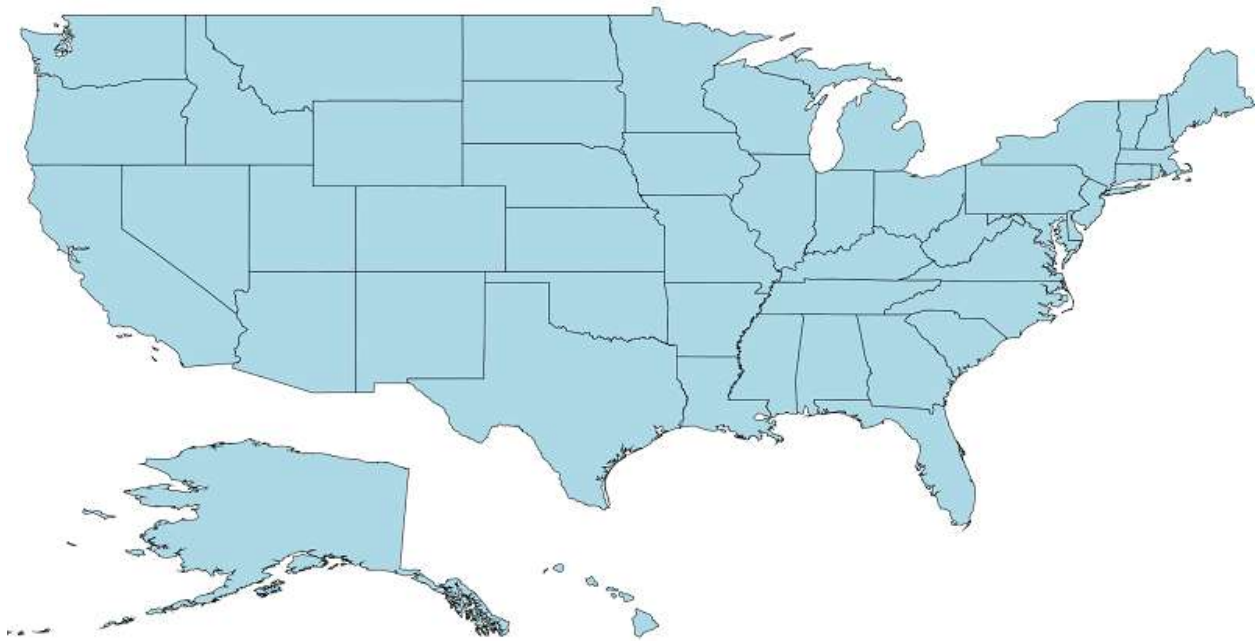
```

C#

```

SfMap maps = new SfMap();
ShapeFileLayer shapeLayer = new ShapeFileLayer();
shapeLayer.Uri = "GettingStarted.ShapeFiles.usa_state.shp";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeFill = new SolidColorBrush(Colors.LightBlue);
shapeSetting.ShapeStroke = new SolidColorBrush(Colors.Black);
shapeSetting.ShapeStrokeThickness = 1;
shapeLayer.ShapeSettings = shapeSetting;
maps.Layers.Add(shapeLayer);
this.Content = maps;

```



Customize selected shapes

The shape selection is enabled when the [EnableSelection](#) property is set to true. To customize the selected shapes alone, use the following properties:

[SelectedShapeColor](#): Sets the color for selected shapes in a map.

[SelectedShapeStroke](#): Sets the border color for selected shapes in a map.

[SelectedShapeStrokeThickness](#): Sets the border thickness for selected shapes in a map.

XML

```

<syncfusion:SfMap >
<syncfusion:SfMap.Layers>

```



```

<syncfusion:ShapeFileLayer EnableSelection="True"
Uri="GettingStarted.ShapeFiles.usa_state.shp">
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting SelectedShapeColor="Green"
SelectedShapeStroke="Black" ShapeStrokeThickness="1" >
</syncfusion:ShapeSetting>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>

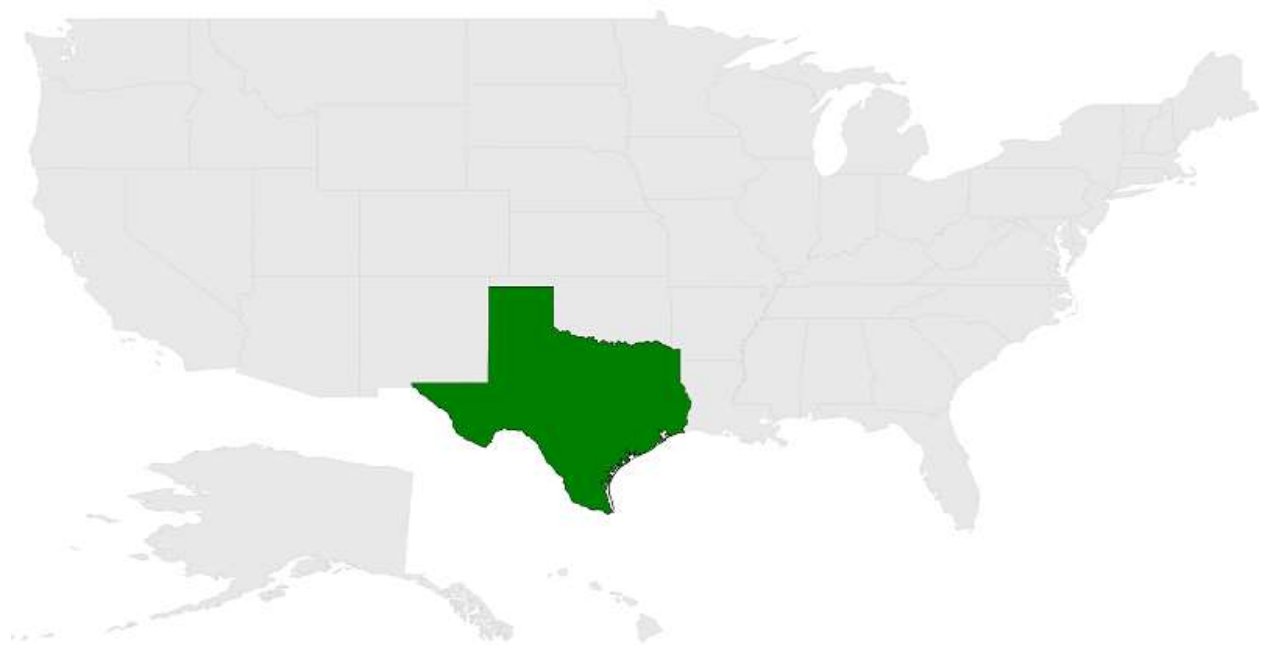
```

C#

```

SfMap maps = new SfMap();
ShapeFileLayer shapeLayer = new ShapeFileLayer();
shapeLayer.EnableSelection = true;
shapeLayer.Uri = "GettingStarted.ShapeFiles.usa_state.shp";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.SelectedShapeColor = new SolidColorBrush(Colors.Green);
shapeSetting.SelectedShapeStroke = new SolidColorBrush(Colors.Black);
shapeSetting.SelectedShapeStrokeThickness = 1;
shapeLayer.ShapeSettings = shapeSetting;
maps.Layers.Add(shapeLayer);

```



Appearance customization

ItemsTemplate is a type of DataTemplate that is used to override the default template for map items.

Data is the property that holds the object for a map item.

XML

```

<syncfusion:SfMap>
<syncfusion:SfMap.Layers>

```

```

<syncfusion:ShapeFileLayer Background="White" ItemsSource="{Binding
Countries}"
ShapeIDPath="Name" ShapeIDTableField="NAME"
Uri="DataMarkers.ShapeFiles.world1.shp">
<syncfusion:ShapeFileLayer.ItemsTemplate>
<DataTemplate>
<Grid Background="#332D2D2D">
<TextBlock Margin="5" Foreground="White" Opacity="1" FontSize="12"
FontWeight="SemiBold" FontFamily="Segoe UI"
Text="{Binding Data.Name}"/>
</Grid>
</DataTemplate>
</syncfusion:ShapeFileLayer.ItemsTemplate>
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeFill="#E5E5E5" ShapeStroke="#C1C1C1"
ShapeStrokeThickness="0.5" ShapeValuePath="Population"
ShapeColorValuePath="Population" >
<syncfusion:ShapeSetting.FillSetting>
<syncfusion:ShapeFillSetting AutoFillColors="False">
<syncfusion:ShapeFillSetting.ColorMappings>
<syncfusion:RangeColorMapping To="1500000000" From="750000000"
Color="#2A91CF"/>
<syncfusion:RangeColorMapping To="750000000" From="1000" Color="#3D9FD8"/>
<syncfusion:RangeColorMapping From="0" To="1000" Color="#C7E9FA"/>
</syncfusion:ShapeFillSetting.ColorMappings>
</syncfusion:ShapeFillSetting>
</syncfusion:ShapeSetting.FillSetting>
</syncfusion:ShapeSetting>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>

```

C#

```

public class Country
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    private double population;
    public double Population
    {
        get { return population; }
        set { population = value; }
    }
}

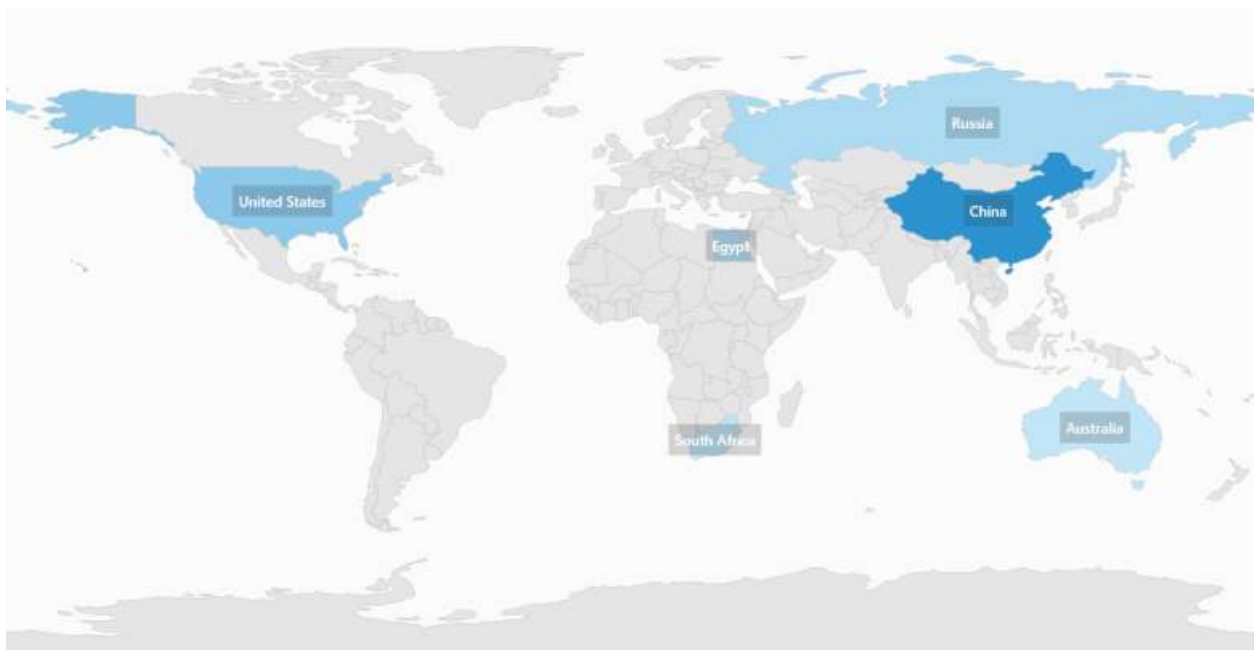
public class ViewModel
{
    private ObservableCollection<Country> countries;
    public ObservableCollection<Country> Countries
    {
        get { return countries; }
    }
}

```

```

set { countries = value; }
}
public ViewModel()
{
    Countries = new ObservableCollection<Country>
    {
        new Country { Name = "Russia", Population = 143228300 },
        new Country { Name = "China", Population = 1347350000 },
        new Country { Name = "Australia", Population = 22789701 },
        new Country { Name = "South Africa", Population = 50586757 },
        new Country { Name = "United States", Population = 314623000 },
        new Country { Name = "Egypt", Population = 82724000 },
    };
}
}

```



ZOrder index for layers

The [BaseMapIndex](#) property allows drill-down from main layer to another layer.

In the ShapeSelected event, the BaseMapIndex property has been used to change the layer when Australia shape is selected.

XML

```

<Grid>
<Grid.RowDefinitions>
<RowDefinition/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Grid.DataContext>
<local:DrilldownViewModel/>
</Grid.DataContext>
<maps:SfMap x:Name="map">
<maps:SfMap.Layers>

```

```

<maps:ShapeFileLayer EnableSelection="True" x:Name="layer1"
Uri="DataMarkers.ShapeFiles.world1.shp"
ItemsSource="{Binding DataSource}" ShapeIDPath="Country"
ShapeIDTableField="NAME"
ShapesSelected="layer1_ShapesSelected">
  <maps:ShapeFileLayer.ShapeSettings>
    <maps:ShapeSetting ShapeColorValuePath="Country" ShapeValuePath="Country">
    </maps:ShapeSetting>
  </maps:ShapeFileLayer.ShapeSettings>
  <maps:ShapeFileLayer.ItemsTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding Data.Country}" IsHitTestVisible="False"/>
    </DataTemplate>
  </maps:ShapeFileLayer.ItemsTemplate>
</maps:ShapeFileLayer>
<maps:ShapeFileLayer x:Name="layer2"
Uri="DataMarkers.ShapeFiles.australia.shp">
  <maps:ShapeFileLayer.ShapeSettings>
    <maps:ShapeSetting ShapeFill="#462A6D"/>
  </maps:ShapeFileLayer.ShapeSettings>
</maps:ShapeFileLayer>
</maps:SfMap.Layers>
</maps:SfMap>
<Label x:Name="label" Grid.Row="1"
HorizontalAlignment="Center" Background="LightGray" Margin="10"
Content="Click on a Australia shape to drill down"/>
</Grid>

```

C#

```

private void layer1_ShapesSelected(object sender, SelectionEventArgs args)
{
    MapShape mapShape = (args.Items as ObservableCollection<MapShape>)[0];
    if (mapShape != null)
    {
        if (mapShape.ShapeValue.ToString() == "Australia")
        {
            this.map.BaseMapIndex = 1;
            label.Visibility = Visibility.Collapsed;
        }
    }
}

public class DrilldownViewModel
{
    public DrilldownViewModel()
    {
        DataSource = new ObservableCollection<DrilldownModel>();
        DataSource.Add(new DrilldownModel("Afghanistan", "Asia"));
        DataSource.Add(new DrilldownModel("Albania", "Europe"));
        DataSource.Add(new DrilldownModel("United Arab Emirates", "Asia"));
        DataSource.Add(new DrilldownModel("Argentina", "South America"));
        DataSource.Add(new DrilldownModel("Armenia", "Asia"));
        DataSource.Add(new DrilldownModel("French Southern and Antarctic Lands",
"Seven seas (open ocean)"));
        DataSource.Add(new DrilldownModel("Australia", "Australia"));
        //..
    }
}

```

```
//...
DataSource.Add(new DrilldownModel("Zambia", "Africa"));
DataSource.Add(new DrilldownModel("Zimbabwe", "Africa"));
}
public ObservableCollection<DrilldownModel> DataSource { get; set; }
}
public class DrilldownModel
{
    public DrilldownModel(string country, string con)
    {
        this.Country = country;
        this.Continent = con;
    }
    public string Continent
    {
        get;
        set;
    }
    public string Country
    {
        get;
        set;
    }
}
```





Events

The [ShapeSelected](#) event will be triggered when a map shape is selected. A corresponding model data is passed as an argument.

The [ShapesUnSelected](#) event will be triggered when a map shape is un selected. A corresponding model data is passed as an argument.

XML

```
<syncfusion:SfMap >
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer EnableSelection="True"
Uri="GettingStarted.ShapeFiles.usa_state.shp"
ShapesSelected="ShapeFileLayer_ShapesSelected"
ShapesUnSelected="ShapeFileLayer_ShapesUnSelected">
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfMap maps = new SfMap();
        ShapeFileLayer shapeLayer = new ShapeFileLayer();
        shapeLayer.ShapesUnSelected += ShapeFileLayer_ShapesUnSelected;
        shapeLayer.ShapesSelected += ShapeFileLayer_ShapesSelected;
    }
}
```

```

shapeLayer.EnableSelection = true;
shapeLayer.Uri = "GettingStarted.ShapeFiles.usa_state.shp";
maps.Layers.Add(shapeLayer);
this.Content = maps;
}
private void ShapeFileLayer_ShapesSelected(object sender, SelectionEventArgs
args)
{
var data = args.Items;
}
private void ShapeFileLayer_ShapesUnSelected(object sender,
SelectionEventArgs args)
{
var data = args.Items;
}
}

```

Note: You can also explore our [WPF Map example](#) to know how to render and configure the map.

see also

[How to customize the markers in maps](#)

[How to drilldown map layers](#)

[How to specify ItemTemplate to shape file layer](#)

[How to render custom data source in SfMap](#)

Shape Types in WPF Maps (SfMaps)

This feature allows you to draw a polygon, polyline, or circle on the map. You can provide input as geo points to draw shapes in two different ways:

- 1.Add shapes using map element collection
- 2.Add shapes using point collection

Add shapes using map element collection

You can provide input as a geo points collection to add multiple shapes in a single layer by using the [MapElements](#) property of shape layer. There are three types of shapes available in map element.

- 1.MapPolygon
- 2.MapPolyline
- 3.MapCircle

MapPolygon

You can add polygon shape on map using [MapPolygon](#) class and provide geo points collection to [Points](#) property. Polygon shape type is used to defines a group of land, water bodies, and other features with a spatial extent.

XML

```

<Grid Margin="20">
<Grid.Resources>
<local:ViewModel x:Key="viewModel"/>
</Grid.Resources>
<maps:SfMap x:Name="Maps" ZoomLevel="4" DataContext="{StaticResource
viewModel}">
<maps:SfMap.Layers>
<maps:ImageryLayer x:Name="layer" Center="30.9709225,-100.2187212">
<maps:ImageryLayer.SubShapeFileLayers>
<maps:SubShapeFileLayer x:Name="subLayer">

```

```

<maps:SubShapeFileLayer.MapElements>
<maps:MapPolygon Points="{Binding PolygonPoints1, Source={StaticResource
viewModel}}" Fill="Blue" Stroke="DarkBlue"
StrokeThickness="4"/>
<maps:MapPolygon Points="{Binding PolygonPoints2, Source={StaticResource
viewModel}}" Fill="Orange" Stroke="Red"
StrokeThickness="4"/>
</maps:SubShapeFileLayer.MapElements>
</maps:SubShapeFileLayer>
</maps:ImageryLayer.SubShapeFileLayers>
</maps:ImageryLayer>
</maps:SfMap.Layers>
</maps:SfMap>
</Grid>

```

C#

```

public partial class MyPage : ContentPage
{
    public MyPage()
    {
        InitializeComponent();
        ViewModel viewModel = new ViewModel();
        SfMap maps = new SfMap();
        maps.ZoomLevel = 4;
        ImageryLayer layer = new ImageryLayer();
        SubShapeFileLayer subLayer = new SubShapeFileLayer();
        layer.Center = new Point(30.9709225, -100.2187212);
        MapPolygon mapPolygon = new MapPolygon();
        mapPolygon.Fill = new SolidColorBrush(Colors.Blue);
        mapPolygon.Stroke = new SolidColorBrush(Colors.DarkBlue);
        mapPolygon.StrokeThickness = 4;
        mapPolygon.Points = viewModel.PolygonPoints1;
        subLayer.MapElements.Add(mapPolygon);
        mapPolygon = new MapPolygon();
        mapPolygon.Fill = new SolidColorBrush(Colors.Orange);
        mapPolygon.Stroke = new SolidColorBrush(Colors.Red);
        mapPolygon.StrokeThickness = 4;
        mapPolygon.Points = viewModel.PolygonPoints2;
        subLayer.MapElements.Add(mapPolygon);
        layer.SubShapeFileLayers.Add(subLayer);
        maps.Layers.Add(layer);
        this.Content = maps;
    }
}

public class ViewModel
{
    public ObservableCollection<Point> PolygonPoints1
    {
        get; set;
    }
    public ObservableCollection<Point> PolygonPoints2
    {
        get; set;
    }
    public ViewModel()

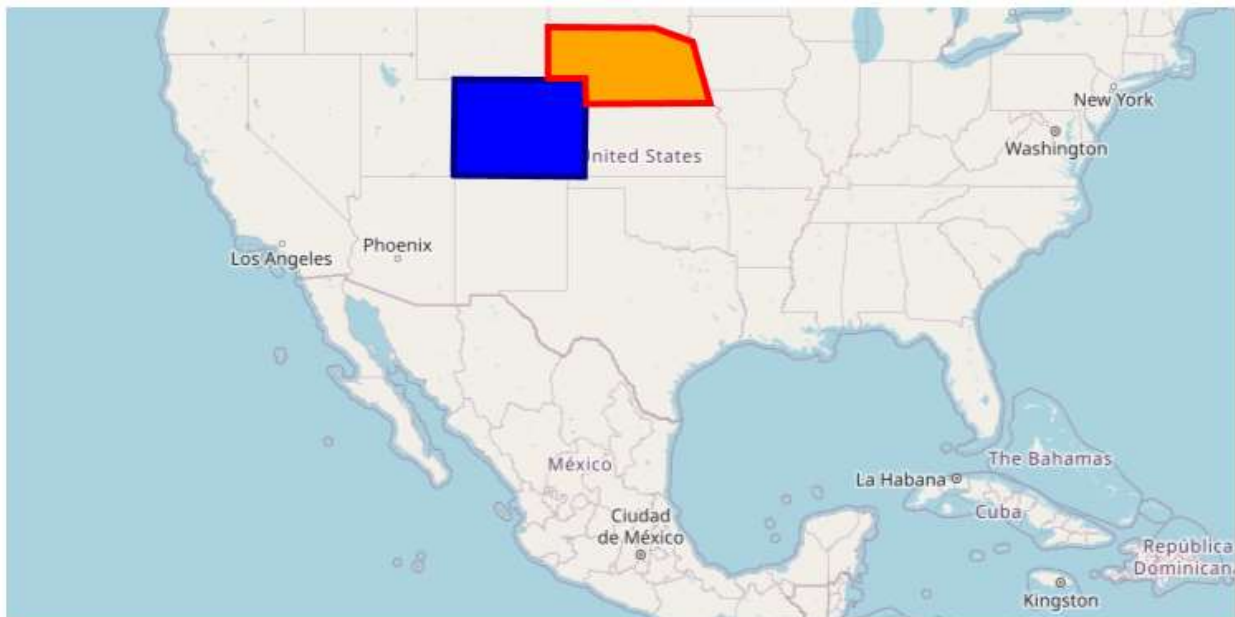
```



```

{
    PolygonPoints1 = new ObservableCollection<Point>()
    {
        new Point(37.042972, -109.085003),
        new Point(40.992567, -109.021030),
        new Point(40.968420, -102.048065),
        new Point(36.991893, -102.144024),
        new Point(37.042972, -109.085003)
    };
    PolygonPoints2 = new ObservableCollection<Point>()
    {
        new Point(41.04621681452063, -104.0625),
        new Point(41.04621681452063, -102.0849609375),
        new Point(40.01078714046552, -102.041015625),
        new Point(40.04443758460856, -95.44921875),
        new Point(42.48830197960227, -96.3720703125),
        new Point(43.03677585761058, -98.4375),
        new Point(43.068887774169625, -104.0625),
        new Point(41.04621681452063, -104.0625),
    };
}

```



MapPolyline

You can add polyline shape on map using [MapPolyline](#) class and provide geo points collection to [Points](#). Polyline are frequently used to define linear features such as roads, rivers, and power lines.

XML

```

<maps:SfMap x:Name="Maps" ZoomLevel="5">
    <maps:SfMap.Layers>
        <maps:ImageryLayer x:Name="layer" Center="49.9709225,10.2187212">
            <maps:ImageryLayer.SubShapeFileLayers>
                <maps:SubShapeFileLayer x:Name="subLayer" >
                    <maps:SubShapeFileLayer.MapElements>

```

```

<maps:MapPolyline Stroke="Black">
  <maps:MapPolyline.Points>
    <Point>
      <Point.X>51.5008</Point.X>
      <Point.Y>-0.1224</Point.Y>
    </Point>
    <Point>
      <Point.X>48.8567</Point.X>
      <Point.Y>2.3508</Point.Y>
    </Point>
    <Point>
      <Point.X>52.5166</Point.X>
      <Point.Y>13.3833</Point.Y>
    </Point>
    <Point>
      <Point.X>48.21327949272514</Point.X>
      <Point.Y>16.388290236693138</Point.Y>
    </Point>
  </maps:MapPolyline.Points>
</maps:MapPolyline>
</maps:SubShapeFileLayer.MapElements>
</maps:SubShapeFileLayer>
</maps:ImageryLayer.SubShapeFileLayers>
</maps:ImageryLayer>
</maps:SfMap.Layers>
</maps:SfMap>

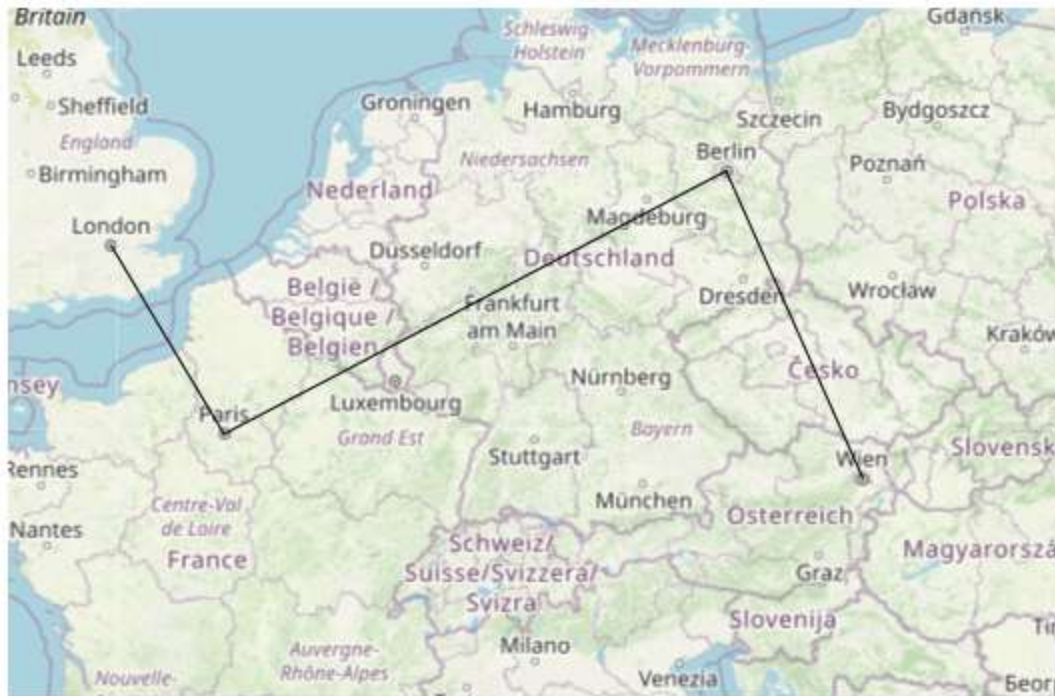
```

C#

```

public partial class MyPage : ContentPage
{
    public MyPage()
    {
        InitializeComponent();
        SfMap maps = new SfMap();
        maps.ZoomLevel = 5;
        ImageryLayer layer = new ImageryLayer();
        layer.Center = new Point(49.9709225, 10.2187212);
        SubShapeFileLayer subLayer = new SubShapeFileLayer();
        MapPolyline mapPolyline = new MapPolyline();
        mapPolyline.Stroke = new SolidColorBrush(Colors.Black);
        mapPolyline.Points = new ObservableCollection<Point>()
        {
            new Point(51.5008, -0.1224),
            new Point(48.8567, 2.3508),
            new Point(52.5166, 13.3833),
            new Point(48.21327949272514, 16.388290236693138)
        };
        subLayer.MapElements.Add(mapPolyline);
        layer.SubShapeFileLayers.Add(subLayer);
        maps.Layers.Add(layer);
        this.Content = maps;
    }
}

```



MapCircle

A circle has a single geo coordinate value and it can be positioned on map using [Center](#) property of [MapCircle](#). The circles are often used to define features such as oil wells, landmarks, and elevations.

- [Center](#) - Holds latitude and longitude values that defines the center of the circle.

XML

```
<maps:SfMap x:Name="Maps" ZoomLevel="4">
  <maps:SfMap.Layers>
    <maps:ImageryLayer x:Name="layer" Center="42.9709225,-90.2187212">
      <maps:ImageryLayer.SubShapeFileLayers>
        <maps:SubShapeFileLayer x:Name="subLayer" >
          <maps:SubShapeFileLayer.MapElements>
            <maps:MapCircle Fill="Blue">
              <maps:MapCircle.Center>
                <Point>
                  <Point.X>43.76140927456403</Point.X>
                  <Point.Y>-79.35451013248883</Point.Y>
                </Point>
              </maps:MapCircle.Center>
            </maps:MapCircle>
            <maps:MapCircle Fill="Blue">
              <maps:MapCircle.Center>
                <Point>
                  <Point.X>40.7324105</Point.X>
                  <Point.Y>-74.4416047</Point.Y>
                </Point>
              </maps:MapCircle.Center>
            </maps:MapCircle>
            <maps:MapCircle Fill="Blue">
```

```

<maps:MapCircle.Center>
<Point>
<Point.X>38.8781708</Point.X>
<Point.Y>-77.1889971</Point.Y>
</Point>
</maps:MapCircle.Center>
</maps:MapCircle>
</maps:SubShapeFileLayer.MapElements>
</maps:SubShapeFileLayer>
</maps:ImageryLayer.SubShapeFileLayers>
</maps:ImageryLayer>
</maps:SfMap.Layers>
</maps:SfMap>

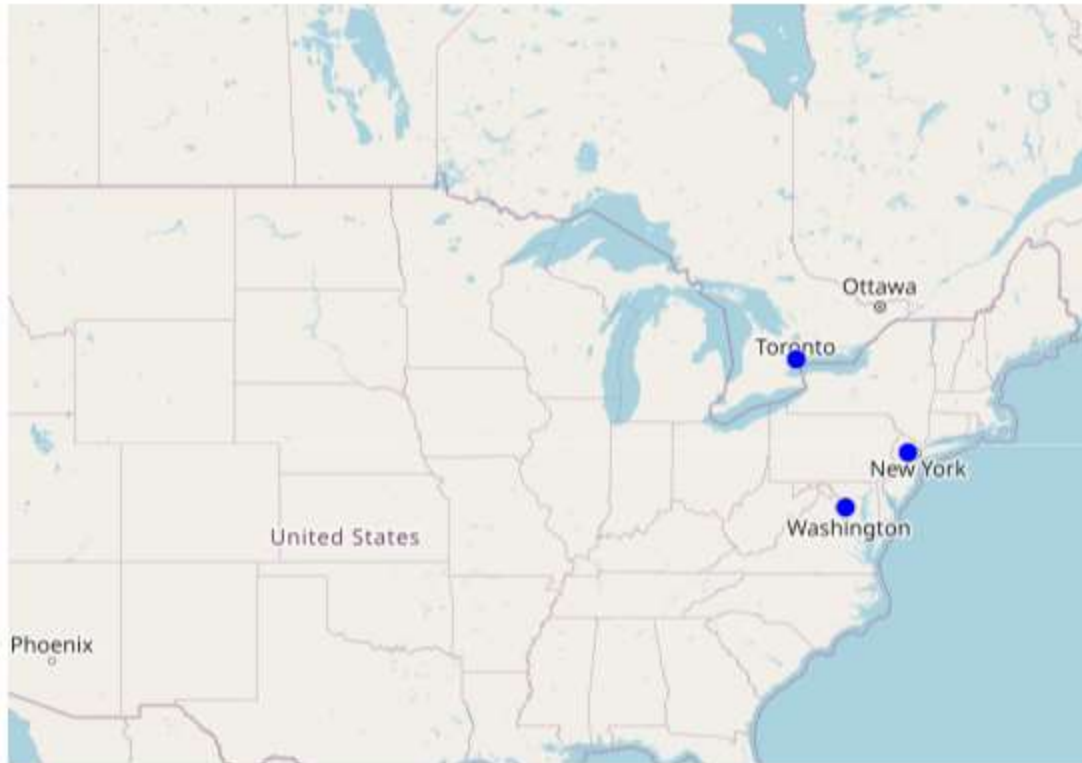
```

C#

```

public partial class MyPage : ContentPage
{
    public MyPage()
    {
        InitializeComponent();
        SfMap maps = new SfMap();
        maps.ZoomLevel = 4;
        ImageryLayer layer = new ImageryLayer();
        layer.Center = new Point(42.9709225, -90.2187212);
        SubShapeFileLayer subLayer = new SubShapeFileLayer();
        MapCircle mapCircle = new MapCircle();
        mapCircle.Fill = new SolidColorBrush(Colors.Blue);
        mapCircle.Center = new Point(43.76140927456403, -79.35451013248883);
        subLayer.MapElements.Add(mapCircle);
        mapCircle = new MapCircle();
        mapCircle.Fill = new SolidColorBrush(Colors.Blue);
        mapCircle.Center = new Point(40.7324105, -74.4416047);
        subLayer.MapElements.Add(mapCircle);
        mapCircle = new MapCircle();
        mapCircle.Fill = new SolidColorBrush(Colors.Blue);
        mapCircle.Center = new Point(38.8781708, -77.1889971);
        subLayer.MapElements.Add(mapCircle);
        layer.SubShapeFileLayers.Add(subLayer);
        maps.Layers.Add(layer);
        this.Content = maps;
    }
}

```



Customization of map elements

The following properties are used to customize [MapPolygon](#), [MapPolyline](#), and [MapCircle](#).

- [Fill](#), [Stroke](#), [StrokeThickness](#) - Used to customize the appearance of the map element's shape UI.
- [Radius](#) - Used to customize radius or range of the map circle. This property is available only for [MapCircle](#).

XML

```
<maps:SfMap x:Name="Maps" ZoomLevel="4">
  <maps:SfMap.Layers>
    <maps:ImageryLayer x:Name="layer" Center="42.9709225,-90.2187212">
      <maps:ImageryLayer.SubShapeFileLayers>
        <maps:SubShapeFileLayer x:Name="subLayer" >
          <maps:SubShapeFileLayer.MapElements>
            <maps:MapCircle Fill="#3eFF0000" Stroke="#73FF0000"
              StrokeThickness="3" Radius="110">
              <maps:MapCircle.Center>
                <Point>
                  <Point.X>43.76140927456403</Point.X>
                  <Point.Y>-79.35451013248883</Point.Y>
                </Point>
              </maps:MapCircle.Center>
            </maps:MapCircle>
          </maps:SubShapeFileLayer.MapElements>
        </maps:SubShapeFileLayer>
      </maps:ImageryLayer.SubShapeFileLayers>
    </maps:ImageryLayer>
  </maps:SfMap.Layers>
```

```
</maps:SfMap>
```

C#

```
public partial class MyPage : ContentPage
{
    public MyPage()
    {
        InitializeComponent();
        SfMap maps = new SfMap();
        maps.ZoomLevel = 4;
        ImageryLayer layer = new ImageryLayer();
        layer.Center = new Point(42.9709225, -90.2187212);
        SubShapeFileLayer subLayer = new SubShapeFileLayer();
        MapCircle mapCircle = new MapCircle();
        mapCircle.Fill = new SolidColorBrush(Color.FromArgb(115, 255, 0, 0));
        mapCircle.StrokeThickness = 3;
        mapCircle.Stroke = new SolidColorBrush(Color.FromArgb(62, 255, 0, 0));
        mapCircle.Radius = 110;
        mapCircle.Center = new Point(43.76140927456403, -79.35451013248883);
        subLayer.MapElements.Add(mapCircle);
        layer.SubShapeFileLayers.Add(subLayer);
        maps.Layers.Add(layer);
        this.Content = maps;
    }
}
```



Add shapes using point collection

You can add more number of shapes on map by using [SubShapeFileLayer](#) and [Points](#) collection property which contains the geo-coordinate points to draw a shape. In shape file layer, there are three type of shapes available and it can be changed by using the [ShapeType](#) property.

1.Polygon 2.Polyline 3.Point icon

You can customize the appearance of the shape UI by setting [ShapeFill](#), [ShapeStroke](#), and [ShapeStrokeThickness](#) properties in [ShapeSetting](#).

Polygon

You can add polygon shape on map by using [Points](#) collection property of shape layer or sub layer and set [ShapeType](#) as [Polygon](#). Polygon shape type defines a group of land, water bodies, and other features with a spatial extent.

XML

```
<maps:SfMap x:Name="maps" ZoomLevel="4">
  <maps:SfMap.Layers>
    <maps:ImageryLayer Center="30.9709225,-100.2187212" >
      <maps:ImageryLayer.SubShapeFileLayers>
        <maps:SubShapeFileLayer x:Name="sublayer1" ShapeType="Polygon"
          Points="{Binding SubLayer1}">
          <maps:SubShapeFileLayer.ShapeSettings>
```



```

<maps:ShapeSetting ShapeFill="Blue" ShapeStroke="DarkBlue"
ShapeStrokeThickness="4" />
</maps:SubShapeFileLayer.ShapeSettings>
</maps:SubShapeFileLayer>
<maps:SubShapeFileLayer x:Name="sublayer2" ShapeType="Polygon"
Points="{Binding SubLayer2}">
<maps:SubShapeFileLayer.ShapeSettings>
<maps:ShapeSetting ShapeFill="Orange" ShapeStroke="Red"
ShapeStrokeThickness="4" />
</maps:SubShapeFileLayer.ShapeSettings>
</maps:SubShapeFileLayer>
</maps:ImageryLayer.SubShapeFileLayers>
</maps:ImageryLayer>
</maps:SfMap.Layers>
</maps:SfMap>

```

C#

```

public partial class MainPage : ContentPage
{
    ViewModel obj = new ViewModel();
    public MainPage()
    {
        InitializeComponent();
        this.DataContext = obj;
        SfMap maps = new SfMap();
        maps.ZoomLevel = 4;
        ImageryLayer layer = new ImageryLayer();
        layer.Center = new Point(30.9709225, -100.2187212);
        SubShapeFileLayer subLayer1 = new SubShapeFileLayer();
        subLayer1.ShapeType = ShapeType.Polygon;
        subLayer1.Points = obj.SubLayer1;
        layer.SubShapeFileLayers.Add(subLayer1);
        SubShapeFileLayer subLayer2 = new SubShapeFileLayer();
        subLayer2.ShapeType = ShapeType.Polygon;
        subLayer2.Points = obj.SubLayer2;
        layer.SubShapeFileLayers.Add(subLayer2);
        ShapeSetting subLayerSetting1 = new ShapeSetting();
        subLayerSetting1.ShapeStrokeThickness = 4;
        subLayerSetting1.ShapeFill = new SolidColorBrush(Colors.Blue);
        subLayerSetting1.ShapeStroke = new SolidColorBrush(Colors.DarkBlue);
        subLayer1.ShapeSettings = subLayerSetting1;
        ShapeSetting subLayerSetting2 = new ShapeSetting();
        subLayerSetting2.ShapeStrokeThickness = 4;
        subLayerSetting2.ShapeFill = new SolidColorBrush(Colors.Orange);
        subLayerSetting2.ShapeStroke = new SolidColorBrush(Colors.Red);
        subLayer2.ShapeSettings = subLayerSetting2;
        subLayer1.Points = obj.SubLayer1;
        subLayer2.Points = obj.SubLayer2;
        maps.Layers.Add(layer);
        this.Content = maps;
    }
}

public class ViewModel
{
    public ObservableCollection<Point> SubLayer1

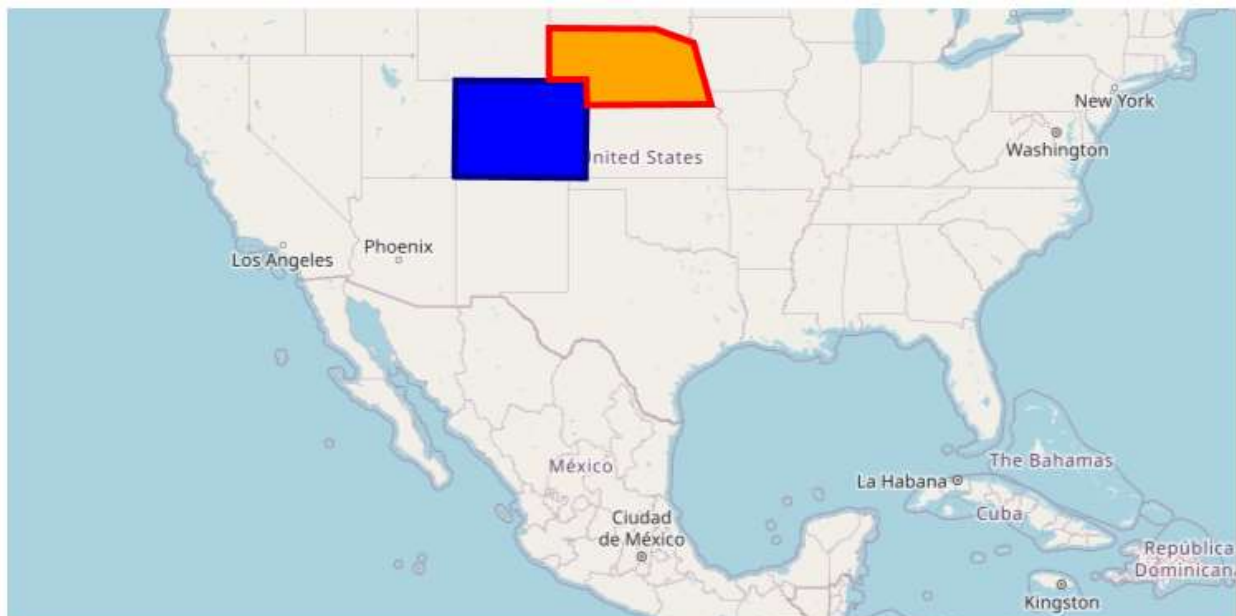
```



```

{
    get; set;
}
public ObservableCollection<Point> SubLayer2
{
    get; set;
}
public ViewModel()
{
    SubLayer1 = new ObservableCollection<Point>()
    {
        new Point(37.042972, -109.085003),
        new Point(40.992567, -109.021030),
        new Point(40.968420, -102.048065),
        new Point(36.991893, -102.144024),
        new Point(37.042972, -109.085003)
    };
    SubLayer2 = new ObservableCollection<Point>()
    {
        new Point(41.04621681452063, -104.0625),
        new Point(41.04621681452063, -102.0849609375),
        new Point(40.01078714046552, -102.041015625),
        new Point(40.04443758460856, -95.44921875),
        new Point(42.48830197960227, -96.3720703125),
        new Point(43.03677585761058, -98.4375),
        new Point(43.068887774169625, -104.0625),
        new Point(41.04621681452063, -104.0625),
    };
}
}

```



Polyline

You can add polyline shape on map by using Points collection property of shape layer or sub layer and set [ShapeType](#) as `Polyline`. Polyline are frequently used to define linear features such as roads, rivers, and power lines.

XML

```
<maps:SfMap x:Name="Maps" ZoomLevel="5">
  <maps:SfMap.Layers>
    <maps:ImageryLayer x:Name="layer" Center="49.9709225,10.2187212">
      <maps:ImageryLayer.SubShapeFileLayers>
        <maps:SubShapeFileLayer x:Name="subLayer" ShapeType="Polyline">
          <maps:SubShapeFileLayer.Points>
            <Point>
              <Point.X>51.5008</Point.X>
              <Point.Y>-0.1224</Point.Y>
            </Point>
            <Point>
              <Point.X>48.8567</Point.X>
              <Point.Y>2.3508</Point.Y>
            </Point>
            <Point>
              <Point.X>52.5166</Point.X>
              <Point.Y>13.3833</Point.Y>
            </Point>
            <Point>
              <Point.X>48.21327949272514</Point.X>
              <Point.Y>16.388290236693138</Point.Y>
            </Point>
          </maps:SubShapeFileLayer.Points>
          <maps:SubShapeFileLayer.ShapeSettings>
            <maps:ShapeSetting x:Name="settings" ShapeFill="Black"
              ShapeStrokeThickness="1"/>
          </maps:SubShapeFileLayer.ShapeSettings>
        </maps:SubShapeFileLayer>
      </maps:ImageryLayer.SubShapeFileLayers>
    </maps:ImageryLayer>
  </maps:SfMap.Layers>
</maps:SfMap>
```

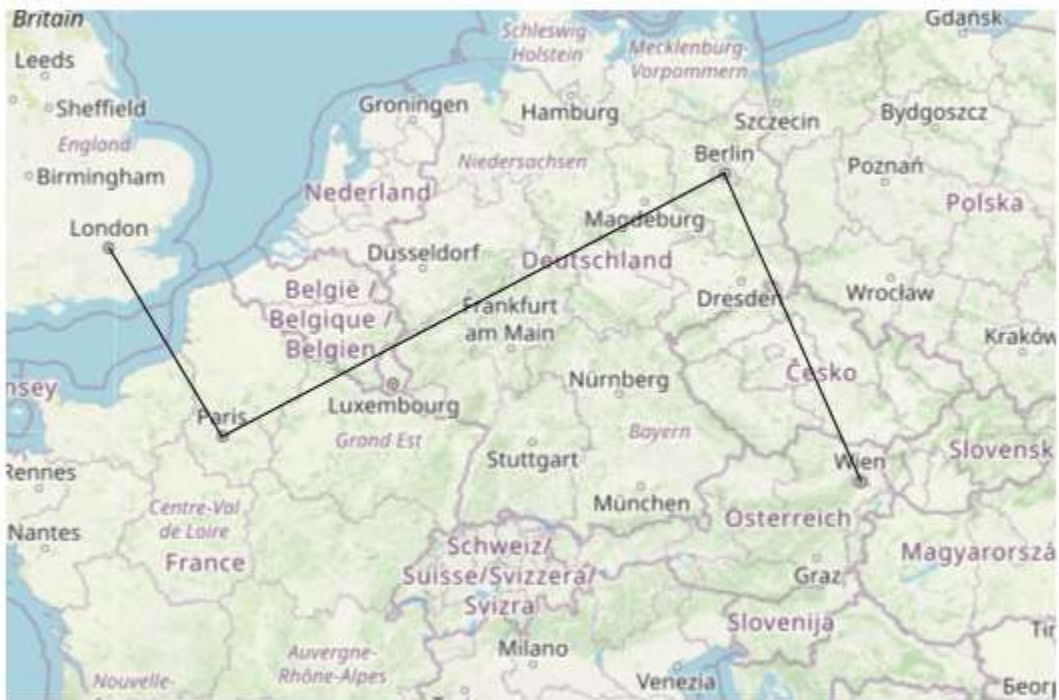
C#

```
public partial class MyPage : ContentPage
{
  public MyPage()
  {
    InitializeComponent();
    SfMap maps = new SfMap();
    maps.ZoomLevel = 5;
    ImageryLayer layer = new ImageryLayer();
    layer.Center = new Point(49.9709225, 10.2187212);
    SubShapeFileLayer subLayer = new SubShapeFileLayer();
    subLayer.Points = new ObservableCollection<Point>()
    {
      new Point(51.5008, -0.1224),
      new Point(48.8567, 2.3508),
    }
  }
}
```

```

new Point(52.5166, 13.3833),
new Point(48.21327949272514, 16.388290236693138)
};
subLayer.ShapeType = ShapeType.Polyline;
ShapeSetting subLayerSetting = new ShapeSetting();
subLayerSetting.ShapeStrokeThickness = 1;
subLayerSetting.ShapeFill = new SolidColorBrush(Colors.Black);
subLayer.ShapeSettings = subLayerSetting;
layer.SubShapeFileLayers.Add(subLayer);
maps.Layers.Add(layer);
this.Content = maps;
}
}

```



Point icon

A point icon has a single geo coordinate value and you can add multi point icon shapes on map by using `Points` collection property of shape layer or sub layer and set `ShapeType` as `PointIcon`. The point icons are often used to define features such as oil wells, landmarks, and elevations.

The size, shape, and alignment of the map points can be customized using the [MapPointIconSize](#), [MapPointIcon](#), [MapPointHorizontalAlignment](#), and [MapPointVerticalAlignment](#) properties of the shape file layer.

XML

```

<maps:SfMap x:Name="Maps" ZoomLevel="4">
  <maps:SfMap.Layers>
    <maps:ImageryLayer x:Name="layer" Center="42.9709225,-90.2187212">
      <maps:ImageryLayer.SubShapeFileLayers>
        <maps:SubShapeFileLayer x:Name="subLayer" MapPointIconSize="10"
          ShapeType="PointIcon">

```

```

<maps:SubShapeFileLayer.Points>
  <Point>
    <Point.X>43.96140927456403</Point.X>
    <Point.Y>-79.85451013248883</Point.Y>
  </Point>
  <Point>
    <Point.X>41.1324105</Point.X>
    <Point.Y>-74.4416047</Point.Y>
  </Point>
  <Point>
    <Point.X>39.2781708</Point.X>
    <Point.Y>-77.5889971</Point.Y>
  </Point>
</maps:SubShapeFileLayer.Points>
<maps:ShapeFileLayer.ShapeSettings>
  <maps:ShapeSetting ShapeFill="Blue"/>
</maps:ShapeFileLayer.ShapeSettings>
</maps:SubShapeFileLayer>
</maps:ImageryLayer.SubShapeFileLayers>
</maps:ImageryLayer>
</maps:SfMap.Layers>
</maps:SfMap>

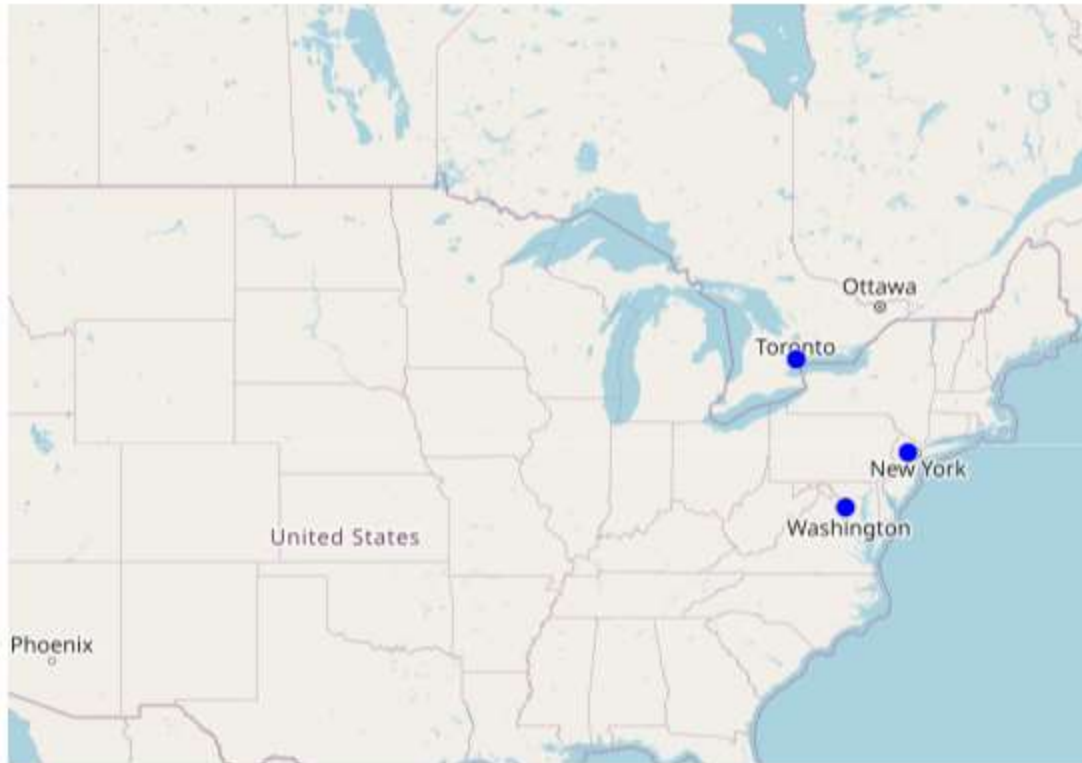
```

C#

```

public partial class MyPage : ContentPage
{
    public MyPage()
    {
        InitializeComponent();
        SfMap maps = new SfMap();
        maps.ZoomLevel = 4;
        ImageryLayer layer = new ImageryLayer();
        layer.Center = new Point(42.9709225, -90.2187212);
        SubShapeFileLayer subLayer = new SubShapeFileLayer();
        subLayer.Points.Add(new Point(43.96140927456403, -79.85451013248883));
        subLayer.Points.Add(new Point(41.1324105, -74.4416047));
        subLayer.Points.Add(new Point(39.2781708, -77.5889971));
        subLayer.MapPointIconSize = 10;
        subLayer.ShapeType = ShapeType.PointIcon;
        subLayer.MapPointIconSize = 10;
        ShapeSetting subLayerSetting = new ShapeSetting();
        subLayerSetting.ShapeFill = new SolidColorBrush(Colors.Blue);
        subLayer.ShapeSettings = subLayerSetting;
        layer.SubShapeFileLayers.Add(subLayer);
        maps.Layers.Add(layer);
        this.Content = maps;
    }
}

```



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Bubbles in WPF Maps (SfMap)

Bubbles in the [WPF Maps](#) control represent the under-bound data values of the map. Bubbles are scattered throughout map shapes that contain bound values.

Bubbles are included when data binding is set as mentioned above and the `BubbleMarkerSetting` is set.

The following properties are available in [BubbleMarkerSetting](#):

Property	Type	Description
AutoFillColor	Boolean (true / false)	Gets or sets whether the colors should be automatically filled.
MaxSize	Double	Get or sets the maximum height and width of the bubble.
MinSize	Double	Gets or sets the minimum height and width of the bubble.
StrokeThickness	Double	Get or sets the border thickness of the bubbles.

ValuePath	String	Gets or sets the name of the under-bound property in ItemsSource.
ColorValuePath	String	Gets or sets colors to bubble shape.
ColorMapping	ObservableCollection<RangeColorMapping>	Gets or sets the tree map colors.
Fill	Brush	Gets or sets the fill brush of the bubble when auto fill color is set to true.
Stroke	Brush	Gets or sets the border color of the bubble.
Opacity	double	Gets or sets the opacity of the bubble in the map.
SizeRatio	double	Gets the Size ratio for the bubbles based on the MinSize and MaxSize.

Adding Bubbles to a Map

To add bubbles to a map, the bubble marker setting has to be added to the shape file layer. Set the [AutoFillColor](#) as true and set the [Fill](#) property. Create the Model and ViewModel as illustrated in the [Data Binding](#) topic and add the following code.

Also set the [MaxSize](#), [MinSize](#), and [ValuePath](#) properties as illustrated in the following code example.

XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer EnableSelection="False" ItemsSource="{Binding Countries}"
      ShapeIDPath="Name" ShapeIDTableField="NAME"
      Uri="DataMarkers.ShapeFiles.world1.shp">
      <syncfusion:ShapeFileLayer.BubbleMarkerSetting>
        <syncfusion:BubbleMarkerSetting AutoFillColor="False" MaxSize="100"
          MinSize="50"
          Fill="#FF26E8FB" StrokeThickness="0"
          ValuePath="Population" >
        </syncfusion:BubbleMarkerSetting>
      </syncfusion:ShapeFileLayer.BubbleMarkerSetting>
      <syncfusion:ShapeFileLayer.ShapeSettings>
        <syncfusion:ShapeSetting ShapeStroke="#C1C1C1" ShapeStrokeThickness="0.5"
          ShapeValuePath="Population" ShapeFill="#E5E5E5"/>
      </syncfusion:ShapeFileLayer.ShapeSettings>
      <syncfusion:ShapeFileLayer.ItemsTemplate>
        <DataTemplate>
          <Border Background="Transparent">
            <TextBlock FontFamily="Segoe UI" FontSize="12" Foreground="White"
```

```

Text="{Binding Data.Population}"/>
</Border>
</DataTemplate>
</syncfusion:ShapeFileLayer.ItemsTemplate>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>

```

C#

```

public class Country
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    private double population;
    public double Population
    {
        get { return population; }
        set { population = value; }
    }
}

public class ViewModel
{
    private ObservableCollection<Country> countries;
    public ObservableCollection<Country> Countries
    {
        get { return countries; }
        set { countries = value; }
    }
    public ViewModel()
    {
        Countries = new ObservableCollection<Country>
        {
            new Country { Name = "Russia", Population = 143228300},
            new Country { Name = "China", Population = 1347350000 },
            new Country { Name = "Australia", Population = 22789701 },
            new Country { Name = "South Africa", Population = 50586757},
            new Country { Name = "United States", Population = 314623000 },
            new Country { Name = "Egypt", Population = 82724000},
        };
    }
}

```



Customizing Bubble Symbol

The shape of the bubble symbol can be modified by using built-in symbols like circle, rectangle, diamond, triangle, trapezoid, star, pentagon, and pushpin that are available in the [BubbleType](#) enum property. Also, bubbles can be customized by setting the [CustomTemplate](#) of the BubbleMarkerSetting.

Property	Type	Description
BubbleType	BubbleType (enum)	Gets or sets the type of bubble symbol needed to be used in maps.
CustomTemplate	DataTemplate	Gets or sets the template to customize the bubble.

XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer x:Name="shapeFileLayer"
      ItemsSource="{Binding Countries}"
      ShapeIDPath="Name"
      ShapeIDTableField="NAME"
      Uri="DataMarkers.ShapeFiles.world1.shp">
      <syncfusion:ShapeFileLayer.BubbleMarkerSetting>
        <syncfusion:BubbleMarkerSetting AutoFillColor="False" MaxSize="100"
          MinSize="50"
          ColorValuePath="Population"
          ValuePath="Population" BubbleType="Star">
          <syncfusion:BubbleMarkerSetting.ColorMappings>
            <syncfusion:RangeColorMapping Color="#7F20BCEE" To="1347350000"
              From="314623001"/>
            <syncfusion:RangeColorMapping Color="#7FA7CE38" To="314623001"
              From="143228301"/>
            <syncfusion:RangeColorMapping Color="#7FF1B21A" To="143228301"
              From="82724090"/>
          </syncfusion:BubbleMarkerSetting.ColorMappings>
        </syncfusion:BubbleMarkerSetting>
      </syncfusion:ShapeFileLayer>
    </syncfusion:SfMap.Layers>
  </syncfusion:SfMap>
```



```

<syncfusion:RangeColorMapping Color="#7F1DA249" To="50586757"
From="22789702"/>
<syncfusion:RangeColorMapping Color="#7FEB737C" To="22789702" From="0"/>
<syncfusion:RangeColorMapping Color="#7FED2D95" To="82724090"
From="50586757"/>
</syncfusion:BubbleMarkerSetting.ColorMappings>
</syncfusion:BubbleMarkerSetting>
</syncfusion:ShapeFileLayer.BubbleMarkerSetting>
<syncfusion:ShapeFileLayer.ItemsTemplate>
<DataTemplate>
<Border Background="Transparent">
<TextBlock FontFamily="Segoe UI" FontSize="12" Foreground="#FF333333"
Text="{Binding Data.Population}"/>
</Border>
</DataTemplate>
</syncfusion:ShapeFileLayer.ItemsTemplate>
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeStroke="#C1C1C1" ShapeStrokeThickness="0.5"
ShapeValuePath="Population" ShapeFill="#E5E5E5"/>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>

```

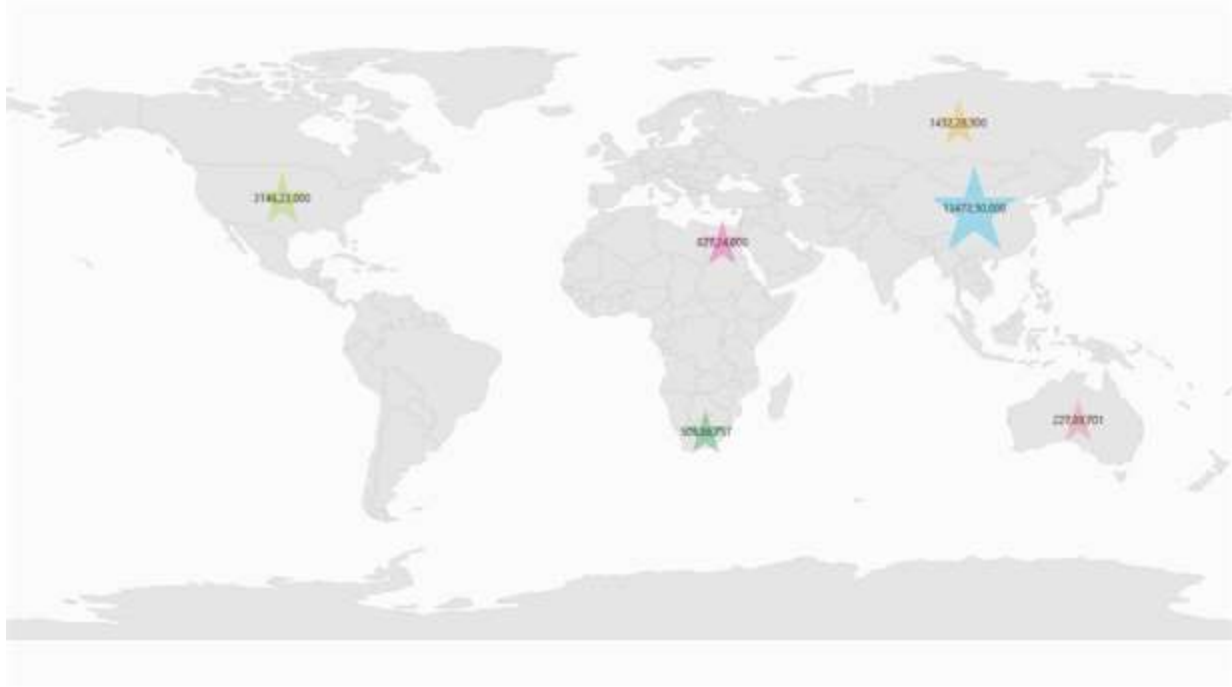
C#

```

ViewModel viewModel = new ViewModel();
SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.world1.shp";
shapeFileLayer.ShapeIDPath = "Name";
shapeFileLayer.ShapeIDTableField = "NAME";
shapeFileLayer.ItemsSource = viewModel.Countries;
map.Layers.Add(shapeFileLayer);
BubbleMarkerSetting bubbleMarkerSetting = new BubbleMarkerSetting();
bubbleMarkerSetting.AutoFillColor = false;
bubbleMarkerSetting.MaxSize = 100;
bubbleMarkerSetting.MinSize = 50;
bubbleMarkerSetting.ColorValuePath = "Population";
bubbleMarkerSetting.ValuePath = "Population";
bubbleMarkerSetting.BubbleType = BubbleType.Star;
shapeFileLayer.BubbleMarkerSetting = bubbleMarkerSetting;
RangeColorMapping rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 314623001;
rangeColorMapping.To = 1347350000;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7F20BCEE");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 143228301;
rangeColorMapping.To = 314623001;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7FA7CE38");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 82724090;

```

```
rangeColorMapping.To = 143228301;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7FF1B21A");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 22789702;
rangeColorMapping.To = 50586757;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7F1DA249");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 0;
rangeColorMapping.To = 22789702;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7FEB737C");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 50586757;
rangeColorMapping.To = 82724090;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7FED2D95");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
shapeFileLayer.ItemsTemplate = grid.Resources["itemTemplate"] as
DataTemplate;
ShapeSetting setting = new ShapeSetting();
setting.ShapeStroke= (SolidColorBrush)(new
BrushConverter().ConvertFrom("#C1C1C1"));
setting.ShapeStrokeThickness = 0.5;
setting.ShapeValuePath = "Population";
setting.ShapeFill = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#E5E5E5"));
shapeFileLayer.ShapeSettings = setting;
grid.Children.Add(map);
```



Range Color Mapping

Range color mapping is one of the features used to differentiate the bubble fill, based on its under-bound value and color ranges. It contains the following properties:

Property	Type	Description
From & To	Double	Gets or sets the value range of the bubble.
Color	Color	Gets or sets the color values for a given range.

The fill color of a particular bubble fill can be determined by its under-bound value and the color range. For example, consider the following color ranges:

XML

```
<syncfusion:BubbleMarkerSetting AutoFillColor="False" MaxSize="100"
MinSize="50"
ColorValuePath="Population"
ValuePath="Population" >
  <syncfusion:BubbleMarkerSetting.ColorMappings>
    <syncfusion:RangeColorMapping Color="#7F20BCEE" To="1347350000"
From="314623001"/>
    <syncfusion:RangeColorMapping Color="#7FA7CE38" To="314623001"
From="143228301"/>
    <syncfusion:RangeColorMapping Color="#7FF1B21A" To="143228301"
From="82724090"/>
    <syncfusion:RangeColorMapping Color="#7F1DA249" To="50586757"
From="22789702"/>
    <syncfusion:RangeColorMapping Color="#7FEB737C" To="22789702" From="0"/>
  </syncfusion:BubbleMarkerSetting.ColorMappings>
</syncfusion:BubbleMarkerSetting>
```

C#

```

RangeColorMapping rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 314623001;
rangeColorMapping.To = 1347350000;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7F20BCEE");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 143228301;
rangeColorMapping.To = 314623001;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7FA7CE38");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 82724090;
rangeColorMapping.To = 143228301;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7FF1B21A");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 22789702;
rangeColorMapping.To = 50586757;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7F1DA249");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 0;
rangeColorMapping.To = 22789702;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7FEB737C");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);
rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 50586757;
rangeColorMapping.To = 82724090;
rangeColorMapping.Color =
(Color)ColorConverter.ConvertFromString("#7FED2D95");
bubbleMarkerSetting.ColorMappings.Add(rangeColorMapping);

```

When the under-bound object value is 22789702, the fill color of the corresponding bubble is set to #7FEB737C. As mentioned earlier, the under-bound value of the bubble is set using the [ValuePath](#) in the BubbleMarkerSetting.

When the under-bound value is under any of the given sorted range or above the sorted range, then the fill is set as **Black**.

[AutoFillColor](#) must be set as **false** to enable range color mapping.



Note: You can also explore our [WPF Map example](#) to know how to render and configure the map.

Shapes Color Customization in WPF Maps (SfMap)

[WPF Maps](#) highly support the customization of the shape's color. The shape's color can be customized by using the following methods:

1. Using the ShapeFill, ShapeStroke and ShapeStrokeThickness properties.
2. Using tree map-like support.
3. Using the color palette.

The important property that makes an impact on shape colors is AutoFillColor. This is a Boolean type property. This property is available in the FillSetting. The use of this property is explained in the following sections.

Properties

The below mentioned properties are available in the ShapesSetting property of the ShapeFileLayer. ShapeSetting defines the basic customization settings of shapes in the map.

ShapeFill

[ShapeFill](#) is a Brush type property that sets the fill color of the shapes in the map.

ShapeStroke

[ShapeStroke](#) is also a brush type property that sets the border color of the shape in the map.

ShapeStrokeThickness

[ShapeStrokeThickness](#) is a double type property that sets the border thickness of the shape in the map.

These setting works only when `AutoFillColor` is set as false.

XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer Uri="DataMarkers.ShapeFiles.usa_state.shp">
```

```

<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeFill="Gray" ShapeStroke="Black"
ShapeStrokeThickness="1">
<syncfusion:ShapeSetting.FillSetting>
<syncfusion:ShapeFillSetting AutoFillColors="False"/>
</syncfusion:ShapeSetting.FillSetting>
</syncfusion:ShapeSetting>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap >

```

C#

```

SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.usa_state.shp";
map.Layers.Add(shapeFileLayer);
ShapeSetting setting = new ShapeSetting();
setting.ShapeStroke = new SolidColorBrush(Colors.Black);
setting.ShapeStrokeThickness = 1;
setting.ShapeFill = new SolidColorBrush(Colors.Gray);
ShapeFillSetting shapeFillSetting = new ShapeFillSetting();
shapeFillSetting.AutoFillColors = false;
setting.FillSetting = shapeFillSetting;
shapeFileLayer.ShapeSettings = setting;
this.Content = map;

```



Tree map-like support

ShapeFill is set based on the under-bound values of the shape. This provides a tree map-like impact on the map UI. The RangeColorMapping property provides a tree map-like fill for the shapes.

Range Color Mapping

Range color mapping is one of the features used to differentiate the shape's fill based on its under-bound value and color ranges. Range color mapping contains the following properties:

Property Table

Property	Type	Description
From	Double	Gets or sets the From value of ColorValuePath
To	Double	Gets or sets the To value of ColorValuePath
Color	Color	Gets or sets the color values for the given range based on the From and To .

The fill color of a particular bubble fill is determined by its under-bound value and color range. To provide a Tree Map like impact on the map, the data binding should work properly. For example, consider the following color ranges.

XML

```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer ItemsSource="{Binding Countries}"
Uri="DataMarkers.ShapeFiles.world1.shp"
ShapeIDPath="Name" ShapeIDTableField="NAME">
<syncfusion:ShapeFileLayer.ShapeSettings >
<syncfusion:ShapeSetting ShapeFill="#E5E5E5" ShapeStroke="Black"
ShapeStrokeThickness="1" ShapeColorValuePath="Population"
ShapeValuePath="Population">
<syncfusion:ShapeSetting.FillSetting>
<syncfusion:ShapeFillSetting AutoFillColors="False">
<syncfusion:ShapeFillSetting.ColorMappings>
<syncfusion:RangeColorMapping To="1500000000" From="750000000"
Color="#2A91CF"/>
<syncfusion:RangeColorMapping To="750000000" From="0" Color="#3D9FD8"/>
<syncfusion:RangeColorMapping To="0" From="0" Color="#C7E9FA"/>
</syncfusion:ShapeFillSetting.ColorMappings>
</syncfusion:ShapeFillSetting>
</syncfusion:ShapeSetting.FillSetting>
</syncfusion:ShapeSetting>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap >
```

C#

```
public class Country
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
```

```

}
private double population;
public double Population
{
    get { return population; }
    set { population = value; }
}
}
public class ViewModel
{
    private ObservableCollection<Country> countries;
    public ObservableCollection<Country> Countries
    {
        get { return countries; }
        set { countries = value; }
    }
    public ViewModel()
    {
        Countries = new ObservableCollection<Country>
        {
            new Country { Name = "Russia", Population = 143228300 },
            new Country { Name = "China", Population = 1347350000 },
            new Country { Name = "Australia", Population = 22789701 },
            new Country { Name = "South Africa", Population = 50586757 },
            new Country { Name = "United States", Population = 314623000 },
            new Country { Name = "Egypt", Population = 82724000 },
        };
    }
}

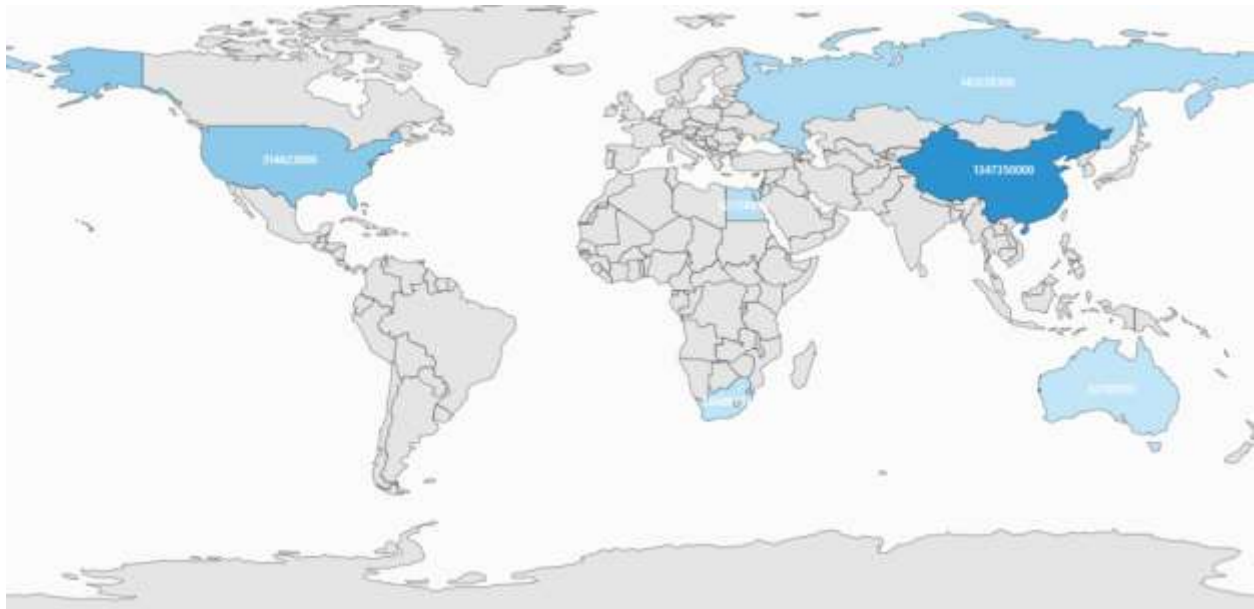
```

When under-bound object value is 750000000, then the fill color of the corresponding bubble is set to #3D9FD8. As mentioned earlier, the under-bound value of the bubble is set through the ShapeValuePath in the ShapeSetting.

When the under-bound value is under any of the given sorted range or above the sorted range, then the fill is set to Black.

AutoFillColor must be set to false to enable the range color mapping.

Note: The shape's under-bound object value must have numeric property and should be mentioned in ShapeValuePath to work on this. The color between the given ranges is applied only to the shapes that have a proper under-bound values. The color for other shapes is the ShapeFill's color.



ColorPalette

[ColorPalette](#) is a set of colors that are applied on the shapes. Map contains two built-in color palettes. They are:

1. Metro
2. CoolBlue

ColorPalette has to be set in the ShapeSetting's ColorPalette property. ColorPalette is the enum property that accepts Metro, CoolBlue and CustomPalette.

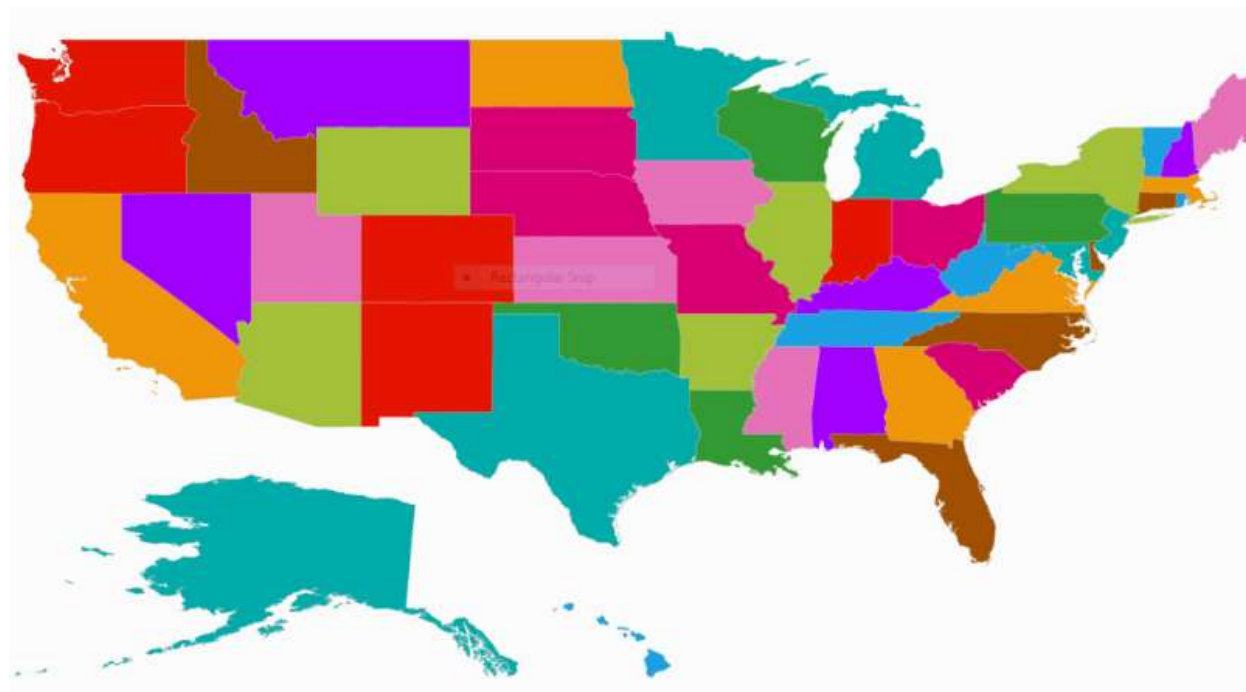
XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer Uri="DataMarkers.ShapeFiles.usa_state.shp">
      <syncfusion:ShapeFileLayer.ShapeSettings>
        <syncfusion:ShapeSetting ColorPalette="Metro">
          <syncfusion:ShapeSetting.FillSetting>
            <syncfusion:ShapeFillSetting AutoFillColors="True"/>
          </syncfusion:ShapeSetting.FillSetting>
        </syncfusion:ShapeSetting>
      </syncfusion:ShapeFileLayer.ShapeSettings>
    </syncfusion:ShapeFileLayer>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.usa_state.shp";
map.Layers.Add(shapeFileLayer);
ShapeSetting setting = new ShapeSetting();
setting.ColorPalette = ColorPalettes.Metro;
```

```
ShapeFillSetting shapeFillSetting = new ShapeFillSetting();
shapeFillSetting.AutoFillColors = true;
setting.FillSetting = shapeFillSetting;
shapeFileLayer.ShapeSettings = setting;
this.Content = map;
```



CustomColorPalette

Besides the built-in the color palettes, the custom colors can be defined for the color palette. The custom colors are defined in the `CustomColors` in `ShapeSetting`. `CustomColors` is the collection property that accepts the `MapColorPalette`. To apply the custom colors, `ColorPalette` must be set to `CustomPalette` and `CustomColors` should be defined.

`MapColorPalette` contains a property named `FillBrush`. This property sets the fill color of the shape when custom palette is set.

XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer Uri="DataMarkers.ShapeFiles.usa_state.shp">
      <syncfusion:ShapeFileLayer.ShapeSettings>
        <syncfusion:ShapeSetting ColorPalette="CustomPalette">
          <syncfusion:ShapeSetting.CustomColors>
            <syncfusion:MapColorPalette FillBrush="Gray"/>
            <syncfusion:MapColorPalette FillBrush="Gold"/>
            <syncfusion:MapColorPalette FillBrush="LightBlue"/>
            <syncfusion:MapColorPalette FillBrush="LightCyan"/>
          </syncfusion:ShapeSetting.CustomColors>
          <syncfusion:ShapeSetting.FillSetting>
            <syncfusion:ShapeFillSetting AutoFillColors="True"/>
          </syncfusion:ShapeSetting.FillSetting>
        </syncfusion:ShapeSetting>
      </syncfusion:ShapeFileLayer>
    </syncfusion:SfMap.Layers>
  </syncfusion:SfMap>
```

```

</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap >

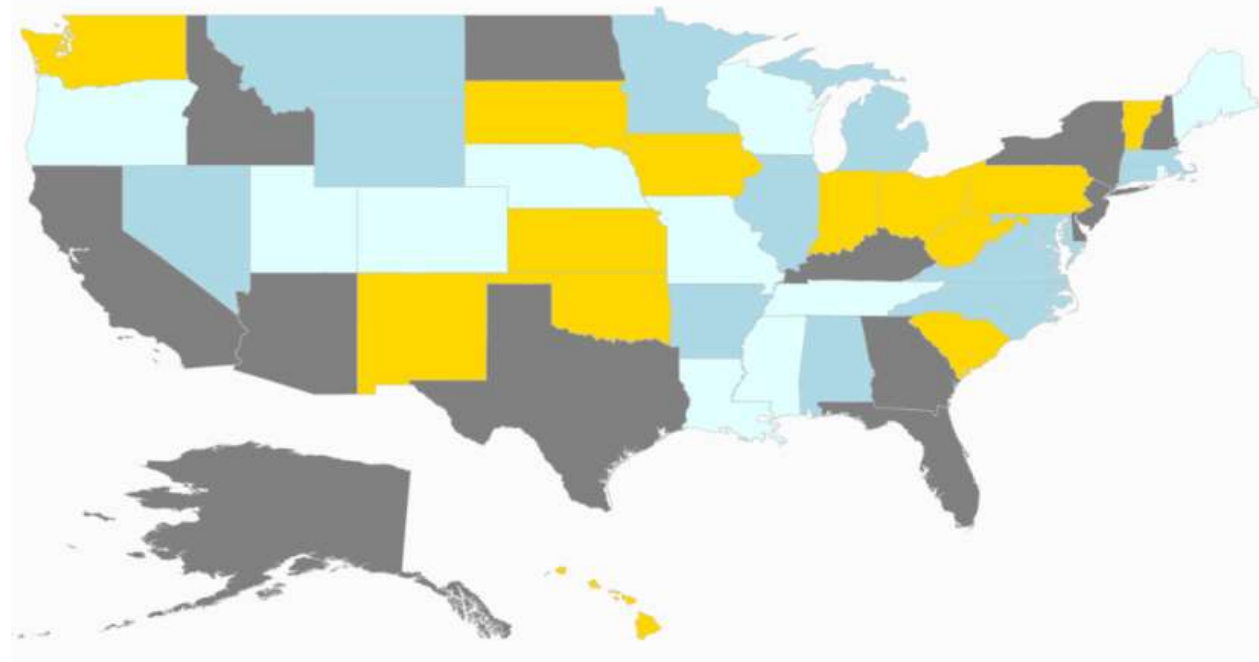
```

C#

```

SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.usa_state.shp";
map.Layers.Add(shapeFileLayer);
ShapeSetting setting = new ShapeSetting();
setting.ColorPalette = ColorPalettes.CustomPalette;
MapColorPalette palette = new MapColorPalette();
palette.FillBrush = new SolidColorBrush(Colors.Gray);
setting.CustomColors.Add(palette);
palette = new MapColorPalette();
palette.FillBrush = new SolidColorBrush(Colors.Gold);
setting.CustomColors.Add(palette);
palette = new MapColorPalette();
palette.FillBrush = new SolidColorBrush(Colors.LightBlue);
setting.CustomColors.Add(palette);
palette = new MapColorPalette();
palette.FillBrush = new SolidColorBrush(Colors.LightCyan);
setting.CustomColors.Add(palette);
ShapeFillSetting shapeFillSetting = new ShapeFillSetting();
shapeFillSetting.AutoFillColors = true;
setting.FillSetting = shapeFillSetting;
shapeFileLayer.ShapeSettings = setting;
this.Content = map;

```



Note: You can also explore our [WPF Map example](#) to know how to render and configure the map.

Annotations in WPF Maps (SfMap)

Annotations are notes that are used to leave some message on the map. In Maps, annotations are denoted by the MapAnnotations. MapAnnotation has two major parts:

1. AnnotationLabel
2. AnnotationSymbol

[AnnotationLabel](#) is a `Text` that shows some information in the text format.

[AnnotationSymbol](#) is a `VisualObject` that shows a note symbolically.

1. [AnnotationLabelForeground](#): Get or sets the foreground color of the annotation label.
2. [AnnotationLabelFontStyle](#): Gets or sets the font style of the annotation label.
3. [AnnotationLabelBackground](#): Gets or sets the background color of the annotation label.
4. [AnnotationLabelFontFamily](#): Gets or sets the font family for the annotation label.
5. [AnnotationLabelFontSize](#): Gets or sets the annotation label font size.
6. [Latitude](#): Gets or sets the Latitude coordinate of the Annotation.
7. [Longitude](#): Gets or set the Longitude coordinate of the Annotation.

XML

```
<syncfusion:SfMap >
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="DataMarkers.ShapeFiles.world1.shp">
<syncfusion:ShapeFileLayer.Annotations>
<syncfusion:MapAnnotations Latitude="-22" Longitude="132"
AnnotationLabel="Australia"
AnnotationLabelFontFamily="Times New Roman"
AnnotationLabelFontSize="20"
AnnotationLabelFontStyle="Oblique"
AnnotationLabelForeground="Red" >
<syncfusion:MapAnnotations.AnnotationSymbol>
<Ellipse Fill="Orange" Height="10" Width="10">
</Ellipse>
</syncfusion:MapAnnotations.AnnotationSymbol>
</syncfusion:MapAnnotations>
<syncfusion:MapAnnotations Latitude="40" Longitude="-98"
AnnotationLabel="Unites States of America"
AnnotationLabelFontFamily="Times New Roman"
AnnotationLabelFontSize="20"
AnnotationLabelFontStyle="Oblique"
AnnotationLabelForeground="Red" >
<syncfusion:MapAnnotations.AnnotationSymbol>
<Ellipse Fill="Orange" Height="10" Width="10">
</Ellipse>
</syncfusion:MapAnnotations.AnnotationSymbol>
</syncfusion:MapAnnotations>
</syncfusion:ShapeFileLayer.Annotations>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap >
```

C#

```
SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.world1.shp";
MapAnnotations annotations = new MapAnnotations();
annotations.Latitude = -22;
annotations.Longitude = 132;
annotations.AnnotationLabel = "Australia";
annotations.AnnotationLabelFontFamily = new FontFamily("Times New Roman");
annotations.AnnotationLabelFontSize = 20;
annotations.AnnotationLabelFontStyle = FontStyles.Italic;
annotations.AnnotationLabelForeground = new SolidColorBrush(Colors.Red);
Ellipse ellipse = new Ellipse();
ellipse.Fill = new SolidColorBrush(Colors.Orange);
ellipse.Height = 10;
ellipse.Width = 10;
annotations.AnnotationSymbol = ellipse;
shapeFileLayer.Annotations.Add(annotations);
MapAnnotations annotations1 = new MapAnnotations();
annotations1.Latitude = 40;
annotations1.Longitude = -98;
annotations1.AnnotationLabel = "Unites States of America";
annotations1.AnnotationLabelFontFamily = new FontFamily("Times New Roman");
annotations1.AnnotationLabelFontSize = 20;
annotations1.AnnotationLabelFontStyle = FontStyles.Italic;
annotations1.AnnotationLabelForeground = new SolidColorBrush(Colors.Red);
Ellipse ellipse1 = new Ellipse();
ellipse1.Fill = new SolidColorBrush(Colors.Orange);
ellipse1.Height = 10;
ellipse1.Width = 10;
annotations1.AnnotationSymbol = ellipse1;
shapeFileLayer.Annotations.Add(annotations1);
map.Layers.Add(shapeFileLayer);
grid.Children.Add(map);
```



Positioning a MapAnnotation

MapAnnotation can be positioned anywhere on the map based on latitude and longitude.

MapAnnotation has two properties called [Latitude](#) and [Longitude](#) that are string types used to set coordinates of the MapAnnotation in the form of latitude and longitude.

Customizing the Annotation Template

The default appearance of the annotation can be customized by using the [AnnotationTemplate](#) property. The annotation template property is available in the ShapeFileLayer.

AnnotationTemplate is a DataTemplate type, used to customize or override the default template of MapAnnotations.

XML

```
<Grid x:Name="grid">
  <Grid.Resources>
    <DataTemplate x:Key="annotationTemplate">
      <Grid >
        <ContentPresenter Content="{Binding AnnotationSymbol}"/>
        <TextBlock Text="{Binding AnnotationLabel}" HorizontalAlignment="Center"
          VerticalAlignment="Center" />
      </Grid>
    </DataTemplate>
  </Grid.Resources>
  <syncfusion:SfMap >
    <syncfusion:SfMap.Layers>
      <syncfusion:ShapeFileLayer Uri="DataMarkers.ShapeFiles.world1.shp"
        AnnotationTemplate="{StaticResource annotationTemplate}">
        <syncfusion:ShapeFileLayer.Annotations>
          <syncfusion:MapAnnotations Latitude="-22" Longitude="132"
            AnnotationLabel="Australia" >
            <syncfusion:MapAnnotations.AnnotationSymbol>
              <Image Height="100" Width="100" Source="note.jpg" />
            </syncfusion:MapAnnotations.AnnotationSymbol>
          </syncfusion:MapAnnotations>
          <syncfusion:MapAnnotations Latitude="40" Longitude="-98"
            AnnotationLabel="USA">
            <syncfusion:MapAnnotations.AnnotationSymbol>
              <Image Height="100" Width="100" Source="note.jpg" />
            </syncfusion:MapAnnotations.AnnotationSymbol>
          </syncfusion:MapAnnotations>
        </syncfusion:ShapeFileLayer.Annotations>
      </syncfusion:ShapeFileLayer>
    </syncfusion:SfMap.Layers>
  </syncfusion:SfMap >
</Grid>
```

C#

```
SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "WpfUG.Assets.ShapeFiles.world1.shp";
shapeFileLayer.AnnotationTemplate = grid.Resources["annotationTemplate"] as
DataTemplate;
MapAnnotations annotations = new MapAnnotations();
annotations.Latitude = -22;
```

```
annotations.Longitude = 132;  
annotations.AnnotationLabel = "Australia";  
Image image = new Image();  
image.Source = new BitmapImage(new Uri(@"Assets/pin.png",  
UriKind.Relative));  
image.Height = 100;  
image.Width = 100;  
annotations.AnnotationSymbol = image;  
shapeFileLayer.Annotations.Add(annotations);  
MapAnnotations annotations1 = new MapAnnotations();  
annotations1.Latitude = 40;  
annotations1.Longitude = -98;  
annotations1.AnnotationLabel = "USA";  
image = new Image();  
image.Source = new BitmapImage(new Uri(@"Assets/pin.png",  
UriKind.Relative));  
image.Height = 100;  
image.Width = 100;  
annotations1.AnnotationSymbol = image;  
shapeFileLayer.Annotations.Add(annotations1);  
map.Layers.Add(shapeFileLayer);  
grid.Children.Add(map);
```



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Map Selection in WPF Maps (SfMap)

Each shape in the map can be selected and unselected when interacted with shapes. There are two ways to select the map shapes:

1. Single Selection
2. Multi Selection

The selected map shapes is differentiate by its fill. [SelectedShapeColor](#) of `ShapeSetting` is the API that is used to get or set the selected shape color.

All selected shapes available in the [SelectedMapShapes](#) of `ShapeFileLayer`.

Single Selection

Single selection allows only one shape to be selected at a time. You can select the shape by clicking or tapping on the shape. Single selection is enabled by the [EnableSelection](#) property of ShapeFileLayer. When `EnableSelection` is set to true, then the map can be selected. When it is set to false, the shapes cannot be selected. When any other shape or the map area is selected, then the shape that is already selected is unselected.

XML

```
<syncfusion:SfMap >
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer x:Name="shapeLayer" EnableSelection="True"
Uri="DataMarkers.ShapeFiles.world1.shp">
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeFill="#E5E5E5" SelectedShapeColor="#1196CD"
ShapeStroke="#C1C1C1" ShapeStrokeThickness="1" />
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.EnableSelection = true;
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.world1.shp";
ShapeSetting setting = new ShapeSetting();
setting.ShapeFill = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#E5E5E5"));
setting.ShapeStroke= (SolidColorBrush)(new
BrushConverter().ConvertFrom("#C1C1C1"));
setting.SelectedShapeColor= (SolidColorBrush)(new
BrushConverter().ConvertFrom("#1196CD"));
setting.ShapeStrokeThickness = 1;
shapeFileLayer.ShapeSettings = setting;
map.Layers.Add(shapeFileLayer);
this.Content = map;
```



Multi Selection

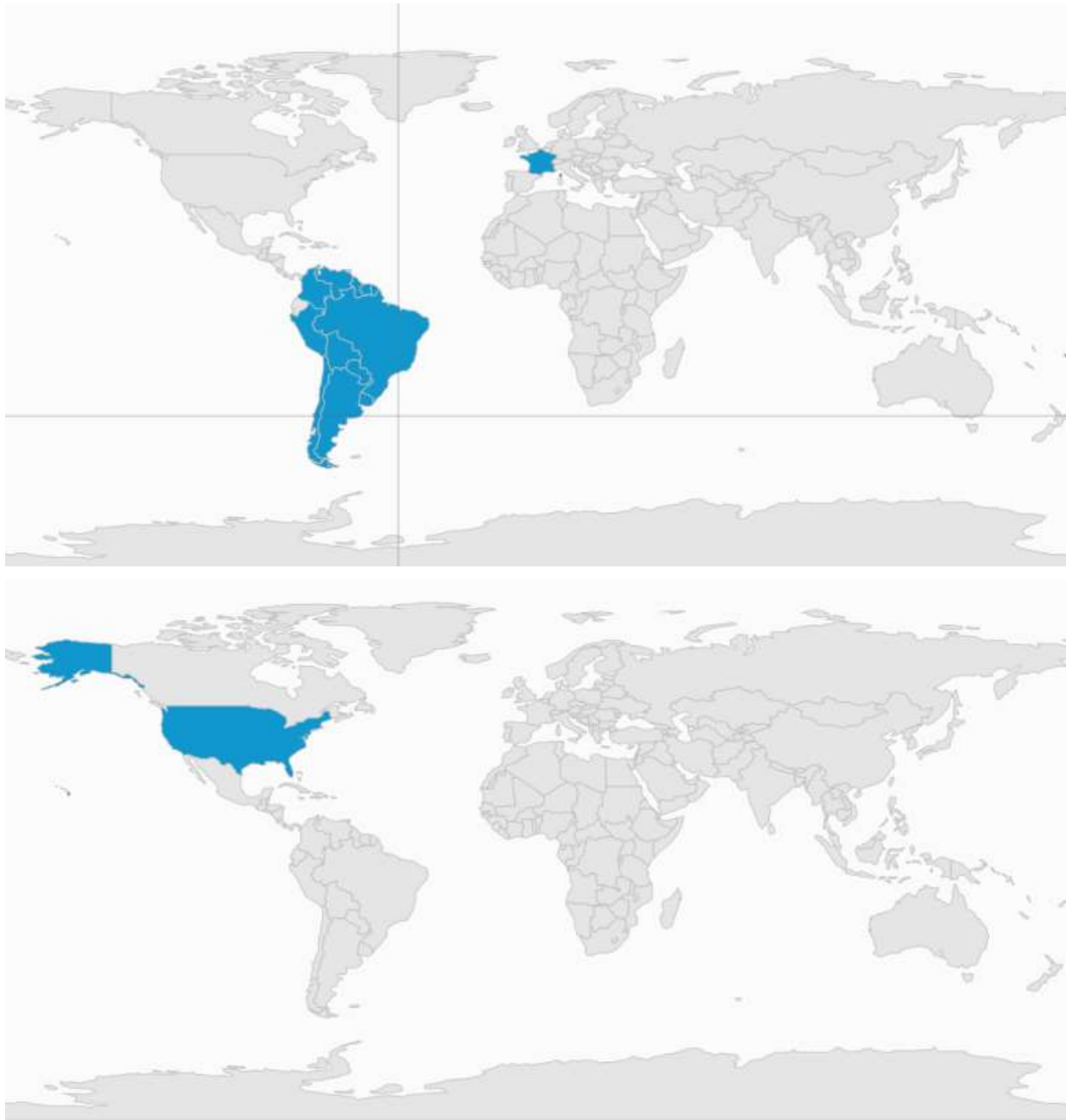
Multiple shapes in the map can be selected when [EnableMultiSelection](#) of ShapeFileLayer is set to true. When EnableMultiSelection is set to true, a cross-hair cursor appears on the map to guide the selection. When you drag on the map, a rectangle appears. The shapes bound that intersect with the rectangle is selected. When EnableMultiSelection is set to true, the panning does not work through interactions.

XML

```
<syncfusion:SfMap >
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer x:Name="shapeLayer" CrossCursorStroke="#686868"
CrossCursorStrokeThickness="0.5" EnableSelection ="True"
EnableMultiSelection="True"
Uri="DataMarkers.ShapeFiles.world1.shp">
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeFill="#E5E5E5" SelectedShapeColor="#1196CD"
ShapeStroke="#C1C1C1" ShapeStrokeThickness="1" />
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.EnableMultiSelection = true;
shapeFileLayer.EnableSelection = true;
shapeFileLayer.CrossCursorStrokeThickness = 0.5;
shapeFileLayer.CrossCursorStroke= (SolidColorBrush) (new
BrushConverter().ConvertFrom("#686868"));
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.world1.shp";
map.Layers.Add(shapeFileLayer);
this.Content = map;
```



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

MapPopup in WPF Maps (SfMap)

MapPopup is a hanging window, displayed when the shape is tapped. It shows additional information from the object bounded with the shape. By default, it takes the property of the bounded object that is referred in the ShapeValuePath and displays its content when the corresponding shape is tapped.

MapPopup is displayed only when [MapPopupVisibility](#) set to true in the shape file layer.

It also customizes the MapPopup template. [PopupCustomTemplate](#) is a DataTemplate type API that is used to expose the custom template for the MapPopup.

XML

```

<syncfusion:SfMap >
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer TranslateZoomLevel="5" ShapeIDPath="Name"
MapPopupVisibility="Visible" Background="#FFCFCECD"
ShapeIDTableField="NAME" ItemsSource="{Binding Countries}"
Uri="DataMarkers.ShapeFiles.world1.shp">
<syncfusion:ShapeFileLayer.PopupCustomTemplate>
<DataTemplate>
<Border>
<Grid Width="200">
<Grid.RowDefinitions>
<RowDefinition />
<RowDefinition />
<RowDefinition />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<Border VerticalAlignment="Center" Padding="10,5,0,0" Height="30"
Background="#FF4B4A4A" Grid.ColumnSpan="2">
<TextBlock Foreground="White" FontFamily="Segoe UI" FontSize="14"
Text="{Binding Name}"/>
</Border>
<Border Padding="3,10,3,3" Background="White" Grid.Row="1"
Grid.ColumnSpan="2">
<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center"
Height="35" Background="White" Orientation="Horizontal">
<TextBlock Foreground="Black" FontFamily="Segoe UI" FontSize="16"
Text="Total Sales:" />
<TextBlock Foreground="Black" FontFamily="Segoe UI" FontSize="16"
Text="{Binding Population}"/>
</StackPanel>
</Border>
<Border HorizontalAlignment="Center" Grid.Row="2">
<StackPanel Orientation="Horizontal">
<ItemsControl Background="White" x:Name="ProductName"
ItemsSource="{Binding Name}" DisplayMemberPath="Name"/>
<ItemsControl Background="White" ItemsSource="{Binding Population}"
DisplayMemberPath="Population"/>
</StackPanel>
</Border>
</Grid>
</Border>
</DataTemplate>
</syncfusion:ShapeFileLayer.PopupCustomTemplate>
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeStroke="#FF1978AA"
ShapeValuePath="Population"
ColorPalette="CustomPalette" ShapeStrokeThickness="0.5" >
<syncfusion:ShapeSetting.CustomColors>
<syncfusion:MapColorPalette FillBrush="#FFC6EAFB"/>
<syncfusion:MapColorPalette FillBrush="#FF93D3F4"/>
<syncfusion:MapColorPalette FillBrush="#FF5FB5E6"/>
<syncfusion:MapColorPalette FillBrush="#FF3E9FD8"/>
<syncfusion:MapColorPalette FillBrush="#FF2991CF"/>

```

```

</syncfusion:ShapeSetting.CustomColors>
<syncfusion:ShapeSetting.FillSetting>
<syncfusion:ShapeFillSetting AutoFillColors="True"/>
</syncfusion:ShapeSetting.FillSetting>
</syncfusion:ShapeSetting>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap >

```

C#

```

public class Country
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    private double population;
    public double Population
    {
        get { return population; }
        set { population = value; }
    }
}

public class ViewModel
{
    private ObservableCollection<Country> countries;
    public ObservableCollection<Country> Countries
    {
        get { return countries; }
        set { countries = value; }
    }
    public ViewModel()
    {
        Countries = new ObservableCollection<Country>
        {
            new Country { Name = "Russia", Population = 143228300},
            new Country { Name = "China", Population = 1347350000 },
            new Country { Name = "Australia", Population = 22789701 },
            new Country { Name = "South Africa", Population = 50586757},
            new Country { Name = "United States", Population = 314623000 },
            new Country { Name = "Egypt", Population = 82724000},
        };
    }
}

```



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Map Shape Labels in WPF Maps (SfMap)

Labels for map shapes can be displayed by using the [LabelPath](#) of ShapeFileLayer. The value of LabelPath must be a field name specified in the .dbf file corresponding to the shapefile.

Property	Type	Description
LabelPath	string	Gets or sets the field name in the database (.dbf) file.

XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer x:Name="shapeFileLayer"
      Uri="DataMarkers.ShapeFiles.world1.shp"
      LabelPath="NAME" FontSize="14">
    </syncfusion:ShapeFileLayer>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.LabelPath = "NAME";
shapeFileLayer.FontSize = 14;
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.world1.shp";
map.Layers.Add(shapeFileLayer);
this.Content = map;
```

The labels can also be customized by modifying the [ItemsTemplate](#) of ShapeFileLayer. The labels can be accessed by using DBFData as follows:

XML

```
<Grid.Resources>
<DataTemplate x:Key="itemtemplate">
<Grid >
<TextBlock Text="{Binding DbfData[NAME]}"
FontSize="14" Margin="10 5"/>
</Grid>
</DataTemplate>
</Grid.Resources>
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="DataMarkers.ShapeFiles.world1.shp"
LabelPath="NAME" ItemsTemplate="{StaticResource itemtemplate}">
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.LabelPath = "NAME";
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.world1.shp";
shapeFileLayer.ItemsTemplate = grid.Resources["itemTemplate"] as
DataTemplate;
map.Layers.Add(shapeFileLayer);
this.Content = map;
```



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Multilayer Support in WPF Maps (SfMap)

Multilayer support allows you to load multiple shapefiles in a single container, enabling maps to display more detail.

Loading Multiple Shapefiles in a Single Container

This feature enables the map to load multiple types of shapefiles in a single container. Basically, shape files contain point shapes in a single container.

Situations arise where, any combination of available shapefiles needs to be loaded in a single container. In such situations this feature enables the map to load multiple shape layers in a single container.

Adding Multiple Layers in the Map

ShapeFileLayer is the core layer for the map. Multiple layers can be added in the [ShapeFileLayer](#) itself. They have to be added in SubShapeFileLayers within the ShapeFileLayer.

SubShapeFileLayers

SubShapeFileLayers is the collection of [SubShapeFileLayer](#). SubShapeFileLayer is also a type of shapefile layer. The following code adds multiple layers in the ShapeFileLayer.

XML

```
<syncfusion:SfMap >
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Background="White" x:Name="shapelayer"
ItemsSource="{Binding MultiLayerList}" ShapeIDPath="NAME"
ShapeIDTableField="Continent" EnableSelection="True"
Uri="DataMarkers.ShapeFiles.continent.shp">
<syncfusion:ShapeFileLayer.BubbleMarkerSetting>
<syncfusion:BubbleMarkerSetting AutoFillColor="False" MaxSize="40"
MinSize="0"
StrokeThickness="0" ValuePath="Weather"
ColorValuePath="Weather">
<syncfusion:BubbleMarkerSetting.ColorMappings>
<syncfusion:RangeColorMapping Color="#8C5FD3" To="30" From="0"/>
<syncfusion:RangeColorMapping Color="#EF6AB3" To="55" From="45"/>
<syncfusion:RangeColorMapping Color="#8AC63C" To="40" From="30"/>
<syncfusion:RangeColorMapping Color="#8AC63C" To="45" From="40"/>
</syncfusion:BubbleMarkerSetting.ColorMappings>
</syncfusion:BubbleMarkerSetting>
</syncfusion:ShapeFileLayer.BubbleMarkerSetting>
<syncfusion:ShapeFileLayer.Annotations>
<syncfusion:MapAnnotations AnnotationLabelFontSize="14"
AnnotationLabelFontFamily="Segoe UI"
AnnotationLabelForeground="#8C1313"
AnnotationLabel="North America"
Latitude="40.4230" Longitude="-112.7372" />
<syncfusion:MapAnnotations AnnotationLabelFontSize="14"
AnnotationLabelFontFamily="Segoe UI"
AnnotationLabelForeground="#8C1313"
AnnotationLabel="Africa"
Latitude="9.1021" Longitude="18.2812" />
<syncfusion:MapAnnotations AnnotationLabelFontSize="14"
AnnotationLabelFontFamily="Segoe UI"
```

```

AnnotationLabelForeground="#8C1313"
AnnotationLabel="Europe"
Latitude="53.0000" Longitude="9.0000" />
<syncfusion:MapAnnotations AnnotationLabelFontSize="14"
AnnotationLabelFontFamily="Segoe UI"
AnnotationLabelForeground="#8C1313"
AnnotationLabel="South America"
Latitude="-19.6048" Longitude="-73.0625" />
<syncfusion:MapAnnotations AnnotationLabelFontSize="14"
AnnotationLabelFontFamily="Segoe UI"
AnnotationLabelForeground="#8C1313"
AnnotationLabel="Asia"
Latitude="49.8380" Longitude="105.8203" />
<syncfusion:MapAnnotations AnnotationLabelFontSize="14"
AnnotationLabelFontFamily="Segoe UI"
AnnotationLabelForeground="#8C1313"
AnnotationLabel="Oceania"
Latitude="-25.3456" Longitude="125.4346" />
</syncfusion:ShapeFileLayer.Annotations>
<syncfusion:ShapeFileLayer.ItemsTemplate>
<DataTemplate>
<Border Visibility="{Binding Data.ItemsVisibility}"
Background="Transparent">
<StackPanel Orientation="Horizontal">
<TextBlock FontFamily="Segoe UI" Foreground="#FF333333"
FontSize="12" Text="{Binding Data.Weather}"/>
<TextBlock FontFamily="Segoe UI" Foreground="#FF333333"
FontSize="12" Text="°C"/>
</StackPanel>
</Border>
</DataTemplate>
</syncfusion:ShapeFileLayer.ItemsTemplate>
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeStroke="#C1C1C1" ShapeStrokeThickness="0.5"
ShapeValuePath="Weather" ShapeFill="#E5E5E5"/>
</syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeFileLayer.SubShapeFileLayers>
<syncfusion:SubShapeFileLayer ShapeIDPath="NAME" ItemsSource="{Binding
AfricaList}"
ShapeIDTableField="Country"
Uri="DataMarkers.ShapeFiles.Africa.shp">
<syncfusion:SubShapeFileLayer.BubbleMarkerSetting>
<syncfusion:BubbleMarkerSetting AutoFillColor="False" MaxSize="40"
MinSize="20" StrokeThickness="0"
ValuePath="Weather" ColorValuePath="Weather">
<syncfusion:BubbleMarkerSetting.ColorMappings>
<syncfusion:RangeColorMapping Color="#8C5FD3" To="30" From="0"/>
<syncfusion:RangeColorMapping Color="#EF6AB3" To="35" From="30"/>
<syncfusion:RangeColorMapping Color="#8AC63C" To="40" From="35"/>
<syncfusion:RangeColorMapping Color="#F79E46" To="47" From="40"/>
</syncfusion:BubbleMarkerSetting.ColorMappings>
</syncfusion:BubbleMarkerSetting>
</syncfusion:SubShapeFileLayer.BubbleMarkerSetting>
<syncfusion:SubShapeFileLayer.ItemsTemplate>
<DataTemplate>
<Border Background="Transparent">
<StackPanel Orientation="Horizontal">

```



```

<TextBlock FontFamily="Segoe UI" Foreground="#FF333333"
FontSize="12" Text="{Binding Data.Weather}"/>
<TextBlock FontFamily="Segoe UI" Foreground="#FF333333"
FontSize="12" Text="°C"/>
</StackPanel>
</Border>
</DataTemplate>
</syncfusion:SubShapeFileLayer.ItemsTemplate>
<syncfusion:SubShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeStroke="#2F8CEA" ShapeValuePath="Weather"
ShapeFill="#8DCEFF">
<syncfusion:ShapeSetting.FillSetting>
<syncfusion:ShapeFillSetting />
</syncfusion:ShapeSetting.FillSetting>
</syncfusion:ShapeSetting>
</syncfusion:SubShapeFileLayer.ShapeSettings>
</syncfusion:SubShapeFileLayer>
<syncfusion:SubShapeFileLayer ShapeIDPath="NAME"
ShapeIDTableField="ADMIN_NAME"
ItemsSource="{Binding OceaniaList}"
Uri="DataMarkers.ShapeFiles.australia.shp">
<syncfusion:SubShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeStroke="#2F8CEA" ShapeValuePath="Weather"
ShapeFill="#8DCEFF">
<syncfusion:ShapeSetting.FillSetting>
<syncfusion:ShapeFillSetting AutoFillColors="False"/>
</syncfusion:ShapeSetting.FillSetting>
</syncfusion:ShapeSetting>
</syncfusion:SubShapeFileLayer.ShapeSettings>
<syncfusion:SubShapeFileLayer.BubbleMarkerSetting>
<syncfusion:BubbleMarkerSetting AutoFillColor="False" MaxSize="40"
MinSize="20" StrokeThickness="0" ValuePath="Weather">
<syncfusion:BubbleMarkerSetting.ColorMappings>
<syncfusion:RangeColorMapping Color="#8C5FD3" To="20" From="0"/>
<syncfusion:RangeColorMapping Color="#EF6AB3" To="25" From="20"/>
<syncfusion:RangeColorMapping Color="#407715" To="30" From="25"/>
<syncfusion:RangeColorMapping Color="#63A028" To="35" From="30"/>
</syncfusion:BubbleMarkerSetting.ColorMappings>
</syncfusion:BubbleMarkerSetting>
</syncfusion:SubShapeFileLayer.BubbleMarkerSetting>
<syncfusion:SubShapeFileLayer.ItemsTemplate>
<DataTemplate>
<Border Background="Transparent">
<StackPanel Orientation="Horizontal">
<TextBlock FontFamily="Segoe UI" Foreground="#FF333333"
FontSize="12" Text="{Binding Data.Weather}"/>
<TextBlock FontFamily="Segoe UI" Foreground="#FF333333"
FontSize="12" Text="°C"/>
</StackPanel>
</Border>
</DataTemplate>
</syncfusion:SubShapeFileLayer.ItemsTemplate>
</syncfusion:SubShapeFileLayer>
</syncfusion:ShapeFileLayer.SubShapeFileLayers>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>

```

C#

```

public class Country : INotifyPropertyChanged
{
    public string NAME { get; set; }
    private Visibility itemsvisibility = Visibility.Visible;
    public Visibility ItemsVisibility
    {
        get { return itemsvisibility; }
        set { itemsvisibility = value; }
    }
    private double weather { get; set; }
    public double Weather
    {
        get
        {
            return weather;
        }
        set
        {
            weather = value;
        }
    }
    private double population { get; set; }
    public double Population
    {
        get
        {
            return population;
        }
        set
        {
            population = value;
            OnPropertyChanged(new PropertyChangedEventArgs("Population"));
        }
    }
    public string PopulationFormat { get; set; }
    public event PropertyChangedEventHandler PropertyChanged;
    public void OnPropertyChanged(PropertyChangedEventArgs e)
    {
        this.PopulationFormat = (String.Format("{0:0,0}",
        this.Population).Trim('$'));
        if (PropertyChanged != null)
        {
            PropertyChanged(this, e);
        }
    }
}

public class MapViewModel
{
    public ObservableCollection<Country> MultiLayerList { get; set; }
    public ObservableCollection<Country> AfricaList { get; set; }
    public ObservableCollection<Country> OceaniaList { get; set; }
    public MapViewModel()
    {
    }
}

```

```

this.OceaniaList = new ObservableCollection<Country>();
this.OceaniaList.Add(new Country() { NAME = "New South Wales", Weather = 26
});
this.OceaniaList.Add(new Country() { NAME = "Queensland", Weather = 30 });
this.OceaniaList.Add(new Country() { NAME = "Tasmania", Weather = 21 });
this.OceaniaList.Add(new Country() { NAME = "Western Australia", Weather =
32 });
this.AfricaList = new ObservableCollection<Country>();
this.AfricaList.Add(new Country() { NAME = "Algeria", Weather = 47 });
this.AfricaList.Add(new Country() { NAME = "Congo (Brazzaville)", Weather =
45 });
this.AfricaList.Add(new Country() { NAME = "Ethiopia", Weather = 50 });
this.AfricaList.Add(new Country() { NAME = "South Africa", Weather = 30 });
this.MuliLayerList = new ObservableCollection<Country>();
this.MuliLayerList.Add(new Country() { NAME = "Asia", Weather = 40 });
this.MuliLayerList.Add(new Country() { NAME = "South America", Weather = 45
});
this.MuliLayerList.Add(new Country() { NAME = "North America", Weather = 52
});
this.MuliLayerList.Add(new Country() { NAME = "Antarctica", ItemsVisibility
= Visibility.Collapsed });
this.MuliLayerList.Add(new Country() { NAME = "Oceania", ItemsVisibility =
Visibility.Collapsed });
this.MuliLayerList.Add(new Country() { NAME = "Europe", ItemsVisibility =
Visibility.Collapsed });
this.MuliLayerList.Add(new Country() { NAME = "Africa", ItemsVisibility =
Visibility.Collapsed });
}
}

```



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Map Points in WPF Maps (SfMap)

Points are one of the record type in shape file layer. Points are used to specify the specific point in the map. For example, used to specify the capital of countries. Points in the shape file is given as latitude and longitude coordinates in the shapes file. Those points should be converted as MapPoints.

Customizing the MapPoint

The default appearance of the MapPoint can be customized by using the [MapPointTemplate](#) property. The MapPointTemplate property is available in the ShapeFileLayer.

About MapPointTemplate Property

MapPointTemplate is a DataTemplate type, used to customize or override the default template of MapPoints.

XML

```
<Grid.Resources>
<DataTemplate x:Key="itemtemplate">
<Ellipse Height="10" Width="10" Stroke="White"
Fill="#8AC63C"/>
</DataTemplate>
</Grid.Resources>
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer ShapeIDPath="NAME" ShapeIDTableField="Continent"
EnableSelection="True"
Uri="DataMarkers.ShapeFiles.continent.shp" MapPointTemplate="{StaticResource
itemtemplate}">
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.ShapeIDPath = "NAME";
shapeFileLayer.ShapeIDTableField = "Continent";
shapeFileLayer.EnableSelection = true;
shapeFileLayer.FontSize = 14;
shapeFileLayer.Uri = "DataMarkers.ShapeFiles.continent.shp";
shapeFileLayer.MapPointTemplate = grid.Resources["itemTemplate"] as
DataTemplate;
map.Layers.Add(shapeFileLayer);
this.Content = map;
```

MapPointIcon

[MapPointIcon](#) is used for customizing points shapes. It can be customized by following shapes:

- Rectangle
- Circle
- Square
- Diamond
- Star

XML

```

<Grid.Resources>
<DataTemplate x:Key="pointTemplate">
<Ellipse
Width="10"
Height="10"
Margin="-10,-10,0,0"
Fill="#8AC63C"
Stroke="White" />
</DataTemplate>
</Grid.Resources>
<syncfusion:SfMap x:Name="maps" Margin="10">
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer
EnableSelection="True"
Uri="WpfUG.Assets.ShapeFiles.states.shp">
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting
ShapeStrokeThickness="1"/>
</syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeFileLayer.SubShapeFileLayers>
<syncfusion:SubShapeFileLayer EnableSelection="True"
Uri="WpfUG.Assets.ShapeFiles.landslide.shp"
MapPointTemplate="{StaticResource pointTemplate}">
</syncfusion:SubShapeFileLayer>
</syncfusion:ShapeFileLayer.SubShapeFileLayers>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>

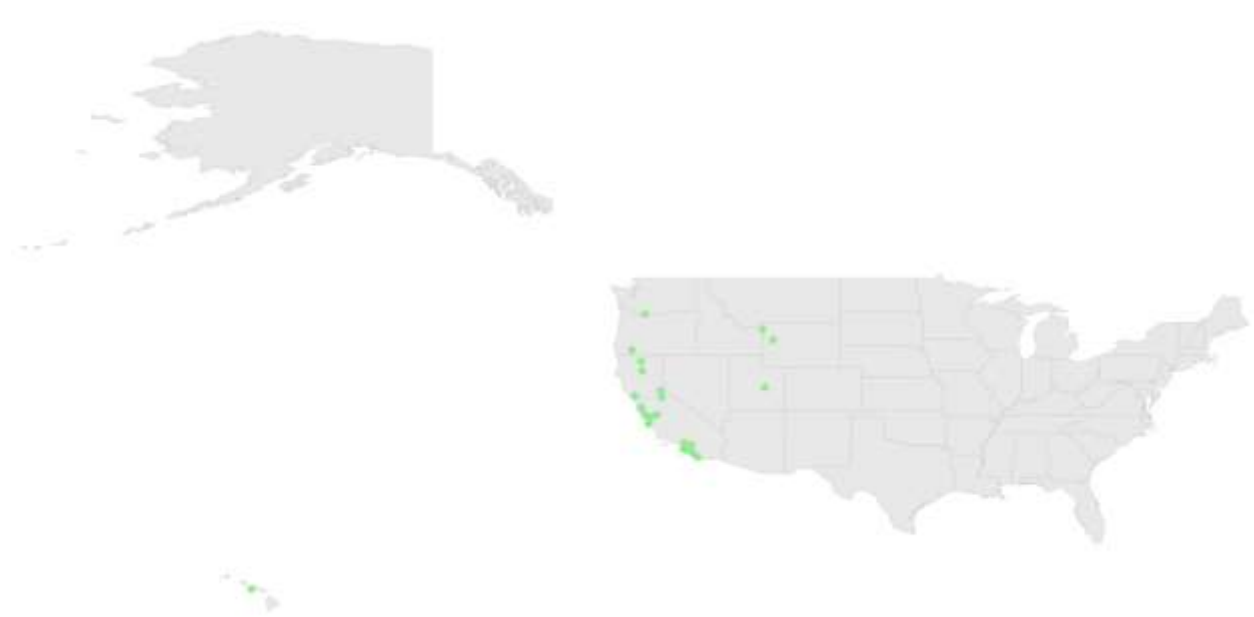
```

C#

```

SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.EnableSelection = true;
shapeFileLayer.Uri = "WpfUG.Assets.ShapeFiles.states.shp";
map.Layers.Add(shapeFileLayer);
ShapeSetting setting = new ShapeSetting();
setting.ShapeStrokeThickness = 1;
shapeFileLayer.ShapeSettings = setting;
SubShapeFileLayer subShapeFileLayer = new SubShapeFileLayer();
subShapeFileLayer.EnableSelection = true;
subShapeFileLayer.Uri = "WpfUG.Assets.ShapeFiles.landslide.shp";
subShapeFileLayer.MapPointTemplate= grid.Resources["pointTemplate"] as
DataTemplate;
shapeFileLayer.SubShapeFileLayers.Add(subShapeFileLayer);
this.Content = map;

```



MapPointPopup

MapPointPopup is a popup, displayed when the point is moved to MapPoint. It shows additional information from the object bounded with the MapPoint. You can get the object from dbf file.

MapPointPopupTemplate

[MapPointPopupTemplate](#) is a DataTemplate, used to expose the template for the MapPoint.

XML

```
<Grid.Resources>
<DataTemplate x:Key="pointTemplate">
<Ellipse
Width="10"
Height="10"
Margin="-10,-10,0,0"
Fill="#8AC63C"
Stroke="White" />
</DataTemplate>
<DataTemplate x:Key="popupTemplate">
<Border
Height="110"
Background="#FF252525"
BorderThickness="0.5"
Opacity="0.9">
<StackPanel Margin="10,5,20,0">
<StackPanel>
<TextBlock
Margin="10,5,0,0"
FontFamily="Segoe UI"
FontSize="20"
Foreground="White"
Text="Landslide Event in USA" />
</StackPanel>
<Grid Margin="10,5,10,0">
```

```

<Rectangle
Height="2"
VerticalAlignment="Center"
Stroke="#FF505050"
StrokeDashArray="6 2" />
</Grid>
<Grid Margin="10,5,0,0">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="0.41*" />
<ColumnDefinition Width="0.1*" />
<ColumnDefinition Width="0.49*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition />
<RowDefinition />
</Grid.RowDefinitions>
<TextBlock
Grid.Row="0"
Grid.Column="0"
FontFamily="Segoe UI"
FontSize="18"
FontWeight="Normal"
Foreground="#FFACACAC"
Text="Location" />
<TextBlock
Grid.Row="0"
Grid.Column="1"
FontFamily="Segoe UI"
FontSize="18"
FontWeight="Normal"
Foreground="#FFACACAC"
Text=" : " />
<TextBlock
Grid.Row="0"
Grid.Column="2"
Margin="5,0,0,0"
FontFamily="Segoe UI"
FontSize="18"
FontWeight="Normal"
Foreground="#FFACACAC"
Text="{Binding [LOCALITY]}" />
<TextBlock
Grid.Row="1"
Grid.Column="0"
FontFamily="Segoe UI"
FontSize="18"
FontWeight="Normal"
Foreground="#FFACACAC"
Text="Year" />
<TextBlock
Grid.Row="1"
Grid.Column="1"
FontFamily="Segoe UI"
FontSize="18"
FontWeight="Normal"
Foreground="#FFACACAC"
Text=" : " />

```

```

<TextBlock
  Grid.Row="1"
  Grid.Column="2"
  Margin="5,0,0,0"
  FontFamily="Segoe UI"
  FontSize="18"
  FontWeight="Normal"
  Foreground="#FFACACAC"
  Text="{Binding [YEAR]}" />
</Grid>
</StackPanel>
</Border>
</DataTemplate>
</Grid.Resources>
<syncfusion:SfMap x:Name="maps" Margin="10">
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer
      EnableSelection="True"
      Uri="WpfUG.Assets.ShapeFiles.states.shp">
      <syncfusion:ShapeFileLayer.ShapeSettings>
        <syncfusion:ShapeSetting
          ShapeStrokeThickness="1"/>
      </syncfusion:ShapeFileLayer.ShapeSettings>
      <syncfusion:ShapeFileLayer.SubShapeFileLayers>
        <syncfusion:SubShapeFileLayer EnableSelection="True"
          Uri="WpfUG.Assets.ShapeFiles.landslide.shp"
          MapPointTemplate="{StaticResource pointTemplate}"
          MapPointPopupTemplate="{StaticResource popupTemplate}">
        </syncfusion:SubShapeFileLayer>
      </syncfusion:ShapeFileLayer.SubShapeFileLayers>
    </syncfusion:ShapeFileLayer>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>

```

C#

```

SfMap map = new SfMap();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.EnableSelection = true;
shapeFileLayer.Uri = "WpfUG.Assets.ShapeFiles.states.shp";
map.Layers.Add(shapeFileLayer);
ShapeSetting setting = new ShapeSetting();
setting.ShapeStrokeThickness = 1;
shapeFileLayer.ShapeSettings = setting;
SubShapeFileLayer subShapeFileLayer = new SubShapeFileLayer();
subShapeFileLayer.EnableSelection = true;
subShapeFileLayer.Uri = "WpfUG.Assets.ShapeFiles.landslide.shp";
subShapeFileLayer.MapPointTemplate= grid.Resources["pointTemplate"] as
DataTemplate;
subShapeFileLayer.MapPointPopupTemplate = grid.Resources["popupTemplate"] as
DataTemplate;
shapeFileLayer.SubShapeFileLayers.Add(subShapeFileLayer);
this.Content = map;

```




Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Legend in WPF Maps (SfMap)

A legend is a key to symbolism used on a map, usually containing swatches of symbols with descriptions. It provides valuable information for interpreting what the map is showing you, and can be represented in various colors and shapes based on the data.

Legend visibility

Legends are visible by setting the [LegendVisibility](#) property of the visibility type as **Visible** in the ShapeFileLayer.

Legend Position

Map legends can be positioned by setting the [LegendPosition](#) property in ShapeFileLayer. Also, the legend can be positioned based on the margin values for the X-axis and the Y-axis with the help of the [LegendPositionX](#) and [LegendPositionY](#) properties available in ShapeFileLayer. For positioning the legend based on margins, corresponding to a map, [LegendPosition](#) must be set with value of **Default**.

Property	Type	Description
LegendPosition	LegendPosition (enum)	Gets or sets the standard position for the legend.
LegendPositionX	Double	Gets or sets the margin value for the x axis.
LegendPositionY	Double	Gets or sets the margin value for the y axis.

Legend header

A header for the legend can be added by setting the [LegendHeader](#) property of the string type.

Legend categories

Legends are categorized as two types:

- Legends for shape layers.
- Legends for bubbles.

These can be set by using the [LegendType](#) property of the type LegendType.

Shapes

Bubble type legends are always bubbles with varying sizes. The size of the bubbles is obtained from the [SizeRatio](#) from the BubbleMarkerSetting.

Layer shape type legends can be different shapes for the legend. The shapes can be set by using the [LegendIcon](#), of the LegendIcon type.

Arranging the Legends

Legends are arranged in matrix format. The number of columns in the arranging matrix can be set by setting the [LegendColumnSplit](#) property of the int type.

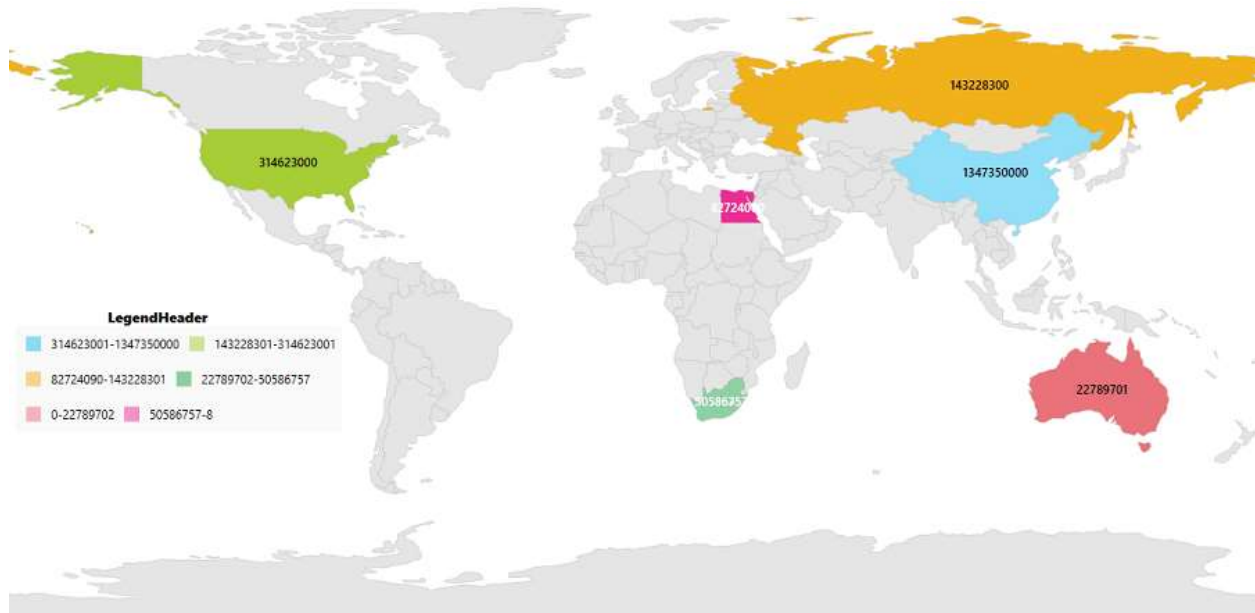
XML

```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer
LegendType="Layers" LegendHeader="LegendHeader"
LegendColumnSplit="2" LegendPositionX="10"
LegendPositionY="300" LegendVisibility="Visible"
LegendIcon="Rectangle" ItemsSource="{Binding Countries}" ShapeIDPath="Name"
ShapeIDTableField="NAME" Uri="DataMarkers.ShapeFiles.world1.shp">
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeColorValuePath="Population"
ShapeFill="#E5E5E5"
ShapeStroke="#C1C1C1" ShapeStrokeThickness="0.5"
ShapeValuePath="Population" >
<syncfusion:ShapeSetting.FillSetting>
<syncfusion:ShapeFillSetting AutoFillColors="False">
<syncfusion:ShapeFillSetting.ColorMappings>
<syncfusion:RangeColorMapping Color="#7F20BCEE" To="1347350000"
From="314623001"/>
<syncfusion:RangeColorMapping Color="#7FA7CE38" To="314623001"
From="143228301"/>
<syncfusion:RangeColorMapping Color="#7FF1B21A" To="143228301"
From="82724090"/>
<syncfusion:RangeColorMapping Color="#7F1DA249" To="50586757"
From="22789702"/>
<syncfusion:RangeColorMapping Color="#7FEB737C" To="22789702" From="0"/>
<syncfusion:RangeColorMapping Color="#7FED2D95" To="82724090"
From="50586757"/>
</syncfusion:ShapeFillSetting.ColorMappings>
</syncfusion:ShapeFillSetting>
</syncfusion:ShapeSetting.FillSetting>
</syncfusion:ShapeSetting>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
public class Country
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    private double population;
    public double Population
    {
        get { return population; }
        set { population = value; }
    }
}

public class ViewModel
{
    private ObservableCollection<Country> countries;
    public ObservableCollection<Country> Countries
    {
        get { return countries; }
        set { countries = value; }
    }
    public ViewModel()
    {
        Countries = new ObservableCollection<Country>
        {
            new Country { Name = "Russia", Population = 143228300},
            new Country { Name = "China", Population = 1347350000 },
            new Country { Name = "Australia", Population = 22789701 },
            new Country { Name = "South Africa", Population = 50586757},
            new Country { Name = "United States", Population = 314623000 },
            new Country { Name = "Egypt", Population = 82724000},
        };
    }
}
```

**XML**

```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer
LegendType="Bubbles" LegendHeader="LegendHeader"
LegendColumnSplit="2" LegendPositionX="10"
LegendPositionY="300" LegendVisibility="Visible"
LegendIcon="Rectangle" ItemsSource="{Binding Countries}" ShapeIDPath="Name"
ShapeIDTableField="NAME" Uri="DataMarkers.ShapeFiles.world1.shp">
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeValuePath="Population"/>
</syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeFileLayer.BubbleMarkerSetting>
<syncfusion:BubbleMarkerSetting AutoFillColor="False" MaxSize="50"
MinSize="20" StrokeThickness="0"
ColorValuePath="Population" ValuePath="Population">
<syncfusion:BubbleMarkerSetting.ColorMappings>
<syncfusion:RangeColorMapping Color="#7F20BCEE" To="1347350000"
From="314623001"/>
<syncfusion:RangeColorMapping Color="#7FA7CE38" To="314623001"
From="143228301"/>
<syncfusion:RangeColorMapping Color="#7FF1B21A" To="143228301"
From="82724090"/>
<syncfusion:RangeColorMapping Color="#7FED2D95" To="82724090"
From="50586757"/>
<syncfusion:RangeColorMapping Color="#7F1DA249" To="50586757"
From="22789702"/>
<syncfusion:RangeColorMapping Color="#7FEB737C" To="22789702" From="0"/>
</syncfusion:BubbleMarkerSetting.ColorMappings>
</syncfusion:BubbleMarkerSetting>
</syncfusion:ShapeFileLayer.BubbleMarkerSetting>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

KML Format in WPF Maps (SfMap)

KML is a file format used for rendering geographical data. It uses a tag-based structure with nested elements and attributes. KML is based on the XML standard, and all tags of a KML file are case-sensitive.

Currently, the SfMaps control supports the following KML elements:

- Place mark
- Point
- LinearRing
- Polygon
- MultiGeometry
- Region
- Style
- StyleMap
- BalloonStyle
- LineStyle
- PolyStyle
- IconStyle
- ExtendedData

KML Shapes Support in ShapeFileLayer

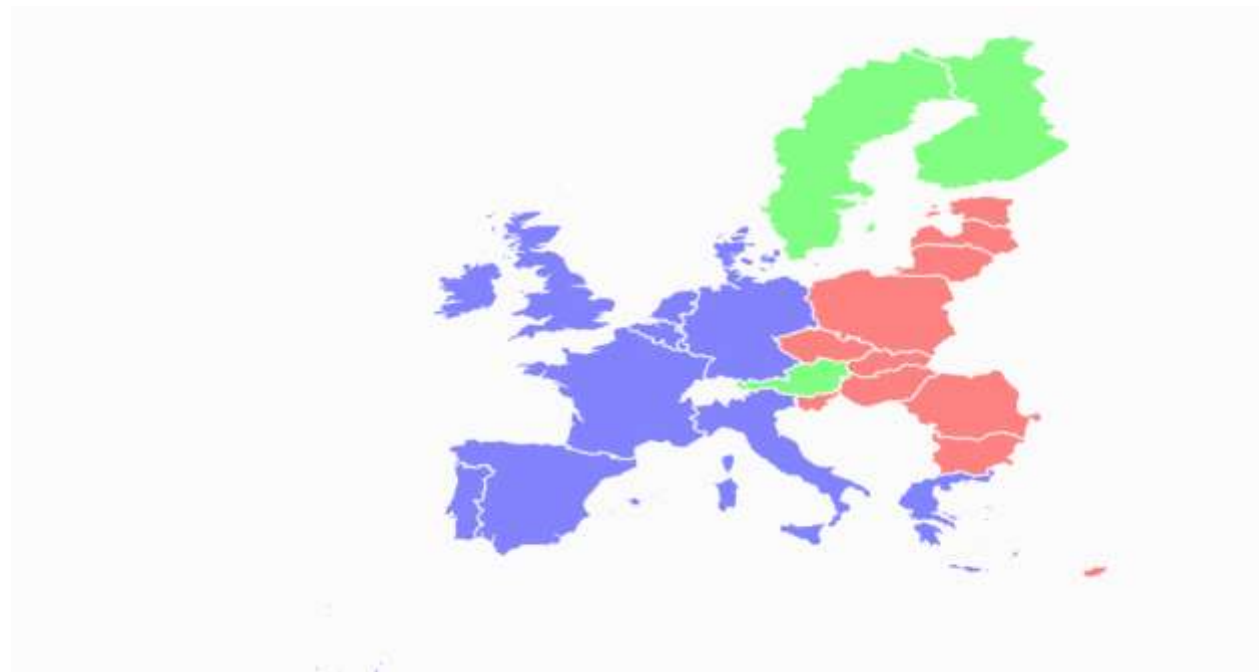
A KML file can be rendered with the help of the ShapeFileLayer in SfMap. The KML file should be added as an Embedded Resource to the application project. The URI of the KML file must be given in the following order:

1. Namespace of project
2. Folder names

3. KmlFileName.kml

XML

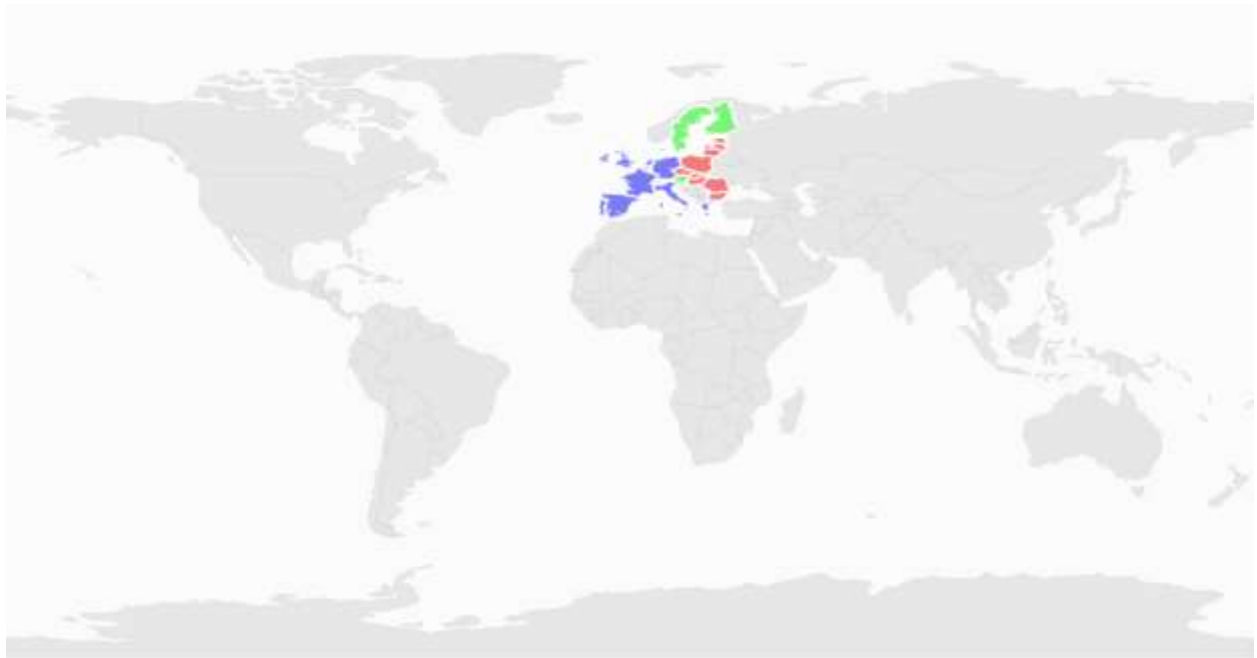
```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="KmlImportDemo.KMLFiles.Eu.kml">
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

**KML Shapes Support in SubShapeFileLayer**

A KML file can be rendered with the help of the SubShapeFileLayer in SfMap. The URI of the KML file given in SubShapeFileLayer as follows.

XML

```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="KmlImportDemo. ShapeFiles.world1.shp">
<syncfusion:ShapeFileLayer.SubShapeFileLayers>
<syncfusion:SubShapeFileLayer Uri="KmlImportDemo.KmlFiles.Eu.kml"/>
</syncfusion:ShapeFileLayer.SubShapeFileLayers>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Map Providers in WPF Maps (SfMap)

SfMap control supports map providers such as OpenStreetMap that can be added to any layers in maps.

Open Street Map

The OpenStreetMap (OSM) is a map of the world built by a community of mappers that is free to use under an open license. It allows you to view geographical data in a collaborative way from anywhere on the Earth. It provides small tile images based on our requests and combines those images into a single one to display the map area in our maps control.

Enable an OSM

You can enable this feature by setting the [LayerType](#) property value as `OSM`.

XML

```
<syncfusion:SfMap ZoomLevel="3">
  <syncfusion:SfMap.Layers>
    <syncfusion:ImageryLayer LayerType="OSM"/>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```



Markers

Markers are used to leave some message on the map.

XML

```
<syncfusion:SfMap x:Name="map">
  <syncfusion:SfMap.Layers>
    <syncfusion:ImageryLayer Markers="{Binding Models}" LayerType="OSM">
    </syncfusion:ImageryLayer>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
public class Model
{
    public string Name { get; set; }
    public string Longitude { get; set; }
    public string Latitude { get; set; }
}

public class MapViewModel
{
    public ObservableCollection<Model> Models { get; set; }
    public MapViewModel()
    {
        this.Models = new ObservableCollection<Model>();
        this.Models.Add(new Model() { Name = "USA ", Latitude = "38.8833N",
        Longitude = "77.0167W" });
        this.Models.Add(new Model() { Name = "Brazil ", Latitude = "15.7833S",
        Longitude = "47.8667W" });
        this.Models.Add(new Model() { Name = "India ", Latitude = "21.0000N",
        Longitude = "78.0000E" });
        this.Models.Add(new Model() { Name = "China ", Latitude = "35.0000N",
        Longitude = "103.0000E" });
        this.Models.Add(new Model() { Name = "Indonesia ", Latitude = "6.1750S",
        Longitude = "106.8283E" });
    }
}
```



```
}
}
```

Customizing the Marker Template

The default appearance of the Marker can be customized by using the [MarkerTemplate](#) property.

XML

```
<syncfusion:SfMap x:Name="map">
  <syncfusion:SfMap.Layers>
    <syncfusion:ImageryLayer Markers="{Binding Models}" LayerType="OSM">
      <syncfusion:ImageryLayer.MarkerTemplate>
        <DataTemplate>
          <Grid Margin="-12,-30,0,0">
            <Canvas>
              <Image Source="pin.png" Height="30"/>
            </Canvas>
            <Grid DataContext="{Binding Data}" Width="265">
              <Grid.RowDefinitions>
                <RowDefinition />
              </Grid.RowDefinitions>
              <Grid.ColumnDefinitions>
                <ColumnDefinition/>
              </Grid.ColumnDefinitions>
              <Canvas Grid.Row="0" Grid.Column="0" Margin="0,0,106,0">
                <Image Source="mappath.png" Width="92" Canvas.Top="25" Canvas.Left="10"/>
                <TextBlock Foreground="White" HorizontalAlignment="Center" FontSize="15"
                  FontFamily="Segoe UI" Text="{Binding Name}" Canvas.Left="25" Canvas.Top="25"
                  RenderTransformOrigin="0.515,-0.3"/>
              </Canvas>
            </Grid>
          </Grid>
        </DataTemplate>
      </syncfusion:ImageryLayer.MarkerTemplate>
    </syncfusion:ImageryLayer>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```



Refer to this [link](#) for customizing marker icons, labels, marker alignment, marker selection and events.

Adding a multiple layers in OSM

Multiple layers can be added in the ImageryLayer itself. They have to be added in SubShapeFileLayers within the ImageryLayer.

SubShapeFileLayers

SubShapeFileLayers is the collection of SubShapeFileLayer. SubShapeFileLayer is also a type of shapefile layer. The following code adds the multiple layers in the ImageryLayer.

XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ImageryLayer LayerType="OSM" >
      <syncfusion:ImageryLayer.SubShapeFileLayers>
        <syncfusion:SubShapeFileLayer Uri="DataMarkers.ShapeFiles.Africa.shp">
          <syncfusion:SubShapeFileLayer.ShapeSettings>
            <syncfusion:ShapeSetting ShapeStroke="#C1C1C1" ShapeStrokeThickness="0.5"
              ShapeFill="Chocolate"/>
          </syncfusion:SubShapeFileLayer.ShapeSettings>
        </syncfusion:SubShapeFileLayer>
      </syncfusion:ImageryLayer.SubShapeFileLayers>
    </syncfusion:ImageryLayer>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

```
</syncfusion:ImageryLayer>  
</syncfusion:SfMap.Layers>  
</syncfusion:SfMap>
```



Bing Map

Bing maps is a map of the entire world owned by Microsoft. As with OSM, it provides map tile images based on our requests and combines those images into a single one to display a map area.

Enable a Bing Map

You can enable this feature by defining the LayerType as **Bing**.

Bing Map Key

The [BingMapKey](#) is provided as input to this key property. The Bing Map key can be obtained from

<http://www.microsoft.com/maps/create-a-bing-maps-key.aspx>.

Maps supports three types of Bing map viewing style options.

1. Aerial
2. AerialWithLabel
3. Road.

The default view of Bing map style is Road.

Aerial View

Aerial view displays satellite images that highlight roads and major landmarks for easy identification. To apply the Aerial view, set [BingMapStyle](#) as `Aerial`, as shown in the following code.

XML

```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ImageryLayer LayerType="Bing" BingMapKey="<Your Bing map key>"
BingMapStyle="Aerial" />
</syncfusion:SfMap.Layers>
</syncfusion:SfMap >
```

C#

```
SfMap map = new SfMap();
ImageryLayer layer = new ImageryLayer();
layer.LayerType = LayerType.Bing;
layer.BingMapKey = "Your Bing map key";
layer.BingMapStyle = BingMapStyle.Aerial;
map.Layers.Add(layer);
this.Content = map;
```

The following screenshot illustrates the Aerial View.



Road View

Road view displays the default map view of roads, buildings, and geography. To apply the Road view, you need to set BingMapStyle as `Road`, as shown in the following code.

XML

```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ImageryLayer LayerType="Bing" BingMapKey="<Your Bing map key>"
BingMapStyle="Road" />
</syncfusion:SfMap.Layers>
</syncfusion:SfMap >
```

C#

```
SfMap map = new SfMap();
ImageryLayer layer = new ImageryLayer();
layer.LayerType = LayerType.Bing;
layer.BingMapKey = "Your Bing map key";
layer.BingMapStyle = BingMapStyle.Road;
map.Layers.Add(layer);
this.Content = map;
```

The following screenshot illustrates the Road view.



AerialWithLabelView

AerialWithLabel view displays the Aerial map areas with labels for continent, country, ocean, etc., names. To apply this type of view style, you need to set BingMapStyle as **AerialWithLabel**, as shown in the following code.

XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ImageryLayer LayerType="Bing" BingMapKey="<Your Bing map key>"
      BingMapStyle="AerialWithLabels" />
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap map = new SfMap();
ImageryLayer layer = new ImageryLayer();
layer.LayerType = LayerType.Bing;
layer.BingMapKey = "Your Bing map key";
layer.BingMapStyle = BingMapStyle.AerialWithLabels;
map.Layers.Add(layer);
this.Content = map;
```

The following screenshot illustrates the AerialWithLabel view.



Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

See also

[How to show google map in WPF SfMap](#)

[How to view bing map using WPF SfMap](#)

[How to customize the markers in maps](#)

[How to drilldown map layers](#)

User interaction in WPF Maps (SfMap)

Tooltip

Tooltip provides additional information about the shapes in the maps. To enable tooltip, set the [ToolTipSettings](#) property in the [ShapeFileLayer](#) and also set the [ValuePath](#) property of [ToolTipSetting](#).

Tooltip is displayed by tapping the following elements:

- Shapes
- Bubbles
- Markers

Tooltip for shapes

XML

```
<maps:SfMap >
  <maps:SfMap.Layers>
    <maps:ShapeFileLayer Uri="/WpfUG;component/Assets/ShapeFiles/usa_state.shp"
      ItemsSource="{Binding Data}" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME" EnableSelection="False"
      LabelPath="State" >
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeStrokeThickness="1" ></maps:ShapeSetting>
      </maps:ShapeFileLayer.ShapeSettings>
    </maps:ShapeFileLayer>
  </maps:SfMap.Layers>
</maps:SfMap>
```

```

<maps:ShapeFileLayer.ToolTipSettings>
  <maps:ToolTipSetting ValuePath="Candidate" x:Name="shapeTooltipSettings" >
  </maps:ToolTipSetting>
</maps:ShapeFileLayer.ToolTipSettings>
<maps:ShapeFileLayer.ItemsTemplate>
  <DataTemplate>
    <Border >
      <TextBlock FontFamily="Segoe UI" FontSize="12" Foreground="#FF333333"
        Text=""/>
    </Border>
  </DataTemplate>
</maps:ShapeFileLayer.ItemsTemplate>
</maps:ShapeFileLayer>
</maps:SfMap.Layers>
</maps:SfMap>

```

C#

```

public class ElectionData
{
    public ElectionData(string state, string candidate, int electors)
    {
        State = state;
        Candidate = candidate;
        Electors = electors;
    }
    public string State
    {
        get;
        set;
    }
    public string Candidate
    {
        get;
        set;
    }
    public int Electors
    {
        get;
        set;
    }
}

public class ViewModel
{
    public ObservableCollection<ElectionData> Data { get; set; }
    public ViewModel()
    {
        Data = new ObservableCollection<ElectionData>();
        Data.Add(new ElectionData("Alabama", "Romney", 9));
        Data.Add(new ElectionData("Alaska", "Romney", 3));
        Data.Add(new ElectionData("Arizona", "Romney", 11));
        Data.Add(new ElectionData("Arkansas", "Romney", 6));
        Data.Add(new ElectionData("California", "Romney", 55));
        Data.Add(new ElectionData("Colorado", "Obama", 9));
        Data.Add(new ElectionData("Connecticut", "Obama", 7));
        Data.Add(new ElectionData("Delaware", "Obama", 3));
    }
}

```



```
Data.Add(new ElectionData("District of Columbia", "Obama", 3));  
Data.Add(new ElectionData("Florida", "Obama", 29));  
Data.Add(new ElectionData("Georgia", "Obama", 16));  
Data.Add(new ElectionData("Hawaii", "Romney", 4));  
Data.Add(new ElectionData("Idaho", "Obama", 4));  
Data.Add(new ElectionData("Illinois", "Romney", 20));  
Data.Add(new ElectionData("Indiana", "Obama", 11));  
Data.Add(new ElectionData("Iowa", "Romney", 6));  
Data.Add(new ElectionData("Kansas", "Obama", 6));  
Data.Add(new ElectionData("Kentucky", "Romney", 8));  
Data.Add(new ElectionData("Louisiana", "Romney", 8));  
Data.Add(new ElectionData("Maine", "Romney", 4));  
Data.Add(new ElectionData("Maryland", "Obama", 10));  
}  
}
```



Tooltip customization

The appearance of the tooltip can be customized using the following properties:

[Foreground](#) : Customizes the text color of tooltip.

[Background](#) : Customizes the background color of tooltip.

[Stroke](#) : Customizes the stroke color of tooltip.

[StrokeThickness](#) : Customizes the stroke width of tooltip.

[ShowDuration](#) : Specifies the duration of tooltip to be displayed.

[Margin](#) : Sets the margin for tooltip.

[FontFamily](#) : Customizes the text font family of tooltip.

[FontStyle](#) : Customizes the font style of tooltip text.

[FontSize](#) : Customizes the font size of tooltip text.

[PointerLength](#) : Customizes the tooltip pointer length.

The following code sample shows all the above customizations.

XML

```
<maps:ShapeFileLayer.ToolTipSettings>
<maps:ToolTipSetting ValuePath="State" PointerLength="18" FontFamily="Segoe
UI" FontStyle="Italic" FontSize="20" Foreground="White" Margin="10"
Background="Green" Stroke="Black" StrokeThickness="2" ShowDuration="2000"
/>
</maps:ShapeFileLayer.ToolTipSettings>
```

C#

```
shapeFileLayer.ToolTipSettings.Background = new
SolidColorBrush(Colors.White);
shapeFileLayer.ToolTipSettings.Background = new
SolidColorBrush(Colors.Green);
shapeFileLayer.ToolTipSettings.Stroke = new SolidColorBrush(Colors.Black);
shapeFileLayer.ToolTipSettings.StrokeThickness = 2;
shapeFileLayer.ToolTipSettings.Margin = new Thickness(10);
shapeFileLayer.ToolTipSettings.ShowDuration = 2000;
shapeFileLayer.ToolTipSettings.FontSize = 20;
shapeFileLayer.ToolTipSettings.FontStyle = FontStyles.Italic;
shapeFileLayer.ToolTipSettings.FontFamily = new FontFamily("Segoe UI");
shapeFileLayer.ToolTipSettings.PointerLength = 18;
```

Note: Similarly we can customize the bubble and marker tooltip also.



Custom template for tooltip

The maps control provides options to design your own template for tooltip using the `ToolTipTemplate` property.

XML

```

<Grid x:Name="grid">
<Grid.Resources>
<DataTemplate x:Key="toolTipTemplate">
<TextBlock Foreground="Yellow" Text="{Binding Data.Candidate}"/>
</DataTemplate>
</Grid.Resources>
<maps:ShapeFileLayer.ToolTipSettings>
<maps:ToolTipSetting ShowDuration="3000" ToolTipTemplate="{StaticResource
ResourceKey=toolTipTemplate}"/>
</maps:ShapeFileLayer.ToolTipSettings>
</Grid>

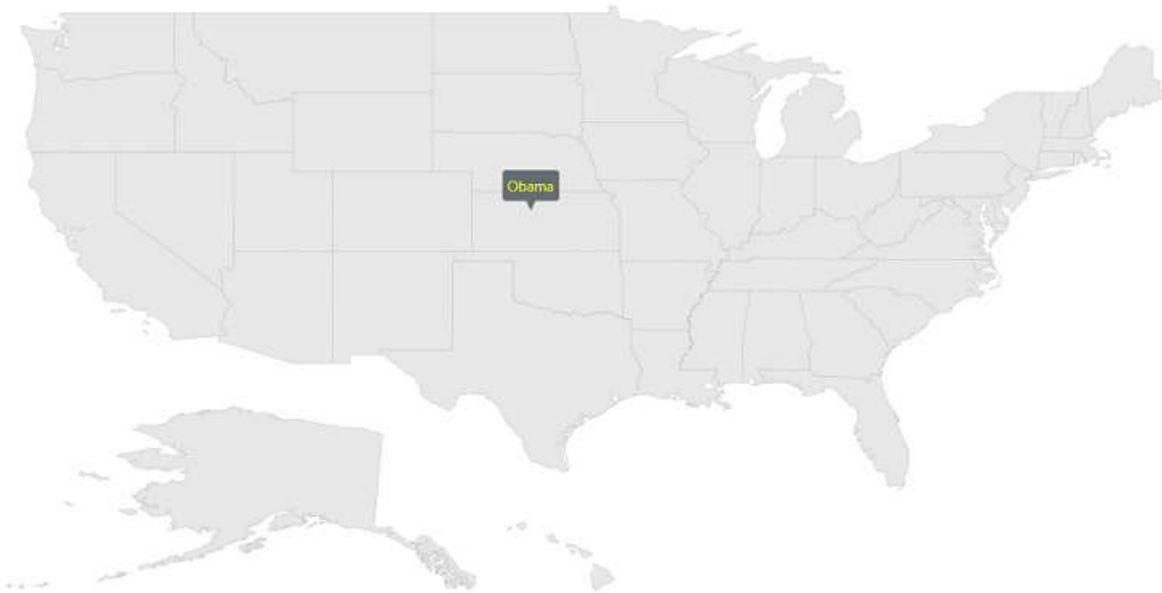
```

C#

```

ToolTipSetting toolTipSetting = new ToolTipSetting();
toolTipSetting.ShowDuration = 3000;
DataTemplate template = this.grid.Resources["toolTipTemplate"] as
DataTemplate;
toolTipSetting.ToolTipTemplate = template;
shapeFile.ToolTipSettings = toolTipSetting;

```



Tooltip for bubbles

XML

```

<maps:SfMap x:Name="map">
<maps:SfMap.Layers>

```

```

<maps:ShapeFileLayer Uri="MapsZoom.ShapeFiles.usa_state.shp"
ItemsSource="{Binding Data}" ShapeIDPath="State"
ShapeIDTableField="STATE_NAME" EnableSelection="False" LabelPath="State">
  <maps:ShapeFileLayer.ItemsTemplate>
    <DataTemplate>
      <Border >
        <TextBlock FontFamily="Segoe UI" FontSize="12" Foreground="#FF333333"
Text="" />
      </Border>
    </DataTemplate>
  </maps:ShapeFileLayer.ItemsTemplate>
  <maps:ShapeFileLayer.BubbleMarkerSetting>
    <maps:BubbleMarkerSetting MinSize="20" MaxSize="50" ValuePath="Electors"
ColorValuePath="Electors" Stroke="Black" StrokeThickness="3">
      <maps:BubbleMarkerSetting.ToolTipSettings >
        <maps:ToolTipSetting ValuePath="Electors">
        </maps:ToolTipSetting>
      </maps:BubbleMarkerSetting.ToolTipSettings>
      <maps:BubbleMarkerSetting.ColorMappings>
        <maps:RangeColorMapping Color="#7F20BCEE" To="0" From="10"/>
        <maps:RangeColorMapping Color="#7FA7CE38" To="10" From="20"/>
        <maps:RangeColorMapping Color="#7FF1B21A" To="20" From="30"/>
        <maps:RangeColorMapping Color="#7F1DA249" To="30" From="40"/>
        <maps:RangeColorMapping Color="#7FEB737C" To="40" From="50"/>
        <maps:RangeColorMapping Color="#7FED2D95" To="50" From="60"/>
      </maps:BubbleMarkerSetting.ColorMappings>
    </maps:BubbleMarkerSetting>
  </maps:ShapeFileLayer.BubbleMarkerSetting>
</maps:ShapeFileLayer>
</maps:SfMap.Layers>
</maps:SfMap>

```

C#

```

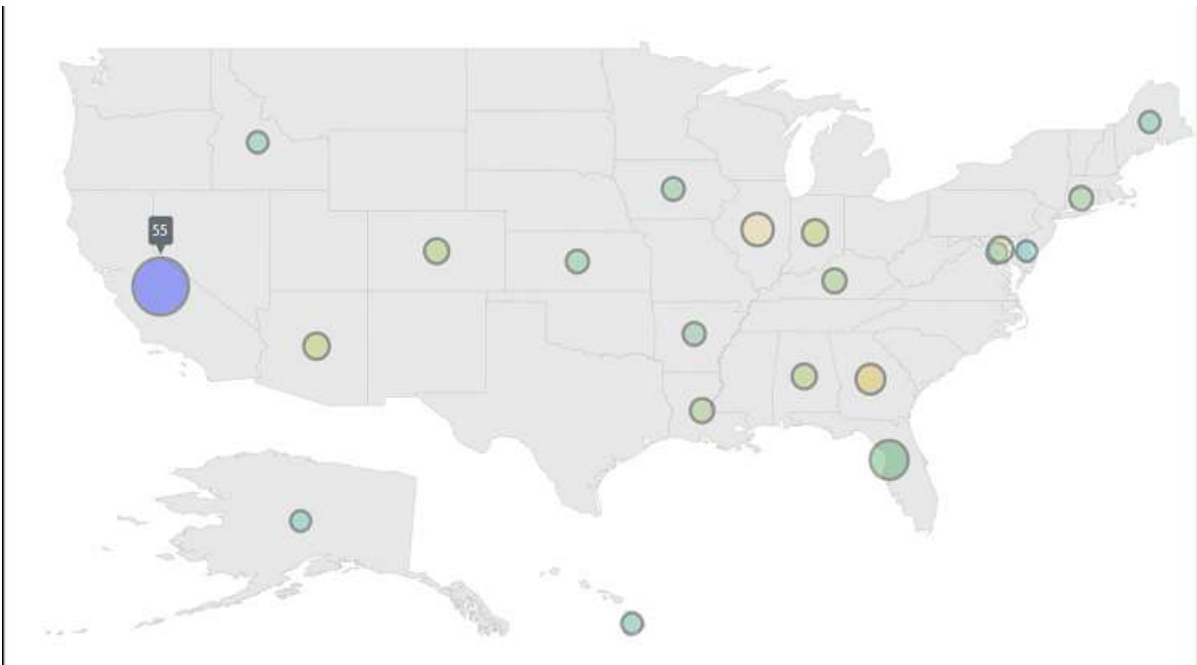
public class ElectionData
{
    public ElectionData(string state, string candidate, int electors)
    {
        State = state;
        Candidate = candidate;
        Electors = electors;
    }
    public string State
    {
        get;
        set;
    }
    public string Candidate
    {
        get;
        set;
    }
    public int Electors
    {
        get;
        set;
    }
}

```

```

    }
    }
    public class ViewModel
    {
        public ObservableCollection<ElectionData> Data { get; set; }
        public ViewModel()
        {
            Data = new ObservableCollection<ElectionData>();
            Data.Add(new ElectionData("Alabama", "Romney", 9));
            Data.Add(new ElectionData("Alaska", "Romney", 3));
            Data.Add(new ElectionData("Arizona", "Romney", 11));
            Data.Add(new ElectionData("Arkansas", "Romney", 6));
            Data.Add(new ElectionData("California", "Romney", 55));
            Data.Add(new ElectionData("Colorado", "Obama", 9));
            Data.Add(new ElectionData("Connecticut", "Obama", 7));
            Data.Add(new ElectionData("Delaware", "Obama", 3));
            Data.Add(new ElectionData("District of Columbia", "Obama", 3));
            Data.Add(new ElectionData("Florida", "Obama", 29));
            Data.Add(new ElectionData("Georgia", "Obama", 16));
            Data.Add(new ElectionData("Hawaii", "Romney", 4));
            Data.Add(new ElectionData("Idaho", "Obama", 4));
            Data.Add(new ElectionData("Illinois", "Romney", 20));
            Data.Add(new ElectionData("Indiana", "Obama", 11));
            Data.Add(new ElectionData("Iowa", "Romney", 6));
            Data.Add(new ElectionData("Kansas", "Obama", 6));
            Data.Add(new ElectionData("Kentucky", "Romney", 8));
            Data.Add(new ElectionData("Louisiana", "Romney", 8));
            Data.Add(new ElectionData("Maine", "Romney", 4));
            Data.Add(new ElectionData("Maryland", "Obama", 10));
        }
    }

```



Tooltip for markers

XML

```

<maps:SfMap>
  <maps:SfMap.Layers>
    <maps:ShapeFileLayer Uri="MapsZoom.ShapeFiles.usa_state.shp"
      Markers="{Binding Models}" >
      <maps:ShapeFileLayer.ItemsTemplate>
        <DataTemplate>
          <Border >
            <TextBlock FontFamily="Segoe UI" FontSize="12" Foreground="#FF333333"
              Text=""/>
            </Border>
          </DataTemplate>
        </maps:ShapeFileLayer.ItemsTemplate>
        <maps:ShapeFileLayer.MarkerToolTipSettings>
          <maps:ToolTipSetting ValuePath="Name"></maps:ToolTipSetting>
        </maps:ShapeFileLayer.MarkerToolTipSettings>
        <maps:ShapeFileLayer.MarkerTemplate>
          <DataTemplate>
            <Grid Margin="-12,-30,0,0">
              <Canvas>
                <Image Source="pin.png" Height="30"/>
              </Canvas>
            </Grid>
          </DataTemplate>
        </maps:ShapeFileLayer.MarkerTemplate>
      </maps:ShapeFileLayer>
    </maps:SfMap.Layers>
  </maps:SfMap>

```

C#

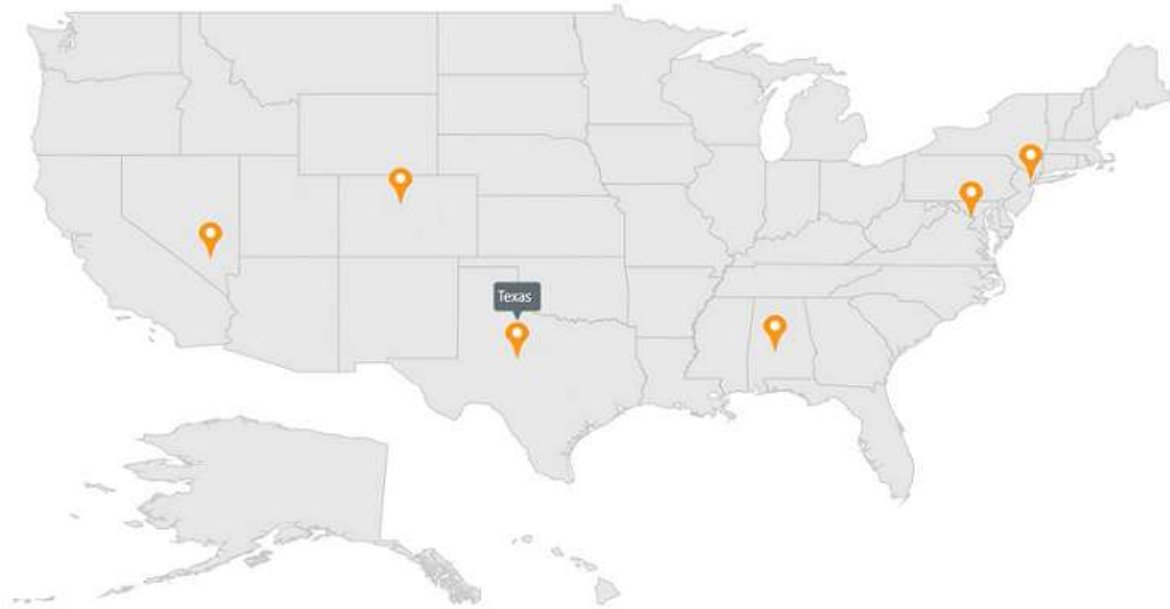
```

public class ViewModel
{
  public ObservableCollection<Model> Models { get; set; }
  public ViewModel()
  {
    this.Models = new ObservableCollection<Model>();
    this.Models.Add(new Model() { Name = "USA ", Latitude = "38.8833N",
      Longitude = "77.0167W" });
    this.Models.Add(new Model() { Name = "Texas ", Latitude = "31.9686N",
      Longitude = "99.9018W" });
    this.Models.Add(new Model() { Name = "Colorado ", Latitude = "39.5501N",
      Longitude = "105.7821W" });
    this.Models.Add(new Model() { Name = "New York ", Latitude = "40.7128N",
      Longitude = "74.0060W" });
    this.Models.Add(new Model() { Name = "Alabama ", Latitude = "32.3182N",
      Longitude = "86.9023W" });
    this.Models.Add(new Model() { Name = "Nevada ", Latitude = "36.8797N",
      Longitude = " 115.3626W" });
  }
}

public class Model
{
  public string Name { get; set; }

```

```
public string Longitude { get; set; }
public string Latitude { get; set; }
}
```



Tooltip for markers in imagery layer

XML

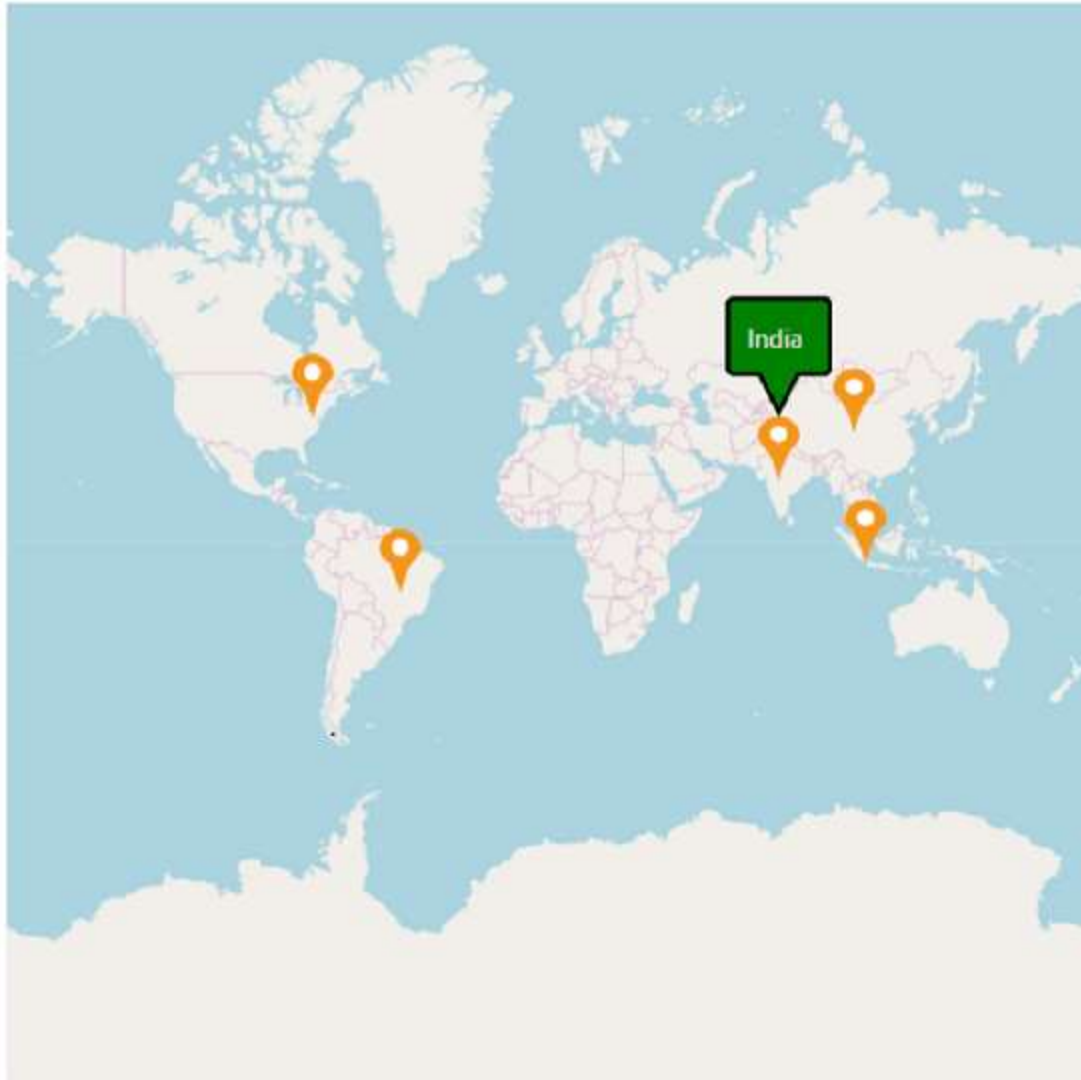
```
<maps:SfMap>
  <maps:SfMap.Layers>
    <maps:ImageryLayer LayerType="OSM" Markers="{Binding Models}" >
      <maps:ImageryLayer.MarkerToolTipSettings>
        <maps:ToolTipSetting ValuePath="Name" PointerLength="18" FontFamily="Segoe
          UI" FontStyle="Italic" FontSize="20" Foreground="White" Margin="10"
          Background="Green" Stroke="Black" StrokeThickness="2" ShowDuration="2000">
          <maps:ToolTipSetting.ToolTipTemplate>
            <DataTemplate>
              <TextBlock Foreground="White" Text="{Binding Value}"></TextBlock>
            </DataTemplate>
          </maps:ToolTipSetting.ToolTipTemplate>
        </maps:ToolTipSetting>
      </maps:ImageryLayer.MarkerToolTipSettings>
      <maps:ImageryLayer.MarkerTemplate>
        <DataTemplate>
          <Grid Margin="-12,-30,0,0">
            <Canvas>
              <Image Source="pin.png" Height="30"/>
            </Canvas>
          </Grid>
        </DataTemplate>
      </maps:ImageryLayer.MarkerTemplate>
    </maps:ImageryLayer>
  </maps:SfMap.Layers>
```

```
</maps:SfMap>
```

C#

```
public class ViewModel
{
    public ObservableCollection<Model> Models { get; set; }
    public ViewModel()
    {
        this.Models = new ObservableCollection<Model>();
        this.Models.Add(new Model() { Name = "USA ", Latitude = "38.8833N",
        Longitude = "77.0167W" });
        this.Models.Add(new Model() { Name = "Brazil ", Latitude = "15.7833S",
        Longitude = "47.8667W" });
        this.Models.Add(new Model() { Name = "India ", Latitude = "21.0000N",
        Longitude = "78.0000E" });
        this.Models.Add(new Model() { Name = "China ", Latitude = "35.0000N",
        Longitude = "103.0000E" });
        this.Models.Add(new Model() { Name = "Indonesia ", Latitude = "6.1750S",
        Longitude = "106.8283E" });
    }
}

public class Model
{
    public string Name { get; set; }
    public string Longitude { get; set; }
    public string Latitude { get; set; }
}
```

Zooming and panning

The Zooming and panning feature of the Maps control allows you to zoom in and out and navigate the map.

Zooming

The zooming feature enables you to zoom in and out of the map to show in-depth information. It is controlled by the [ZoomLevel](#) property of the map. When the zoom level of the Map control is increased, the map is zoomed in. When the zoom level is decreased, then the map is zoomed out.

Properties related to zooming

The following properties are related to the zooming feature of the Maps control:

1. ZoomLevel
2. EnableZoom
3. MinZoom
4. MaxZoom

ZoomLevel

[ZoomLevel](#) is the primary property of the zooming feature. It controls the map's scale size while zooming. Initially, the [ZoomLevel](#) is set to 1. zoom level cannot be less than 1.

EnableZoom

The [EnableZoom](#) property enables or disables the zooming feature. A [True](#) value for this property enables the zooming feature, while [False](#) disables the zooming feature.

MinZoom

The [MinZoom](#) property is used to set the minimum zoom level of the map.

MaxZoom

The [MaxZoom](#) property is used to set the maximum zoom level of the Map control.

Sample code for setting zooming feature properties:

XML

```
<syncfusion:SfMap ZoomLevel="3" MinZoom="1" MaxZoom="20" EnableZoom="True">
</syncfusion:SfMap >
```

C#

```
SfMap maps = new SfMap();
maps.ZoomLevel = 3;
maps.MinZoom = 1;
maps.MaxZoom = 20;
maps.EnableZoom = true;
this.Content = maps;
```

Zooming in ShapeFileLayer

Methods to zoom the map

Maps can be zoomed by using the following methods:

1. By changing the ZoomLevel.
2. Using the Zoom method.
3. Using the mouse scroll.

Changing the ZoomLevel

A map can be zoomed by changing the zoom level of the Map control. Incrementing the ZoomLevel, zooms in the map, while decrementing the ZoomLevel, zooms it out.

Using the Zoom method

Maps can be zoomed using the [Zoom](#) method. The Zoom method has the parameter zoom value. The map can be zoomed or scaled with the zoom value parameter.

C#

```
SfMap syncMap = new SfMap();
ShapeFileLayer shapeLayer = new ShapeFileLayer();
shapeLayer.Uri = "MapApp.ShapeFiles.world1.shp";
syncMap.Layers.Add(shapeLayer);
syncMap.Zoom(5);
```

Using a mouse wheel event

In addition to the pinching event, the map can be zoomed using the mouse events. The map is zoomed in when the mouse is scrolled up. The map zooms out when the mouse is scrolled down.



Panning the map

The panning feature enables navigation using the map.

Enable and disable pan

The [EnablePan](#) property enables or disables the panning feature of the map. A `True` value enables the panning feature. A `False` value disables the panning feature of the map.

XML

```
<syncfusion:SfMap ShowCoords="True" LatitudeLongitudeType="Decimal"
EnablePan="True">
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer Uri="MapApp.world1.shp">
    </syncfusion:ShapeFileLayer>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap >
```

C#

```
SfMap map = new SfMap();
map.ShowCoords = true;
map.LatitudeLongitudeType = LatLonType.Decimal;
map.EnablePan = true;
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "MapApp.world1.shp";
map.Layers.Add(shapeFileLayer);
this.Content = map;
```

Ways to pan the map

There are two methods for panning the map. They are:

1. Using the Pan method.
2. By dragging the map.

Through the Pan method

The map can be panned with the [Pan](#) method in the Maps control. The Pan method has two parameters: x and y. The map is translated with respect to the x and y parameters.

Code sample for the Pan method

C#

```
SfMap syncMap = new SfMap();
syncMap.EnablePan = true;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "App2.world1.shp";
syncMap.Layers.Add(layer);
syncMap.Pan(200, 200);
```

Dragging the map

The map can be panned by dragging the map using the mouse interactions.

Note: The map can be panned only when some parts of the map are outside the view of the control.



Zooming in ImageryLayer

Calculate a zoom level

This feature allows you to automatically set the initial zoom level automatically in two ways:

- Distance Radius(KM/miles)
- Geo-bounds(Northeast, Southwest)

Distance radius

Note: [DistanceType](#) default value is KiloMeter.

Calculate the initial zoom level automatically based on the [Radius](#) and [DistanceType](#) properties of ImageryLayer.

- [Center](#) - Represents center point of ImageryLayer.

XML

```
<Window.Resources>
<ResourceDictionary >
<DataTemplate x:Key="markerTemplate">
<Grid Margin="-12,-30,0,0">
<Canvas>
<Image Source="pin.png" Height="30"/>
</Canvas>
</Grid>
</DataTemplate>
</ResourceDictionary>
</Window.Resources>
<Grid>
<maps:SfMap>
<maps:SfMap.Layers>
<maps:ImageryLayer MarkerTemplate="{StaticResource
ResourceKey=markerTemplate}" Markers="{Binding Models}" Center="38.909804, -
77.043442" Radius="5" DistanceType="KiloMeter" >
</maps:ImageryLayer>
</maps:SfMap.Layers>
</maps:SfMap>
</Grid>
```

C#

```
SfMap maps = new SfMap();
ImageryLayer layer = new ImageryLayer();
layer.Center = new Point(38.909804, -77.043442);
layer.Radius = 5;
layer.DistanceType = DistanceType.KiloMeter;
layer.Markers = obj.Models;
layer.MarkerTemplate = this.Resources["markerTemplate"] as DataTemplate;
maps.Layers.Add(layer);
public class Model
{
public string Longitude { get; set; }
public string Latitude { get; set; }
}
public class ViewModel
{
public ObservableCollection<Model> Models { get; set; }
public ViewModel()
{
this.Models = new ObservableCollection<Model>();
this.Models.Add(new Model() { Latitude = "38.909804", Longitude = "-
77.043442" });
}
}
```

Geo-bounds

Calculate the initial zoom level automatically based on the [LatLngBounds](#) of ImageryLayer.

`LatLngBounds` has the below properties.

[Northeast](#) - Gets or sets the northeast corner of the geo bounds.

[Southwest](#) - Gets or sets the southwest corner of the geo bounds.

XML

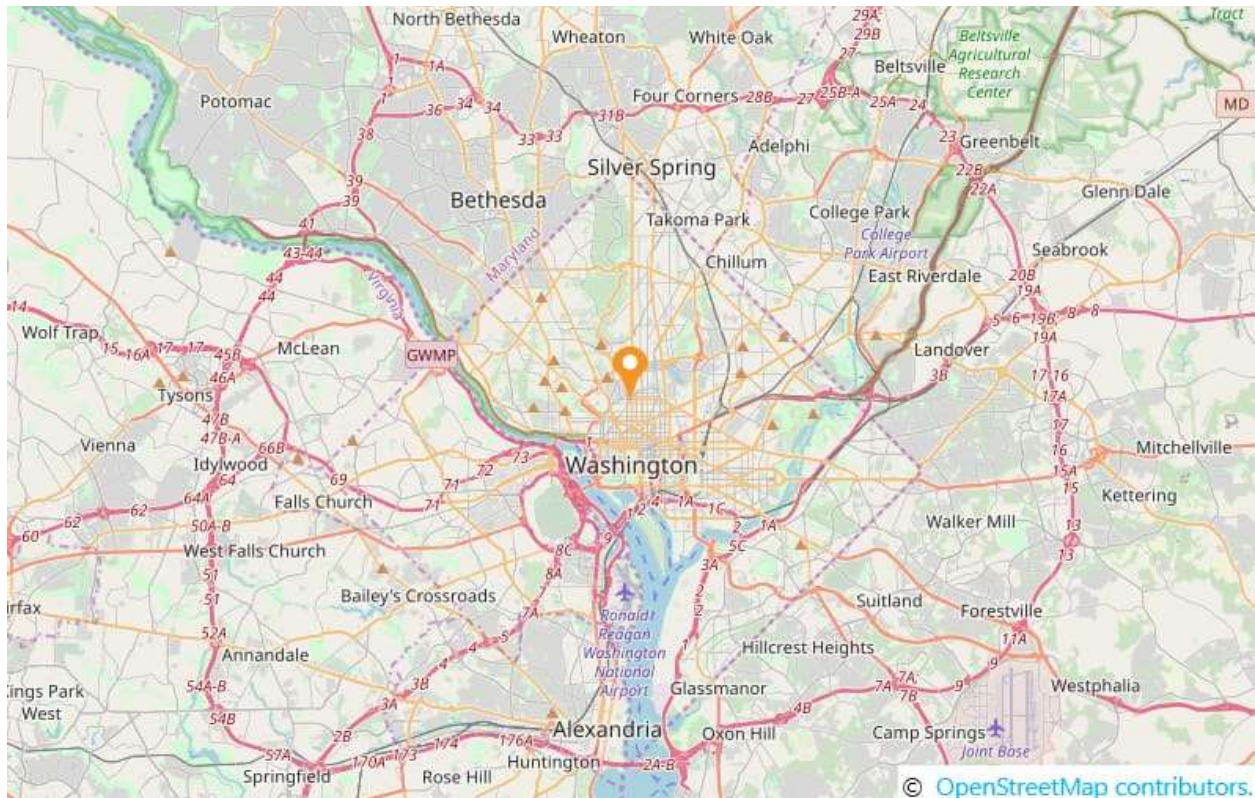
```
<Window.Resources>
<ResourceDictionary >
<DataTemplate x:Key="markerTemplate">
<Grid Margin="-12,-30,0,0">
<Canvas>
<Image Source="pin.png" Height="30"/>
</Canvas>
</Grid>
</DataTemplate>
</ResourceDictionary>
</Window.Resources>
<Grid>
<maps:SfMap>
<maps:SfMap.Layers>
<maps:ImageryLayer LayerType="OSM" MarkerTemplate="{StaticResource
ResourceKey=markerTemplate}" Markers="{Binding Models}" >
<maps:ImageryLayer.LatLngBounds>
<maps:LatLngBounds Northeast="38.909804, -77.043442" Southwest="38.909804, -
77.043442" >
</maps:LatLngBounds>
</maps:ImageryLayer.LatLngBounds>
</maps:ImageryLayer>
</maps:SfMap.Layers>
</maps:SfMap>
</Grid>
```

C#

```
SfMap maps = new SfMap();
ImageryLayer layer = new ImageryLayer();
LatLngBounds bounds = new LatLngBounds();
bounds.Northeast = new Point(38.909804, -77.043442);
bounds.Southwest = new Point(38.909804, -77.043442);
layer.LatLngBounds = bounds;
layer.Markers = obj.Models;
layer.MarkerTemplate = this.Resources["markerTemplate"] as DataTemplate;
maps.Layers.Add(layer);
public class Model
{
public string Longitude { get; set; }
public string Latitude { get; set; }
}
public class ViewModel
{
public ObservableCollection<Model> Models { get; set; }
public ViewModel()
```

```
{
    this.Models = new ObservableCollection<Model>();
    this.Models.Add(new Model() { Latitude = "38.909804", Longitude = "-77.043442" });
}
}
```

Note: When setting the [LatLngBounds](#) and [DistanceType](#) at the same time, the priority is distance radius, and calculate zoom level based on [Radius](#) value.



Calculate the map tile layer bounds

Calculate the imagery layer pixel bounds while zooming, panning, and changing the Geo-Coordinate value.

XML

```
<maps:SfMap>
  <maps:SfMap.Layers>
    <maps:ImageryLayer x:Name="layer" Center="30.9709225, -100.2187212"
      CenterChanged="layer_CenterChanged">
    </maps:ImageryLayer>
  </maps:SfMap.Layers>
</maps:SfMap>
```

C#

```
public partial class MapBound : ContentPage
{
    ImageryLayer layer = new ImageryLayer();
}
```



```

public MapBound()
{
    InitializeComponent();
    SfMap maps = new SfMap();
    layer.Center = new Point(30.9709225, -100.2187212);
    layer.CenterChanged += layer_CenterChanged;
    maps.Layers.Add(layer);
    this.Content = maps;
}
private void layer_CenterChanged(object sender, CenterChangedEventArgs e)
{
    var pixelbounds = layer.MapBounds;
}
}

```

Pinch zooming in ImageryLayer

If you want to zoom the imagery layer with your fingers, you must enable the [EnableZoom](#) and `IsManipulationEnabled` property of map control.

XML

```

<Window.Resources>
<ResourceDictionary>
<DataTemplate x:Key="markerTemplate">
<Grid>
<Canvas Margin="-12,-30,0,0">
<Image Source="pin.png" Height="30" />
<TextBlock HorizontalAlignment="Center" Margin="0,30,0,0" FontSize="20"
FontFamily="Segoe UI" Text="{Binding Label}"/>
</Canvas>
</Grid>
</DataTemplate>
</ResourceDictionary>
</Window.Resources>
<Grid>
<Grid>
<syncfusion:SfMap ZoomLevel="3" IsManipulationEnabled="True"
EnableZoom="True">
<syncfusion:SfMap.Layers>
<syncfusion:ImageryLayer Markers="{Binding Models}"
MarkerTemplate="{StaticResource markerTemplate}">
</syncfusion:ImageryLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>

```

C#

```

SfMap maps = new SfMap();
ImageryLayer layer = new ImageryLayer();
layer.Markers = view.Models;
maps.ZoomLevel = 3;
layer.MarkerTemplate = Resources["markerTemplate"] as DataTemplate;
maps.IsManipulationEnabled = true;
maps.EnableZoom = true;

```



```

maps.Layers.Add(layer);
this.Content = maps;
public class ViewModel
{
    public ObservableCollection<Model> Models { get; set; }
    public ViewModel()
    {
        this.Models = new ObservableCollection<Model>();
        this.Models.Add(new Model() { Label = "USA", Latitude = "38.8833N",
        Longitude = "77.0167W" });
        this.Models.Add(new Model() { Label = "Brazil ", Latitude = "15.7833S",
        Longitude = "47.8667W" });
        this.Models.Add(new Model() { Label = "India ", Latitude = "21.0000N",
        Longitude = "78.0000E" });
        this.Models.Add(new Model() { Label = "China ", Latitude = "35.0000N",
        Longitude = "103.0000E" });
        this.Models.Add(new Model() { Label = "Indonesia ", Latitude = "6.1750S",
        Longitude = "106.8283E" });
    }
}
public class Model
{
    public string Label { get; set; }
    public string Longitude { get; set; }
    public string Latitude { get; set; }
}

```



You can cancel the pinch zooming in ImageryLayer by setting the [Cancel](#) property as true in [ZoomLevelChanging](#) event argument, as shown in the following code sample.

XML

```

<syncfusion:SfMap EnableZoom="True" IsManipulationEnabled="True">
    <syncfusion:SfMap.Layers>
        <syncfusion:ImageryLayer Markers="{Binding Models}"

```

```
ZoomLevelChanging="ImageryLayer_ZoomLevelChanging"/>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
private void ImageryLayer_ZoomLevelChanging(object sender,
ZoomLevelChangingEventArgs e)
{
    e.Cancel = true;
}
```

Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

Markers in WPF Maps (SfMap)

Markers are used to show some messages on maps.

Adding the marker

Any number of markers can be added to the shape file layer or the imagery layer using the [Markers](#) property. Each marker contains the following properties:

Note: You must create a model that contains properties such as Latitude and Longitude to add a marker in maps. If you want to add text with default marker, add Label property with your model.

Label: Displays some messages on maps.

Latitude: Specifies y-axis position of the marker.

Longitude: Specifies x-axis position of the marker.

XML

```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="Maps.ShapeFiles.world1.shp"
Markers="{Binding Models}" >
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        ViewModel view = new ViewModel();
        this.DataContext = view;
        SfMap maps = new SfMap();
        ShapeFileLayer shape = new ShapeFileLayer();
        shape.Uri = "Maps.ShapeFiles.world1.shp";
        shape.Markers = view.Models;
        maps.Layers.Add(shape);
    }
}
```

```
this.Content = maps;
}
}
public class ViewModel
{
    public ObservableCollection<Model> Models { get; set; }
    public ViewModel()
    {
        this.Models = new ObservableCollection<Model>();
        this.Models.Add(new Model() { Label = "USA", Latitude = "38.8833N",
        Longitude = "77.0167W" });
        this.Models.Add(new Model() { Label = "Brazil ", Latitude = "15.7833S",
        Longitude = "47.8667W" });
        this.Models.Add(new Model() { Label = "India ", Latitude = "21.0000N",
        Longitude = "78.0000E" });
        this.Models.Add(new Model() { Label = "China ", Latitude = "35.0000N",
        Longitude = "103.0000E" });
        this.Models.Add(new Model() { Label = "Indonesia ", Latitude = "6.1750S",
        Longitude = "106.8283E" });
    }
}
public class Model
{
    public string Label { get; set; }
    public string Longitude { get; set; }
    public string Latitude { get; set; }
}
```



Add a custom marker

The Maps control provides the support for defining the custom markers using the [MarkerTemplate](#) property.

XML

```

<Window.Resources>
<ResourceDictionary>
<DataTemplate x:Key="markerTemplate">
<Grid>
<Canvas Margin="-12,-30,0,0">
<Image Source="pin.png" Height="30" />
<TextBlock HorizontalAlignment="Center" Margin="0,30,0,0" FontSize="30"
FontFamily="Segoe UI" Text="{Binding Label}"/>
</Canvas>
</Grid>
</DataTemplate>
</ResourceDictionary>
</Window.Resources>
<Grid>
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="Maps.ShapeFiles.world1.shp"
Markers="{Binding Models}" MarkerTemplate="{StaticResource
markerTemplate}">
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>

```

C#

```

SfMap maps = new SfMap();
ShapeFileLayer shape = new ShapeFileLayer();
shape.Uri = "Maps.ShapeFiles.world1.shp";
shape.MarkerTemplate=Resources["markerTemplate"] as DataTemplate;
shape.Markers = view.Models;
maps.Layers.Add(shape);
this.Content = maps;

```



Customizing marker icons

The size and color of marker icons can be customized using the [MarkerIconSize](#) and [MarkerIconFill](#) properties.

Icon types

The shape of a marker icons can be customized using the [MarkerIconType](#) property. The maps control supports the following types of marker icons:

- Circle
- Diamond
- Image
- Rectangle
- Square

XML

```
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="Maps.ShapeFiles.world1.shp"
MarkerIconType="Diamond" MarkerIconSize="30,20" MarkerIconFill="Green"
Markers="{Binding Models}" >
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap maps = new SfMap();
ShapeFileLayer shape = new ShapeFileLayer();
shape.Uri = "Maps.ShapeFiles.world1.shp";
shape.Markers = view.Models;
shape.MarkerIconType = MarkerIcon.Diamond;
shape.MarkerIconFill = new SolidColorBrush(Colors.Green);
shape.MarkerIconSize = new Size(30, 20);
maps.Layers.Add(shape);
this.Content = maps;
```



Setting an image marker icon

You can set the image as marker icon by setting the icon type as an Image and set [MarkerIconSource](#) to get the image from local path.

XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer Uri="Maps.ShapeFiles.world1.shp"
      MarkerIconType="Image" MarkerIconSource="pin.png" MarkerIconSize="30,30"
      Markers="{Binding Models}" >
    </syncfusion:ShapeFileLayer>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap maps = new SfMap();
ShapeFileLayer shape = new ShapeFileLayer();
shape.Uri = "Maps.ShapeFiles.world1.shp";
shape.MarkerIconSize = new Size(30, 30);
shape.Markers = view.Models;
shape.MarkerIconType = MarkerIcon.Image;
BitmapImage bimage = new BitmapImage();
bimage.BeginInit();
bimage.UriSource = new Uri("../..\\pin.png", UriKind.Relative);
bimage.EndInit();
shape.MarkerIconSource = bimage;
maps.Layers.Add(shape);
this.Content = maps;
```



Customizing labels

You can customize the marker labels using the [MarkerLabelFontSize](#), [MarkerLabelForeground](#), [MarkerLabelFontStyle](#), and [MarkerLabelFontWeight](#) and [MarkerLabelFontFamily](#) properties.

XML

```
<syncfusion:SfMap>
  <syncfusion:SfMap.Layers>
    <syncfusion:ShapeFileLayer Uri="Maps.ShapeFiles.world1.shp"
      MarkerLabelFontSize="30" MarkerLabelForeground="Red"
      MarkerLabelFontStyle="Italic" MarkerLabelFontWeight="Bold"
      MarkerLabelFontFamily="Segoe UI" Markers="{Binding Models}" >
    </syncfusion:ShapeFileLayer>
  </syncfusion:SfMap.Layers>
</syncfusion:SfMap>
```

C#

```
SfMap maps = new SfMap();
ShapeFileLayer shape = new ShapeFileLayer();
shape.Uri = "Maps.ShapeFiles.world1.shp";
shape.Markers = view.Models;
shape.MarkerLabelFontSize = 30;
shape.MarkerLabelForeground = new SolidColorBrush(Colors.Red);
shape.MarkerLabelFontStyle = FontStyles.Italic;
shape.MarkerLabelFontWeight = FontWeights.Bold;
shape.MarkerLabelFontFamily = new FontFamily("Segoe UI");
maps.Layers.Add(shape);
```



Marker Alignment

You can align the maps marker horizontally and vertically using the [MarkerHorizontalAlignment](#) and [MarkerVerticalAlignment](#) properties.

Setting a horizontal alignment

The [MarkerHorizontalAlignment](#) property is used to position the marker icon horizontally. The marker icon can be positioned using the following ways:

- **Near:** Specifies the near position of the marker icon for the given latitude and longitude values.
- **Center:** Specifies the center position of the marker icon for the given latitude and longitude values.
- **Far:** Specifies the far position of the marker icon for the given latitude and longitude values.

XML

```
<ResourceDictionary>
<DataTemplate x:Key="markerTemplate">
<Grid>
<StackPanel Margin="-12,-20,0,0" Height="60" Width="100"
Orientation="Horizontal" Background="Transparent">
<Image Source="pin.png" Height="60" />
</StackPanel>
</Grid>
</DataTemplate>
</ResourceDictionary>
<Grid>
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer MarkerTemplate="{StaticResource
markerTemplate}" Uri="Maps.ShapeFiles.usa_state.shp"
MarkerHorizontalAlignment="Near" Markers="{Binding Models}" >
<syncfusion:ShapeFileLayer.ShapeSettings>
```



```
<syncfusion:ShapeSetting ShapeFill="LightGray" ShapeStroke="Black"
ShapeStrokeThickness="1">
</syncfusion:ShapeSetting>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>
```

C#

```
SfMap maps = new SfMap();
ShapeFileLayer shape = new ShapeFileLayer();
shape.Uri = "Maps.ShapeFiles.usa_state.shp";
shape.MarkerTemplate = Resources["markerTemplate"] as DataTemplate;
shape.Markers = view.Models;
shape.MarkerHorizontalAlignment = MarkerAlignment.Near;
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeStrokeThickness = 1;
shapeSetting.ShapeStroke = new SolidColorBrush(Colors.Black);
shapeSetting.ShapeFill = new SolidColorBrush(Colors.LightGray);
shape.ShapeSettings = shapeSetting;
maps.Layers.Add(shape);
```



Setting a vertical alignment

The [MarkerVerticalAlignment](#) property is used to position the marker icon vertically. The marker icon can be positioned using the following ways:

- **Near**: Specifies the near position of the marker icon for the given latitude and longitude values.

- **Center**: Specifies the center position of the marker icon for the given latitude and longitude values.
- **Far**: Specifies the far position of the marker icon for the given latitude and longitude values.

XML

```
<ResourceDictionary>
<DataTemplate x:Key="markerTemplate">
<Grid>
<StackPanel Margin="-12,-20,0,0" Height="60" Width="100"
Orientation="Horizontal" Background="Transparent">
<Image Source="pin.png" Height="60" />
</StackPanel>
</Grid>
</DataTemplate>
</ResourceDictionary>
<Grid>
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer MarkerTemplate="{StaticResource
markerTemplate}" Uri="Maps.ShapeFiles.usa_state.shp"
MarkerVerticalAlignment="Near" Markers="{Binding Models}" >
<syncfusion:ShapeFileLayer.ShapeSettings>
<syncfusion:ShapeSetting ShapeFill="LightGray" ShapeStroke="Black"
ShapeStrokeThickness="1">
</syncfusion:ShapeSetting>
</syncfusion:ShapeFileLayer.ShapeSettings>
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>
```

C#

```
SfMap maps = new SfMap();
ShapeFileLayer shape = new ShapeFileLayer();
shape.Uri = "Maps.ShapeFiles.usa_state.shp";
shape.MarkerTemplate = Resources["markerTemplate"] as DataTemplate;
shape.Markers = view.Models;
shape.MarkerVerticalAlignment = MarkerAlignment.Near;
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeStrokeThickness = 1;
shapeSetting.ShapeStroke = new SolidColorBrush(Colors.Black);
shapeSetting.ShapeFill = new SolidColorBrush(Colors.LightGray);
shape.ShapeSettings = shapeSetting;
maps.Layers.Add(shape);
```



Note: The default marker icon position for VerticalAlignment and HorizontalAlignment is Center.

Selection Mode

If you add any view for marker using the [MarkerTemplate](#) property from [MarkerSelected](#) event, then the corresponding view will be applied to the selected marker. Custom view will be added continuously for all the selected marker, but do not have option to reset the old one. Now, you can achieve this using the [MarkerSelectionMode](#) property. If set selection mode as **Single**, then it will be removed the old view of marker and load the initially rendered marker. If set selection mode as **Multiple**, then the marker template does not reset.

XML

```
<Window.Resources>
  <ResourceDictionary>
    <DataTemplate x:Key="markerTemplate">
      <Grid>
        <StackPanel Margin="-12,-30,0,0" Height="50" Orientation="Horizontal">
          <Image Source="pin.png" Height="30" />
          <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
            Margin="10" FontSize="30" FontFamily="Segoe UI" Text="{Binding Label}"/>
        </StackPanel>
      </Grid>
    </DataTemplate>
  </ResourceDictionary>
</Window.Resources>
<Grid>
  <syncfusion:SfMap>
    <syncfusion:SfMap.Layers>
      <syncfusion:ShapeFileLayer Uri="Maps.ShapeFiles.world1.shp"
        MarkerSelected="ShapeFileLayer_MarkerSelected" MarkerSelectionMode="Single"
        Markers="{Binding Models}" />
    </syncfusion:ShapeFileLayer>
  </syncfusion:SfMap>
</Grid>
```

```

</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>

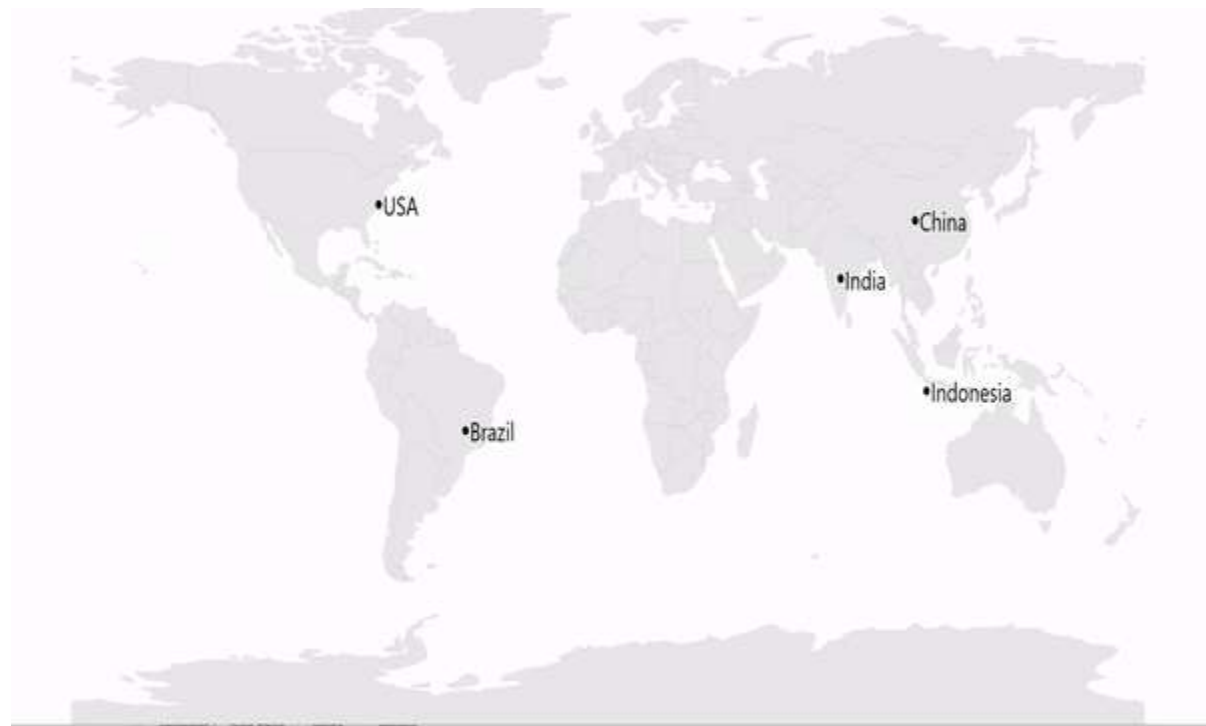
```

C#

```

SfMap maps = new SfMap();
ShapeFileLayer shape = new ShapeFileLayer();
shape.Uri = "Maps.ShapeFiles.world1.shp";
shape.Markers = view.Models;
shape.MarkerSelected += ShapeFileLayer_MarkerSelected;
maps.Layers.Add(shape);
this.Content = maps;
private void ShapeFileLayer_MarkerSelected(object sender,
MarkerSelectedEventArgs e)
{
    e.MarkerTemplate = Resources["markerTemplate"] as DataTemplate;
}

```



Events

The [MarkerSelected](#) event is fired when a marker is selected. The [MarkerTemplate](#), [SelectedMarker](#), and [IsSelected](#) will be passed to [MarkerSelectedEventArgs](#).

If you set any view for the [MarkerTemplate](#) property of [MarkerSelectedEventArgs](#), the corresponding view will be applied to the selected marker.

[SelectedMarker](#): Contains selected marker data.

[IsSelected](#): Used to determine whether a marker is selected or unselected.

[CanBringToTop](#) : Used to bring the selected marker to the top position.

XML

```

<Window.Resources>
<ResourceDictionary>
<DataTemplate x:Key="markerTemplate">
<Grid>
<StackPanel Margin="-12,-30,0,0" Height="50" Orientation="Horizontal">
<Image Source="pin.png" Height="30" />
<TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
Margin="10" FontSize="30" FontFamily="Segoe UI" Text="{Binding Label}"/>
</StackPanel>
</Grid>
</DataTemplate>
</ResourceDictionary>
</Window.Resources>
<Grid>
<syncfusion:SfMap>
<syncfusion:SfMap.Layers>
<syncfusion:ShapeFileLayer Uri="Maps.ShapeFiles.world1.shp"
MarkerSelected="ShapeFileLayer_MarkerSelected" Markers="{Binding Models}" >
</syncfusion:ShapeFileLayer>
</syncfusion:SfMap.Layers>
</syncfusion:SfMap>
</Grid>

```

C#

```

SfMap maps = new SfMap();
ShapeFileLayer shape = new ShapeFileLayer();
shape.Uri = "Maps.ShapeFiles.world1.shp";
shape.Markers = view.Models;
shape.MarkerSelected += ShapeFileLayer_MarkerSelected;
maps.Layers.Add(shape);
this.Content = maps;
private void ShapeFileLayer_MarkerSelected(object sender,
MarkerSelectedEventArgs e)
{
e.MarkerTemplate = Resources["markerTemplate"] as DataTemplate;
object selectedMarker = e.SelectedMarker;
bool MarkerSelected = e.IsSelected;
}

```

![Marker Selected Event Image](Marker_images/Marker Selected Event.png)

Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

See also

[How to customize the markers in maps](#)

Commands in WPF Maps (SfMap)

The [WPF Map](#) control contains the following commands:

- ZoomIn Command
- ZoomOut Command

- Pan Command
- ZoomReset Command
- PanReset Command

ZoomIn Command

[ZoomIn](#) command zooms in the map.

ZoomOut Command

[ZoomOut](#) command zooms out the map.

Pan Command

[PanCommand](#) is used to navigate on the Map control. The direction of the navigation is the Command parameter for the Pan Command.

ZoomReset Command

The [ZoomResetCommand](#) resets the ZoomLevel value of the Map control.

PanReset Command

The [PanResetCommand](#) resets the map at its initial position; it resets the pan values.

XML

```
<Grid Margin="10">
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition Height="Auto"/>
  </Grid.RowDefinitions>
  <syncfusion:SfMap x:Name="Map">
    <syncfusion:SfMap.Layers>
      <syncfusion:ShapeFileLayer x:Name="shapeControl"
        Uri="DataMarkers.ShapeFiles.world1.shp">
      </syncfusion:ShapeFileLayer>
    </syncfusion:SfMap.Layers>
  </syncfusion:SfMap >
  <StackPanel Grid.Row="1" Orientation="Horizontal" Margin="10">
    <StackPanel.Resources>
      <Style TargetType="Button">
        <Setter Property="Margin" Value="5"/>
        <Setter Property="Padding" Value="5"/>
      </Style>
    </StackPanel.Resources>
    <Button Content="ZoomIn" Command="{Binding
      ElementName=Map, Path=ZoomInCommand}" />
    <Button Content="ZoomOut" Command="{Binding
      ElementName=Map, Path=ZoomOutCommand}" />
    <!--Panning commands-->
    <Button Content="LeftPan" Command="{Binding
      ElementName=Map, Path=PanCommand}" CommandParameter="left"/>
    <Button Content="RightPan" Command="{Binding
      ElementName=Map, Path=PanCommand}" CommandParameter="right"/>
    <Button Content="TopPan" Command="{Binding ElementName=Map, Path=PanCommand}"
      CommandParameter="top"/>
    <Button Content="BottomPan" Command="{Binding
      ElementName=Map, Path=PanCommand}" CommandParameter="bottom"/>
    <Button Content="ZoomReset" Command="{Binding
      ElementName=Map, Path=ZoomResetCommand}" />
  </StackPanel>
</Grid>
```

```
<Button Content="PanReset" Command="{Binding
ElementName=Map, Path=PanResetCommand}" />
</StackPanel>
</Grid>
```

Note: You can also explore our [WPF Map example](#) to know how to render and configure the map.

Events in WPF Maps (SfMap)

- [ZoomedIn](#) - Occurs whenever zooming the map.
- [ZoomedOut](#) - Occurs when zoomed out the map.
- [Panning](#) - Occurs while panning the map.
- [Panned](#) - Occurs after panned the map.
- [MapToolTipOpening](#) - Occurs when any tooltip on the SfMap control is opened.

Tooltip opening event

[MapToolTipOpening](#) event occurs whenever you select a shape, bubble, or marker. You will get the **Data** and **ToolTipType** properties as arguments from **ToolTipOpeningEventArgs** handler, and you can cancel the event for a particular shape using the **Cancel** property.

XML

```
<maps:SfMap MapToolTipOpening="Map_MapToolTipOpening" >
  <maps:SfMap.Layers>
    <maps:ShapeFileLayer Uri="MapsZoom.ShapeFiles.usa_state.shp"
      ItemsSource="{Binding Data}"
      ShapeIDPath="State" ShapeIDTableField="STATE_NAME"
      EnableSelection="False" LabelPath="State">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeStrokeThickness="1" ></maps:ShapeSetting>
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.ToolTipSettings>
        <maps:ToolTipSetting ValuePath="Candidate" x:Name="shapeToolTipSettings" >
        </maps:ToolTipSetting>
      </maps:ShapeFileLayer.ToolTipSettings>
    </maps:ShapeFileLayer>
  </maps:SfMap.Layers>
</maps:SfMap>
```

C#

```
private void SfMaps_TooltipOpening(object sender, ToolTipOpeningEventArgs e)
{
    if ((e.Data is MapsZoom.ElectionData) && (e.Data as
MapsZoom.ElectionData).State == "North Dakota")
    {
        e.Cancel = true;
    }
}
```

Note: You can refer to our [WPF Map](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Map example](#) to know how to render and configure the map.

How to

Transform latitude and longitude value to pixel value and vice-versa

SfMap offers two utility methods to transform the pixel values to longitude and latitude values and vice-versa. This method is used for both ShapeFileLayer and ImageryLayer.

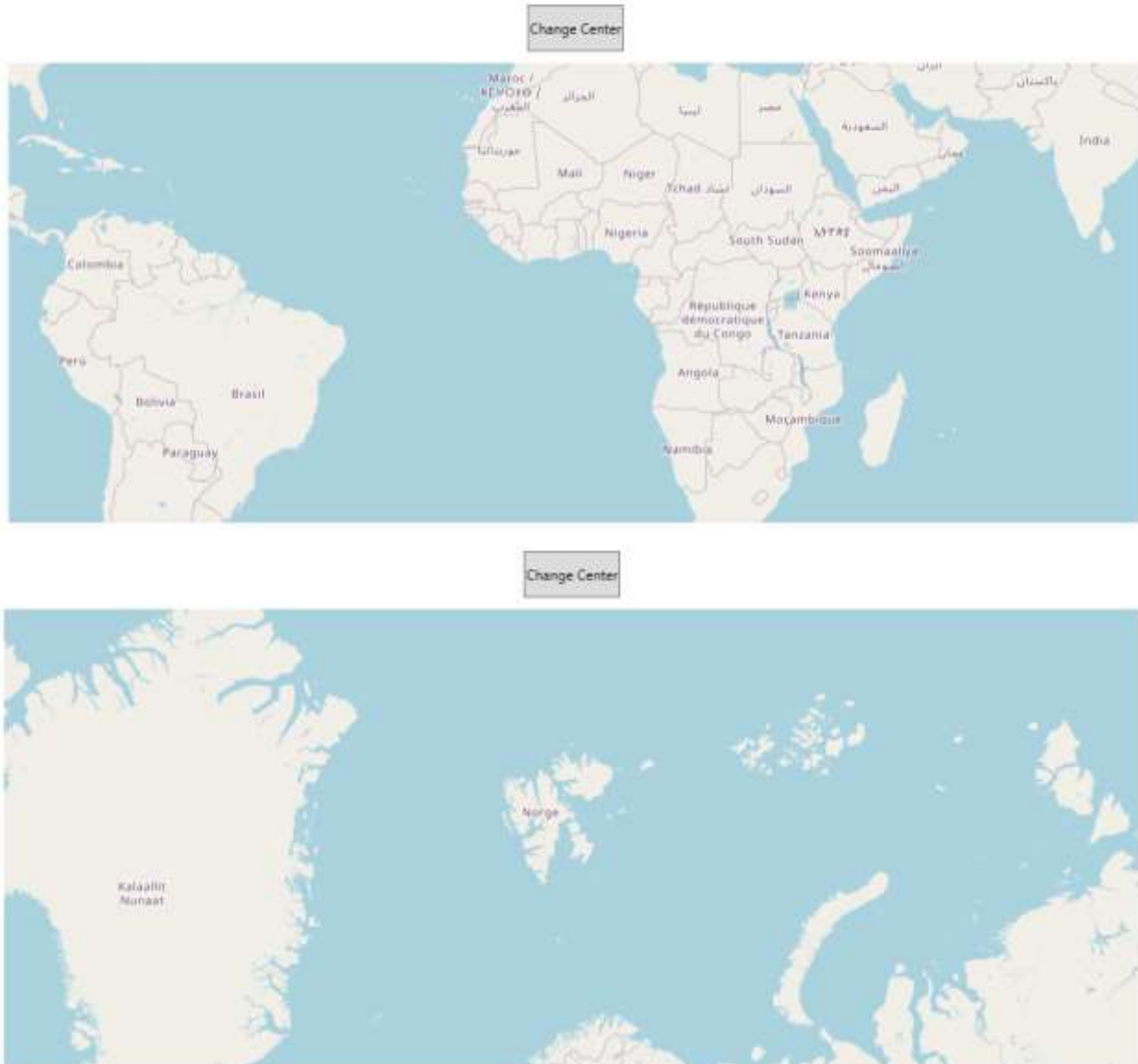
1. [GeopointToViewPoint](#) - Converts the latitude and longitude values to screen point. Here, pass the parameters as latitude and longitude values, from that values we can get screen points x and y.
2. [GetLatLonFromPoint](#) - Converts the screen point to longitude and latitude values. Here, pass the parameters as screen points x and y, from that points we can get longitude(Point.X) and latitude(Point.Y) values.

XML

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="60"/>
<RowDefinition Height="*/>
</Grid.RowDefinitions>
<Button x:Name="but" Content="Change Center"
HorizontalAlignment="Center"
Click="but_Click" Margin="10"/>
<maps:SfMap Grid.Row="1" ZoomLevel="3">
<maps:SfMap.Layers>
<maps:ImageryLayer x:Name="layer">
</maps:ImageryLayer>
</maps:SfMap.Layers>
</maps:SfMap>
</Grid>
```

C#

```
private void but_Click(object sender, RoutedEventArgs e)
{
    Point pixelPoint = layer.GeopointToViewPoint(21.00, 78.00);
    Point longitudeLatitude = layer.GetLatLonFromPoint(pixelPoint);
    layer.Center = longitudeLatitude;
}
```

SfMaskedEdit

WPF MaskedTextBox (SfMaskedEdit) Overview

The [WPF MaskedTextBox](#) (SfMaskedEdit) is an advanced version of the input control that restricts your input of certain types such as characters, text, and numbers by using a mask pattern. This control is used to create a template for providing information such as telephone numbers, IP addresses, product IDs, and so on.

Debit Card Number

5639-87XX-XXXX-XXXX

Expiry Month Expiry Year CVV

MM YYYY XXX

Promo Code

OFFR-10XX

Name on card

Enter name

Mobile Number

+1-XXX-XXX-XXXX

Key features

Mask types - Provides different set of mask types. The types are Simple, Regular and RegEx.

PromptChar - Provides support for setting the prompt characters manually.

Validation - Provides support to validate the input values can be done either during each key press or when the control lost its focus.

Value - Provides support to enter the Values and clipboard operations can be used with or without literal and prompt characters.

Watermark - Provides support to Watermark text can be used to display an instruction or important information.

Customization - Provides support to customize the UI of masked text box.

Getting Started with WPF MaskedTextBox (SfMaskedEdit)

This section explains how to create a [WPF MaskedTextBox](#) (SfMaskedEdit) and explains about its structure and features.

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.

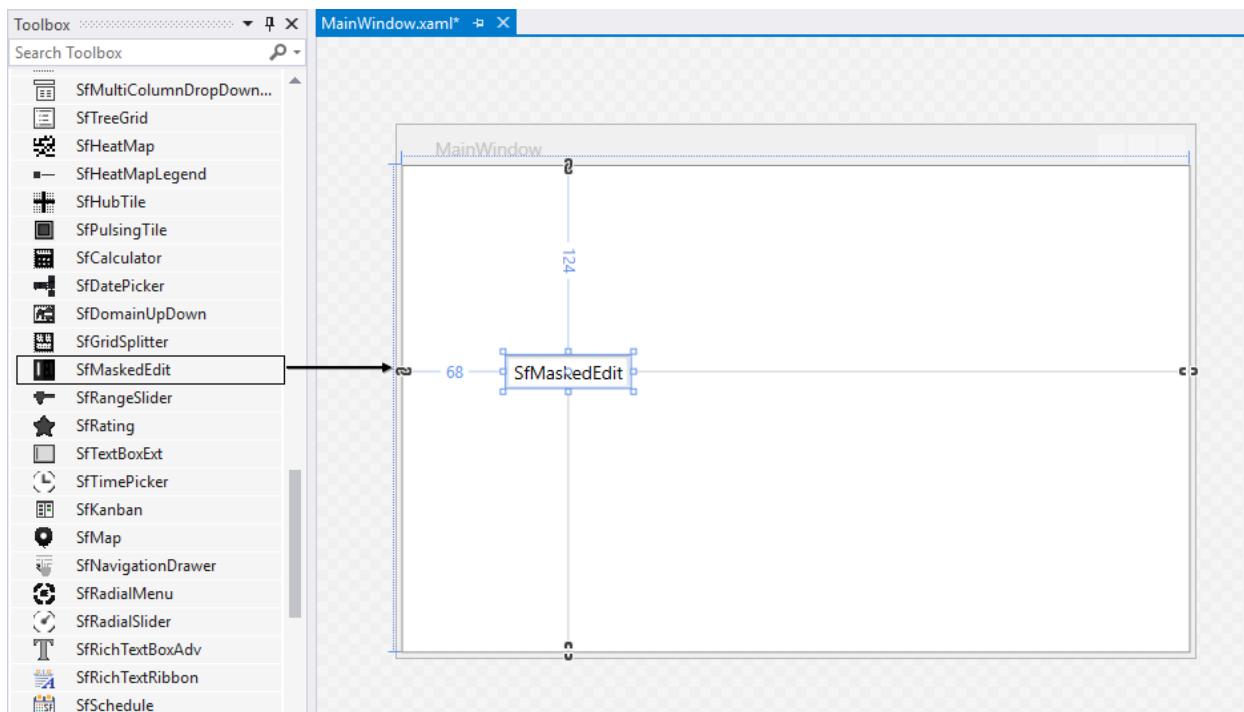
You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

Adding WPF SfMaskedEdit via designer

You can add the **SfMaskedEdit** 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.SfInput.WPF
- Syncfusion.SfShared.WPF



Adding WPF SfMaskedEdit via XAML

To add the **SfMaskedEdit** 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,
 - Syncfusion.SfInput.WPF
 - Syncfusion.SfShared.WPF
3. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> and declare the **SfMaskedEdit** control in XAML page.
4. Declare the **SfMaskedEdit** 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="SfMaskedEditSample.MainWindow"
Title="SfMaskedEdit Sample" Height="350" Width="525">
<Grid>
<!--Adding SfMaskedEdit control -->
<syncfusion:SfMaskedEdit x:Name="sfMaskedEdit"
Width="100"
Height="25" />
```

```
</Grid>
</Window>
```

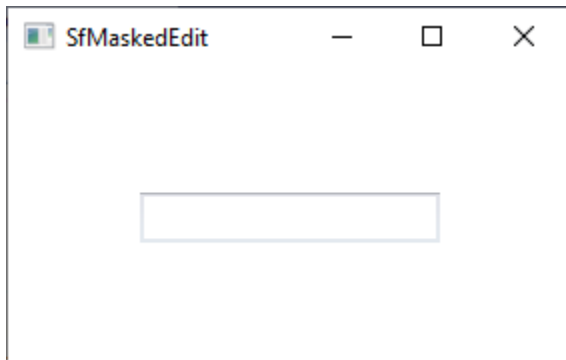
Adding WPF SfMaskedEdit via C\#

To add the **SfMaskedEdit** 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.SfInput.WPF
 - Syncfusion.SfShared.WPF
3. Include the required namespace and create an instance of **SfMaskedEdit** and add it to the window.
4. Declare the **SfMaskedEdit** control using C#.

C#

```
using Syncfusion.Windows.Controls.Input;
public partial class MainWindow : Window {
    public MainWindow() {
        InitializeComponent();
        //Creating an instance of SfMaskedEdit control
        SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
        sfMaskedEdit.Width = 100;
        sfMaskedEdit.Height = 25;
        //Adding SfMaskedEdit as window content
        this.Content = sfMaskedEdit;
    }
}
```



Note: View [Sample](#) in GitHub

Restrict the user to enter valid data

You can restrict the user to enter the valid input without any custom validation by creating the mask pattern as your requirement. You can enable the mask by setting the mask pattern to the **Mask** property and set the **MaskType** property value as **Regex**. The default value of **Mask** property is **null** and **MaskType** property is **Simple**.


XML

```
<syncfusion:SfMaskedEdit Mask="-?\d+\.\d*" />
```

```
MaskType="Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"-?\d+\.\d*";
```



Here, the **SfMaskedEdit** accept the positive and negative whole or float type numbers.

Note: Please refer the [Restrict the user to enter valid data](#) page to know more about the various mask pattern with examples.

Note: View [Sample](#) in GitHub

Setting the value

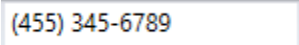
You can set the value for the **SfMaskedEdit** by using the **Value** property. Based on the mask, the value of **Value** property is formatted. The default value of **Value** property is **null**.

XML

```
<syncfusion:SfMaskedEdit Value="4553456789"
Mask="\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.Value="4553456789";
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}";
```



Note: View [Sample](#) in GitHub

Get the value in various formats

By default, the **Value** property holds your input characters, prompt characters, and the literals defined in the mask. You can modify this and allow the **Value** property to hold the characters without prompt and literals by setting the **ValueMaskFormat** property. The value can be formatted by any one of the following formatting options,

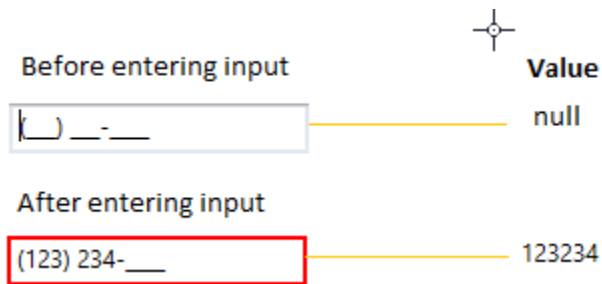
- ExcludePromptAndLiterals
- IncludeLiterals
- IncludePrompt
- IncludePromptAndLiterals

XML

```
<syncfusion:SfMaskedEdit ValueMaskFormat="ExcludePromptAndLiterals"
Mask="\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValueMaskFormat = MaskFormat.ExcludePromptAndLiterals;
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}";
```



Note: View [Sample](#) in GitHub

Indicates error on invalid input

After input validation failed, you can indicate to the user about the invalid input by the showing error border. The error border automatically disappeared when the input validation is succeed. You can change the error border color by using the [ErrorBorderBrush](#) property. The default value of [ErrorBorderBrush](#) property is Red.

XML

```
<syncfusion:SfMaskedEdit ErrorBorderBrush="Yellow"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ErrorBorderBrush = Brushes.Yellow;
```



Note: View [Sample](#) in GitHub

Indicates the missed input

You can indicate to the user to enter the missed input by using the prompt character. You can change the prompt character by using the [PromptChar](#) property. The default prompt character is `_`.

XML

```
<syncfusion:SfMaskedEdit PromptChar="X"
MaskType="Regex"
```

```
Mask="\+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.PromptChar = 'X';
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}";
```

Default prompt character = '_'

+1 112-1_-__

Prompt character = 'X'

+1 112-1XX-XXXX

Note: View [Sample](#) in GitHub

Value changed notification

you can notified when changing the value of `SfMaskedEdit.Value` property by using the [ValueChanged](#) event.

Note: Your valid input character is updated to the `Value` property based on the `ValidationMode` property.

Refer [Input Validation](#) to know more about the `ValidationMode`.

XML

```
<syncfusion:SfMaskedEdit ValueChanged="SfMaskedEdit_ValueChanged"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValueChanged += SfMaskedEdit_ValueChanged;
```

You can handle the event as follows,

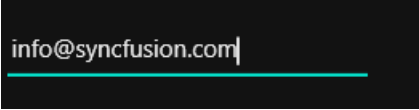
C#

```
private void SfMaskedEdit_ValueChanged(object sender, EventArgs e) {
    MessageBox.Show("Value changed");
}
```

Theme

SfMaskedEdit supports various built-in themes. Refer to the below links to apply themes for the SfMaskedEdit,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Note: View [Sample](#) in GitHub

Input Restriction in WPF MaskedTextBox (SfMaskedEdit)

This section explains accessing the [SfMaskedEdit](#) properties and events associated with it.

Restrict the user to enter valid data

You can restrict the user to enter the valid input without any custom validation by creating the mask pattern as your requirement. You can enable the mask by setting the mask pattern to the [Mask](#) property and set the [MaskType](#) property value as [Regex](#). The default value of [Mask](#) property is [null](#) and [MaskType](#) property is [Simple](#). The following table demonstrates the mask elements with its description,

Elements	Description
[]	It denotes the set to define the particular input elements
[ABC]	Accepts any single character included in the specified set of characters.
[^ABC]	Accepts any single character not included in the specified set of characters.
[A-Z]	Accepts any character between two specified set of characters inclusive.
\d	Accepts any digit. Same as [0-9].
\D	Accepts any non-digit. Same as [^0-9].
\w	Accepts any word character. \w is the same as [a-zA-Z_0-9].
\W	Accepts any non-word character. \W is the same as [^a-zA-Z_0-9].
\s	Accepts any white space characters.
\S	Accepts any non-white space characters.
(?=subexpression)	Matches a group after the main expression without including it in the result.
(?!subexpression)	Specifies a group that cannot match after the main expression.
{n}	Accepts the input for n number of times.
{n,}	Accepts the input for 'n' and more than 'n' number of times.
{n,m}	Accepts the input for n minimum number of times and m maximum number of times.
+	Accepts one or more matches for the preceding character.
*	Accepts zero or more matches for the preceding character.
?	Optional input (Zero or one occurrence of the matching input).
	Acts like a Boolean OR. Matches the expression before or after the .

.	Accepts any character. It can be changed based on culture.
---	--

Allow specific values

If you want to allow the user to enter only the specific values, use the `(?=subexpression)` with the mask. It will match the input with the mask when value is same as the subexpression.

XML

```
<syncfusion:SfMaskedEdit Mask="(?=123)\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"(?=123)\d{3}";
```

For the input mask `(?=123)\d{3}`. It allows first two digit. While entering the third digit, it will validate the regex and restrict the input. It allows the output only when the MaskText is "123" for this given input.

RegEx Input	SfMaskedEdit	Example
<code>(?=123)\d{3}</code>	123	123

Restrict specific values

If you want to restrict the user to enter some specific values, use the `(?!subexpression)` with the mask. It will not allow the input when value is same as the subexpression.

XML

```
<syncfusion:SfMaskedEdit Mask="(?!55) (?!000) (?!666)\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"(?!55) (?!000) (?!666)\d{3}";
```





For the input mask `(?!55)(?!000)(?!666)\d{3}` – `\d{3}` denotes that the output will be of 3 character length. `(?55)` denotes the value will not begin with 55. `(?!000)(?!666)` denotes the output cannot be 000 or 666.

RegEx Input	SfMaskedEdit	Example
<code>(?!55)(?!000)(?!666)\d{3}</code>	<input type="text" value="123"/>	Value will be 3 digit and the value cannot be 000 or 666. Also will not begins with 55

Basic Mask examples

You can create your custom mask as follows,

Mask use case	Pattern	Example
Positive whole number	<code>\d+</code>	<input type="text"/>
Negative whole number	<code>-\d+</code>	<input type="text"/>
Positive and negative whole number	<code>-?\d+</code>	<input type="text"/>
Positive and negative float number	<code>-?\d+\.\d*</code>	<input type="text"/>
Percent value	<code>\d+\.\d{2}%</code>	<input type="text"/>
Currency value	<code>\$ \d+\.\d{2}"</code>	<input type="text"/>
Alphanumeric with space	<code>[a-zA-Z0-9]*</code>	<input type="text"/>
Alphanumeric without space	<code>[a-zA-Z0-9]*</code>	<input type="text"/>
Email ID	<code>[A-Za-z0-9._%~]+@[A-Za-z0-9]+\.[A-Za-z]{2,3}</code>	<input type="text"/>
Hexadecimal number	<code>\\x[0-9A-Fa-f]{1,2}</code>	<input type="text"/>
Octal number	<code>\\0[0-7]{1,3}</code>	<input type="text"/>
Hexadecimal color code	<code>#[A-Fa-f0-9]{6}</code>	<input type="text"/>
Time	<code>(0[0-9] 1[0-9] 2[0-3]):[0-5][0-9]</code>	<input type="text"/>
Phone number	<code>\\([0-9]\\d{2})\\ [0-9]\\d{2}-[0-9]\\d{3}</code>	<input type="text"/>
Product key	<code>[A-Z\\d]{5}-[A-Z\\d]{5}-[A-Z\\d]{5}-[A-Z\\d]{5}-[A-Z\\d]{5}</code>	<input type="text"/>

Zip code	\d{5}-\d{5}	
Bank SWIFT code	[A-Z]{6}[0-9]{2}[A-Z]{3}	
Bank Account Number	\d{8,12}	
Credit Card Number	\d{4} \d{4} \d{4} \d{4}	

Note: View [Sample](#) in GitHub

Setting the value

You can set the value for the **SfMaskedEdit** by using the **Value** property. Based on the mask, the value of **Value** property is formatted. The default value of **Value** property is **null**.

XML

```
<syncfusion:SfMaskedEdit Value="4553456789"
Mask="\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.Value="4553456789";
sfMaskedEdit.MaskType = MaskType.Regex;
sfMaskedEdit.Mask = @"\"([0-9]\d{2}) [0-9]\d{2}-[0-9]\d{3}";
```

(455) 345-6789

Note: View [Sample](#) in GitHub

Get the value in various formats

By default, the **Value** property holds your input characters, prompt characters, and the literals defined in the mask. You can modify this and allow the **Value** property to hold the characters without prompt and literals by setting the **ValueMaskFormat** property. The value can be formatted by any one of the following formatting options,

- ExcludePromptAndLiterals
- IncludeLiterals
- IncludePrompt
- IncludePromptAndLiterals

Value without prompt character and literals

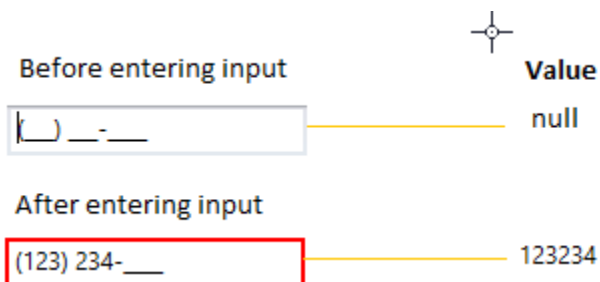
If you want to get only the text entered by the user, use the **ValueMaskFormat** property as **ExcludePromptAndLiterals**. It does not include prompt and literals characters in the **Value** property.

XML

```
<syncfusion:SfMaskedEdit ValueMaskFormat="ExcludePromptAndLiterals"
Mask="\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValueMaskFormat = MaskFormat.ExcludePromptAndLiterals;
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}";
```



Note: View [Sample](#) in GitHub

Value with literals

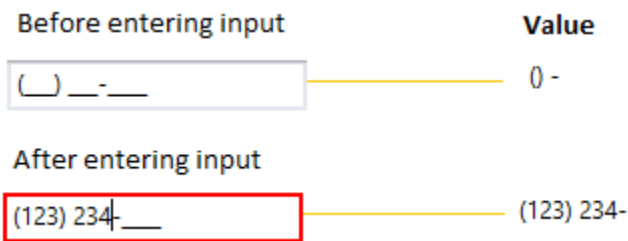
If you want to get the text entered by the user as well as any literal characters defined in the mask, use the **ValueMaskFormat** property as **IncludeLiterals**. It does not include prompt characters in the **Value** property.

XML

```
<syncfusion:SfMaskedEdit ValueMaskFormat="IncludeLiterals"
Mask="\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValueMaskFormat = MaskFormat.IncludeLiterals;
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}";
```



Note: View [Sample](#) in GitHub

Value with prompt character

If you want to get the text entered by the user as well as any prompt characters defined in the mask, use the `ValueMaskFormat` property as `IncludePrompt`. It does not include literals characters in the `Value` property.

XML

```
<syncfusion:SfMaskedEdit ValueMaskFormat="IncludePrompt"
Mask="\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValueMaskFormat = MaskFormat.IncludePrompt;
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\"([0-9]\d{2}) [0-9]\d{2}-[0-9]\d{3}";
```

Before entering input

Value

() _ . _

After entering input

(123) 234 _

123234 _

Note: View [Sample](#) in GitHub

Value with prompt characters and literals

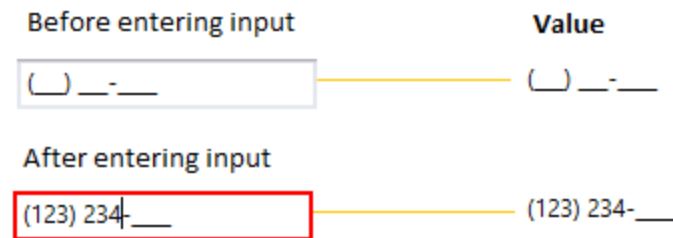
If you want to get the text entered by the user as well as any literals and the prompt character defined in the mask, use the `ValueMaskFormat` property as `IncludePromptAndLiterals`. It includes the prompt and literals characters in the `Value` property.

XML

```
<syncfusion:SfMaskedEdit ValueMaskFormat="IncludePromptAndLiterals"
Mask="\([0-9]\d{2}\) [0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValueMaskFormat = MaskFormat.IncludePromptAndLiterals;
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\"([0-9]\d{2}) [0-9]\d{2}-[0-9]\d{3}";
```



Note: View [Sample](#) in GitHub

Input validation

You can validate the user input on key press or control lost focus.

Input validation on each input entering

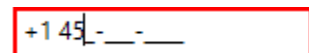
You can validate the user input on each input key press. You can enable it by setting the [ValidationMode](#) property as `KeyPress`. The default value of `ValidationMode` property is `KeyPress`.

XML

```
<syncfusion:SfMaskedEdit ValidationMode="KeyPress"
Mask="\+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValidationMode = InputValidationMode.KeyPress;
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}";
```



Note: View [Sample](#) in GitHub

Input validation on lost focus

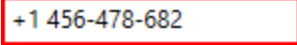
You can validate the user input on when control lost focus. You can enable it by setting the `ValidationMode` property as `LostFocus`.

XML

```
<syncfusion:SfMaskedEdit ValidationMode="LostFocus"
Mask="\+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValidationMode = InputValidationMode.LostFocus;
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}";
```



Note: View [Sample](#) in GitHub

Get the validation result

You can check whether the input validation is succeed or failed by using the [HasError](#) property on once validation is completed. The `HasError` property returns the following results,

- **True** - If validation is successful.
- **False** - If validation is failed.

XML

```
<syncfusion:SfMaskedEdit ValidationMode="LostFocus"
Mask="\+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}"
MaskType = "Regex"
LostFocus="SfMaskedEdit_LostFocus"
Name="sfMaskedEdit"/>
```

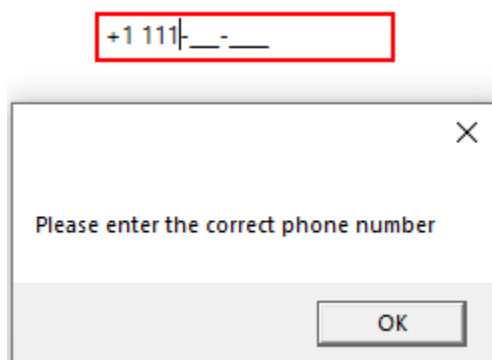
C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValidationMode = InputValidationMode.LostFocus;
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"\+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}";
sfMaskedEdit.LostFocus += SfMaskedEdit_LostFocus;
```

You can Get the validation result as follows,

C#

```
private void SfMaskedEdit_LostFocus(object sender, RoutedEventArgs e) {
if ((sender as SfMaskedEdit).HasError) {
MessageBox.Show("Please enter the correct phone number");
}
}
```



Indicates error on invalid input

After input validation failed, you can indicate to the user about the invalid input by the showing error border. The error border automatically disappeared when the input validation is succeed. You can

change the error border color by using the [ErrorBorderBrush](#) property. The default value of `ErrorBorderBrush` property is `Red`.

XML

```
<syncfusion:SfMaskedEdit ErrorBorderBrush="Yellow"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ErrorBorderBrush = Brushes.Yellow;
```



Note: View [Sample](#) in GitHub

Indicates the missed input

You can indicate to the user to enter the missed input by using the prompt character. You can change the prompt character by using the [PromptChar](#) property. The default prompt character is `_`.

XML

```
<syncfusion:SfMaskedEdit PromptChar="X"
MaskType="Regex"
Mask="+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.PromptChar = 'X';
sfMaskedEdit.MaskType = MaskType.Regex;
sfMaskedEdit.Mask = @"+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}";
```

Default prompt character = `'_'`

+1 112-1|_ _ _

Prompt character = `'X'`

+1 112-1|X-X-XXX

Note: View [Sample](#) in GitHub

Display the prompt character on got focus

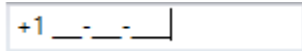
By default, the prompt character shown only on while entering the input. If you want to show the prompt character on when control got focus, use the [ShowPromptOnFocus](#) property value as `true`. The default vale of `ShowPromptOnFocus` property is `false`.

XML

```
<syncfusion:SfMaskedEdit ShowPromptOnFocus="True"
PromptChar="X"
MaskType="Regex"
Mask="+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}"
Name="sfMaskedEdit"/>
```


C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ShowPromptOnFocus = true;
sfMaskedEdit.PromptChar = 'X';
sfMaskedEdit.MaskType = MaskType.RegEx;
sfMaskedEdit.Mask = @"^+1 [0-9]\d{2}-[0-9]\d{2}-[0-9]\d{3}";
```



Note: View [Sample](#) in GitHub

Setting the watermark

You can prompt the user with instructions or important information when control is not on focus and any valid character is not entered. You can set watermark by using the [Watermark](#) property. The default value of [Watermark](#) property is `null`.

XML

```
<syncfusion:SfMaskedEdit Watermark="Type here"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.Watermark = "Type here";
```



Note: View [Sample](#) in GitHub

Custom template for Watermark

You can change the template of the [Watermark](#) by using the [WatermarkTemplate](#) property.

XML

```
<syncfusion:SfMaskedEdit Watermark="Type here"
Name="sfMaskedEdit">
<syncfusion:SfMaskedEdit.WatermarkTemplate >
<DataTemplate>
<Border Background="Yellow">
<TextBlock Foreground="Blue"
FontWeight="Bold"
Text="{Binding}"
TextAlignment="Center"/>
</Border>
</DataTemplate>
</syncfusion:SfMaskedEdit.WatermarkTemplate>
</syncfusion:SfMaskedEdit>
```



Note: View [Sample](#) in GitHub

Value changed notification

you can notified when changing the value of `SfMaskedEdit.Value` property by using the [ValueChanged](#) event.

Note: Your valid input character is updated to the `Value` property based on the `ValidationMode` property.

Refer [Input Validation](#) to know more about the `ValidationMode`.

XML

```
<syncfusion:SfMaskedEdit ValueChanged="SfMaskedEdit_ValueChanged"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.ValueChanged += SfMaskedEdit_ValueChanged;
```

You can handle the event as follows,

C#

```
private void SfMaskedEdit_ValueChanged(object sender, EventArgs e) {
    MessageBox.Show("Value changed");
}
```

Appearance in WPF MaskedTextBox (SfMaskedEdit)

This section explains different UI customization and theming options available in [SfMaskedEdit](#).

Setting the background

You can change the default background color and selection color of `SfMaskedEdit` by using the `Background` and `SelectionBrush` property. The default value of `Background` property is `White` and `SelectionBrush` property is `Royal Blue`.

XML

```
<syncfusion:SfMaskedEdit Background="Yellow"
SelectionBrush="Green"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.Background = Brushes.Yellow;
sfMaskedEdit.SelectionBrush = Brushes.Green;
```



Note: View [Sample](#) in GitHub

Setting the foreground

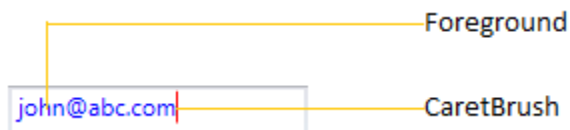
You can change the foreground color by using the **Foreground** property and can change the caret color by using the **CaretBrush** property. The default value of **Foreground** property is **Black** and **CaretBrush** property is **null**.

XML

```
<syncfusion:SfMaskedEdit Foreground="Blue"
CaretBrush="Red"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.Foreground = Brushes.Blue;
sfMaskedEdit.CaretBrush = Brushes.Red;
```



Note: View [Sample](#) in GitHub

Setting the border color

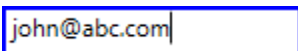
You can change the default border color of **SfMaskedEdit** by using the **BorderBrush** property. The default value of **BorderBrush** property is **Lavender**.

XML

```
<syncfusion:SfMaskedEdit BorderBrush="Blue"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();
sfMaskedEdit.BorderBrush = Brushes.Blue;
```



Note: View [Sample](#) in GitHub

Change flow direction

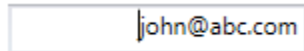
You can change the flow direction of the **SfMaskedEdit** layout from right to left by setting the **FlowDirection** property value as **RightToLeft**. The default value of **FlowDirection** property is **LeftToRight**.

XML

```
<syncfusion:SfMaskedEdit FlowDirection="RightToLeft"
Name="sfMaskedEdit"/>
```

C#

```
SfMaskedEdit sfMaskedEdit = new SfMaskedEdit();  
sfMaskedEdit.FlowDirection = FlowDirection.RightToLeft;
```

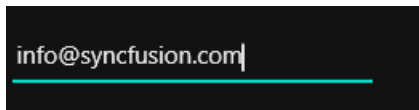


Note: View [Sample](#) in GitHub

Theme

SfMaskedEdit supports various built-in themes. Refer to the below links to apply themes for the SfMaskedEdit,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



MenuAdv

WPF Menu (MenuAdv) Overview

A MenuAdv control allows the hierarchical organization of elements that are associated with commands and event handlers. This control contains a collection of MenuItemAdv, which can be expanded to display additional MenuItemAdv's or to perform a specific action when being clicked.

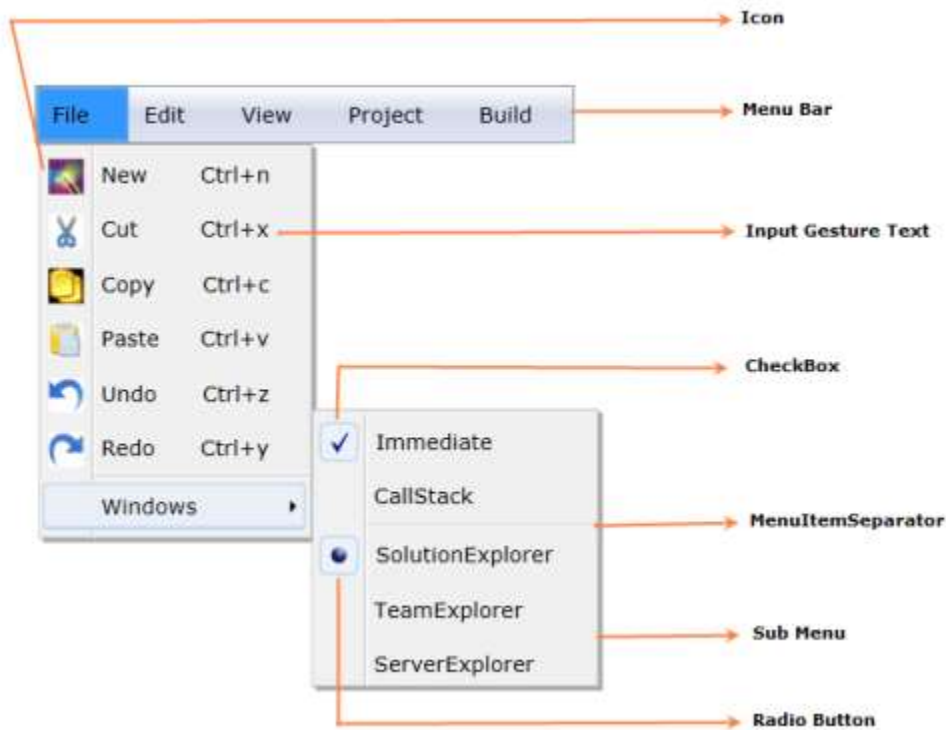
The main features of MenuAdv include:

- Binding and DataTemplate support
- Command Binding and Command Target support
- Orientation support
- Expand Modes support
- Icon support
- Check Box and Radio Button support
- InputGestureText support
- MenuItemSeparator
- Boundary Detection
- Scroll support
- Animation support
- Keyboard Navigation support
- Blendability

Getting Started with WPF Menu (MenuAdv)

This section provides a quick overview for working with the Menu ([MenuAdv](#)).

Structure of the MenuAdv 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 [MenuAdv](#) 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](#)

Important

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

Creating Application with MenuAdv control

In this walk through, user will create a WPF application that contains [MenuAdv](#) 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 using [MenuAdv](#).

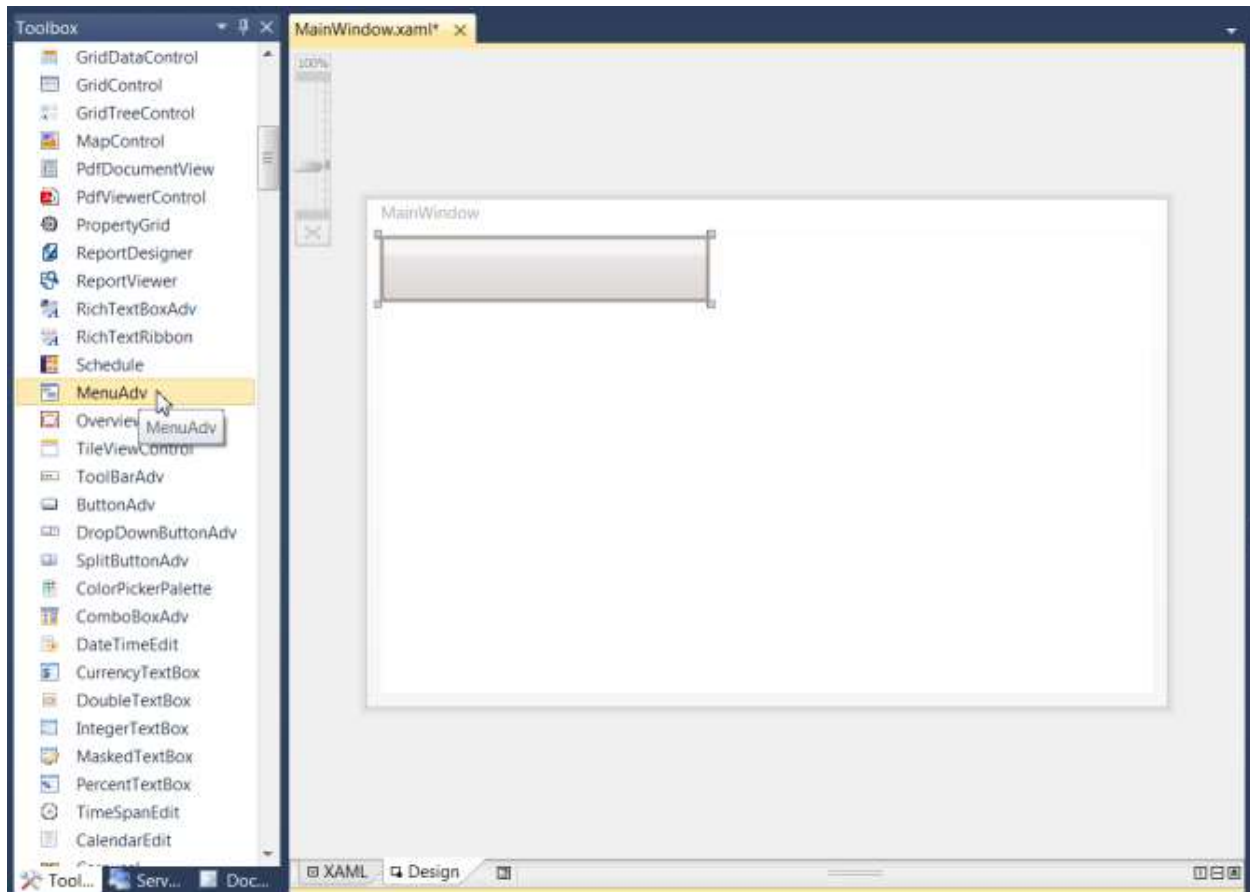
Adding control via designer

The [MenuAdv](#) control can be added to the application in VisualStudio or ExpressionBlend by dragging it from Toolbox and dropping it in designer. The required assembly will be added automatically.

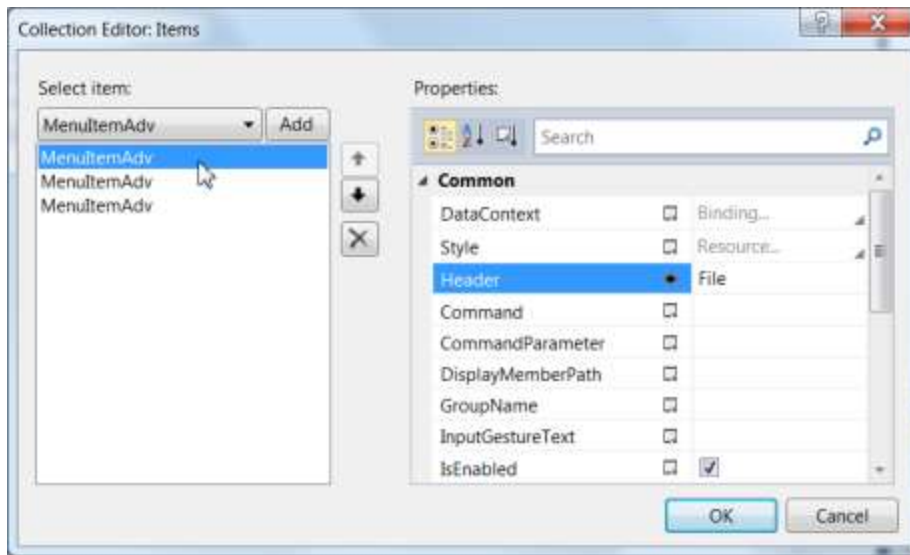
Through Visual Studio

The following are the steps to create the [MenuAdv](#) control using Visual Studio.

1. Drag [MenuAdv](#) from the Visual Studio Toolbox and drop it in the designer.



2. Select the [MenuAdv](#) and go to properties.
3. Click the three dotted button given in Items property. Collection Editor window will open.

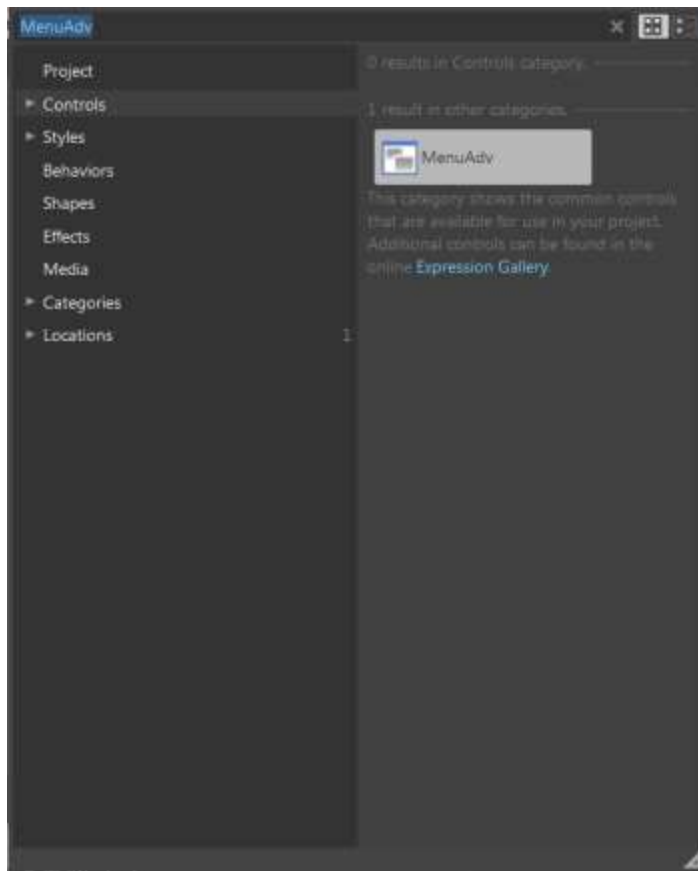


4. Using the Collection Editor, add the GroupBarItem and configure their properties.

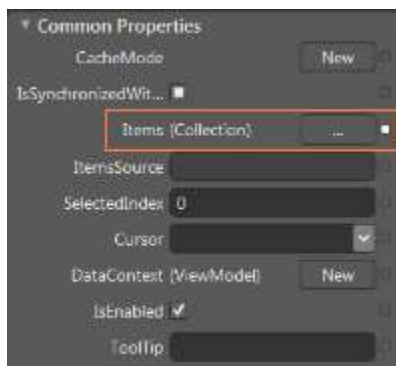
Through Expression Blend

The [MenuAdv](#) control can also be created and configured using Expression Blend by following steps.

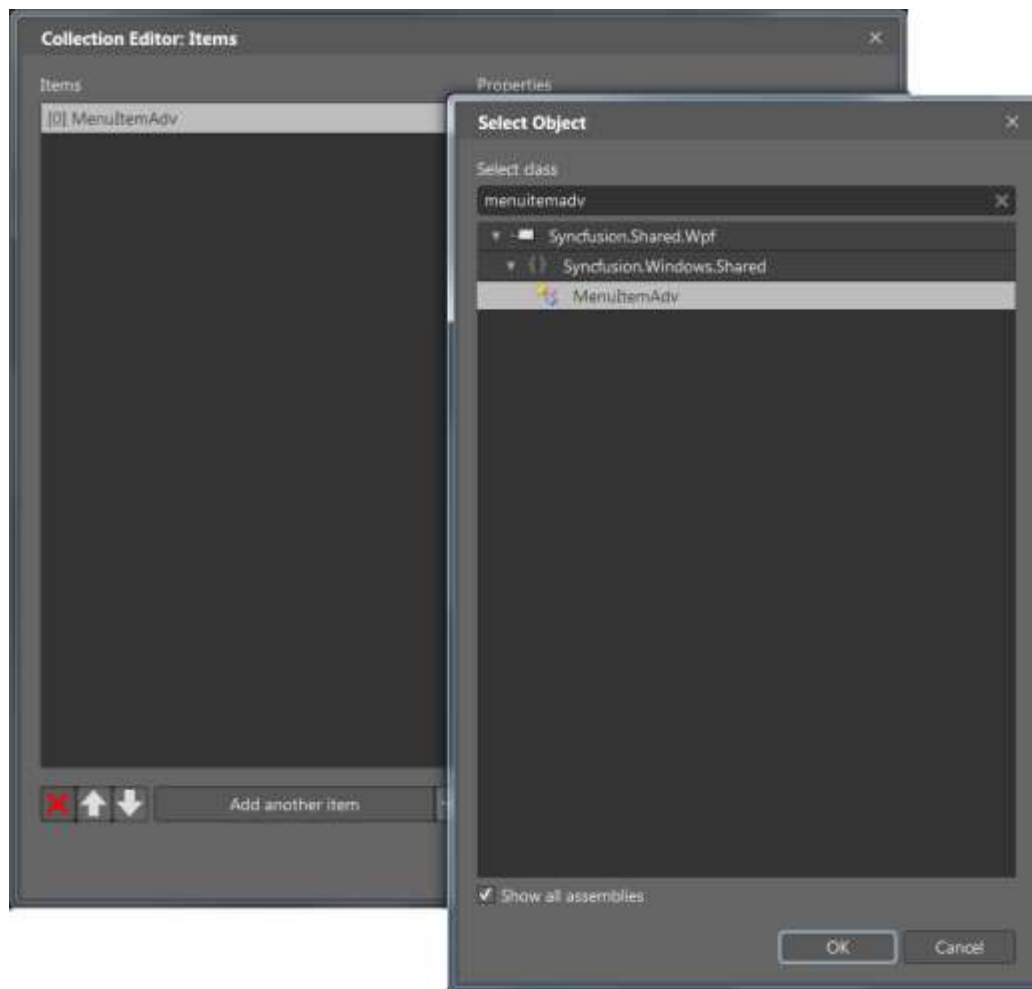
1. Create a WPF project in the Expression Blend and refer the following assembly.
 - Syncfusion.Shared.Wpf
2. Search for [MenuAdv](#) in the Toolbox.



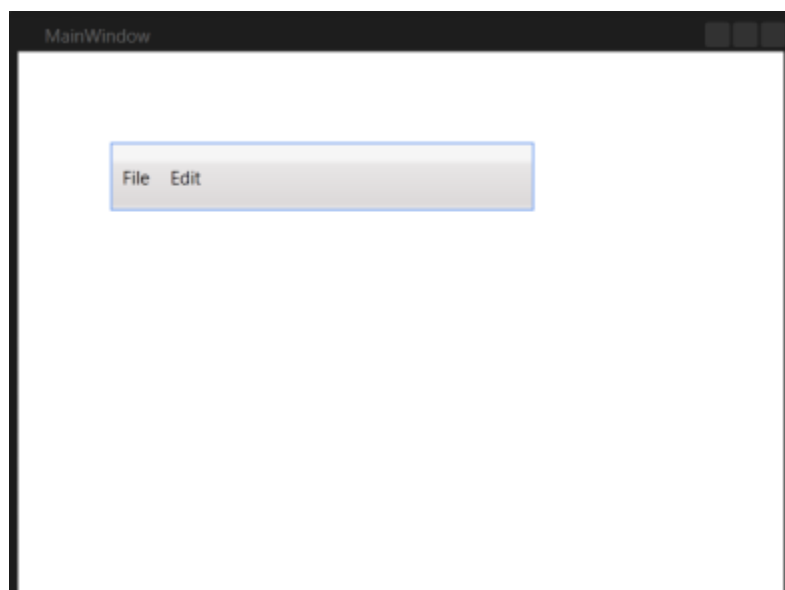
3. Drag the [MenuAdv](#) to designer to generate an empty menu bar.
4. To add the [MenuItemAdv](#) to the [MenuAdv](#) control, select the [MenuAdv](#) and go to Properties area.
5. Click Items (Collection) under Common Properties.



6. Once the Collection Editor opens, click Add another item. The Select Object window will open.
7. Select [MenuItemAdv](#) by typing [MenuItemAdv](#) in the search box, and then click OK.



8. Configure the properties (such as header or icon) of the [MenuItemAdv](#) using the properties in the Collection Editor. This will generate the following control.



Note: You can customize the appearance of the [MenuItemAdv](#) using the template-editing feature available in the Expression Blend.

Adding control manually in XAML

In order to add [MenuAdv](#) control manually in XAML, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.Shared.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page.
3. Declare [MenuAdv](#) in XAML page.

XML

```
<syncfusion:MenuAdv Height="25" VerticalAlignment="Top">
<syncfusion:MenuItemAdv Header="Products" />
<syncfusion:MenuItemAdv Header="Business Intelligence" />
<syncfusion:MenuItemAdv Header="User Interface" >
<syncfusion:MenuItemAdv Header="WPF ">
<syncfusion:MenuItemAdv Header="Tools"/>
<syncfusion:MenuItemAdv Header="Chart"/>
<syncfusion:MenuItemAdv Header="Grid"/>
<syncfusion:MenuItemAdv Header="Diagram"/>
<syncfusion:MenuItemAdv Header="Gauge"/>
<syncfusion:MenuItemAdv Header="Schedule"/>
<syncfusion:MenuItemAdv Header="Edit"/>
</syncfusion:MenuItemAdv>
<syncfusion:MenuItemAdv Header="Silverlight "/>
<syncfusion:MenuItemAdv Header="Reporting" />
</syncfusion:MenuItemAdv>
</syncfusion:MenuAdv>
```

Adding control manually in C#

In order to add [MenuAdv](#) control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.Shared.WPF
2. Import MenuAdv namespace **Syncfusion.Windows.Shared**.

C#

```
using Syncfusion.Windows.Shared;
```

3. Create MenuAdv control instance and add it to the page.

C#

```
MenuAdv mAdv = new MenuAdv();
MenuItemAdv product = new MenuItemAdv() { Header = "Products" };
MenuItemAdv bi = new MenuItemAdv() { Header = "Business Intelligence" };
MenuItemAdv ui = new MenuItemAdv() { Header = "User Interface" };
MenuItemAdv wpf = new MenuItemAdv() { Header = "WPF" };
```

```
MenuItemAdv tools = new MenuItemAdv() { Header = "Tools" };
MenuItemAdv chart = new MenuItemAdv() { Header = "Chart" };
MenuItemAdv grid = new MenuItemAdv() { Header = "Grid" };
MenuItemAdv diagram = new MenuItemAdv() { Header = "Diagram" };
MenuItemAdv gauge = new MenuItemAdv() { Header = "Gauge" };
MenuItemAdv schedule = new MenuItemAdv() { Header = "Schedule" };
MenuItemAdv edit = new MenuItemAdv() { Header = "Edit" };
MenuItemAdv sl = new MenuItemAdv() { Header = "Silverlight" };
MenuItemAdv reporting = new MenuItemAdv() { Header = "Reporting" };
wpf.Items.Add(tools);
wpf.Items.Add(chart);
wpf.Items.Add(grid);
wpf.Items.Add(diagram);
wpf.Items.Add(gauge);
wpf.Items.Add(schedule);
wpf.Items.Add(edit);
ui.Items.Add(wpf);
ui.Items.Add(sl);
product.Items.Add(bi);
product.Items.Add(ui);
product.Items.Add(reporting);
mAdv.Items.Add(product);
this.Content = mAdv;
```

[View sample in GitHub](#)

Set icon for Menu item

You can display image on left of the [MenuItemAdv](#) control by setting the image source as value for [Icon](#) property.

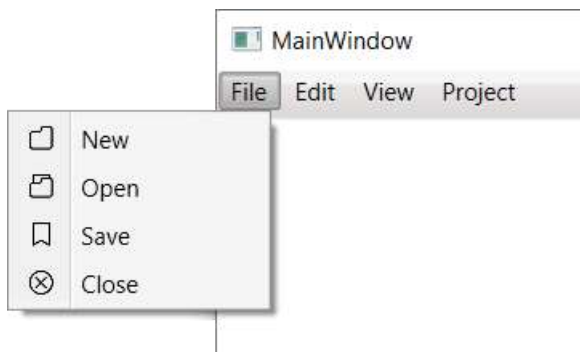
XML

```
<Grid>
<syncfusion:MenuAdv Height="25" VerticalAlignment="Top">
<syncfusion:MenuItemAdv Header="File" >
<syncfusion:MenuItemAdv Header="New">
<syncfusion:MenuItemAdv.Icon>
<Image Source="NewIcon.png" Width="15" Height="15" />
</syncfusion:MenuItemAdv.Icon>
</syncfusion:MenuItemAdv>
<syncfusion:MenuItemAdv Header="Open">
<syncfusion:MenuItemAdv.Icon>
<Image Source="OpenIcon.png" Width="15" Height="15" />
</syncfusion:MenuItemAdv.Icon>
</syncfusion:MenuItemAdv>
<syncfusion:MenuItemAdv Header="Save">
<syncfusion:MenuItemAdv.Icon>
<Image Source="SaveIcon.png" Width="15" Height="15" />
</syncfusion:MenuItemAdv.Icon>
</syncfusion:MenuItemAdv>
<syncfusion:MenuItemAdv Header="Close">
<syncfusion:MenuItemAdv.Icon>
<Image Source="CloseIcon.png" Width="15" Height="15"/>
</syncfusion:MenuItemAdv.Icon>
</syncfusion:MenuItemAdv>
</syncfusion:MenuItemAdv>
</syncfusion:MenuAdv>
```

```
<syncfusion:MenuItemAdv Header="Edit" />
<syncfusion:MenuItemAdv Header="View" />
<syncfusion:MenuItemAdv Header="Project" />
</syncfusion:MenuAdv>
</Grid>
```

```
string path;
path = Path.GetFullPath(@"../.." + "NewIcon.png");
MenuAdv menuAdv = new MenuAdv()
{
    VerticalAlignment = VerticalAlignment.Top,
    Height = 25
};
Image image1 = new Image() { Height=15, Width = 15 };
Image image2= new Image() { Height = 15, Width = 15 };
Image image3 = new Image() { Height = 15, Width = 15 };
Image image4 = new Image() { Height = 15, Width = 15 };
MenuItemAdv File = new MenuItemAdv() { Header = "File" };
MenuItemAdv New = new MenuItemAdv() { Header = "New" };
MenuItemAdv Open = new MenuItemAdv() { Header = "Open" };
MenuItemAdv Save = new MenuItemAdv() { Header = "Save" };
MenuItemAdv Close = new MenuItemAdv() { Header = "Close" };
MenuItemAdv Edit = new MenuItemAdv() { Header = "Edit" };
MenuItemAdv View = new MenuItemAdv() { Header = "View" };
MenuItemAdv Project = new MenuItemAdv() { Header = "Project" };
image1.Source = new BitmapImage(new Uri(path));
New.Icon = image1;
path = Path.GetFullPath(@"../.." + "OpenIcon.png");
image2.Source = new BitmapImage(new Uri(path));
Open.Icon = image2;
path = Path.GetFullPath(@"../.." + "SaveIcon.png");
image3.Source = new BitmapImage(new Uri(path));
Save.Icon = image3;
path = Path.GetFullPath(@"../.." + "CloseIcon.png");
image4.Source = new BitmapImage(new Uri(path));
```

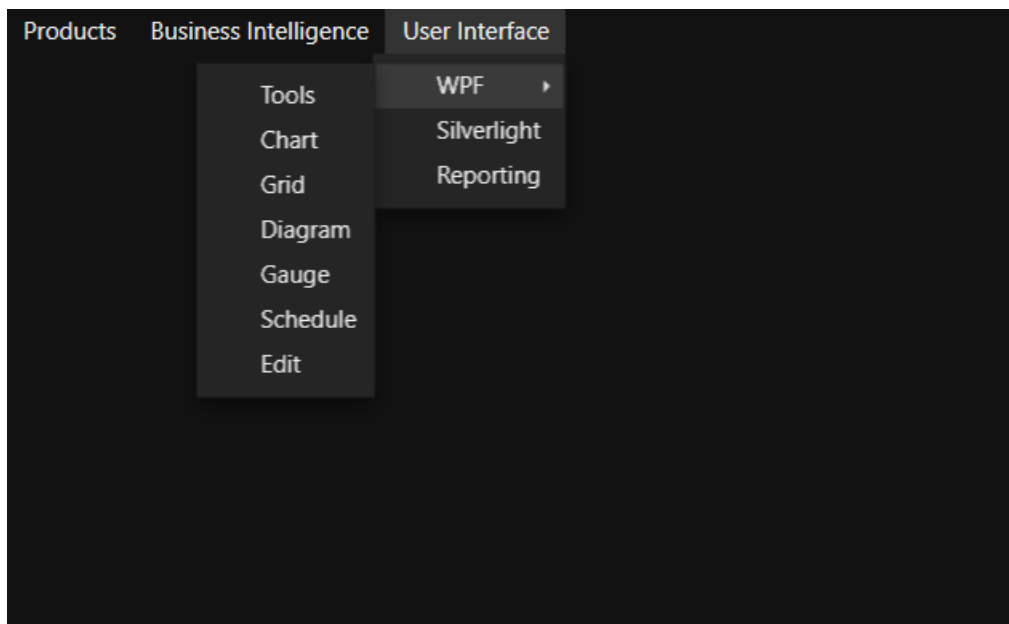
```
Close.Icon = image4;  
File.Items.Add(New);  
File.Items.Add(Open);  
File.Items.Add(Save);  
File.Items.Add(Close);  
menuAdv.Items.Add(File);  
menuAdv.Items.Add(Edit);  
menuAdv.Items.Add(View);  
menuAdv.Items.Add(Project);
```



Theme

MenuAdv supports various built-in themes. Refer to the below links to apply themes for the MenuAdv,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Data Binding in WPF Menu (MenuAdv)

The topics under this section explain the data binding support for the MenuAdv control.

Data-Binding to Objects

The MenuAdv control also supports binding to objects. The following example shows this.

1. Create a class that act as a model for MenuAdv.

CSHARP

```
public class Model
{
    public Model()
    {
        SubItems = new ObservableCollection<Model>();
    }
    public string Header { get; set; }
    public ObservableCollection<Model> SubItems { get; set; }
}
```

2. Create a ViewModel class and initialize the items.

CSHARP

```
public class ViewModel
{
    public ViewModel()
    {
        MenuItems = new ObservableCollection<Model>();
        PopulateData();
    }
    public ObservableCollection<Model> MenuItems { get; set; }
    private void PopulateData()
    {
        Model product = new Model() { Header = "Products" };
        PopulateSubSubItems(product);
        MenuItems.Add(product);
    }
    private void PopulateSubSubItems(Model product)
    {
        Model bi = new Model() { Header = "Business Intelligence" };
        Model ui = new Model() { Header = "User Interface" };
        Model wpf = new Model() { Header = "WPF" };
        Model tools = new Model() { Header = "Tools" };
        Model chart = new Model() { Header = "Chart" };
        Model grid = new Model() { Header = "Grid" };
        Model diagram = new Model() { Header = "Diagram" };
        Model gauge = new Model() { Header = "Gauge" };
        Model schedule = new Model() { Header = "Schedule" };
        Model edit = new Model() { Header = "Edit" };
        wpf.SubItems.Add(tools);
        wpf.SubItems.Add(chart);
        wpf.SubItems.Add(grid);
        wpf.SubItems.Add(diagram);
    }
}
```

```

wpf.SubItems.Add(gauge);
wpf.SubItems.Add(schedule);
wpf.SubItems.Add(edit);
Model sl = new Model() { Header = "Silverlight" };
ui.SubItems.Add(wpf);
ui.SubItems.Add(sl);
Model reporting = new Model() { Header = "Reporting" };
product.SubItems.Add(bi);
product.SubItems.Add(ui);
product.SubItems.Add(reporting);
}
}

```

3. Create a ViewModel instance and use it as DataContext for the root window.

XML

```

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

```

4. Now configure the ItemsSource and ItemTemplate of MenuAdv.

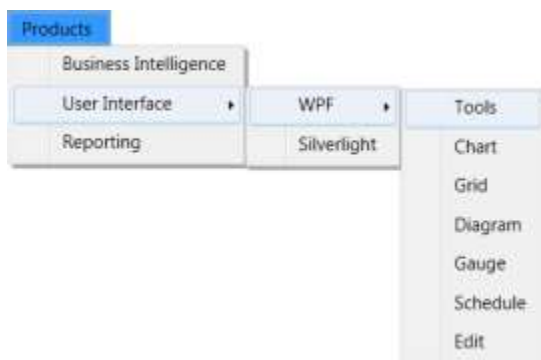
XML

```

<syncfusion:MenuAdv ItemsSource="{Binding MenuItems}">
<syncfusion:MenuAdv.ItemTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding.SubItems}">
<TextBlock Text="{Binding.Header}" />
</HierarchicalDataTemplate>
</syncfusion:MenuAdv.ItemTemplate>
</syncfusion:MenuAdv>

```

Implementing the above code will generate the following control.



Data-Binding with XML

An XML file can also be used as *ItemsSource* for the MenuAdv control. The following example illustrates this.

1. Create an XML file with the following details as follows and name it as Data.xml.

XML

```
<Categories>
<Root Name="Products" >
<SubItem Name="User Interface" >
<SubItem Name="ASP .NET"/>
<SubItem Name="ASP .NET MVC"/>
<SubItem Name="WPF">
<SubItem Name="Tools"/>
<SubItem Name="Chart"/>
<SubItem Name="Grid"/>
<SubItem Name="Diagram"/>
<SubItem Name="Gauge"/>
<SubItem Name="Schedule"/>
<SubItem Name="Edit"/>
</SubItem>
<SubItem Name="Silverlight"/>
<SubItem Name="Mobile MVC"/>
<SubItem Name="Windows Phone"/>
<SubItem Name="Windows Forms"/>
</SubItem>
<SubItem Name="Business Intelligence">
<SubItem Name="WPF"/>
<SubItem Name="ASP.NET"/>
<SubItem Name="ASP.NET MVC"/>
<SubItem Name="Silverlight"/>
</SubItem>
<SubItem Name="Reporting">
<SubItem Name="WPF"/>
<SubItem Name="Windows Forms"/>
</SubItem>
</Root>
</Categories>
```

2. Add XmlDataProvider for the above XML document.

XML

```
<XmlDataProvider Source="Data.xml"
x:Key="xmlSource" XPath="Categories"/>
```

3. Set ItemsSource property for the MenuAdv.

XML

```
<syncfusion:MenuAdv ItemsSource="{Binding Source={StaticResource
xmlSource}, XPath=Root}" >
<syncfusion:MenuAdv.ItemTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding
XPath=SubItem}" >
<TextBlock Text="{Binding XPath=@Name}" />
```

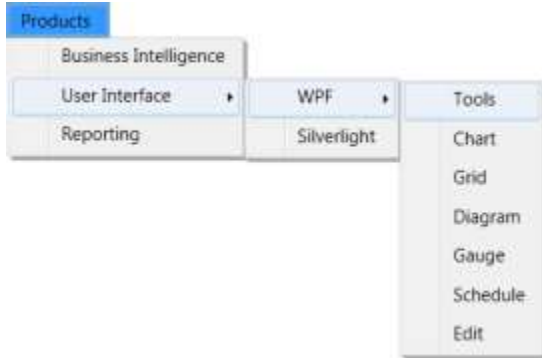


```

</HierarchicalDataTemplate>
</syncfusion:MenuAdv.ItemTemplate>
</syncfusion:MenuAdv>

```

This will create the following MenuAdv control.



Customizing Data Templates in WPF Menu (MenuAdv)

Data templates can be customized for items of the MenuAdv control. The next section explains how to customize the MenuItemAdv using data templates.

Item Template

You can customize how a business object is displayed as MenuItemAdv using ItemTemplate of MenuAdv. The MenuAdv displays hierarchical data. HierarchicalDataTemplate is used to define the ItemTemplate. The following code example shows the usage of ItemTemplate.

XML

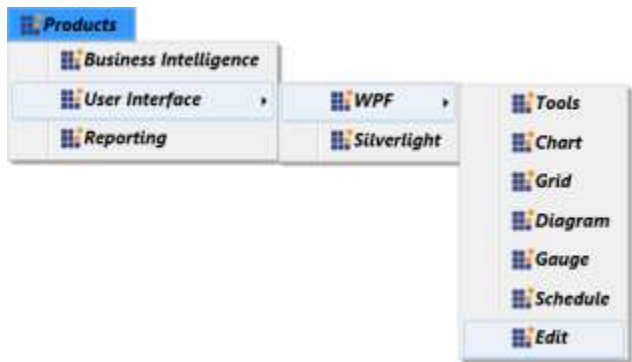
```

<syncfusion:MenuAdv ItemsSource="{Binding MenuItems}" >
<syncfusion:MenuAdv.ItemTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding SubItems}">
<StackPanel Orientation="Horizontal">
<Image Source="App.ico" Width="15" Height="15"/>
<TextBlock Text="{Binding Header}" FontWeight="Bold"
FontStyle="Italic" />
</StackPanel>
</HierarchicalDataTemplate>
</syncfusion:MenuAdv.ItemTemplate>
</syncfusion:MenuAdv>

```

Note: [View sample in GitHub](#)

Implementing the above code will generate the following MenuAdv control.



Binding and DataTemplate Support in WPF Menu (MenuAdv)

Data Binding is the process of establishing a connection between the application User Interface (UI) and Business logic. Data Binding can be unidirectional (source to target or target to source) or bidirectional (source to target and target to source). Users can bind a collection of Business objects by using the ItemsSource property of MenuItemAdv. The Hierarchical data template allows users to bind data that is hierarchical and self-referencing.

Use Case Scenario

[WPF Menu](#) (MenuAdv) helps users to bind a collection of business objects to MenuItemAdv and bind hierarchical levels of data to MenuItemAdv.

Adding the Binding and DataTemplate Support to an Application

The data source can be bound to MenuItemAdv by using the ItemsSource property. When binding the DataTemplate or HierarchicalDataTemplate to the MenuItemAdv, the Business objects can be bound by using the ItemTemplate property of MenuItemAdv. Binding and DataTemplate support can be added to an application, as shown in the following code snippets.

XML

```
<shared:MenuAdv x:Name="Menu">
  <shared:MenuItemAdv Header="File"/>
  <shared:MenuItemAdv Header="Edit"/>
  <shared:MenuItemAdv Header="View"/>
  <shared:MenuItemAdv Header="Project"/>
  <shared:MenuItemAdv Header="Data Binding">
    <shared:MenuItemAdv Header="DataTemplate Example">
      <shared:MenuItemAdv.ItemsSource>
        <local:MenuListCollection/>
      </shared:MenuItemAdv.ItemsSource>
      <shared:MenuItemAdv.ItemTemplate>
        <DataTemplate>
          <TextBlock Text="{Binding Caption}"/>
        </DataTemplate>
      </shared:MenuItemAdv.ItemTemplate>
    </shared:MenuItemAdv>
    <shared:MenuItemAdv Header="HierarchicalDataTemplateExample">
      <shared:MenuItemAdv.ItemsSource>
        <local:CustomItemsSource/>
      </shared:MenuItemAdv.ItemsSource>
      <shared:MenuItemAdv.ItemTemplate>
        <hdt:HierarchicalDataTemplate ItemsSource="{Binding Items}">
          <TextBlock Text="{Binding myString}"/>
        </hdt:HierarchicalDataTemplate>
      </shared:MenuItemAdv.ItemTemplate>
    </shared:MenuItemAdv>
  </shared:MenuItemAdv>
</shared:MenuAdv>
```

```

</shared:MenuItemAdv.ItemTemplate>
</shared:MenuItemAdv>
</shared:MenuItemAdv>
</shared:MenuAdv><

```

C#

```

public class MenuList
{
    public int MenuID { get; set; }
    public string Caption { get; set; }
    public MenuList() { }
    public MenuList(string capt, int id)
    {
        Caption = capt;
        MenuID = id;
    }
}

public class MenuListCollection : ObservableCollection<MenuList>
{
    public MenuListCollection()
    {
        for (int i = 0; i < 5; i++)
        {
            this.Add(new MenuList()
            {
                MenuID = i, Caption = "Menu " + (i + 1).ToString() });
        }
    }
}

public class ObjectModel
{
    public ObjectModel(string myString1, params ObjectModel[] myItems)
    {
        myString = myString1;
        ObservableCollection<ObjectModel> itemsObservableCollection = new
        ObservableCollection<ObjectModel>();
        foreach (var item in myItems)
        {
            itemsObservableCollection.Add(item);
        }
        Items = itemsObservableCollection;
    }
    public string myString { get; set; }
    public ObservableCollection<ObjectModel> Items { get; set; }}

public class CustomItemSource : ObservableCollection<ObjectModel>
{
    public CustomItemSource()
    {
        this.Add(new ObjectModel("User Interface Edition",
        new ObjectModel("ASP.NET", new ObjectModel("Tools",
        new ObjectModel("Menu"), new ObjectModel("Toolbar"),
        new ObjectModel("RangeSlider")), new ObjectModel("Diagram"),
        new ObjectModel("Gauge"), new ObjectModel("Chart")),
        new ObjectModel("ASP.NET MVC"), new ObjectModel("Windows Forms"),
        new ObjectModel("WPF", new ObjectModel("Tools", new

```

```

ObjectModel("Menu"), new ObjectModel("Toolbar"), new
ObjectModel("RangeSlider")), new ObjectModel("Diagram"), new
ObjectModel("Gauge"), new ObjectModel("Chart"), new
ObjectModel("Silverlight", new ObjectModel("Tools", new
ObjectModel("Menu"), new ObjectModel("Toolbar"), new
ObjectModel("RangeSlider")), new ObjectModel("Diagram"), new
ObjectModel("Gauge"), new ObjectModel("Chart"))));
this.Add(new ObjectModel("Reporting Edition",
new ObjectModel("Essential Reports"), new ObjectModel
("Essential XlsIO"), new ObjectModel("Essential DocIO"),
new ObjectModel("Essential PDF"), new ObjectModel("Essential
Calculate"), new ObjectModel("Essential Grouping"))));
this.Add(new ObjectModel("Business Intelligence",
new ObjectModel("Essential BI Chart"), new ObjectModel
("Essential BI Client"), new ObjectModel("Essential BI Grid"),
new ObjectModel("Essential BI Gauge"))));
}
}

```



Sample Link

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

Command Binding and Command Target Support in WPF Menu (MenuAdv)

Commands are a way to bind the UI to the logic that performs the action, which is to be executed. MenuAdv supports command binding. When users press the Enter key or click to select an item, the command will be triggered. This can be attained by using the Command and CommandParameter properties of MenuItemAdv.

The command target is the element on which the command is to be executed with regards to a RoutedCommand and routing of the Executed and CanExecute starts. This can be attained by using the CommandTarget property of MenuItemAdv.

Use Case Scenarios

MenuAdv helps users handle any command that can be routed outside the boundaries of the logical tree and do not require handling logic in code behind.

Using the Command Binding Support in an Application

To use the Command Binding support in an application users have to create a DelegateCommand class, which is obtained from the ICommand interface in the ViewModel sample class, which can be used to bind the command in the sample WPF application. The Command can be bound to MenuItemAdv by using the Command property and the target element can be bound to MenuItemAdv by using the CommandTarget property, as shown in the following code snippets.

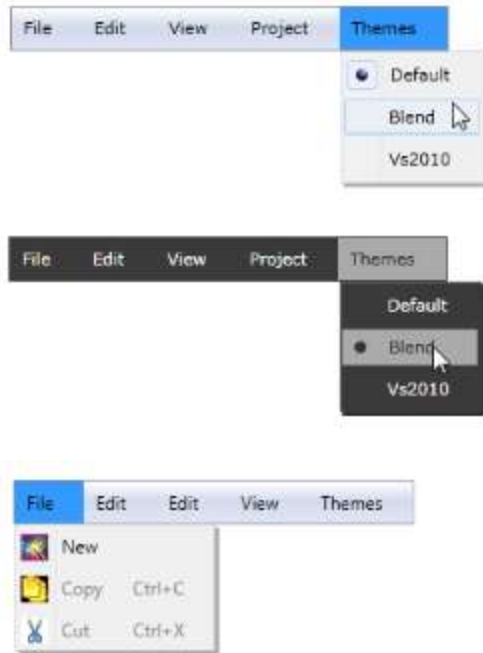
XML

```
<shared:MenuAdv x:Name="Menu">
  <shared:MenuItemAdv Header="File">
    <shared:MenuItemAdv Header="New">
      <shared:MenuItemAdv.Icon>
        <Image Source="/MenuControlDemo; component/Images/NewIcon.jpg"/>
      </shared:MenuItemAdv.Icon>
    </shared:MenuItemAdv>
    <shared:MenuItemAdv Command="Copy" CommandTarget="{Binding
      ElementName=textbox}">
      <shared:MenuItemAdv.Icon>
        <Image Source="/MenuControlDemo; component/Images/CopyIcon.jpg"/>
      </shared:MenuItemAdv.Icon>
    </shared:MenuItemAdv>
    <shared:MenuItemAdv Command="Cut" CommandTarget="{Binding
      ElementName=textbox}">
      <shared:MenuItemAdv.Icon>
        <Image Source="/MenuControlDemo; component/Images/CutIcon.jpg"/>
      </shared:MenuItemAdv.Icon>
    </shared:MenuItemAdv>
    <shared:MenuItemAdv Header="Edit"/>
    <shared:MenuItemAdv Header="View"/>
    <shared:MenuItemAdv Header="Themes">
      <shared:MenuItemAdv IsCheckable="True" Header="Default"
        CheckIconType="RadioButton" GroupName="Themes" IsChecked="True"
        CommandParameter="Default" Command="{Binding ElementName=root,
          Path=MyCommand}" />
      <shared:MenuItemAdv IsCheckable="True" Header="Blend"
        CheckIconType="RadioButton" GroupName="Themes" IsChecked="False"
        CommandParameter="Blend" Command="{Binding ElementName=root,
          Path=MyCommand}" />
      <shared:MenuItemAdv IsCheckable="True" Header="Vs2010"
        CheckIconType="RadioButton" GroupName="Themes" IsChecked="False"
        CommandParameter="Vs2010" Command="{Binding ElementName=root,
          Path=MyCommand}" />
    </shared:MenuItemAdv>
  </shared:MenuAdv>
</shared:MenuAdv><TextBox x:Name="textbox"/>
```

C#

```
public partial class MainPage : UserControl
{
  public MainPage ()
  {
```

```
InitializeComponent();
}
private DelegateCommand myCommand;
public DelegateCommand MyCommand
{
    get
    {
        if (myCommand == null)
        {
            myCommand = new DelegateCommand(ApplyTheme);
        }
        return myCommand;
    }
}
private void ApplyTheme(object visualStyle)
{
    SkinManager.SetVisualStyle(this, (Syncfusion.Windows.Controls.Theming.VisualStyle) (visualStyle));
}
}
public class DelegateCommand : ICommand
{
    public event EventHandler CanExecuteChanged;
    readonly Predicate<Object> _canExecute = null;
    readonly Action<Object> _executeAction = null;
    public DelegateCommand(Action<object> executeAction, Predicate<Object> canExecute)
    {
        _executeAction = executeAction;
        _canExecute = canExecute;
    }
    public DelegateCommand(Action<object> executeAction) : this(executeAction, null)
    {
        _executeAction = executeAction;
    }
    public void UpdateCanExecute()
    {
        if (CanExecuteChanged != null)
        CanExecuteChanged(this, new EventArgs());
    }
    public bool CanExecute(object parameter)
    {
        return _canExecute == null || _canExecute(parameter);
    }
    public void Execute(object parameter)
    {
        if (_executeAction != null)
        _executeAction(parameter);
        UpdateCanExecute();
    }
}
```



Properties

The properties for the Command Binding support are described in the following tabulation:

Property	Description	Type	Data Type
Command	Gets or sets the Command of MenuItemAdv.	DependencyProperty	ICommand(null)
CommandParameter	Gets or sets the CommandParameter of MenuItemAdv.	DependencyProperty	String(null)
CommandTarget	Gets or sets the CommandTarget of MenuItemAdv.	DependencyProperty	IInputElement(null)

Sample Link

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

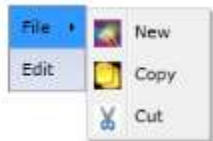
Orientation Support in WPF Menu (MenuAdv)

The MenuAdv control can align its content vertically and horizontally by using the Orientation property of the MenuAdv class.

When the value of the Orientation property is set to Horizontal, the Items of MenuAdv will be arranged horizontally.



Similarly, when the value of the Orientation property is set to Vertical, the Items of MenuAdv will be arranged vertically.



Use Case Scenarios

MenuAdv helps users to set the Menu items in Horizontal or Vertical orientations.

Adding the Orientation Support to an Application

Users can add the Orientation support to MenuAdv used in the application as mentioned in the code snippet below.

XML

```
<shared:MenuAdv x:Name="Menu" Orientation="Horizontal"/>
<shared:MenuItemAdv Header="File">
  <shared:MenuItemAdv Header="New">
    <shared:MenuItemAdv.Icon>
      <Image Source="/MenuControlDemo;component/Images/NewIcon.jpg"/>
    </shared:MenuItemAdv.Icon>
  </shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Copy">
    <shared:MenuItemAdv.Icon>
      <Image Source="/MenuControlDemo;component/Images/CopyIcon.jpg"/>
    </shared:MenuItemAdv.Icon>
  </shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Cut">
    <shared:MenuItemAdv.Icon>
      <Image Source="/MenuControlDemo;component/Images/CutIcon.jpg"/>
    </shared:MenuItemAdv.Icon>
  </shared:MenuItemAdv>
</shared:MenuItemAdv>
<shared:MenuItemAdv Header="Edit"/>
</shared:MenuAdv>
```

Properties

The property for the Orientation support is described in the following tabulation:

Property	Description	Type	Data Type
Orientation	Gets or sets the Orientation of MenuAdv.	DependencyProperty	Orientation(Horizontal)

[Sample Link](#)

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

Expand Modes Support

Expand Modes in MenuAdv is used to open the submenu of each MenuItemAdv, which is added in MenuAdv by doing click to open the submenu or by doing mouse hover to open the submenu. MenuAdv supports two kinds of Expand Modes, namely ExpandOnClick and ExpandOnMouseOver, which can be obtained by using the ExpandMode property.

[Use Case Scenarios](#)

MenuAdv helps users to use Expand Modes to open the submenu of MenuItemAdv added in MenuAdv.

[Using the Expand Modes Support in an Application](#)

When the ExpandMode property is set to ExpandOnClick, you can open the submenu of each MenuItemAdv, which is added MenuAdv by clicking it. This type of expand mode is used to open menus in Windows operating system. Similarly, when the ExpandMode property is set to ExpandOnMouseOver, you can open the submenu of each MenuItemAdv, which is added in MenuAdv by moving the mouse pointer over it. Therefore, you need not click to open the submenu. You can achieve this functionality by using the ExpandMode property, as shown in the following code snippet.

XML

```
<shared:MenuAdv x:Name="Menu" ExpandMode="ExpandOnMouseOver"/>
<shared:MenuItemAdv Header="File">
  <shared:MenuItemAdv Header="New">
    <shared:MenuItemAdv.Icon>
      <Image Source="/MenuControlDemo;component/Images/NewIcon.jpg"/>
    </shared:MenuItemAdv.Icon>
  </shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Copy">
    <shared:MenuItemAdv.Icon>
      <Image Source="/MenuControlDemo;component/Images/CopyIcon.jpg"/>
    </shared:MenuItemAdv.Icon>
  </shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Cut">
    <shared:MenuItemAdv.Icon>
      <Image Source="/MenuControlDemo;component/Images/CutIcon.jpg"/>
    </shared:MenuItemAdv.Icon>
  </shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Edit"/>
</shared:MenuAdv>
```

[Properties](#)

The property for the Expand Modes support is described in the following tabulation:

Property	Description	Type	Data Type
ExpandMode	Gets or sets the ExpandMode of MenuAdv.	DependencyProperty	ExpandModes(ExpandOnClick)

[Sample Link](#)

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

Icon Support in WPF Menu (MenuAdv)

MenuItemAdv allows users to display an image on the left of the control. Icon for MenuItemAdv can be set by providing the image source as a value for the Icon property of the MenuItemAdv class.

Use Case Scenarios

MenuAdv helps users to display an image on the left of the control.

Adding the Icon Support to an Application

The Icon support can be added to an application by using the Icon property of MenuItemAdv, as shown in the following code snippet.

XML

```
<shared:MenuAdv x:Name="Menu"/>
<shared:MenuItemAdv Header="File">
  <shared:MenuItemAdv Header="New">
    <shared:MenuItemAdv.Icon>
      <Image Source="/MenuControlDemo;component/Images/NewIcon.jpg"/>
    </shared:MenuItemAdv.Icon>
  </shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Copy">
    <shared:MenuItemAdv.Icon>
      <Image Source="/MenuControlDemo;component/Images/CopyIcon.jpg"/>
    </shared:MenuItemAdv.Icon>
  </shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Cut">
    <shared:MenuItemAdv.Icon>
      <Image Source="/MenuControlDemo;component/Images/CutIcon.jpg"/>
    </shared:MenuItemAdv.Icon>
  </shared:MenuItemAdv>
  </shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Edit"/>
</shared:MenuAdv>
```



Properties

The property for the Icon support is described in the following tabulation:

Property	Description	Type	Data Type
Icon	Gets or sets the Icon of MenuItemAdv.	DependencyProperty	Object(null)

Sample Link

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

Check Box and Radio Button Support in WPF Menu (MenuAdv)

MenuAdv provides a support for selecting several items. MenuItemAdv can be checked by setting the IsCheckable property of the MenuItemAdv to "true". You can change the icon type (Check Box or Radio Button) by using the CheckIcon property also it can be checked by using the IsChecked property.

Use Case Scenarios

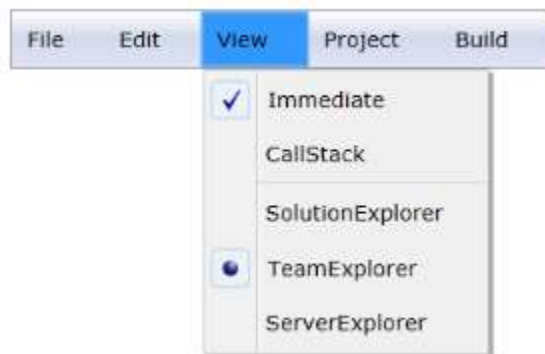
MenuAdv helps users to use MenuItemAdv with the CheckBox or RadioButton support.

Using the Check Box and Radio Button Support in an Application

If you set the CheckIcon property to RadioButton, then MenuItemAdv's will be grouped based on the value of the GroupName property and end user can select only one item from the group. Similarly, if you set the CheckIcon property to CheckBox, then MenuItemAdv can be checked and unchecked. The Check Box and Radio Button support can be used in an application, as shown in the following the code snippet.

XML

```
<shared:MenuAdv x:Name="Menu" Margin="10">
  <shared:MenuItemAdv Header="File"/>
  <shared:MenuItemAdv Header="Edit"/>
  <shared:MenuItemAdv Header="View">
    <shared:MenuItemAdv Header="Immediate"
      IsCheckable="True" CheckIconType="CheckBox" IsChecked="True"/>
    <shared:MenuItemAdv Header="CallStack"
      IsCheckable="True" CheckIconType="CheckBox" IsChecked="False"/>
    <shared:MenuItemSeparator/>
    <shared:MenuItemAdv Header="SolutionExplorer"
      IsCheckable="True" CheckIconType="RadioButton" GroupName="group1"
      IsChecked="False"/>
    <shared:MenuItemAdv Header="TeamExplorer"
      IsCheckable="True" CheckIconType="RadioButton" GroupName="group1"
      IsChecked="True"/>
    <shared:MenuItemAdv Header="ServerExplorer"
      IsCheckable="True" CheckIconType="RadioButton" GroupName="group1"
      IsChecked="False"/>
  </shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Project"/>
  <shared:MenuItemAdv Header="Build"/>
</shared:MenuAdv>
```



Properties

The properties for the Check box and Radio button support are described in the following tabulation:

Property	Description	Type	Data Type
IsCheckable	Gets or sets the value of IsCheckable in MenuItemAdv.	DependencyProperty	bool(false)
CheckIconType	Gets or sets the CheckIconType of MenuItemAdv.	DependencyProperty	CheckIconType(CheckBox)
IsChecked	Gets or sets the IsChecked value of MenuItemAdv.	DependencyProperty	bool(false)

[Sample Link](#)

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

[InputGestureText Support in WPF Menu \(MenuAdv\)](#)

InputGestureText is used to set MenuItemAdv to display shortcut keys along with its Header. This support can be utilized by using the InputGestureText property. The value given by using this property will be displayed along with the Header of MenuItemAdv. Also, InputGestureText can be displayed in MenuItemAdv by using the command support. If you set the value of the Command property by using ApplicationCommands the corresponding InputGestureText will be displayed along with its header value automatically.

[Use Case Scenarios](#)

MenuAdv helps users to display the shortcut keys along with the MenuItemAdv header.

[Adding the InputGestureText Support to an Application](#)

The value assigned by using the InputGestureText property will be displayed in MenuItemAdv along with the Header property value of MenuItemAdv. The InputGestureText support can be added to an application, as shown in the following code snippet.

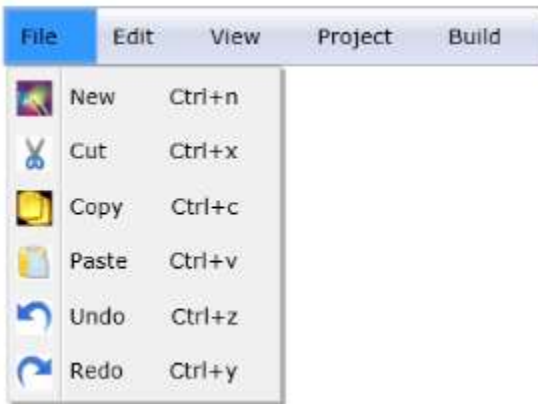
XML

```
<shared:MenuAdv x:Name="Menu" Margin="10">
  <shared:MenuItemAdv Header="File">
    <shared:MenuItemAdv Header="New" InputGestureText="Ctrl+n">
      <shared:MenuItemAdv.Icon>
        <Image Source="/MenuControlDemo;component/Images/NewIcon.jpg"/>
      </shared:MenuItemAdv.Icon>
    </shared:MenuItemAdv>
    <shared:MenuItemAdv Header="Cut" InputGestureText="Ctrl+x">
      <shared:MenuItemAdv.Icon>
        <Image Source="/MenuControlDemo;component/Images/CutIcon.jpg"/>
      </shared:MenuItemAdv.Icon>
    </shared:MenuItemAdv>
    <shared:MenuItemAdv Header="Copy" InputGestureText="Ctrl+c">
      <shared:MenuItemAdv.Icon>
        <Image Source="/MenuControlDemo;component/Images/CopyIcon.jpg"/>
      </shared:MenuItemAdv.Icon>
    </shared:MenuItemAdv>
    <shared:MenuItemAdv Header="Paste" InputGestureText="Ctrl+v">
      <shared:MenuItemAdv.Icon>
        <Image Source="/MenuControlDemo;component/Images/PasteIcon.jpg"/>
      </shared:MenuItemAdv.Icon>
    </shared:MenuItemAdv>
  </shared:MenuItemAdv>
</shared:MenuAdv>
```

```

<shared:MenuItemAdv Header="Undo" InputGestureText="Ctrl+z">
<shared:MenuItemAdv.Icon>
<Image Source="/MenuControlDemo;component/Images/UndoIcon.jpg"/>
</shared:MenuItemAdv.Icon>
</shared:MenuItemAdv>
<shared:MenuItemAdv Header="Redo" InputGestureText="Ctrl+y">
<shared:MenuItemAdv.Icon>
<Image Source="/MenuControlDemo;component/Images/RedoIcon.jpg"/>
</shared:MenuItemAdv.Icon>
</shared:MenuItemAdv>
<shared:MenuItemSeparator/>
</shared:MenuItemAdv>
<shared:MenuItemAdv Header="Edit"/>
<shared:MenuItemAdv Header="View"/>
<shared:MenuItemAdv Header="Project"/>
<shared:MenuItemAdv Header="Build"/>
</shared:MenuAdv>

```



Properties

The property for the InputGestureText support is described in the following tabulation:

Property	Description	Type	Data Type
InputGestureText	Gets or sets the value of InputGestureText of MenuItemAdv.	DependencyProperty	String(string.Empty)

Sample Link

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

MenuItemSeparator Support in WPF Menu (MenuAdv)

MenuItemSeparator is a line, which is used to separate MenuItemAdv's. MenuItemSeparator can be included in the items list of MenuItemAdv.

Use Case Scenarios

MenuAdv helps users to separate MenuItemAdv's by using MenuItemSeparator. In the case of separating the radio button group of items from other items, separator can be used.

Adding the MenuItemSeparator Support to an Application

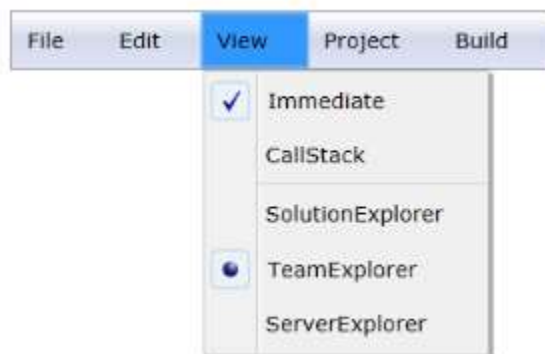
MenuItemSeparator can be added to an application, as shown in the following code snippet.

XML

```

<shared:MenuAdv x:Name="Menu" Margin="10">
<shared:MenuItemAdv Header="File"/>
<shared:MenuItemAdv Header="Edit"/>
<shared:MenuItemAdv Header="View">
<shared:MenuItemAdv Header="Immediate"
IsCheckable="True" CheckIconType="CheckBox" IsChecked="True"/>
<shared:MenuItemAdv Header="CallStack"
IsCheckable="True" CheckIconType="CheckBox" IsChecked="False"/>
<shared:MenuItemSeparator/>
<shared:MenuItemAdv Header="SolutionExplorer"
IsCheckable="True" CheckIconType="RadioButton" GroupName="group1"
IsChecked="False"/>
<shared:MenuItemAdv Header="TeamExplorer"
IsCheckable="True" CheckIconType="RadioButton" GroupName="group1"
IsChecked="True"/>
<shared:MenuItemAdv Header="ServerExplorer"
IsCheckable="True" CheckIconType="RadioButton" GroupName="group1"
IsChecked="False"/>
</shared:MenuItemAdv>
<shared:MenuItemAdv Header="Project"/>
<shared:MenuItemAdv Header="Build"/>
</shared:MenuAdv>

```

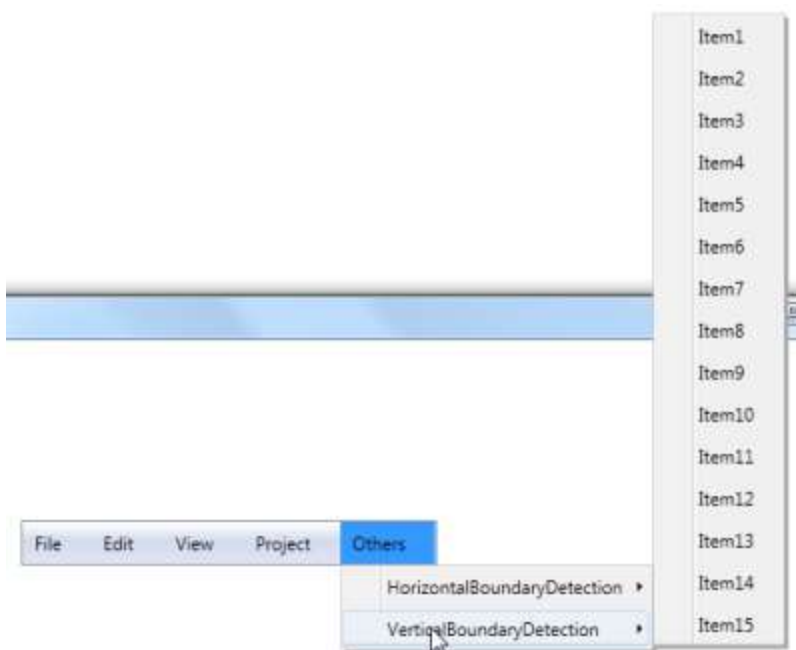
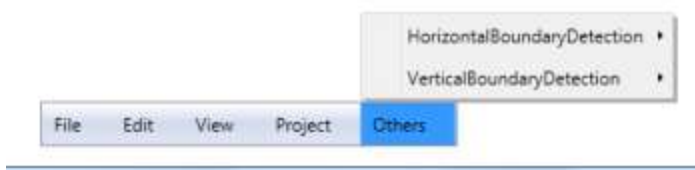
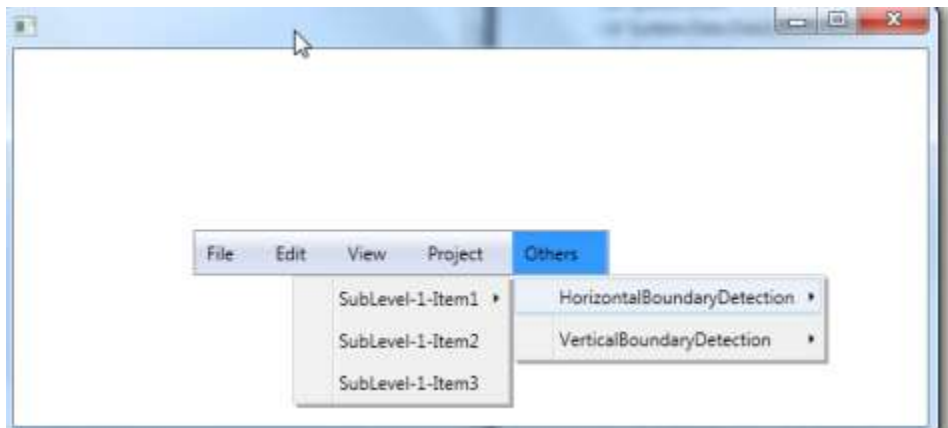


[Sample Link](#)

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

Boundary Detection in WPF Menu (MenuAdv)

MenuItemAdv detects the boundaries and opens its submenu in the opposite direction, when the submenu crosses the boundary in both horizontal and vertical directions. In the case of Vertical direction, if the MenuItemAdv which opens the submenu is not the item of MenuAdv the submenu will only adjust its position and it will not open in opposite direction.



Use Case Scenarios

MenuAdv will be very useful when the number of submenu item levels are more and the opening of the submenu crosses the boundary, this feature allows the submenu be always visible by adjusting its position.

Sample Link

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

Scroll Support in WPF Menu (MenuAdv)

MenuAdv allows users to scroll through the submenu items so that all the items of the submenu are visible even if the submenu crosses the vertical boundary. The Scroll support can be enabled by setting the `IsScrollEnabled` property to true. If the `IsScrollEnabled` property is set to false, users will not be able to scroll through the submenu items and the items that cross the vertical boundary will not be visible.



Use Case Scenarios

MenuAdv will be very useful in the case of adding more number of items to the single `MenuItemAdv` and the size of the submenu crosses the vertical boundary.

Using the Scroll Support in an Application

If the value of the `IsScrollEnabled` property is set to true, users can scroll through the submenu items by using the `TopScrollButton`, `BottomScrollButton`, and mouse wheel. If the value of the `IsScrollEnabled` property is set to false, users will not be able to scroll through the submenu items.

Properties

The property for the Scroll support is described in the following tabulation:

Property	Description	Type	Data Type
<code>IsScrollEnabled</code>	Gets or sets <code>IsScrollEnabled</code> of <code>MenuAdv</code> .	<code>DependencyProperty</code>	<code>bool(true)</code>

Sample Link

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

Animation Support in WPF Menu (MenuAdv)

MenuAdv supports animation types to open the submenu pop-up. The following animation types are supported by MenuAdv:

- Fade
- Slide
- Scroll

The Animation support can be used by using the `PopUpAnimationType` property. If the `PopUpAnimationType` property is set to `None`, the submenu will open without any animation.

Adding the Animation Support to an Application

If the `PopUpAnimationType` property is set to `Fade`, the submenu will open with faded animation. If the `PopUpAnimationType` property is set to `Slide`, the submenu will open like the slide. If the `PopUpAnimationType` property is set to `Scroll`, the submenu popup open with scroll animation. The Animation support can be added to an application, as shown in the following code snippet.

XML

```
<shared:MenuAdv x:Name="Menu" Margin="10" PopUpAnimationType="Slide">
  <shared:MenuItemAdv Header="File"/>
  <shared:MenuItemAdv Header="Edit"/>
  <shared:MenuItemAdv Header="View"/>
  <shared:MenuItemAdv Header="Project"/>
  <shared:MenuItemAdv Header="VerticalAnimation">
  <shared:MenuItemAdv Header="HorizontalAnimation">
  <shared:MenuItemAdv Header="Item1"/>
  <shared:MenuItemAdv Header="Item2"/>
  <shared:MenuItemAdv Header="Item3"/>
  <shared:MenuItemAdv Header="Item4"/>
  <shared:MenuItemAdv Header="Item5"/>
</shared:MenuItemAdv>
  <shared:MenuItemAdv Header="Item1"/>
  <shared:MenuItemAdv Header="Item2"/>
  <shared:MenuItemAdv Header="Item3"/>
  <shared:MenuItemAdv Header="Item4"/>
  <shared:MenuItemAdv Header="Item5"/>
</shared:MenuItemAdv>
</shared:MenuAdv>
```

Properties

The property for the Animation support is described in the following tabulation:

Property	Description	Type	Data Type
PopUpAnimationType	Gets or sets the PopUpAnimationType of MenuAdv.	DependencyProperty	PopUpAnimationType(None)

Sample Link

WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

Keyboard Navigation Support in WPF Menu (MenuAdv)

The MenuAdv control supports keyboard navigation, which allows users to select MenuItemAdv to open and close the submenu. The following keys are used for keyboard navigation:

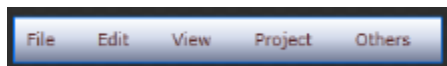
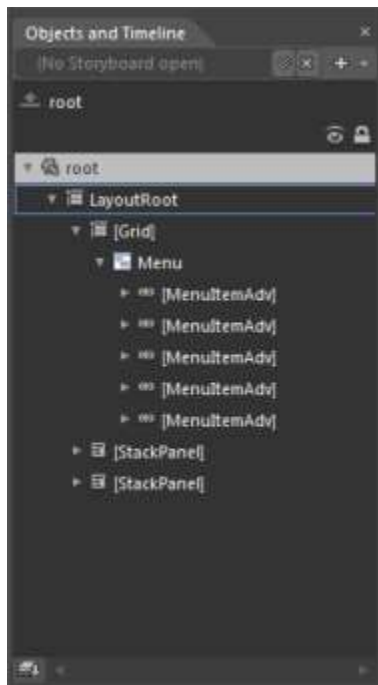
- Arrow keys- The Arrow keys can be used to navigate to items in the submenu and to open the submenu.
- Enter key - The Enter key can be used to select an item and to open the submenu.
- Esc key - The Esc key can be used to close the submenu.
- Tab key - The Tab key can be used to navigate to items inside the submenu.

[Sample Link](#)

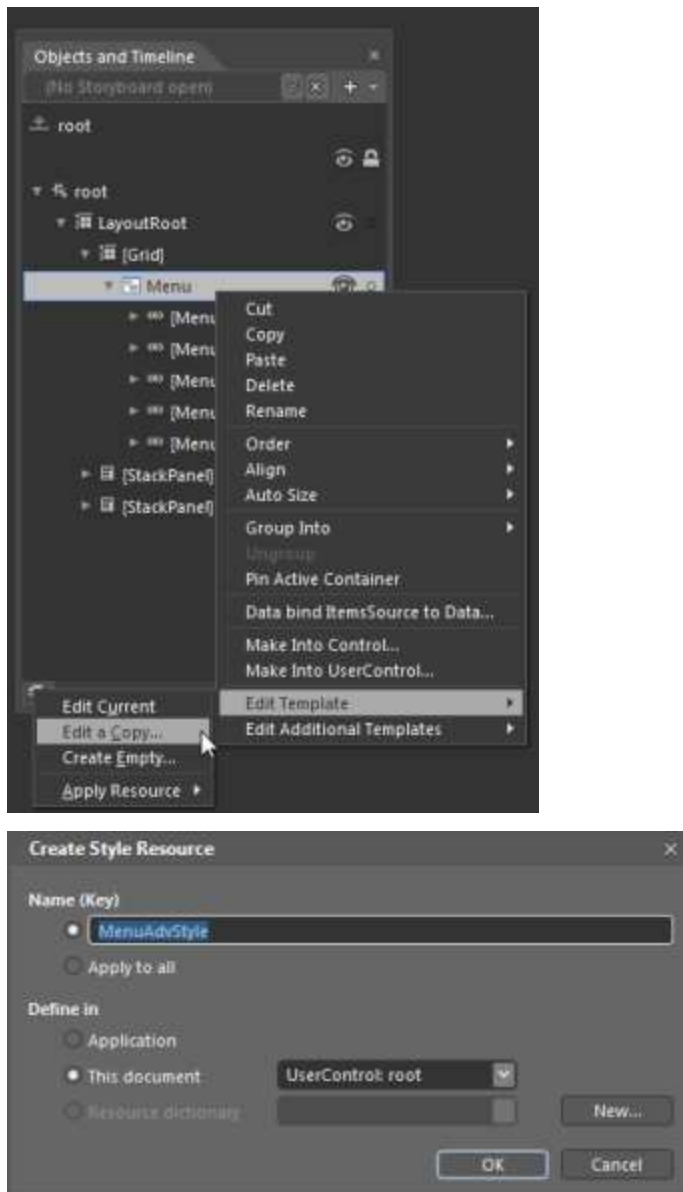
WPF Sample Browser-> Tools -> MenuAdv -> MenuAdv Demo

Blendability Support in WPF Menu (MenuAdv)

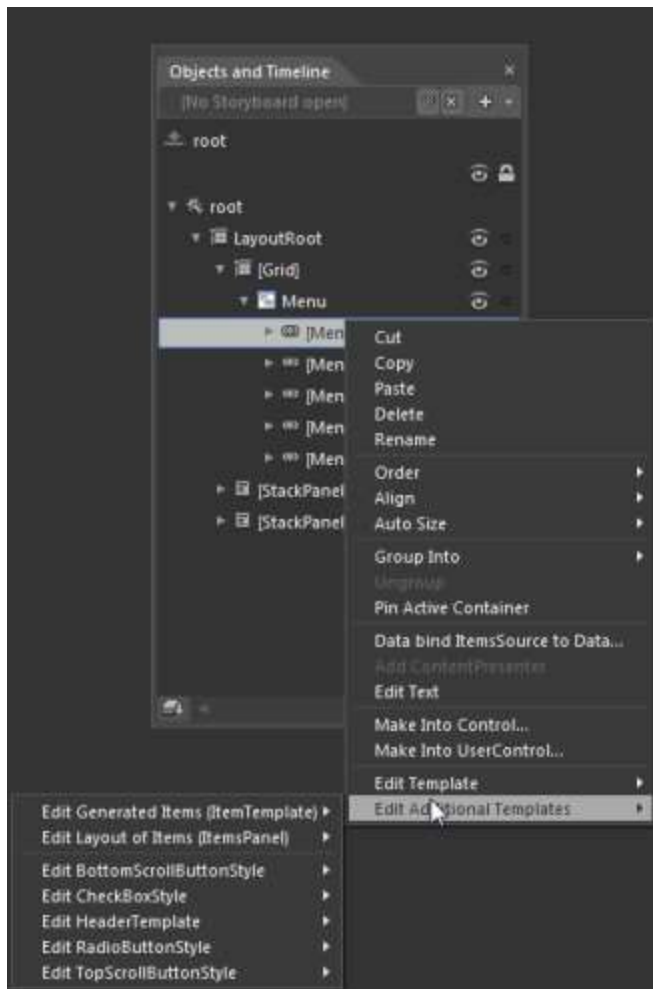
You can customize MenuAdv and MenuItemAdv in Expression Blend. After adding MenuAdv and MenuItemAdv to the design view, you can see MenuAdv and MenuItemAdv in the Objects and Timeline window.



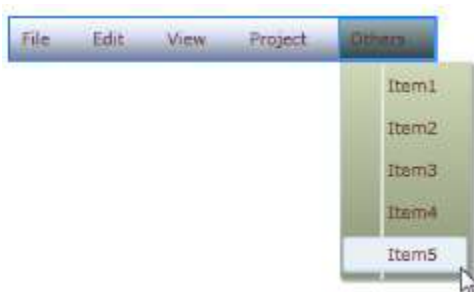
Right-click MenuAdv and in EditTemplate option select Edit a Copy and assign the key name to the resource edited. The same process can be repeated to edit the MenuItemAdv style.



User can also customize the appearance of the TopScrollButtonStyle, BottomScrollButtonStyle, CheckBoxStyle & RadioButtonStyle using the Edit Additional Templates option by doing right click on the MenuItemAdv and selecting the respective styles from the option.



The following screen shot shows the MenuAdv and MenuItemAdv templates edited and customized by using Expression Blend.

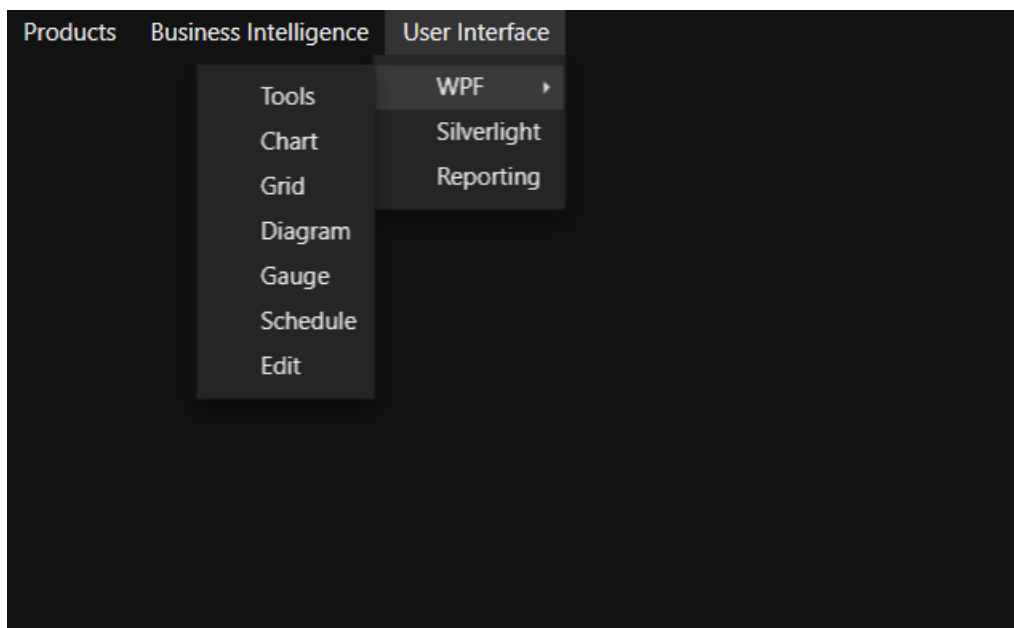


Styling in WPF Menu (MenuAdv)

Theme

MenuAdv supports various built-in themes. Refer to the below links to apply themes for the MenuAdv,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



SfMultiColumnDropDown

WPF Multi Column Dropdown (SfMultiColumnDropDown) Overview

Syncfusion's MultiColumnDropDown control is combined with an Editor (TextBox Control) and powerful SfDataGrid (DataGrid Control) to search and select items by using the DropDownGrid. The Editor accepts free-flow text that is used to filter the DropDownGrid.

Following are the key features of SfMultiColumnDropDownControl

- **Data binding** – Supports to bind different types of data sources.
- **Columns** – Support for various column types.
- **Editing** – Support for editing and autocomplete.
- **Selection** – Support for single and also multi selection.
- **Filtering** – Support for Case sensitive filtering and filtering delay.
- **Styling** – Support for customizing styles of popup.

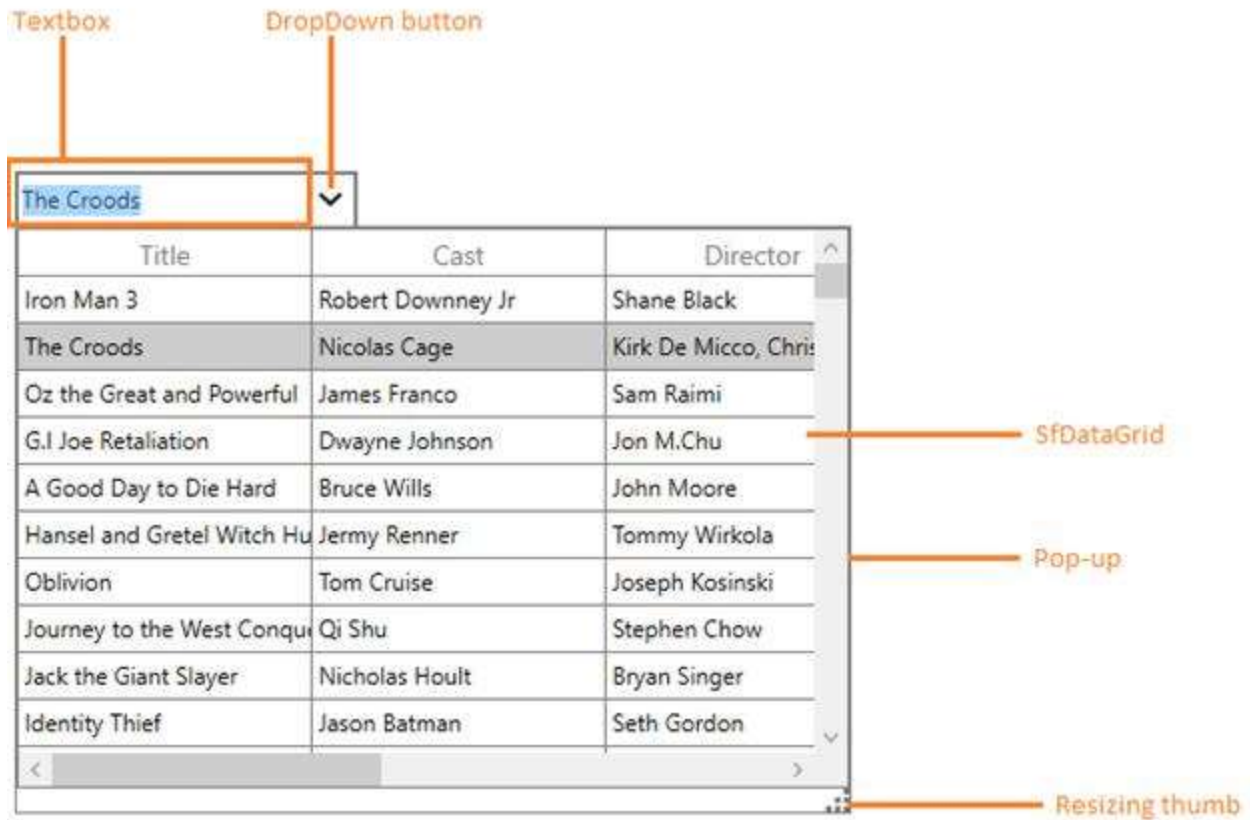
Getting Started with WPF Multi Column Dropdown (SfMultiColumnDropDown)

SfMultiColumnDropDownControl displays multiple columns in dropdown by embedding SfDataGrid control for rich look up selection.

Features,

- Auto complete support
- Support to customize the columns to be displayed
- Filter the data based on typed text
- Ability to customize popup width, height and resize at runtime
- Editor can be Read-only

You can [refer here](#) to know more about SfDataGrid.



Assembly deployment

The following list of assemblies needs to be added as reference to use **SfMultiColumnDropDownControl** control in any application,

Required assemblies	Description
Syncfusion.Data.WPF	Syncfusion.Data.WPF assembly contains fundamental and base classes for CollectionViewAdv 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 SfMultiColumnDropDownControl, DropDownGrid. SfMultiColumnDropDownControl control present in Syncfusion.UI.Xaml.Grid namespace. This namespace also added in http://schemas.syncfusion.com/wpf Syncfusion WPF schema.

Creating simple application with SfMultiColumnDropDownControl

In this walk through, you will create WPF application that contains **SfMultiColumnDropDownControl**.

1. [Creating project](#)
2. [Adding control via Designer](#)
3. [Adding control manually in XAML](#)
4. [Adding control manually in C#](#)

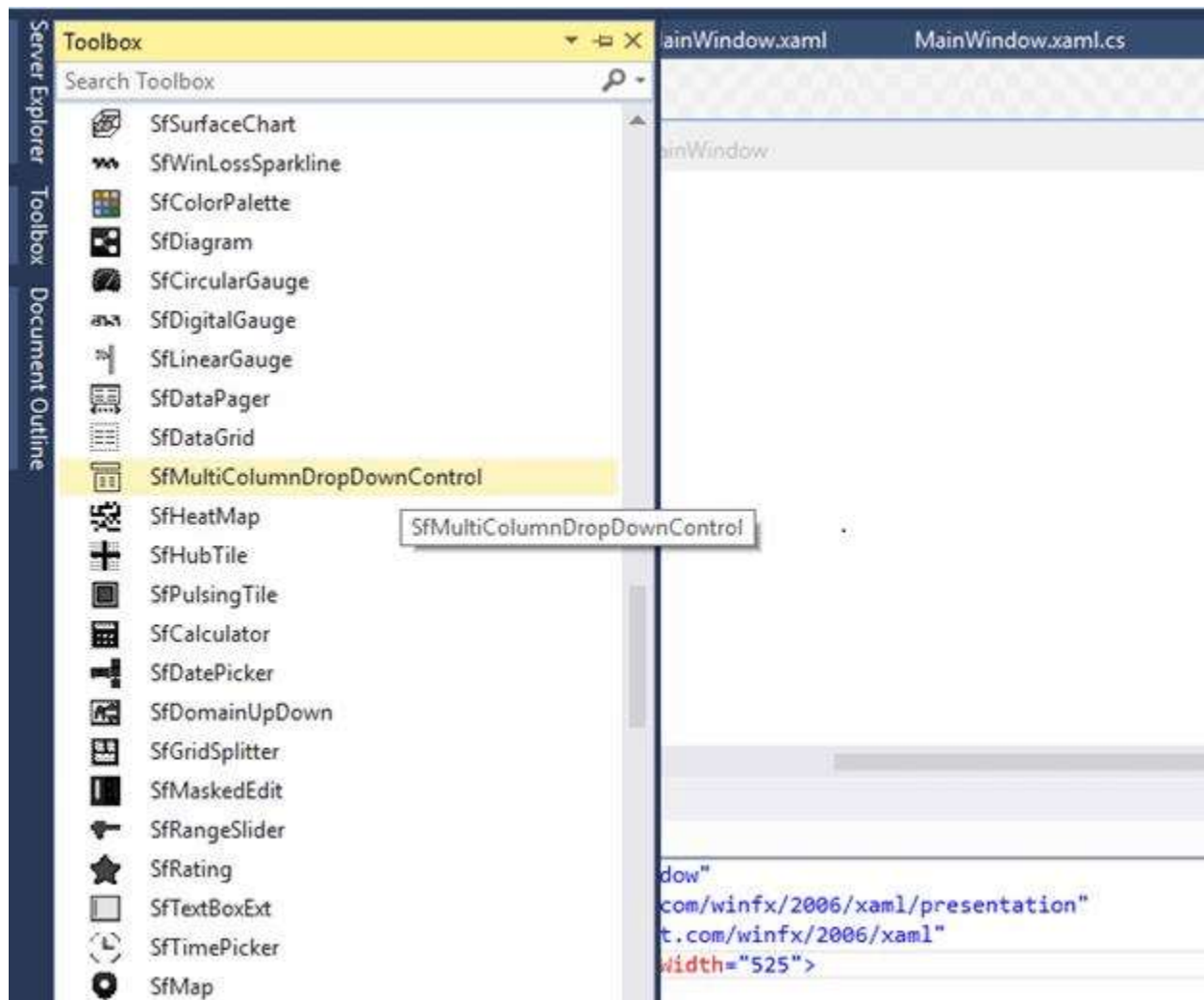
5. [Creating Data Model for application](#)
6. [Binding to Data](#)
7. [Defining DisplayMember and ValueMember](#)
8. [Defining Columns](#)
9. [Auto complete and filtering](#)

Creating the project

Create new WPF Project in Visual Studio to display SfMultiColumnDropDownControl with data objects.

Adding control via Designer

SfMultiColumnDropDownControl 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

2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> or SfMultiColumnDropDownControl namespace **Syncfusion.UI.Xaml.Grid** in XAML page.
3. Declare **SfMultiColumnDropDownControl** 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:SfMultiColumnDropDownControl x:Name="sfmultiColumn"/>
</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 references to the project,
 - o Syncfusion.Data.WPF
 - o Syncfusion.SfGrid.WPF
2. Import SfMultiColumnDropDownControl namespace Syncfusion.UI.Xaml.Grid.
3. Create **SfMultiColumnDropDownControl** instance and add it to the Page.

C#

```
using Syncfusion.UI.Xaml.Grid;
namespace WpfApplication1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfMultiColumnDropDownControl sfMultiColumn = new
            SfMultiColumnDropDownControl();
            Root_Grid.Children.Add(sfMultiColumn);
        }
    }
}
```

Creating Data Model for sample application

Before binding **ItemsSource** 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;
}
}

```

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 Trujilo", "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

You can populate the drop down list for SfMultiColumnDropDownControl by setting [ItemsSource](#) property.

Bind the collection created in previous step to `ItemsSource` property by setting ViewModel as DataContext.

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"
    xmlns:local="clr-namespace:WpfApplication1"
    Title="MainWindow" Height="350" Width="525">
    <Window.DataContext>
    <local:ViewModel/>
    </Window.DataContext>
    <Grid x:Name="Root_Grid">
    <syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
    ItemsSource="{Binding Orders}"
    DisplayMember="OrderID"
    Width="175"
    Height="30"
    SelectedIndex="2"/>
    </Grid>
</Window>

```

C#

```
ViewModel viewModel = new ViewModel();
sfMultiColumn.ItemsSource = viewModel.Orders;
```



OrderID	CustomerID	CustomerN
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 Citeau
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Leblanc
1010	BOTTM	Elizabeth Lincoln

Defining DisplayMember and ValueMember

[DisplayMember](#) denotes the path to a value on the data object for visual presentation of the selected object in editor and [ValueMember](#) denotes the path to a value on the data object for [SelectedValue](#).

Refer [here](#) to know more about the [DisplayMember](#) and [ValueMember](#).

Defining Columns

By default, the SfMultiColumnDropDownControl generates the columns automatically based on [ItemsSource](#) property. You can prevent the automatic column generation by setting [AutoGenerateColumns](#) property to `false`. When [AutoGenerateColumns](#) property is false, you have to define the columns to be displayed as below,

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
Width="175"
Height="30"
SelectedIndex="0"
AutoGenerateColumns="false"
DisplayMember="OrderID"
ItemsSource="{Binding Orders}">
  <syncfusion:SfMultiColumnDropDownControl.Columns>
    <syncfusion:GridTextColumn MappingName="OrderID" />
    <syncfusion:GridTextColumn MappingName="CustomerID" />
    <syncfusion:GridTextColumn MappingName="Country" />
  </syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```

C#

```
SfMultiColumnDropDownControl sfMultiColumn = new
SfMultiColumnDropDownControl();
sfMultiColumn.AutoGenerateColumns = false;
sfMultiColumn.Columns.Add(new GridTextColumn() { MappingName = "OrderID" });
sfMultiColumn.Columns.Add(new GridTextColumn() { MappingName = "CustomerID"
});
sfMultiColumn.Columns.Add(new GridTextColumn() { MappingName = "Country" });
```

Editing and filtering

SfMultiColumnDropDownControl provides support to auto append the text from **ItemsSource** when end-user edits in the TextBox by setting [AllowAutoComplete](#) to true.

Also, it provides support to filter the items displayed in the drop-down based on text in the TextBox by setting [AllowIncrementalFiltering](#) to true.

Theme

SfMultiColumnDropDownControl supports various built-in themes. Refer to the below links to apply themes for the SfMultiColumnDropDownControl,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

Title	Cast	D
Iron Man 3	Robert Downney Jr	Shane Bla
The Croods	Nicolas Cage	Kirk De M
Oz the Great and Powerful	James Franco	Sam Raim
G.I Joe Retaliation	Dwayne Johnson	Jon M.Chu
A Good Day to Die Hard	Bruce Wills	John Moo
Hansel and Gretel Witch H	Jermy Renner	Tommy W
Oblivion	Tom Cruise	Joseph Ko
Journey to the West Conq	Qi Shu	Stephen C
Jack the Giant Slayer	Nicholas Hoult	Bryan Sing
Identity Thief	Jason Batman	Seth Gord

Data Binding in WPF Multi Column Dropdown (SfMultiColumnDropDown)

You can populate the drop down list for SfMultiColumnDropDownControl by setting [ItemsSource](#) property.

[DisplayMember](#) denotes the path to a value on the data object for visual presentation of the Textbox and [ValueMember](#) denotes the path to a value on the data object to get the [SelectedValue](#).

XML

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="400" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
Width="175"
Height="30"
Grid.Row="0"
SelectedIndex="0"
DisplayMember="Title"
ValueMember="OrderID"
ItemsSource="{Binding Orders}" />
<StackPanel Grid.Row="1" Margin="100,0,0,0">
<TextBlock FontSize="16" Text="SelectedItem (Display Member) " />
<TextBlock FontSize="22"
FontWeight="Bold"
Text="{Binding ElementName= "sfMultiColumn",
Mode=TwoWay,
Path=SelectedItem.Title}" />
<TextBlock FontSize="16" Text="SelectedValue (Value Member) " />
<TextBlock FontSize="22"
FontWeight="Bold"
Text="{Binding ElementName= "sfMultiColumn",
Mode=TwoWay,
Path=SelectedValue}" />
</StackPanel>
</Grid>
```

Here, **DisplayMember** is **Title** hence the Title property in underlying data object is displayed in the textbox and the **ValueMember** is **OrderID** hence the OrderID property in underlying data object is considered as the **SelectedValue**.



SelectedItem (Display Member)

The Croods

SelectedValue (Value Member)

2

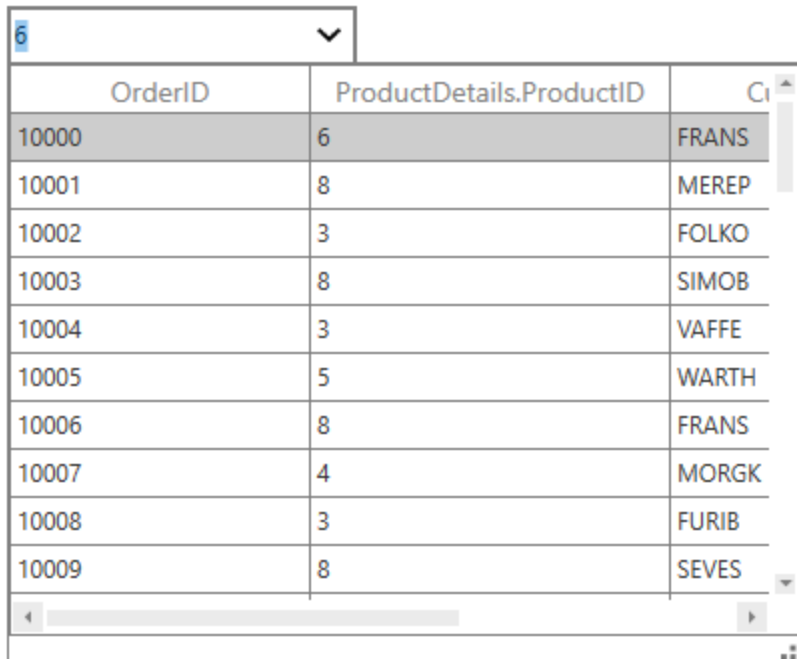
Binding with complex and indexer properties

SfMultiColumnDropDownControl provides support to display complex and indexer properties in its columns and also you can set complex and indexer properties as path to **DisplayMember** and **ValueMember** properties.

Binding with complex properties

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
Width="175"
Height="30"
SelectedIndex="0"
AutoGenerateColumns="false"
DisplayMember="ProductDetails.ProductID"
ValueMember="ProductDetails.ProductID"
ItemsSource="{Binding Orders}">
  <syncfusion:SfMultiColumnDropDownControl.Columns>
    <syncfusion:GridTextColumn MappingName="OrderID" />
    <syncfusion:GridTextColumn MappingName="ProductDetails.ProductID" />
    <syncfusion:GridTextColumn MappingName="CustomerID" />
    <syncfusion:GridTextColumn MappingName="Country" />
  </syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```

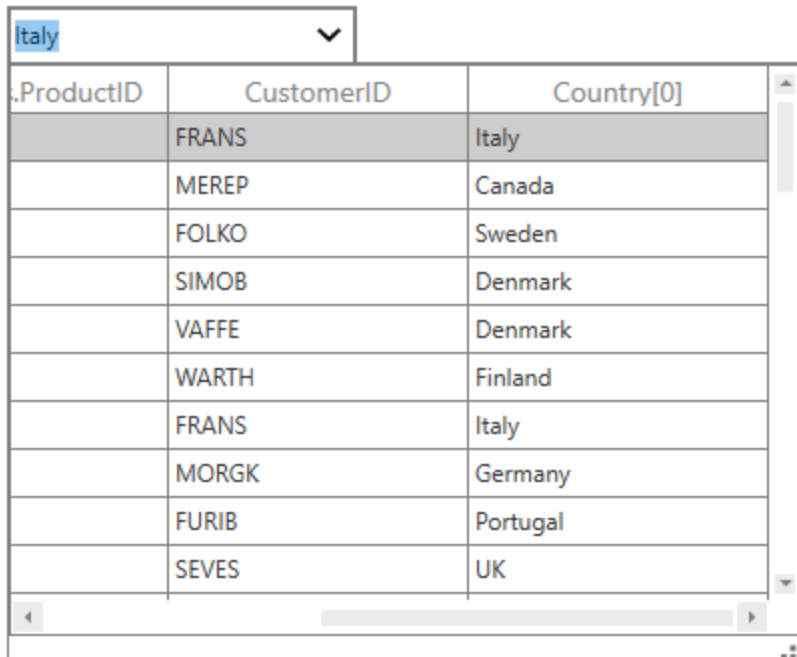


OrderID	ProductDetails.ProductID	Country
10000	6	FRANS
10001	8	MEREP
10002	3	FOLKO
10003	8	SIMOB
10004	3	VAFFE
10005	5	WARTH
10006	8	FRANS
10007	4	MORGK
10008	3	FURIB
10009	8	SEVES

Binding with indexer properties

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
Width="175"
Height="30"
SelectedIndex="0"
AutoGenerateColumns="false"
DisplayMember="Country[0]"
ValueMember="Country[0]"
ItemsSource="{Binding Orders}">
  <syncfusion:SfMultiColumnDropDownControl.Columns>
    <syncfusion:GridTextColumn MappingName="OrderID" />
    <syncfusion:GridTextColumn MappingName="ProductID" />
    <syncfusion:GridTextColumn MappingName="CustomerID" />
    <syncfusion:GridTextColumn MappingName="Country[0]" />
  </syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```



ProductID	CustomerID	Country[0]
	FRANS	Italy
	MEREP	Canada
	FOLKO	Sweden
	SIMOB	Denmark
	VAFFE	Denmark
	WARTH	Finland
	FRANS	Italy
	MORGK	Germany
	FURIB	Portugal
	SEVES	UK

Columns in WPF Multi Column Dropdown (SfMultiColumnDropDown)

SfMultiColumnDropDownControl enables you to define the columns as like in SfDataGrid. You can let the SfMultiColumnDropDownControl to create columns or you can manually defined 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 [SfMultiColumnDropDownControl.AutoGenerateColumns](#).

You can [refer here](#) to know more about the automatic column generation in SfMultiColumnDropDownControl.

Manually defining columns

SfMultiColumnDropDownControl control allows you to define the columns manually by adding desired column to the [SfMultiColumnDropDownControl.Columns](#) collection.

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
Width="175"
Height="30"
AutoGenerateColumns="false"
DisplayMember="OrderID"
ItemsSource="{Binding Orders}"
SelectedIndex="0"
ValueMember="OrderID">
  <syncfusion:SfMultiColumnDropDownControl.Columns>
    <syncfusion:GridColumnMapping MappingName="OrderID" />
  </syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```



```
<syncfusion:GridTextColumn MappingName="CustomerID" />
<syncfusion:GridTextColumn MappingName="Country" />
</syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```

C#

```
SfMultiColumnDropDownControl sfMultiColumn = new
SfMultiColumnDropDownControl();
sfMultiColumn.AutoGenerateColumns = false;
sfMultiColumn.Columns.Add(new GridCurrencyColumn() { MappingName = "OrderID"
});
sfMultiColumn.Columns.Add(new GridTextColumn() { MappingName = "CustomerID"
});
sfMultiColumn.Columns.Add(new GridTextColumn() { MappingName = "Country" });
```

1001		
OrderID	CustomerID	Country
\$1001.00	ALFKI	Germany
\$1002.00	ANATR	Mexico
\$1003.00	ANTON	Mexico
\$1004.00	AROUT	UK
\$1005.00	BERGS	Sweden
\$1006.00	BLAUS	Germany
\$1007.00	BLONP	France
\$1008.00	BOLID	Spain
\$1009.00	BONAP	France
\$1010.00	BOTTM	Canada

Customize auto-generated columns

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.SfMultiColumn.AutoGeneratingColumn +=
SfMultiColumn_AutoGeneratingColumn;
private void SfMultiColumn_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 auto generation of specific column by handling `AutoGeneratingColumn` event.

In the below code, column generation for `OrderID` property is canceled by setting `Cancel` property to `true`.

C#

```
this.SfMultiColumn.AutoGeneratingColumn +=  
SfMultiColumn_AutoGeneratingColumn;  
private void SfMultiColumn_AutoGeneratingColumn(object sender,  
AutoGeneratingColumnArgs e)  
{  
    if (e.Column.MappingName == "OrderID")  
        e.Cancel = true;  
}
```

Changing column type

You can change the column type while auto generation by setting the instance of column with type 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.SfMultiColumn.AutoGeneratingColumn +=  
SfMultiColumn_AutoGeneratingColumn;  
private void SfMultiColumn_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.SfMultiColumn.AutoGeneratingColumn +=  
SfMultiColumn_AutoGeneratingColumn;  
private void SfMultiColumn_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.

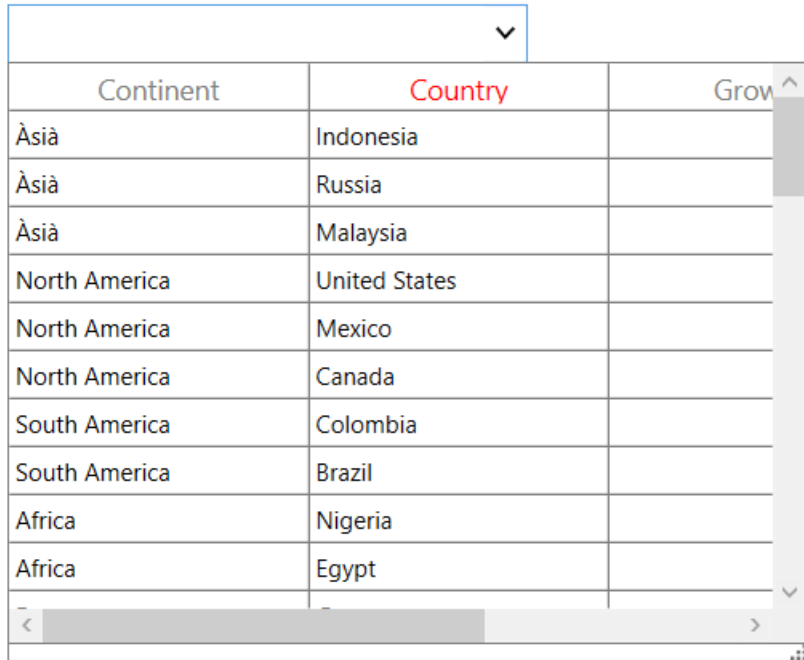
XML

```
<Window.Resources>
<DataTemplate x:Key="headerTemplate">
<TextBlock Text="{Binding}" TextWrapping="Wrap" Foreground="Red"/>
</DataTemplate>
</Window.Resources>
```

C#

```
this.SfMultiColumn.AutoGeneratingColumn +=
SfMultiColumn_AutoGeneratingColumn;
private void SfMultiColumn_AutoGeneratingColumn(object sender,
AutoGeneratingColumnArgs e)
{
if (e.Column.MappingName == "Country")
{
e.Column.HeaderTemplate = this.FindResource("headerTemplate") as
DataTemplate;
}
}
```

Below screenshot shows the customized header template loaded on the header of Country column.



Continent	Country	Grow
Àsià	Indonesia	
Àsià	Russia	
Àsià	Malaysia	
North America	United States	
North America	Mexico	
North America	Canada	
South America	Colombia	
South America	Brazil	
Africa	Nigeria	
Africa	Egypt	

Column sizing

You can also set the column width based on certain logic by setting [SfMultiColumnDropDownControl.GridColumnSizer](#). You can refer here to know more about the [GridColumn.ColumnSizer](#).

In the below code, `GridLengthUnitType.Star` is sets as `GridColumnSizer` for equally sets the column widths.

You can [refer here](#) to know more about the column sizing.

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
Width="175"
Height="25"
AutoGenerateColumns="False"
DisplayMember="Title"
GridColumnSizer="Star"
ItemsSource="{Binding MoviesLists}"
SelectedIndex="2"
ValueMember="Title">
<syncfusion:SfMultiColumnDropDownControl.Columns>
<syncfusion:GridTextColumn MappingName="Title" />
<syncfusion:GridTextColumn MappingName="Cast" />
<syncfusion:GridTextColumn MappingName="Director" />
<syncfusion:GridTextColumn MappingName="Rating" />
</syncfusion:SfMultiColumnDropDownControl.Columns>
```

```
</syncfusion:SfMultiColumnDropDownControl>
```

Oz the Great and Powerful ▼			
Title	Cast	Director	Rating
Iron Man 3	Robert Downney	Shane Black	PG-13
The Croods	Nicolas Cage	Kirk De Micco,	PG
Oz the Great and	James Franco	Sam Raimi	PG
G.I Joe Retaliatio	Dwayne Johnson	Jon M.Chu	PG-13
A Good Day to	Bruce Wills	John Moore	R
Hansel and Gret	Jermy Renner	Tommy Wirkola	R
Oblivion	Tom Cruise	Joseph Kosinski	PG-13
Journey to the V	Qi Shu	Stephen Chow	R
Jack the Giant S	Nicholas Hoult	Bryan Singer	PG-13
Identity Thief	Jason Batman	Seth Gordon	R
Iron Man 3	Robert Downney	Shane Black	PG-13

Editing and AutoComplete in WPF Multi Column Dropdown

SfMultiColumnDropDownControl allows you to edit via TextBox. You can make the editor as read-only by setting [SfMultiColumnDropDownControl.ReadOnly](#) property as `true`.

Auto Completion of Text

SfMultiColumnDropDownControl auto append the text based on `ItemsSource` when end-user edits in the TextBox when setting [AllowAutoComplete](#) to `true`.

1001 ▼		
OrderID	CustomerID	Country
\$1001.00	ALFKI	Germany
\$1002.00	ANATR	Mexico
\$1003.00	ANTON	Mexico
\$1004.00	AROUT	UK
\$1005.00	BERGS	Sweden
\$1006.00	BLAUS	Germany
\$1007.00	BLONP	France
\$1008.00	BOLID	Spain
\$1009.00	BONAP	France
\$1010.00	BOTTM	Canada

Limitations

When setting DataTable as `ItemsSource`, `AllowAutoComplete` is not supported.

Auto Increment

You can allow end-user to change the SelectedItem while mouse wheel over the control by setting [AllowSpinOnMouseWheel](#) as `true`.

Null Value Support

You can allow the null values in editor by setting [AllowNullInput](#) as `true`. While deleting the text in Textbox, then the SelectedItem will be null.

Note: `AllowNullInput` will work only when the underlying property type is nullable.

Open popup while editing

You can open the popup while typing the value in editor itself by setting [AllowImmediatePopup](#) as `true`.

Open popup while loading

You can specify whether the popup is need to open or close by setting [IsDropDownOpen](#) as `true`. You can open the popup while loading the SfMultiColumnDropDownControl itself by setting `IsDropDownOpen` as `true`.

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfmultiColumn"
Width="175"
Height="30"
IsDropDownOpen="True"
AutoGenerateColumns="false"
DisplayMember="OrderID"
ItemsSource="{Binding Orders}"
SelectedIndex="0"
ValueMember="OrderID">
  <syncfusion:SfMultiColumnDropDownControl.Columns>
    <syncfusion:GridCurrencyColumn MappingName="OrderID" />
    <syncfusion:GridTextColumn MappingName="CustomerID" />
    <syncfusion:GridTextColumn MappingName="Country" />
  </syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```

C#

```
sfMultiColumn.Loaded += sfMultiColumn_Loaded;
void sfMultiColumn_Loaded(object sender, RoutedEventArgs e)
{
    (sender as SfMultiColumnDropDownControl).IsDropDownOpen = true;
}
```

Keyboard Interactions

SfMultiColumnDropDownControl provides keyboard support for interaction. You can open the `DropDownGrid` without clicking the toggle button in SfMultiColumnDropDownControl by using the keyboard shortcuts.

Keyboard shortcut	Description
-------------------	-------------

Enter	If the popup is opened, then closes the popup and sets the SelectedItem .
Esc	If the popup is opened, then closes the popup and reverts the modified value.
F4, ALT + UP, ALT + Down	Shortcuts to open and close the popup.

Getting editor text

You can access the text displayed in the Textbox by using [SfMultiColumnDropDownControl.Text](#) property.

XML

```
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="400" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<syncfusion:SfMultiColumnDropDownControl x:Name="sfmultiColumn"
Width="175"
Height="30"
Grid.Column="0"
SelectedIndex="0"
DisplayMember="Cast"
ValueMember="Title"
ItemsSource="{Binding Orders}" />
<StackPanel Grid.Column="1" Margin="0,100,0,0">
<TextBlock FontSize="16" Text="MultiColumnDropDownControl text " />
<TextBlock FontSize="22"
FontWeight="Bold"
Text="{Binding ElementName= sfmultiColumn,
Mode=TwoWay,
Path=Text}" />
</StackPanel>
</Grid>
```

C#

```
var text = this.sfMultiColumn.Text;
```



MultiColumnDropDownControl text
The Croods

Selecting text when editor got focus

You can select the text displayed in the textbox when it got the focus from any other control by setting [TextSelectionOnFocus](#) property.

Selection in WPF Multi Column Dropdown (SfMultiColumnDropDown)

[SfMultiColumnDropDownControl](#) allows you to select one or more rows based on the SelectionMode. You can get the selected item in the SfDataGrid by using [SelectedItem](#) property and the selected index by using [SelectedIndex](#) property.

By using [SelectedValue](#) property, you can get the selected value from the selected item based on the ValueMember property.

Its recommend to use the [SelectedItem](#) and [SelectedValue](#) instead of using [SelectedIndex](#) to get the selected value.

By default, you can select only one item at a time from the dropdown, as the default value of [SelectionMode](#) is Single.

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="MultiColumnDropDown"
ItemsSource="{Binding Orders}"
ValueMember="CustomerID"
DisplayMember="CustomerName"
SelectionMode="Single">
</syncfusion:SfMultiColumnDropDownControl>
```

C#

```
using Syncfusion.UI.Xaml.Grid;
this.MultiColumnDropDown.SelectionMode = DropDownSelectionMode.Single;
```

The Croods		
Title	Cast	Director
Iron Man 3	Robert Downney Jr	Shane Black
The Croods	Nicolas Cage	Kirk De Micco, Chris Sander
Oz the Great and Powerful	James Franco	Sam Raimi
G.I Joe Retaliation	Dwayne Johnson	Jon M.Chu
A Good Day to Die Hard	Bruce Wills	John Moore
Hansel and Gretel Witch Hu	Jermy Renner	Tommy Wirkola
Oblivion	Tom Cruise	Joseph Kosinski
Journey to the West Conquer	Qi Shu	Stephen Chow
Jack the Giant Slayer	Nicholas Hoult	Bryan Singer
Identity Thief	Jason Batman	Seth Gordon
< >		

Multi-Selection

You can select multiple rows at same time by setting [SelectionMode](#) as **Multiple**. Further, you can select multiple rows in the following ways

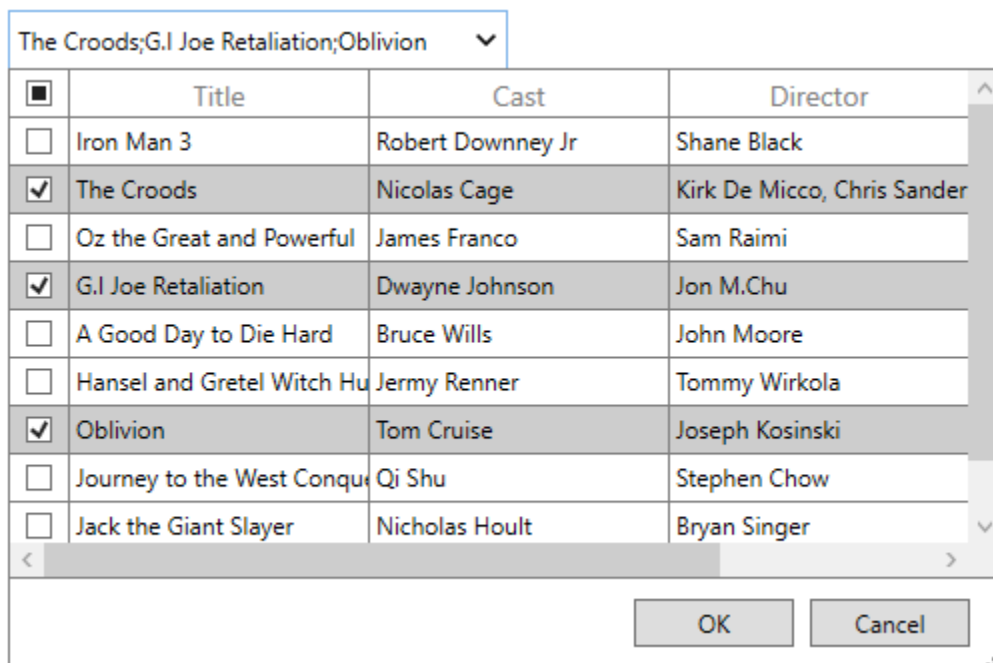
1. By clicking on the respective rows.
2. By dragging mouse on the dropdown grid.
3. By using **Space** key.
4. By interacting with the checkbox in the **Selector** column.

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="MultiColumnDropDown"
ItemsSource="{Binding Orders}"
ValueMember="CustomerID"
DisplayMember="CustomerName"
SelectionMode="Multiple">
</syncfusion:SfMultiColumnDropDownControl>
```

C#

```
using Syncfusion.UI.Xaml.Grid;
this.MultiColumnDropDown.SelectionMode = DropDownSelectionMode.Multiple;
```



Note: SelectedItem will be added, only after clicking on Ok button in the dropdown or by pressing Enter key. SelectedItem, SelectedIndex and SelectedValue denotes the first selected row.

Separator string customization

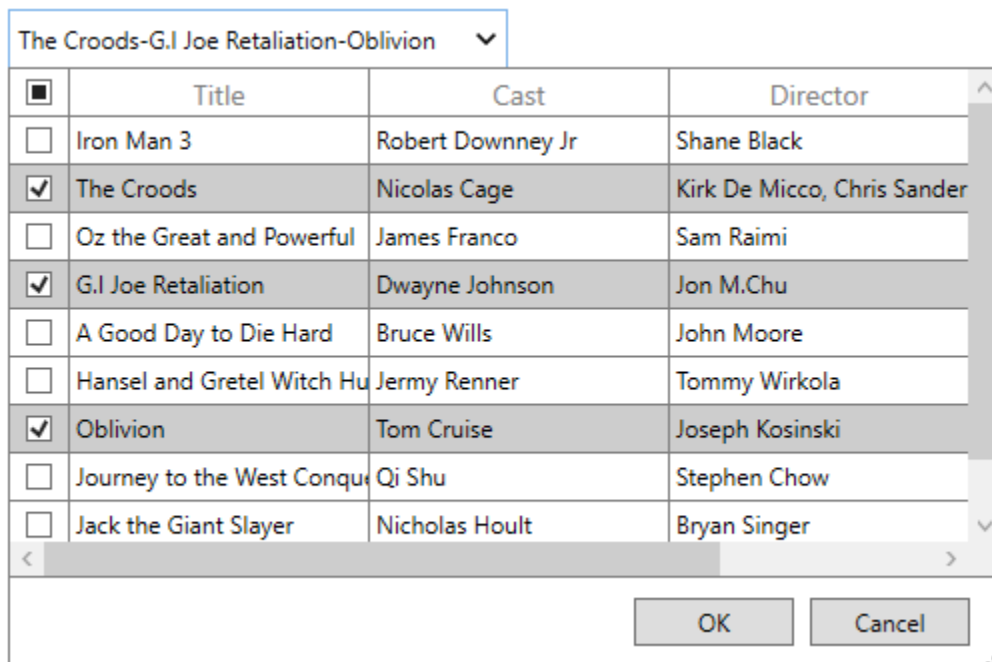
By default, selected values in the editor are separated by **;**. You can change this string by setting [SeparatorString](#) property.

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="MultiColumnDropDown"
ItemsSource="{Binding Orders}"
SelectionMode="Multiple"
ValueMember="CustomerID"
DisplayMember="CustomerName"
SeparatorString="-">
</syncfusion:SfMultiColumnDropDownControl>
```

C#

```
using Syncfusion.UI.Xaml.Grid;
this.MultiColumnDropDown.SeparatorString = "-";
```

*Accessing the selected items*

The selected items can be retrieved by using the [SelectedItems](#) property. The selection can also be added programmatically by using the [SelectedItems](#) property.

Load custom control in drop-down

You can add custom header to the dropdown by setting [HeaderTemplate](#) property. For example, you can add textbox at the header of dropdown to search and filter items.

XML

```
xmlns:interactivity="http://schemas.microsoft.com/expression/2010/interactivity"
<Window.Resources>
<local:MultiConverter x:Key="multiConverter"/>
<DataTemplate x:Key="headerTemplate">
<Border BorderThickness="0,0,0,1" BorderBrush="Gray">
<TextBox x:Name="searchTextBox" Margin="3" >
```

```

<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="TextChanged">
<interactivity:InvokeCommandAction Command="{Binding
Path=DataContext.TextChanged, Source={x:Reference
Name=MultiColumnDropDown}}" >
<interactivity:InvokeCommandAction.CommandParameter>
<MultiBinding Converter="{StaticResource multiConverter}">
<Binding Source="{x:Reference Name=MultiColumnDropDown}"/>
<Binding RelativeSource="{RelativeSource Mode=FindAncestor,
AncestorType=TextBox}"/>
</MultiBinding>
</interactivity:InvokeCommandAction.CommandParameter>
</interactivity:InvokeCommandAction>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</TextBox>
</Border>
</DataTemplate>
</Window.Resources>
<syncfusion:SfMultiColumnDropDownControl x:Name="MultiColumnDropDown"
DisplayMember="Title"
HeaderTemplate="{StaticResource headerTemplate}"
ItemsSource="{Binding MoviesLists}"
ValueMember="Cast"
SelectionMode="Multiple">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="PopupOpening">
<interactivity:InvokeCommandAction
Command="{Binding Path=DataContext.PopupOpening,
ElementName=MultiColumnDropDown}"
CommandParameter="{Binding ElementName=MultiColumnDropDown}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:SfMultiColumnDropDownControl>

```

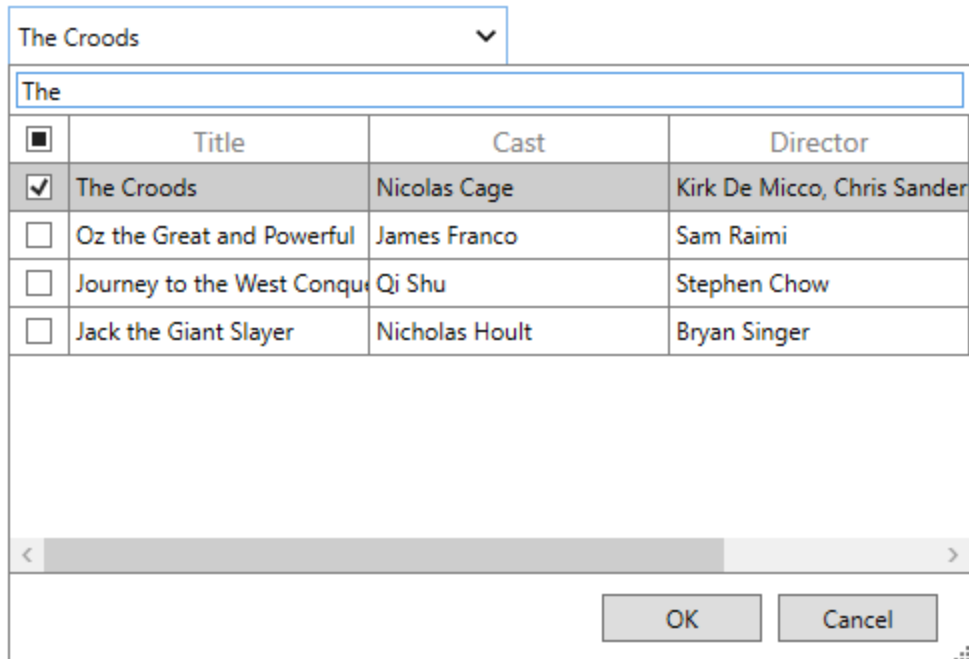
C#

```

using Syncfusion.UI.Xaml.Grid;
using Syncfusion.UI.Xaml.Utility;
public BaseCommand TextChanged
{
    get
    {
        if (textChanged == null)
            textChanged = new BaseCommand(OnTextChangedExecuted);
        return textChanged;
    }
}
public BaseCommand PopupOpening
{
    get
    {
        if (popupOpeningCommand == null)
            popupOpeningCommand = new BaseCommand(OnPopupOpening);
        return popupOpeningCommand;
    }
}

```

```
}
private void OnTextChangedExecuted(object obj)
{
    var param = (object[])obj;
    var multiColumnDropDown = param[0] as SfMultiColumnDropDownControl;
    searchTextBox = param[1] as TextBox;
    var grid = multiColumnDropDown.GetDropDownGrid();
    if (grid != null && grid.View != null && grid.View.Filter != null)
        grid.View.RefreshFilter();
}
private void OnPopupOpening(object obj)
{
    var multiColumnDropDown = obj as SfMultiColumnDropDownControl;
    var grid = multiColumnDropDown.GetDropDownGrid();
    if (grid != null)
        grid.View.Filter = CustomerFilter;
}
private bool CustomerFilter(object item)
{
    var movie = item as GrossingMoviesList;
    return movie.Title.ToLower().Contains(searchTextBox.Text.ToLower());
}
internal class MultiConverter : IMultiValueConverter
{
    public object Convert(object[] values, Type targetType, object parameter,
        CultureInfo culture)
    {
        return values.ToArray();
    }
    public object[] ConvertBack(object value, Type[] targetTypes, object
        parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```



You can refer [Search with TextBox in DropDown](#) section in this [sample](#).

Note: You can load custom control in drop-down only when SelectionMode is Multiple.

Events

You can handle the selection operations in SfMultiColumnDropDownControl by using [SelectionChanged](#) event.

SelectionChanged

[SelectionChanged](#) event is fired when select the item in SfDataGrid. You can use this event to get the SelectedItem, SelectedValue. [SelectionChangedEventArgs](#) provides data for [SelectionChanged](#) event.

C#

```
MultiColumnDropDown.SelectionChanged +=
MultiColumnDropDown_SelectionChanged;
void MultiColumnDropDown_SelectionChanged(object sender,
Syncfusion.UI.Xaml.Grid.SelectionChangedEventArgs args)
{
    var selectedValue = (sender as SfMultiColumnDropDownControl).SelectedValue;
}
```

Note: SelectionChanged event will not be triggered when SelectionMode is Multiple.

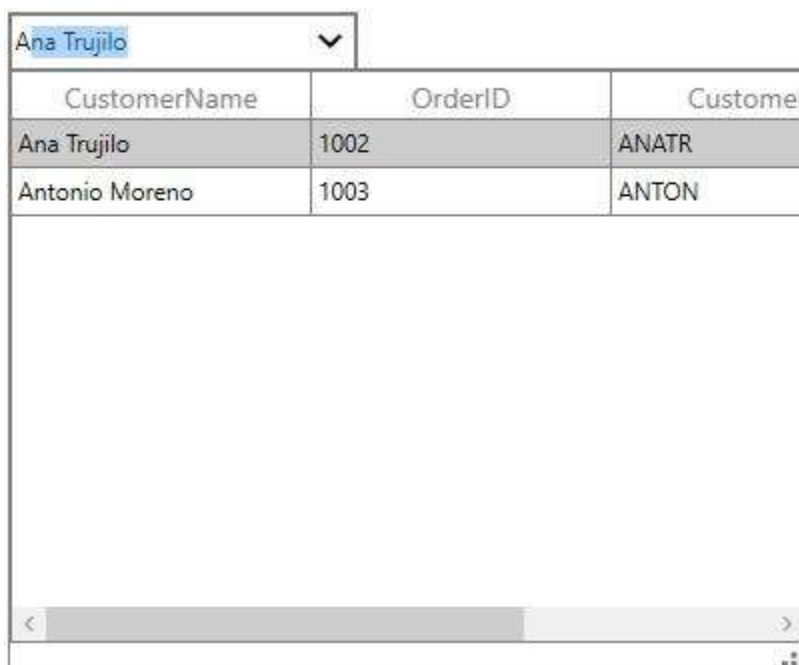
Filtering in WPF Multi Column Dropdown (SfMultiColumnDropDown)

SfMultiColumnDropDownControl provides support to filter the drop down display list based on typed text by setting [AllowIncrementalFiltering](#) as true. The records are filter based on [DisplayMember](#).

By default, drop down list is filtered based on [SearchCondition.StartsWith](#) condition. You can change the filtering search condition by setting [SearchCondition](#) (StartsWith, Contains, Equals options).

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
Width="175"
Height="30"
AutoGenerateColumns="False"
DisplayMember="CustomerName"
ItemsSource="{Binding Orders}"
SelectedIndex="4"
ValueMember="CustomerName">
<syncfusion:SfMultiColumnDropDownControl.Columns>
<syncfusion:GridTextColumn MappingName="CustomerName" />
<syncfusion:GridTextColumn MappingName="OrderID" />
<syncfusion:GridTextColumn MappingName="CustomerID" />
<syncfusion:GridTextColumn MappingName="Country" />
</syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```



Case-Sensitive Filtering

You can decide whether the automatic completion of text and the filtering are case-sensitive or not by setting [AllowCaseSensitiveFiltering](#) as true.

Ignore Diacritic Sensitivity

By default, SfMultiColumnDropDownControl filter or auto-complete the data based on an input character only. For example, if we type normal character in editor, it will not filter or auto-complete the record containing a diacritic character. You can disable the [AllowDiacriticSensitiveFiltering](#) property if you want to filter or auto-complete data that also contains a diacritic character while typing normal character in the editor.

XML

```
<Syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
HorizontalAlignment="Left"
AutoGenerateColumns="False"
AllowIncrementalFiltering="True"
```

```
AllowImmediatePopup="True"
AllowDiacriticSensitiveFiltering="False"
VerticalAlignment="Top"
DisplayMember="Continent"
ItemsSource="{Binding PopulationDetails}">
</Syncfusion:SfMultiColumnDropDownControl>
```



How to filter SfMultiColumnDropDownControl based on various column values

By default, SfMultiColumnDropDownControl filter the text based on **DisplayMember** (considers single column text only). You can also filter the text based on multiple columns by overriding the [FilterRecord](#) method in **SfMultiColumnDropDownControl** and use the [SearchText](#) property to get the entered text in editor.

C#

```
public class CustomMultiColumnControl : SfMultiColumnDropDownControl
{
    /// <summary>
    /// Returns true if the item is displayed in the Filtered List, otherwise
    /// returns false.
    /// </summary>
    /// <param name="item"></param>
    /// <returns></returns>
    protected override bool FilterRecord(object item)
```

```

{
    var _item = item as GrossingMoviesList;
    var result = (_item.Title.Contains(this.SearchText)) ||
    (_item.Cast.Contains(this.SearchText));
    return result;
}
}

```

XML

```

<local:CustomMultiColumnControl x:Name="sfMultiColumn"
Width="175"
Height="25"
AutoGenerateColumns="False"
DisplayMember="Title"
ItemsSource="{Binding MoviesLists}"
SelectedIndex="2"
ValueMember="Title">
<local:CustomMultiColumnControl.Columns>
<syncfusion:GridTextColumn MappingName="Title" />
<syncfusion:GridTextColumn MappingName="Cast" />
<syncfusion:GridTextColumn MappingName="Director" />
<syncfusion:GridTextColumn MappingName="Rating" />
</local:CustomMultiColumnControl.Columns>
</local:CustomMultiColumnControl>

```

Here, **Title** is defined as a **DisplayMember**. But it also searches the match case from the **Cast** column also while filtering.



Note: Excel-like filtering is not supported in SfMultiColumnDropDownControl. You can customize the SfMultiColumnDropDownControl ControlTemplate to enable the Excel-like filtering by setting **AllowFiltering** as true in SfDataGrid.

Filtering Delay

By default, filtering and auto-completion operations performed while typing in the editor will be delayed for a specified amount of time (milliseconds). The period of delaying filtering and auto-completion operations can be specified by the `FilterDelay` property. The default value for the `FilterDelay` property is 500.

C#

```
this.sfMultiColumnDropDown.FilterDelay = 2000;
```

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumnDropDown"
Width="175"
Height="30"
ItemsSource="{Binding Orders}"
DisplayMember="OrderID"
AutoGenerateColumns="False"
FilterDelay="2000"
SelectedIndex="2">
  <syncfusion:SfMultiColumnDropDownControl.Columns>
    <syncfusion:GridTextColumn MappingName="OrderID" />
    <syncfusion:GridTextColumn MappingName="CustomerID" />
    <syncfusion:GridTextColumn MappingName="Country" />
  </syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```

Limitations

When setting `DataTable` as `ItemsSource`, `AllowIncrementalFiltering` is not supported.

Styles and Templates in WPF Multi Column Dropdown

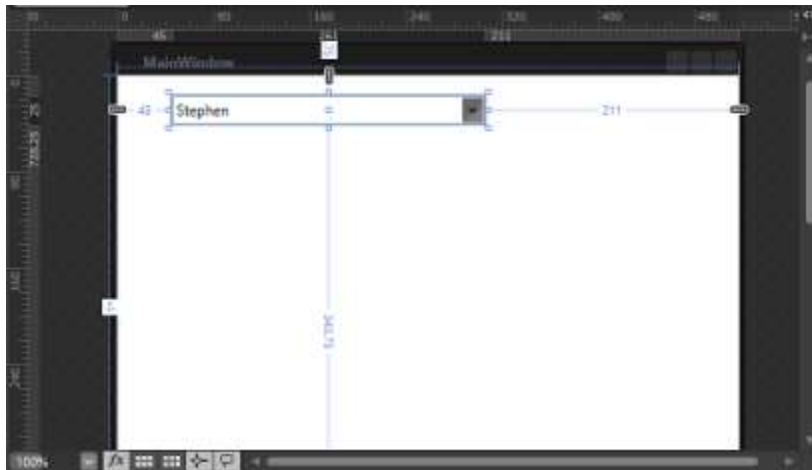
WPF styles and templates refer to a suite of features (styles and templates) that allows you to create visually compelling effects and to create consistent appearance for the products.

This section elaborates the information to understand the possible ways, by which you can change the visual appearance of the `SfMultiColumnDropDownControl`. In addition, you can edit the structure of `SfMultiColumnDropDownControl` by using Blend and VisualStudio that enables you to customize the appearance.

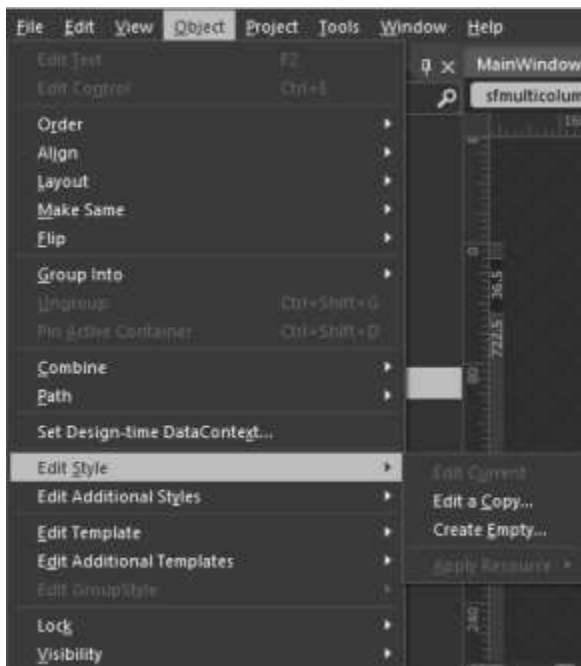
Edit Appearance in Expression Blend

This section explains you how to edit the `SfMultiColumnDropDownControl` structure in Expression Blend. To edit the Control Template in Expression Blend, refer to the following steps.

1. Open your application in Expression Blend.
2. Select the `SfMultiColumnDropDownControl` from the window

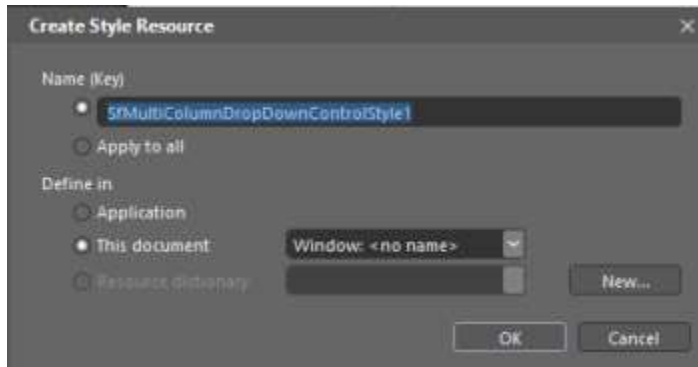


3. Go to the Menu bar and Choose Object > EditStyle.



4. You get two options in sub menu bar.

1. Edit a Copy- Edit a copy of the default style, when you select this option, a new dialog window opens.



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.

Now, press 'Ok'. Expression Blend generates the style of the SfMultiColumnDropDownControl in the Resource section. You can edit the generated code in the XAML view.

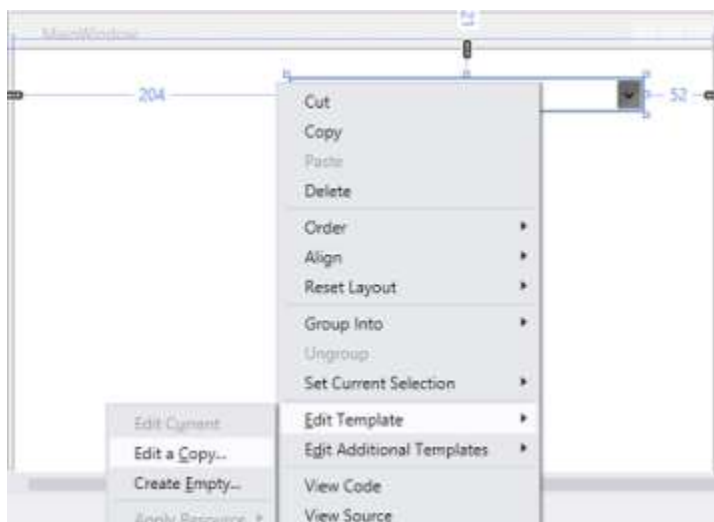
2. Create Empty- create an empty style for the SfMultiColumnDropDownControl. When you select this option, the same Create style Resources dialog opens. You should enter the name or change the name of style and choose the location for the Style.

Now, press 'Ok'. Expression Blend generates the empty style of the SfMultiColumnDropDownControl in the Resource section. You can edit the generated code in the XAML view.

[Edit Appearance in VisualStudio](#)

This section explain you how to edit a SfMultiColumnDropDownControl style in Visual Studio Design View. To Edit the control style in Visual Studio, refer to the following steps,

1. Open your application in Visual Studio.
2. Open Design view > Select SfMultiColumnDropDownControl >Right Click on SfMultiColumnDropDownControl, Menu options is displayed.



3. Click Edit Template, you have two options in sub menu bar.

4. Edit a Copy –Edit a copy of the default style, when you select this option, a new dialog window opens as follows.



The Create ControlTemplate 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.

Now press 'OK'. Visual Studio generates the style of SfMultiColumnDropDownControl in the Resource section. You can edit the generated code in the XAML view.

2. Create Empty- Create an empty style for the SfMultiColumnDropDownControl. When you select this option, the same Create ControlTemplate Resources dialog opens. Enter the name or change the name of style and choose the location for the style.

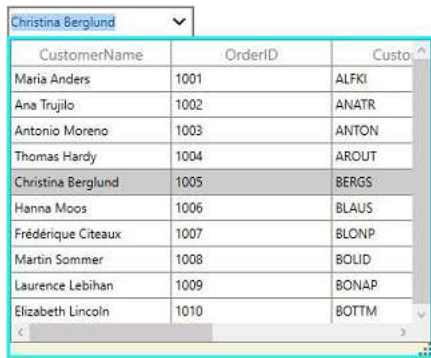
Now press 'OK'. Visual Studio generates the empty style of SfMultiColumnDropDownControl in the Resource section. You can edit the generated code in the XAML view.

Popup customization in WPF Multi Column Dropdown

SfMultiColumnDropDownControl allows you to customize the Popup appearance by setting [PopupBackground](#), [PopupBorderBrush](#), [PopupDropDownGridBackground](#) and [PopupBorderThickness](#) properties.

XML

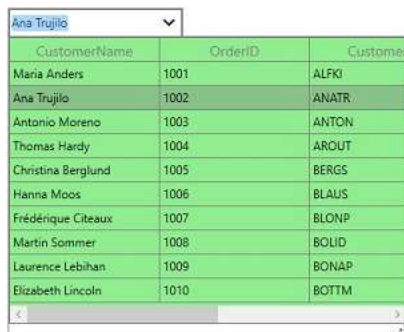
```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
Width="175"
Height="30"
DisplayMember="CustomerName"
ItemsSource="{Binding Orders}"
PopupBackground="Beige"
PopupBorderBrush="Aqua"
PopupBorderThickness="3"
ValueMember="CustomerName">
<syncfusion:SfMultiColumnDropDownControl.Columns>
<syncfusion:GridTextColumn MappingName="CustomerName" />
<syncfusion:GridTextColumn MappingName="OrderID" />
<syncfusion:GridTextColumn MappingName="CustomerID" />
<syncfusion:GridTextColumn MappingName="Country" />
</syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```



You can change the drop down grid background by setting [PopupDropDownGridBackground](#).

XML

```
<syncfusion:SfMultiColumnDropDownControl x:Name="sfMultiColumn"
Width="175"
Height="30"
DisplayMember="CustomerName"
ItemsSource="{Binding Orders}"
PopupDropDownGridBackground="LightGreen"
ValueMember="CustomerName">
  <syncfusion:SfMultiColumnDropDownControl.Columns>
    <syncfusion:GridTextColumn MappingName="CustomerName" />
    <syncfusion:GridTextColumn MappingName="OrderID" />
    <syncfusion:GridTextColumn MappingName="CustomerID" />
    <syncfusion:GridTextColumn MappingName="Country" />
  </syncfusion:SfMultiColumnDropDownControl.Columns>
</syncfusion:SfMultiColumnDropDownControl>
```



Popup sizing

You can change the pop-up height and width by setting [PopupWidth](#) and [PopupHeight](#) properties. 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](#) will be set based on [PopupMinHeight](#) and [PopupMaxHeight](#) when the value is not between them. The default value of [PopupMinHeight](#) is 300.0.

Auto sizing

SfMultiColumnDropDownControl can automatically adjust the popup width and height based on the width and height of SfDataGrid when setting [IsAutoPopupSize](#) as `true`.

Note: When `IsAutoPopupSize` is `true`, then the popup width and height is not calculated based on `PopupHeight` and `PopupWidth`.

Resizing popup

You can allow the end-user to resize the drop-down popup through resizing thumb by setting [ResizingThumbVisibility](#) property to `Visible`.

Keep DropDownPopup as StaysOpen

You can keep the popup in `SfMultiColumnDropDownControl` as always open by using `StaysOpen` property. You can get the popup from template of `SfMultiColumnDropDownControl` in its loaded event and set the `StaysOpen` property as `true`.

C#

```
sfMultiColumn.Loaded += (o, e) =>
{
    var popup = sfMultiColumn.Template.FindName("PART_Popup", sfMultiColumn) as
    Popup;
    popup.StaysOpen = true;
};
```

Events

`SfMultiColumnDropDownControl` provides the following events for popup customizations,

PopupClosing event

[PopupClosing](#) event is fired when the popup is closing. You can use this event to skip the popup closing and [PopupClosingEventArgs](#) provides data for `PopupClosing` event. You can skip the popup closing by setting `args.Cancel` as `true`.

C#

```
sfMultiColumn.PopupClosing += sfMultiColumn_PopupClosing;
void sfMultiColumn_PopupClosing(object sender, PopupClosingEventArgs args)
{
}
```

PopupClosed event

[PopupClosed](#) event is fired when the popup is closed. [PopupClosedEventArgs](#) provides data for `PopupClosed` event

C#

```
sfMultiColumn.PopupClosed += sfMultiColumn_PopupClosed;
void sfMultiColumn_PopupClosed(object sender, PopupClosedEventArgs args)
{
}
```

PopupOpening event

[PopupOpening](#) event is fired when the popup is opening. You can use this event to skip the popup opening and [PopupOpeningEventArgs](#) provides data for `PopupOpening` event. You can skip the popup opening by setting `args.Cancel` as `true`.

C#

```
sfMultiColumn.PopupOpening += sfMultiColumn_PopupOpening;
void sfMultiColumn_PopupOpening(object sender, PopupOpeningEventArgs args)
{
}
```

PopupOpened event

[PopupOpened](#) event is fired when the popup is opened. [PopupOpenedEventArgs](#) provides data for [PopupOpened](#) event

C#

```
sfMultiColumn.PopupOpened += sfMultiColumn_PopupOpened;
void sfMultiColumn_PopupOpened(object sender, PopupOpenedEventArgs args)
{
}
```

UIAutomation

SfMultiColumnDropDownControl supports the following UIAutomations,

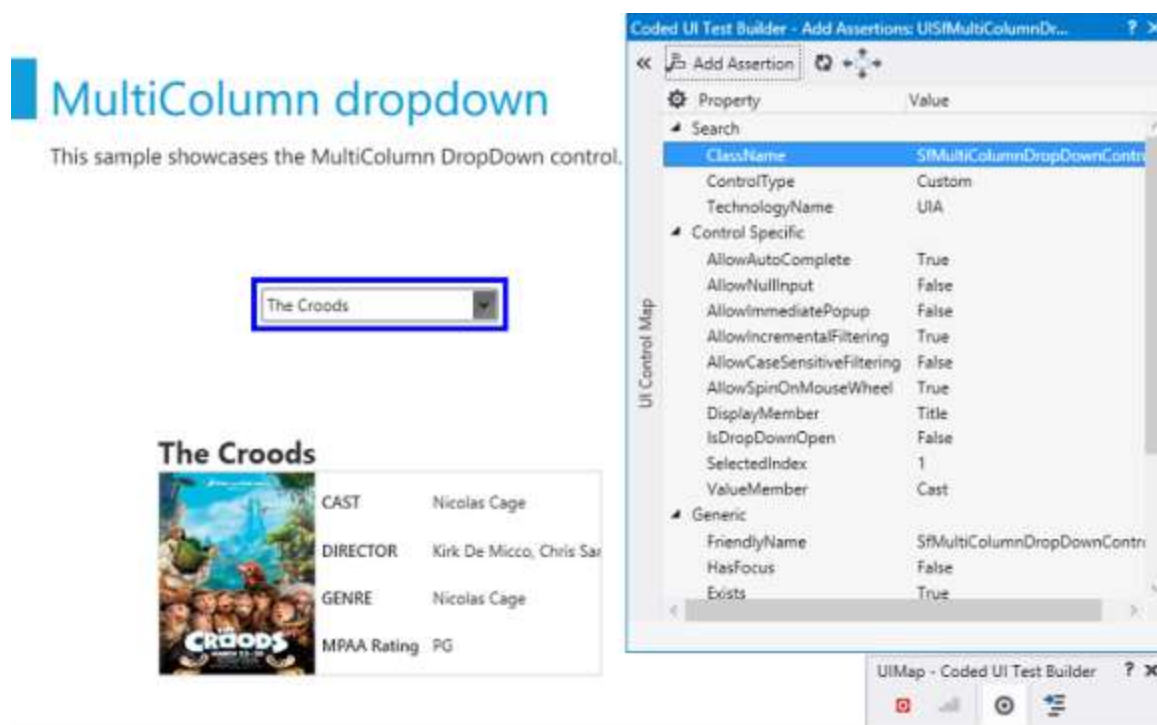
Coded UI

SfMultiColumnDropDownControl supports CodedUITest automation that enables you to create an automation test with SfMultiColumnDropDownControl elements and record the sequence of actions.

There are three level of support in CodedUITest for SfMultiColumnDropDownControl.

Levels	Description
Level 1	Record and Detecting the UI Elements when you do the actions in the control.
Level 2	Provide custom properties for UI elements when you drag the Cross hair 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.

The following screenshot illustrates the SfMultiColumnDropDownControl properties, when you drag the crosshair to the SfMultiColumnDropDown control.



Following are the properties for SfMultiColumnDropDownControl.

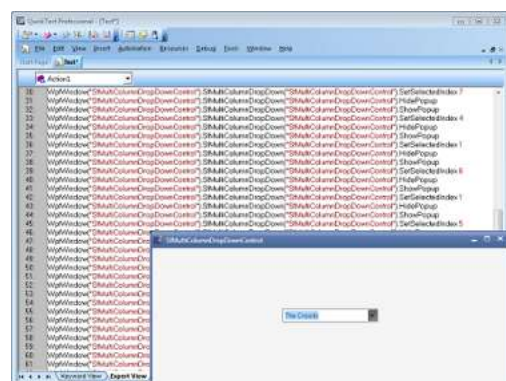
Property Table

UI Element	Properties
SfMultiColumnDropDownControl	<i>AllowAutoComplete AllowNullInput AllowImmediatePopup AllowIncrementalFiltering AllowCaseSensitiveFiltering AllowSpinOnMouseWheel DisplayMember IsDropDownOpen SelectedIndex ValueMember</i>

Quick Test Professional

SfMultiColumnDropDownControl supports for QTP test. You can record the actions performed in the control, by mentioning the corresponding method name with Syncfusion namespace. To know more about QTP test refer to the [link](#).

The following screenshot illustrates the QTP Test for SfMultiColumnDropDownControl.



Following are methods for SfMultiColumnDropDownControl.

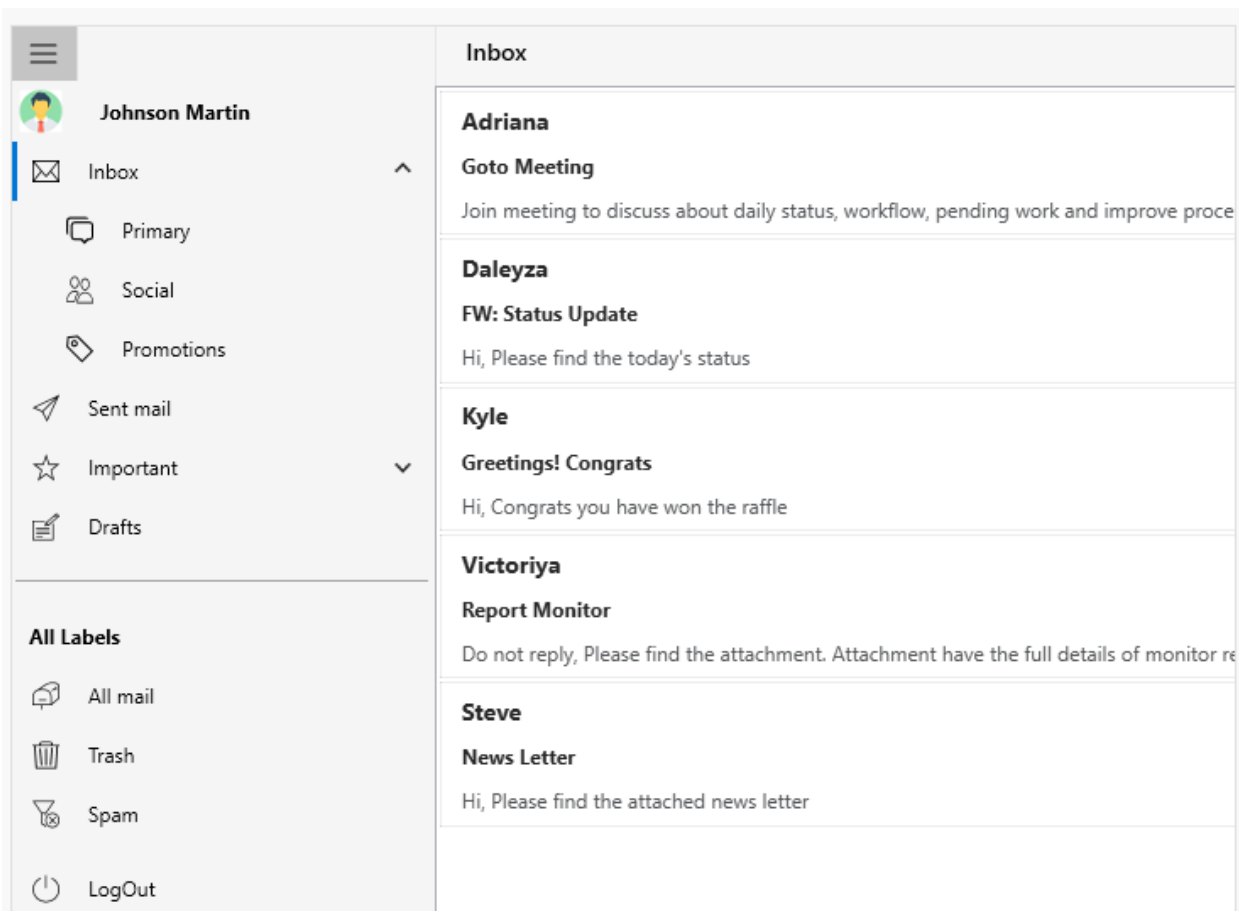
Methods Table

Method	Description	Parameters	Return Type
void SetSelectedIndex(int index);	To set the SelectedIndex from the Popup	Int index	void
void ShowPopup();	To open the Popup	NA	void
void HidePopup();	To close the Popup	NA	void

SfNavigationDrawer

WPF Navigation Drawer (SfNavigationDrawer) Overview

The WPF Navigation Drawer control is a sidebar navigation view that is used to create a navigation menu for easy navigation. It provides compact and extended display modes with built-in navigation view items with ability to switch between both modes based on available size. It also provides default mode which allows to have a custom pane view.



SfNavigationDrawer

Use case scenarios

The Navigation Drawers are used in applications where navigating to the major module or page is a basic requirement. The Navigation Drawer is available in the following apps that signify the importance of navigating through pages:

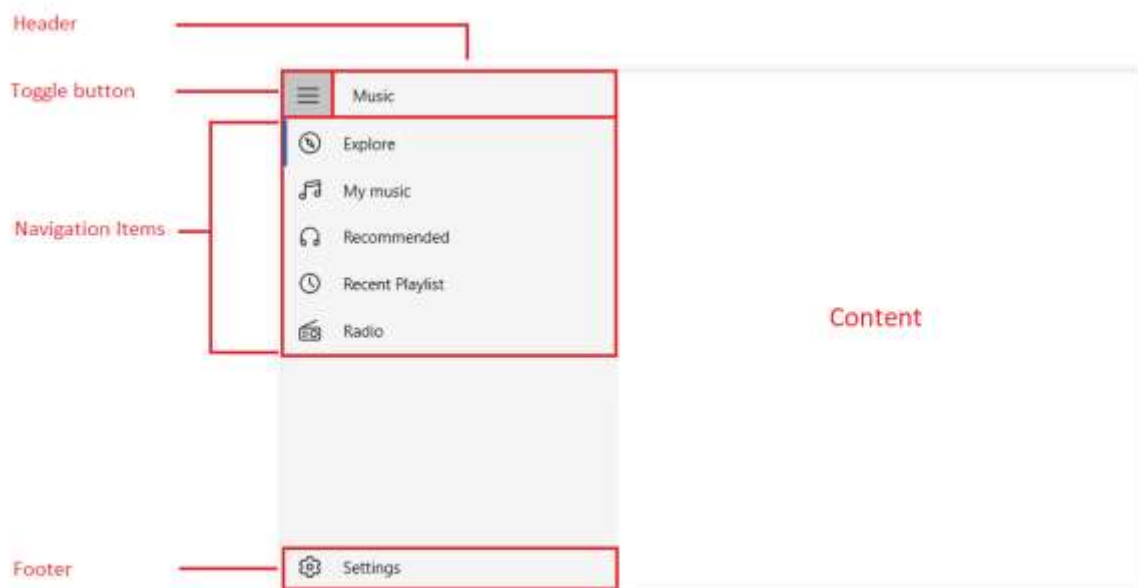
1. Facebook
2. Play Store
3. e-Commerce Apps
4. Banking Apps

Key features

- A sidebar menu to provide Modern UI navigation for application.
- Navigation menu can be placed on any side of the screen.
- Compact and extended display modes with support to best fit the content based on available size.
- Built-in navigation menu items: tab, button, header, multi-level hierarchal items and footer items.
- Custom views also can be added as the header and footer of the drawer.

Visual Structure

This section describes the visual elements of the NavigationDrawer control and defines terms and concepts used in the DisplayMode.



- **Header** — Represents the header of the drawer
- **Footer** — Represents the footer of the drawer.

- **NavigationDrawer Items** — Built-in items used to populate the items in the drawer pane body and footer.
- **ToggleButton** — Built-in toggle button used to collapse and expand the drawer menu.

Getting Started with WPF Navigation Drawer (SfNavigationDrawer)

This section helps you to build your application with SfNavigationDrawer.

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 SfNavigationDrawer

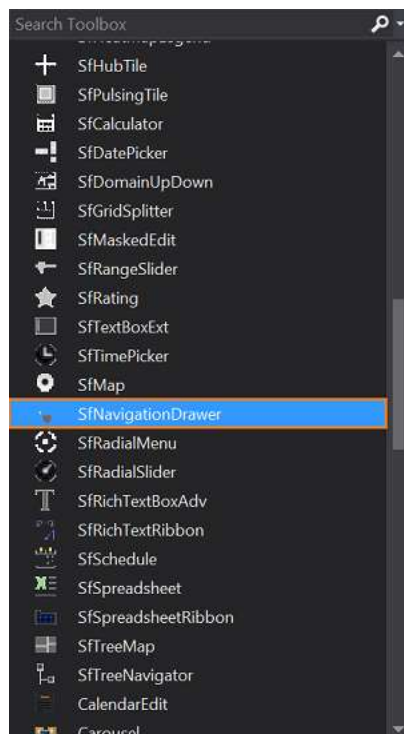
- Creating the project
- Adding SfNavigationDrawer from Toolbox
- Adding SfNavigationDrawer control from XAML
- Adding control manually in C#
- Adding content to the control
- Adding sidebar menu items

Creating the project

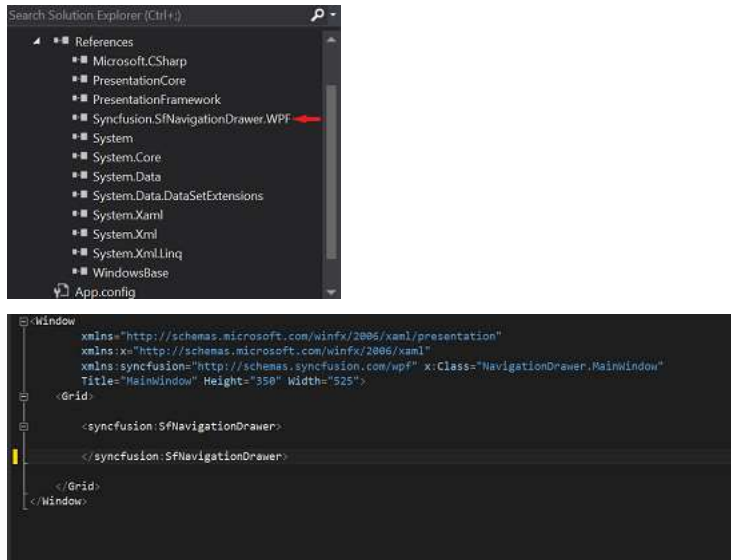
Create new WPF Project in Visual Studio to display SfNavigationDrawer with data objects.

Adding SfNavigationDrawer from Toolbox

Drag and drop the SfNavigationDrawer control from the Toolbox to your application.



Now, the SyncfusionControls for UWP XAML reference is added to the application references and the xmlns namespace code is generated in MainWindow.xaml as below.



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.SfNavigationDrawer.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page.
3. Declare SfNavigationDrawer control in XAML page.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"/>
</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.SfNavigationDrawer.WPF
2. Import SfTreeView namespace **Syncfusion.UI.Xaml.NavigationDrawer**.
3. Create SfNavigationDrawer control instance and add it to the Page.

C#

```

using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Windows;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
            this.Content = navigationDrawer;
        }
    }
}

```

Adding content to the control

XML

```

<syncfusion:SfNavigationDrawer>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```

C#

```

SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
Label label = new Label();
label.Content = "Content View";
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;

```

Adding sidebar menu items

The sidebar menu can be populated using the built-in items. The items will be of type [NavigationItem](#) having many properties for customization.

See also [Different display modes](#) topic in SfNavigationDrawer.

XML

```

<syncfusion:SfNavigationDrawer
x:Name="navigationDrawer"
DisplayMode="Expanded">

```

```

<syncfusion:NavigationItem
Header="Inbox"
IsExpanded="True"
IsSelected="True">
<syncfusion:NavigationItem.Icon>
<Path Data="M32.032381, 16.445429 L25.410999, ... />
</syncfusion:NavigationItem.Icon>
<syncfusion:NavigationItem Header="Primary">
<syncfusion:NavigationItem.Icon>
<Path Data="M9.5189972,7.3780194C8.3389893,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Social">
<syncfusion:NavigationItem.Icon>
<Path Data="M22.133972,14.194015C17.582977,... />
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Promotions">
<syncfusion:NavigationItem.Icon>
<Path Data="M9.4614787,7.2521966C8.897512,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Sent mail">
<syncfusion:NavigationItem.Icon>
<Path Data="M42.128046,6.7269681 L18.53705,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Important">
<syncfusion:NavigationItem.Icon>
<Path Data="M25.085007,5.9780004 L20.577011,...../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Drafts">
<syncfusion:NavigationItem.Icon>
<Path Data="M6.9999996,48.353 L19,48.353 19,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem ItemType="Separator" />
<syncfusion:NavigationItem Header="All Labels" ItemType="Header" />
<syncfusion:NavigationItem Header="Starred">
<syncfusion:NavigationItem.Icon>
<Path Data="M25.085007,5.9780004 L20.577011,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="All mail">
<syncfusion:NavigationItem.Icon>
<Path Data="M12,32.999999 L26,32.999999 26,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Trash">
<syncfusion:NavigationItem.Icon>
<Path Data="M17,12 L19,12 19,42 17,42z M10.998,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Spam">
<syncfusion:NavigationItem.Icon>

```

```

<Path Data="M33.671003,29.293001 L39.214003,..."/>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```

C#

```

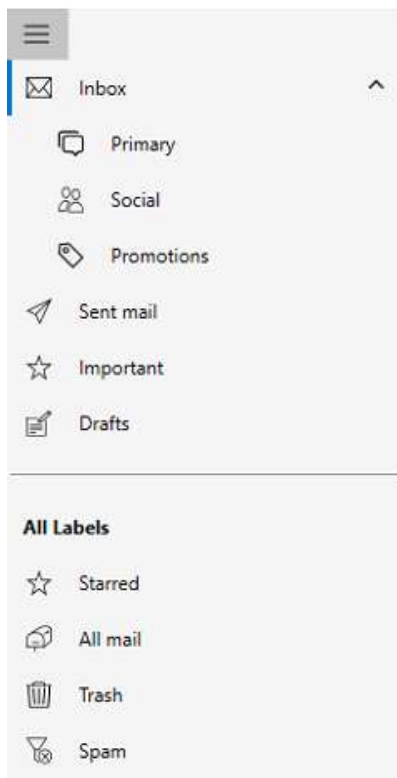
SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Expanded;
NavigationItemsCollection navigationSubItems = new
NavigationItemsCollection();
navigationSubItems.Add(new NavigationItem()
{
Header = "Primary",
Icon = new Path()
{
Data = Geometry.Parse("M9.5189972,7.3780194C8.3389893,...),
.....
},
});
navigationSubItems.Add(new NavigationItem()
{
Header = "Social",
Icon = new Path()
{
Data = Geometry.Parse("M22.133972,14.194015C17.582977,...),
.....
},
});
navigationSubItems.Add(new NavigationItem()
{
Header = "Promotions",
Icon = new Path()
{
Data = Geometry.Parse("M9.4614787,7.2521966C8.897512,...),
.....
},
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "Inbox",
Icon = new Path()
{
Data = Geometry.Parse("M32.032381, 16.445429 L25.410999, ....),
.....
},
Items = navigationSubItems,
IsExpanded = true,

```

```
IsSelected=true
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Sent mail",
    Icon = new Path()
    {
        Data = Geometry.Parse("M42.128046,6.7269681 L18.53705,....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Drafts",
    Icon = new Path()
    {
        Data = Geometry.Parse("M6.9999996,48.353 L19,....),
        ....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    ItemType = ItemType.Separator
});
navigationDrawer.Items.Add(new NavigationItem()
{
    ItemType = ItemType.Header,
    Header= "All Labels"
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Starred",
    Icon = new Path()
    {
        Data = Geometry.Parse("M25.085007,5.9780004 L20.577011,....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "All mail",
    Icon = new Path()
    {
        Data = Geometry.Parse("M12,32.999999 L26,32.999999 26,....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Trash",
    Icon = new Path()
    {
        Data = Geometry.Parse("M17,12 L19,12 19,42 17,42z M10.998,....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
```



```
{
Header = "Spam",
Icon = new Path()
{
Data = Geometry.Parse("M33.671003,29.293001 L39.214003,....),
.....
}
});
Label label = new Label();
label.Content = "Content View";
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;
```



Note: View [sample](#) in GitHub

Different display modes in WPF Navigation Drawer (SfNavigationDrawer)

The WPF Navigation drawer provides default, compact and extended [display modes](#) to create navigation menu for an application. Compact and Extended display modes support to populate the navigation menu using built items with different types.

See also [different item types](#) topic in Navigation Drawer.

Compact display mode

A navigation sidebar is shown as a narrow bar to the width set to the [CompactModeWidth](#) property. The navigation menu gets expanded on clicking the built-in toggle button and appears as an overlay above the main content to the width set to the [ExpandedModeWidth](#) property.

Note: The navigation menu will get toggled back to compact width when any interaction performed on the main content area.

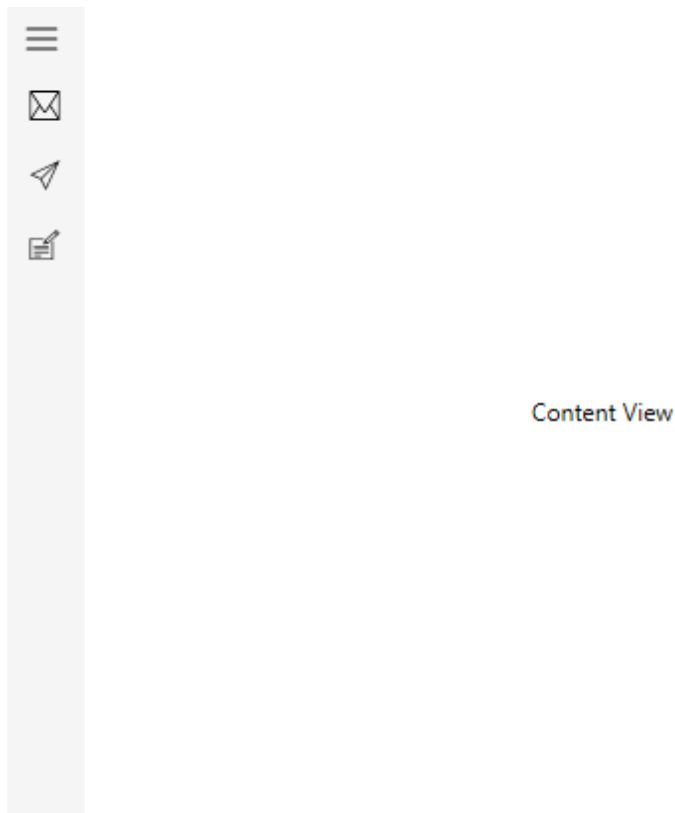
XML

```
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
    DisplayMode="Compact">
    <syncfusion:NavigationItem Header="Inbox">
    <syncfusion:NavigationItem.Icon>
    <Path Data="M32.032381, 16.445429 L25.410999,... />
    </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Sent mail">
    <syncfusion:NavigationItem.Icon>
    <Path Data="M42.128046,6.7269681 L18.53705,...../>
    </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Drafts">
    <syncfusion:NavigationItem.Icon>
    <Path Data="M6.9999996,48.353 L19,...../>
    </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:SfNavigationDrawer.ContentView>
    <Label
    Content="Content View" .../>
    </syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>
```

C#

```
SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Compact;
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Inbox",
    Icon = new Path()
    {
        Data = Geometry.Parse("M32.032381, 16.445429 L25.410999,.....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Sent mail",
    Icon = new Path()
    {
        Data = Geometry.Parse("M42.128046,6.7269681 L18.53705,.....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
```

```
{
Header = "Drafts",
Icon = new Path()
{
Data = Geometry.Parse("M6.9999996,48.353 L19,.....),
.....
}
});
Label label = new Label();
label.Content = "Content View";
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;
```



Expanded display mode

The navigation sidebar stays open to the left or right of the window based on the [Position](#) property pushing the main content alongside. The [ExpandedModeWidth](#) property defines the width of the drawer.

When the drawer menu is toggled using the built-in toggle button, the drawer menu gets collapsed and shown as a narrow bar to the width set to the [CompactModeWidth](#) property.

XML

```

<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
DisplayMode="Expanded">
<syncfusion:NavigationItem Header="Inbox">
<syncfusion:NavigationItem.Icon>
<Path Data="M32.032381, 16.445429 L25.410999,..../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Sent mail">
<syncfusion:NavigationItem.Icon>
<Path Data="M42.128046,6.7269681 L18.53705,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Drafts">
<syncfusion:NavigationItem.Icon>
<Path Data="M6.9999996,48.353 L19,...../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Content="Content View" ...../>
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```

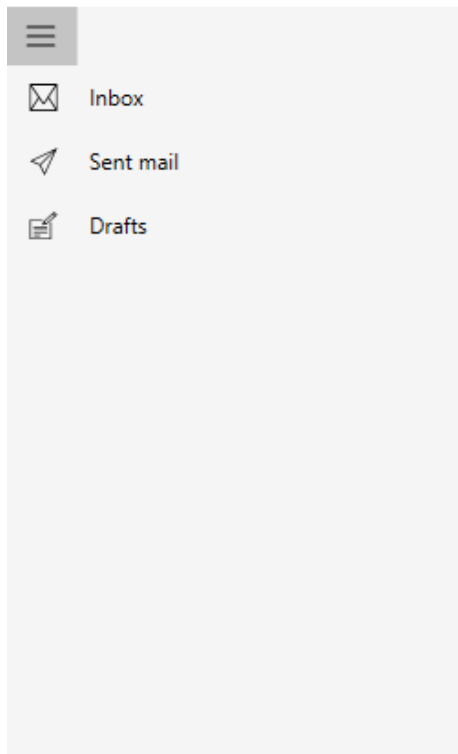
C#

```

SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Expanded;
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Inbox",
    Icon = new Path()
    {
        Data = Geometry.Parse("M32.032381, 16.445429 L25.410999,.....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Sent mail",
    Icon = new Path()
    {
        Data = Geometry.Parse("M42.128046,6.7269681 L18.53705,.....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Drafts",
    Icon = new Path()
    {
        Data = Geometry.Parse("M6.9999996,48.353 L19,48.353 19,.....),
        .....
    }
});
Label label = new Label();
label.Content = "Content View";

```

```
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;
```



Auto display mode change

The Navigation Drawer handles its display mode based on the values set to the [ExpandedModeThresholdWidth](#) property. This can be enabled by setting the [AutoChangeDisplayMode](#) property to `True`.

This switches the display mode to compact when the application window size is less than the [ExpandedModeThresholdWidth](#) and switches to expanded mode when the application window size is larger.

XML

```
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
    DisplayMode="Expanded" AutoChangeDisplayMode="True"
    ExpandedModeThresholdWidth="600">
    <syncfusion:NavigationItem Header="Inbox">
    <syncfusion:NavigationItem.Icon>
    <Path
    Data="M32.032381, 16.445429 L25.410999,..... />
    </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Sent mail">
    <syncfusion:NavigationItem.Icon>
    <Path
```

```

Data="M42.128046,6.7269681 L18.53705,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Drafts">
<syncfusion:NavigationItem.Icon>
<Path Data="M6.9999996,48.353 L19,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Content="Content View" ...../>
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

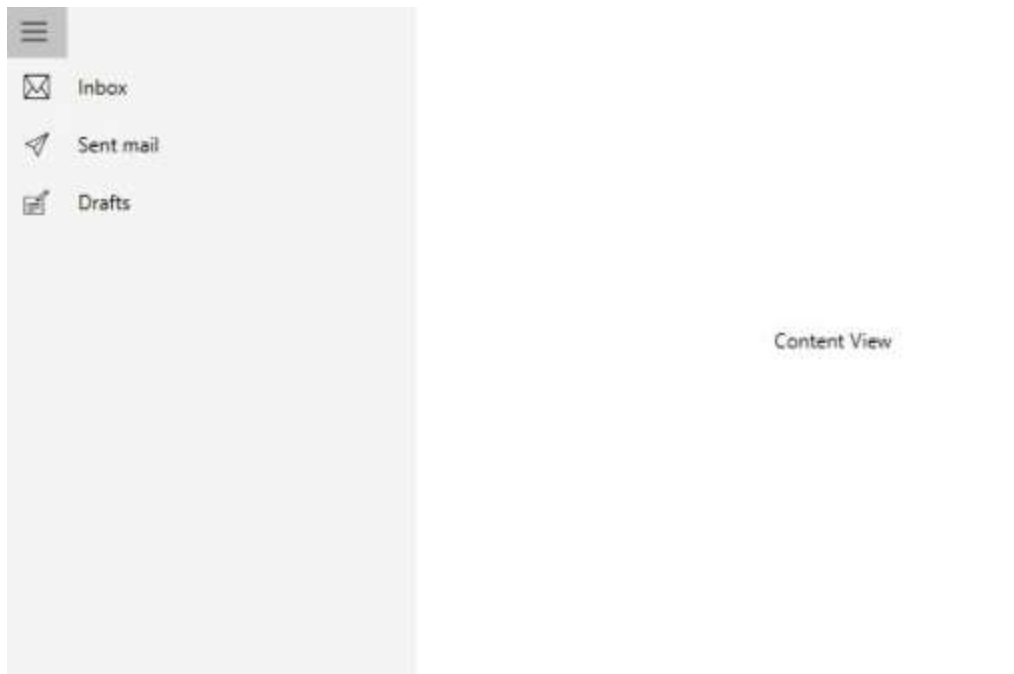
```

C#

```

SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Expanded;
navigationDrawer.AutoChangeDisplayMode = true;
navigationDrawer.ExpandedModeThresholdWidth = 600;
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Inbox",
    Icon = new Path()
    {
        Data = Geometry.Parse("M32.032381, 16.445429 L25.410999,.....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Sent mail",
    Icon = new Path()
    {
        Data = Geometry.Parse("M42.128046,6.7269681 L18.53705,.....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Drafts",
    Icon = new Path()
    {
        Data = Geometry.Parse("M6.9999996,48.353 L19,.....),
        .....
    }
});
Label label = new Label();
label.Content = "Content View";
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;

```



Note: View [sample](#) in GitHub

Collapsible drawer mode

A collapsible drawer can be achieved using the Navigation Drawer by setting the display mode to [Default mode](#). In this display mode, the drawer menu is populated using [custom views](#).

See also [Custom Views](#) topic in Navigation Drawer.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
  <Window.DataContext>
    <local:ViewModel/>
  </Window.DataContext>
  <syncfusion:SfNavigationDrawer x:Name="navigationDrawer" DrawerWidth="300">
    <syncfusion:SfNavigationDrawer.ContentView>
      <Grid x:Name="mainContentView"
Background="White">
        <Grid.RowDefinitions>
          <RowDefinition Height="auto"/>
        </RowDefinitions>
        <StackPanel Background="#1aa1d6"
Orientation="Horizontal">
          <Button x:Name="hamburgerButton"
Height="50" Width="50"
```

```

HorizontalAlignment="Left"
Click="HamburgerButton_Click">
<Image Source="hamburger_icon.png"
Height="20"
Width="20" />
</Button>
<Label x:Name="headerLabel"
Height="50"
Content="Home"
Background="#1aa1d6"/>
</StackPanel>
<Label Grid.Row="1"
x:Name="contentLabel"
Content="Content View"
Foreground="Black"/>
</Grid>
</syncfusion:SfNavigationDrawer.ContentView>
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View" />
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
<syncfusion:SfNavigationDrawer.DrawerContentView>
<Grid>
<ListBox x:Name="list"
ItemsSource="{Binding Contents}">
<ListBox.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Black"/>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerContentView>
<syncfusion:SfNavigationDrawer.DrawerFooterView>
<Grid Background="#31ade9">
<Label Content="Footer View" />
</Grid>
</syncfusion:SfNavigationDrawer.DrawerFooterView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

using System.Collections.Generic;
using System.Windows;
namespace NavigationDrawerWPF
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void HamburgerButton_Click(object sender, RoutedEventArgs e)

```



```

{
navigationDrawer.ToggleDrawer();
}
}
public class ViewModel
{
public ViewModel()
{
Contents = new List<Model>();
Contents.Add(new Model() { Name = "Home" });
Contents.Add(new Model() { Name = "Profile" });
Contents.Add(new Model() { Name = "Inbox" });
Contents.Add(new Model() { Name = "Outbox" });
Contents.Add(new Model() { Name = "Sent" });
Contents.Add(new Model() { Name = "Trash" });
Contents.Add(new Model() { Name = "Sign Out" });
}
public List<Model> Contents { get; set; }
}
public class Model
{
public string Name { get; set; }
}
}

```

Populating data in WPF Navigation Drawer (SfNavigationDrawer)

This section explains how to populate the drawer menu.

Populating using built-in items

The WPF Navigation Drawer sidebar provides a built-in items support of type [NavigationItem](#) that can be populated using the [Items] property.

The below properties are available in the [NavigationItem](#) and can be used to define each item of the navigation menu.

- **Header** - Represents the content of the [NavigationItem](#).
- **Icon** - Represents the icon in the [NavigationItem](#).
- **IconTemplate** - Used to display the custom icon in the [NavigationItem](#). Also see the [IconTemplate](#) section.
- **IconMemberPath** - Used to display the icon for sub-items, while providing the [ItemsSource](#).
- **DisplayMemberPath** - Used to display the content for sub-items, while providing the [ItemsSource](#). Also see the [Hierarchical Data Binding](#) section.
- **ExpanderTemplate** - Used to provide a custom view for the expander icon in both collapsed and expanded state.
- **Command** — Executes when the item gets clicked. See also [Commands](#) section.
- **CommandParameter** — [CommandParameter](#) is user defined data value that can be passed to the [Command](#) when it is executed.
- **IsChildSelected** — Gets whether any sub item is selected or not.
- **ItemType** — Defines the type of navigation item. See also [Different item types](#) section.
- **IsExpanded** — Gets whether the item is in expanded or collapsed state.
- **IsSelected** — Gets whether the item is selected or not.

- **SelectionBackground** — Used to customize the selection indicator in **NavigationItem**.
- **Items** - Used to populate the sub-items.

XML

```

<syncfusion:SfNavigationDrawer
x:Name="navigationDrawer"
DisplayMode="Expanded">
<syncfusion:NavigationItem
Header="Inbox"
IsExpanded="True"
IsSelected="True">
<syncfusion:NavigationItem.Icon>
<Path Data="M32.032381, 16.445429 L25.410999, ... />
</syncfusion:NavigationItem.Icon>
<syncfusion:NavigationItem Header="Primary">
<syncfusion:NavigationItem.Icon>
<Path Data="M9.5189972,7.3780194C8.3389893,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Social">
<syncfusion:NavigationItem.Icon>
<Path Data="M22.133972,14.194015C17.582977,... />
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Promotions">
<syncfusion:NavigationItem.Icon>
<Path Data="M9.4614787,7.2521966C8.897512,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Sent mail">
<syncfusion:NavigationItem.Icon>
<Path Data="M42.128046,6.7269681 L18.53705,..../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Important">
<syncfusion:NavigationItem.Icon>
<Path Data="M25.085007,5.9780004 L20.577011,...../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Drafts">
<syncfusion:NavigationItem.Icon>
<Path Data="M6.9999996,48.353 L19,48.353 19,..../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem ItemType="Separator" />
<syncfusion:NavigationItem Header="All Labels" ItemType="Header" />
<syncfusion:NavigationItem Header="Starred">
<syncfusion:NavigationItem.Icon>
<Path Data="M25.085007,5.9780004 L20.577011,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="All mail">
<syncfusion:NavigationItem.Icon>
<Path Data="M12,32.999999 L26,32.999999 26,.../>

```

```

</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Trash">
<syncfusion:NavigationItem.Icon>
<Path Data="M17,12 L19,12 19,42 17,42z M10.998,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Spam">
<syncfusion:NavigationItem.Icon>
<Path Data="M33.671003,29.293001 L39.214003,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```

C#

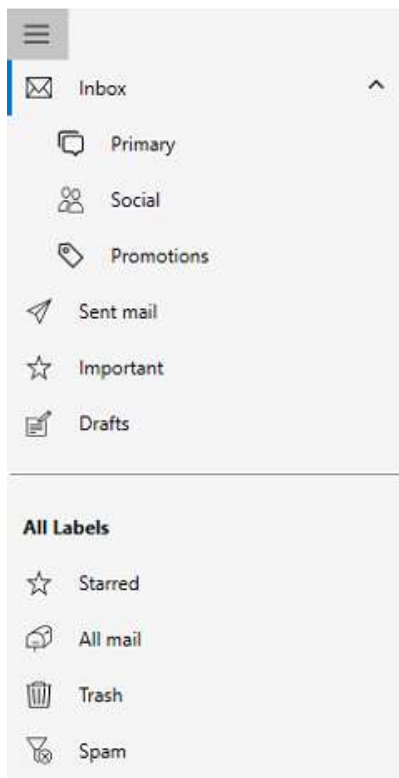
```

SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Expanded;
NavigationItemsCollection navigationSubItems = new
NavigationItemsCollection();
navigationSubItems.Add(new NavigationItem()
{
Header = "Primary",
Icon = new Path()
{
Data = Geometry.Parse("M9.5189972,7.3780194C8.3389893,...),
.....
},
});
navigationSubItems.Add(new NavigationItem()
{
Header = "Social",
Icon = new Path()
{
Data = Geometry.Parse("M22.133972,14.194015C17.582977,...),
.....
},
});
navigationSubItems.Add(new NavigationItem()
{
Header = "Promotions",
Icon = new Path()
{
Data = Geometry.Parse("M9.4614787,7.2521966C8.897512,...),
.....
},
});
navigationDrawer.Items.Add(new NavigationItem()

```

```
{
Header = "Inbox",
Icon = new Path()
{
Data = Geometry.Parse("M32.032381, 16.445429 L25.410999, ....),
.....
},
Items = navigationSubItems,
IsExpanded = true,
IsSelected=true
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "Sent mail",
Icon = new Path()
{
Data = Geometry.Parse("M42.128046,6.7269681 L18.53705,....),
.....
}
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "Drafts",
Icon = new Path()
{
Data = Geometry.Parse("M6.9999996,48.353 L19,....),
....
}
});
navigationDrawer.Items.Add(new NavigationItem()
{
ItemType = ItemType.Separator
});
navigationDrawer.Items.Add(new NavigationItem()
{
ItemType = ItemType.Header,
Header= "All Labels"
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "Starred",
Icon = new Path()
{
Data = Geometry.Parse("M25.085007,5.9780004 L20.577011,....),
.....
}
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "All mail",
Icon = new Path()
{
Data = Geometry.Parse("M12,32.999999 L26,32.999999 26,....),
.....
}
});
navigationDrawer.Items.Add(new NavigationItem()
```

```
{
Header = "Trash",
Icon = new Path()
{
Data = Geometry.Parse("M17,12 L19,12 19,42 17,42z M10.998,....),
.....
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "Spam",
Icon = new Path()
{
Data = Geometry.Parse("M33.671003,29.293001 L39.214003,....),
.....
});
Label label = new Label();
label.Content = "Content View";
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;
```



Note: View [sample](#) in GitHub

IconTemplate

The *IconTemplate* is used to provide the custom icon for the *NavigationItem*.

Model

C#

```
public class Category
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    private object icon;
    public object Icon
    {
        get { return icon; }
        set { icon = value; }
    }
}
```

ViewModel

C#

```
public class ViewModel
{
    public ObservableCollection<Category> Categories { get; set; }
    public ViewModel()
    {
        Categories = new ObservableCollection<Category>();
        Categories.Add(new Category()
        {
            Name = "Inbox",
            Icon = "Inbox.png"
        });
        Categories.Add(new Category()
        {
            Name = "Sent",
            Icon = "Sent.png"
        });
        Categories.Add(new Category()
        {
            Name = "Draft",
            Icon = "Draft.png"
        });
        Categories.Add(new Category()
        {
            Name = "Spam",
            Icon = "Spam.png"
        });
    }
}
```

XML

```

<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Window.Resources>
<Style x:Key="ItemStyle" TargetType="syncfusion:NavigationItem">
<Setter Property="IconTemplate">
<Setter.Value>
<DataTemplate>
<Image
Width="16"
Height="16"
Source="{Binding}" />
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
<syncfusion:SfNavigationDrawer
x:Name="navigationDrawer"
DisplayMemberPath="Name"
DisplayMode="Compact"
IconMemberPath="Icon"
ItemContainerStyle="{StaticResource ItemStyle}"
ItemsSource="{Binding Categories}">
<syncfusion:SfNavigationDrawer.ContentView>
<Label
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="ContentView" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

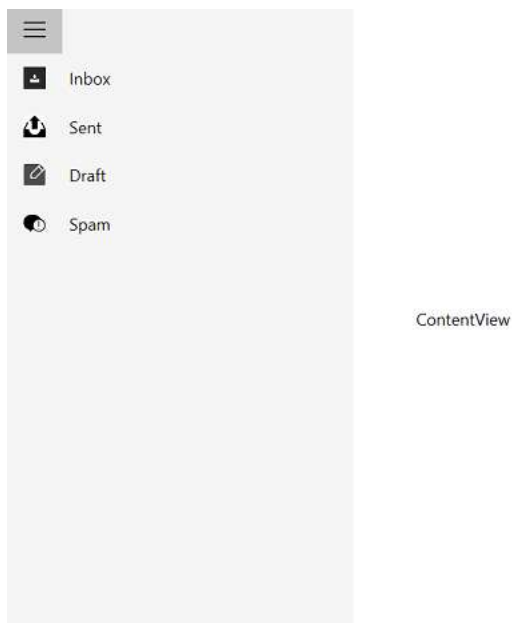
```

C#

```

ViewModel viewModel = new ViewModel();
DataContext = viewModel;
Style navigationItemStyle = new Style(typeof(NavigationItem));
FrameworkElementFactory image = new FrameworkElementFactory(typeof(Image));
image.SetValue(Image.WidthProperty, (double)16);
image.SetValue(Image.HeightProperty, (double)16);
image.SetValue(Image.SourceProperty, new Binding());
DataTemplate dataTemplate = new DataTemplate { VisualTree = image };
navigationItemStyle.Setters.Add(new
Setter(NavigationItem.IconTemplateProperty, dataTemplate));
SfNavigationDrawer navigationDrawer =new SfNavigationDrawer();
navigationDrawer.DisplayMemberPath = "Name";
navigationDrawer.DisplayMode = DisplayMode.Compact;
navigationDrawer.IconMemberPath = "Icon";
navigationDrawer.ItemContainerStyle = navigationItemStyle;
navigationDrawer.ItemsSource = viewModel.Categories;
Label label = new Label();
label.Content = "ContentView";
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;

```



Different item types

The WPF Navigation Drawer has four built-in item types that can be set to each **NavigationItem**.

- **Tab** — Interactions can be performed in this type and can have selection. This type supports multi-level population having an expand and collapse icon.
- **Button** — This type is similar to a button control behavior. Interactions can be performed and does not have selection. This type supports multi-level population having an expand and collapse icon.
- **Header** — This item type does not have any interaction or selection and acts like a header label. This item will get visible only in drawer menu expanded state. This item can be added as a sub item also but cannot have a sub item.
- **Separator** — This item type does not have any interaction or selection and acts as a separator line. This item can be added as a sub item also but cannot have a sub item.

See also [Populating using built in items](#) section.

Note: ItemType default value is Tab.

Data Binding

The drawer menu can be populated by [ItemsSource](#) property. It can support, bound collection of objects through the **ItemsSource**.

[ItemTemplate](#) property used to customize the content of the item, while using the ItemsSource and the icon of the item showing by used the [IconMemberPath](#) property.

See also [Hierarchical Data Binding](#) section.

Model

C#

```
public class Model  
{
```



```

private string item;
public string Item
{
    get { return item; }
    set { item = value; }
}
private object icon;
public object Icon
{
    get { return icon; }
    set { icon = value; }
}
}

```

ViewModel

C#

```

public class ViewModel
{
    public ObservableCollection<Model> Items { get; set; }
    public ViewModel()
    {
        Items = new ObservableCollection<Model>();
        Items.Add(new Model()
        {
            Item = "Explore",
            Icon = new Path()
            {
                Data = Geometry.Parse("M6.0033803,5.705333 L7.0853577,....),
                .....
            }
        });
        Items.Add(new Model()
        {
            Item = "My music",
            Icon = new Path()
            {
                Data = Geometry.Parse("M2.5,10.701 C1.6729736,10.701 1,....),
                .....
            }
        });
        Items.Add(new Model()
        {
            Item = "Recommended",
            Icon = new Path()
            {
                Data = Geometry.Parse("M11.5,10 C11.224,10 11,....),
                .....
            }
        });
        Items.Add(new Model()
        {
            Item = "Recent Playlist",
            Icon = new Path()
            {
                Data = Geometry.Parse("M6.9990103,3.0000041 L7.9990076,....),

```

```

.....
}
});
Items.Add(new Model()
{
    Item = "Radio",
    Icon = new Path()
    {
        Data = Geometry.Parse("M2.5,11.472009 L6.5,11.472009 C6.776,....),
        .....
    }
});
}
}
}
}

```

XML

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<syncfusion:SfNavigationDrawer
x:Name="navigationDrawer"
DisplayMode="Compact"
IconMemberPath="Icon"
ItemsSource="{Binding Items}">
<syncfusion:SfNavigationDrawer.ItemTemplate>
<DataTemplate>
<Label Content="{Binding Item}"/>
</DataTemplate>
</syncfusion:SfNavigationDrawer.ItemTemplate>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```

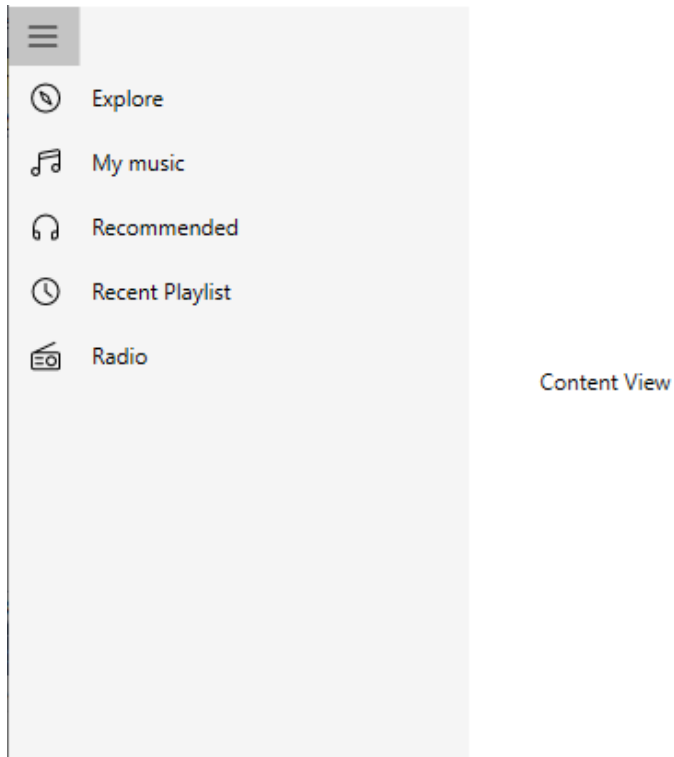
C#

```

ViewModel viewModel = new ViewModel();
this.DataContext = viewModel;
SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Compact;
navigationDrawer.IconMemberPath = "Icon";
FrameworkElementFactory textBlock = new
FrameworkElementFactory(typeof(TextBlock));
textBlock.SetBinding(TextBlock.TextProperty, new Binding("Item"));
DataTemplate template = new DataTemplate { VisualTree = textBlock };
navigationDrawer.ItemTemplate = template;
navigationDrawer.ItemsSource = viewModel.Items;
Label label = new Label();
label.Content = "Content View";
label.HorizontalAlignment = HorizontalAlignment.Center;

```

```
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;
```



Hierarchical Data Binding

The control populates the items using the [ItemsSource](#), also it allows the sub items.

When using the `ItemsSource` to show the content of the item achieved by [DisplayMemberPath](#) property and [ItemsContainerStyle](#) property used to customize the items and allows to set the `ItemsSource` and `DisplayMemberPath` properties of the sub items.

Model

C#

```
public class Model
{
    public ObservableCollection<Model> SubItems { get; set; }
    private string item;
    public string Item
    {
        get { return item; }
        set { item = value; }
    }
    private object icon;
    public object Icon
    {
        get { return icon; }
        set { icon = value; }
    }
}
```

```
}
}
```

*ViewModel***C#**

```
public class ViewModel
{
    public ObservableCollection<Model> Items { get; set; }
    ObservableCollection<Model> SubItems = new ObservableCollection<Model>();
    public ViewModel()
    {
        Items = new ObservableCollection<Model>();
        SubItems.Add(new Model()
        {
            Item = "Item1",
        });
        SubItems.Add(new Model()
        {
            Item = "Item2",
        }); SubItems.Add(new Model()
        {
            Item = "Item3",
        });
        Items.Add(new Model()
        {
            Item = "Explore",
            Icon = new Path()
            {
                Data = Geometry.Parse("M6.0033803,5.705333 L7.0853577,...),
                .....
            },
            SubItems=SubItems
        });
        Items.Add(new Model()
        {
            Item = "My music",
            Icon = new Path()
            {
                Data = Geometry.Parse("M2.5,10.701 C1.6729736,10.701 1,...),
                .....
            }
        });
        Items.Add(new Model()
        {
            Item = "Recommended",
            Icon = new Path()
            {
                Data = Geometry.Parse("M11.5,10 C11.224,10 11,10.225 11,...),
                .....
            }
        });
        Items.Add(new Model()
        {
            Item = "Recent Playlist",
            Icon = new Path()
```

```

{
    Data = Geometry.Parse("M6.9990103,3.0000041 L7.9990076,....),
    .....
}
});
Items.Add(new Model()
{
    Item = "Radio",
    Icon = new Path()
    {
        Data = Geometry.Parse("M2.5,11.472009 L6.5,11.472009 C6.776,....),
        .....
    }
});
}
}

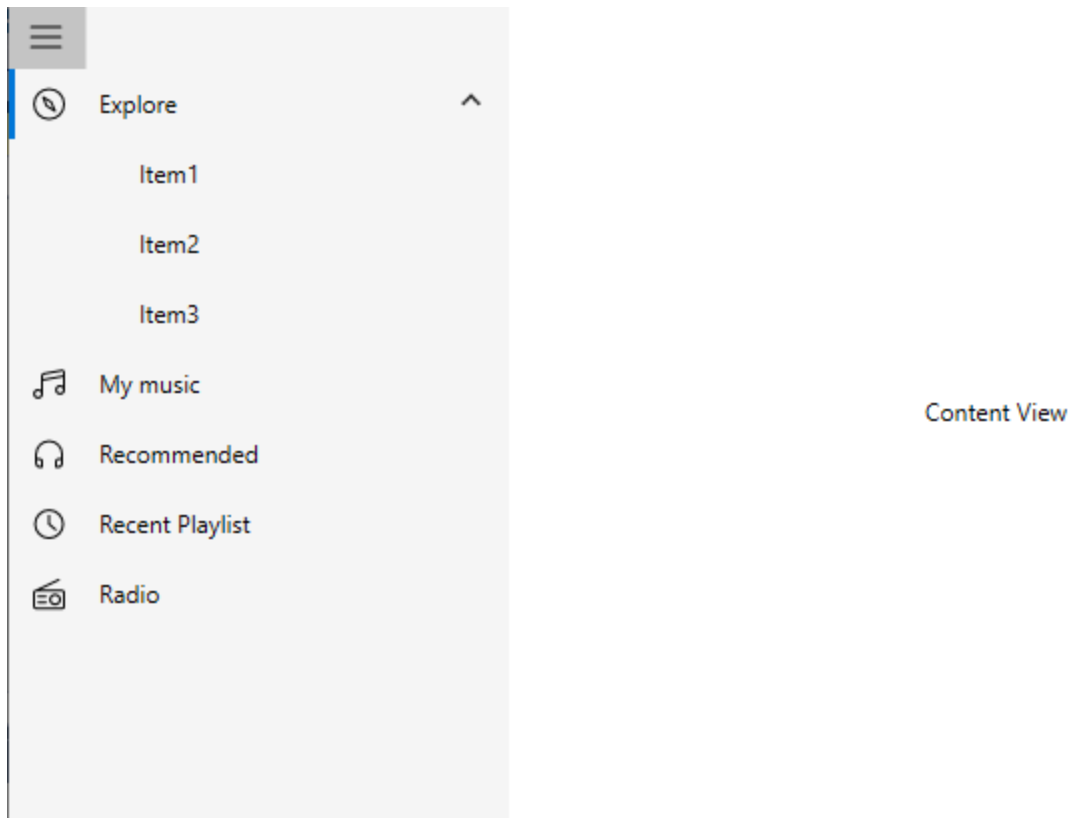
```

XML

```

<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Window.Resources>
<Style x:Key="ItemStyle" TargetType="syncfusion:NavigationItem">
<Setter Property="Icon" Value="{Binding Icon}" />
<Setter Property="DisplayMemberPath" Value="Item" />
<Setter Property="ItemsSource" Value="{Binding SubItems}" />
</Style>
</Window.Resources>
<syncfusion:SfNavigationDrawer
x:Name="navigationDrawer"
DisplayMemberPath="Item"
DisplayMode="Expanded"
ItemContainerStyle="{StaticResource ItemStyle}"
ItemsSource="{Binding Items}">
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```



Note: View [sample](#) in GitHub

IndentationWidth

This property used to change the horizontal position of sub items. Left margin of the sub items depends on the [IndentationWidth](#) property.

XML

```
<syncfusion:SfNavigationDrawer
x:Name="navigationDrawer"
DisplayMode="Compact"
IndentationWidth="40">
  <syncfusion:NavigationItem Header="Inbox">
    <syncfusion:NavigationItem.Icon>
      <Path Data="M32.032381, 16.445429 L25.410999, ..... />
    </syncfusion:NavigationItem.Icon>
    <syncfusion:NavigationItem Header="Primary">
      <syncfusion:NavigationItem.Icon>
        <Path Data="M9.5189972,7.3780194C8.3389893,...../>
      </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Social">
      <syncfusion:NavigationItem.Icon>
        <Path Data="M22.133972,14.194015C17.582977,...../>
      </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Promotions">
      <syncfusion:NavigationItem.Icon>
        <Path Data="M9.4614787,7.2521966C8.897512,...../>
      </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
  </syncfusion:SfNavigationDrawer>
```

```

</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Sent mail">
<syncfusion:NavigationItem.Icon>
<Path Data="M42.128046,6.7269681 L18.53705,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Drafts">
<syncfusion:NavigationItem.Icon>
<Path Data="M6.9999996,48.353 L19,48.353 19,...../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```

C#

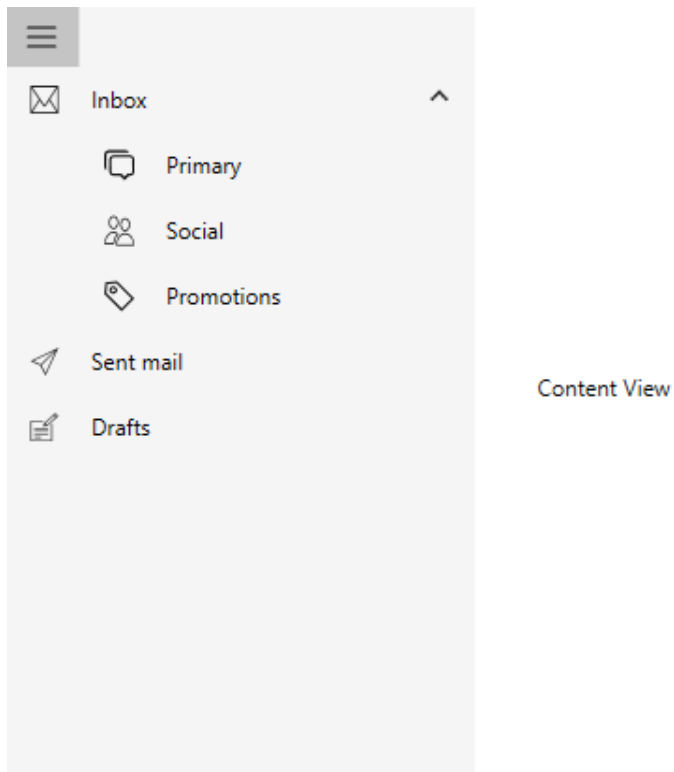
```

SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Compact;
navigationDrawer.IndentationWidth = 40;
NavigationItemsCollection navigationSubItems = new
NavigationItemsCollection();
navigationSubItems.Add(new NavigationItem()
{
Header = "Primary",
Icon = new Path()
{
Data = Geometry.Parse("M9.5189972,7.3780194C8.3389893,....),
.....
},
});
navigationSubItems.Add(new NavigationItem()
{
Header = "Social",
Icon = new Path()
{
Data = Geometry.Parse("M22.133972,14.194015C17.582977,....),
.....
},
});
navigationSubItems.Add(new NavigationItem()
{
Header = "Promotions",
Icon = new Path()
{
Data = Geometry.Parse("M9.4614787,7.2521966C8.897512,....),
.....
},
});

```

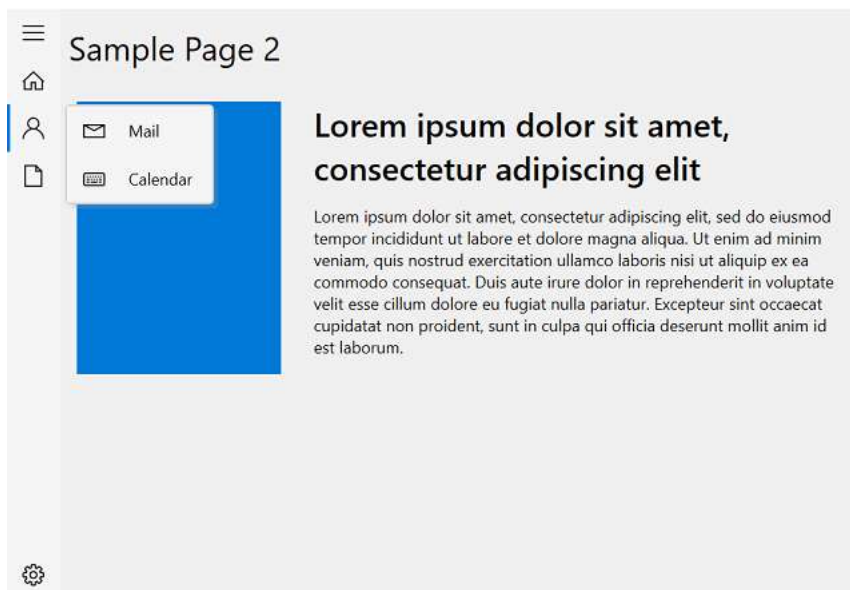
```
});  
navigationDrawer.Items.Add(new NavigationItem()  
{  
    Header = "Inbox",  
    Icon = new Path()  
    {  
        Data = Geometry.Parse("M32.032381, 16.445429 L25.410999,....),  
        .....  
    },  
    Items = navigationSubItems  
});  
navigationDrawer.Items.Add(new NavigationItem()  
{  
    Header = "Sent mail",  
    Icon = new Path()  
    {  
        Data = Geometry.Parse("M42.128046,6.7269681 L18.53705,....),  
        .....  
    }  
});  
navigationDrawer.Items.Add(new NavigationItem()  
{  
    Header = "Drafts",  
    Icon = new Path()  
    {  
        Data = Geometry.Parse("M6.9999996,48.353 L19,48.353 19,....),  
        .....  
    }  
});  
Label label = new Label();  
label.Content = "Content View";  
label.HorizontalAlignment = HorizontalAlignment.Center;  
label.VerticalAlignment = VerticalAlignment.Center;  
label.Height = 30;  
label.Width = 150;  
navigationDrawer.ContentView = label;  
this.Content = navigationDrawer;
```

See also [Populating using built in items](#) section.



Popup support

The sub-items will be displayed in the popup when the drawer menu is collapsed in the compact and expanded display modes.



Handling Selection in WPF Navigation Drawer (SfNavigationDrawer)

This section explains the handling of selection in SfNavigationDrawer.

The [SelectedItem](#) property is used for getting or setting the currently selected item of the Navigation Drawer. There are two cases in accessing the SelectedItem property.

The first is when the Navigation Drawer is populated with NavigationItems, then the SelectedItem property is of type NavigationItem.

C#

```
var selectedItem = this.navigationDrawer.SelectedItem as NavigationItem;
```

The other one is when the Navigation Drawer is bound to a collection of custom objects, the SelectedItem is of the type of the custom object.

C#

```
var selectedItem = this.navigationDrawer.SelectedItem as NavigationModel;
```

Note: View [sample](#) in GitHub

Header and Footer in WPF Navigation Drawer (SfNavigationDrawer)

The Navigation menu's header and footer parts are fully customizable. This section explains the customization of header and footer.

Customizing the Toggle button and Header

The Navigation Drawer built-in toggle button and header can be customized using the [ToggleButtonContentTemplate](#), [ToggleButtonIconTemplate](#), [IsToggleButtonVisible](#) properties.

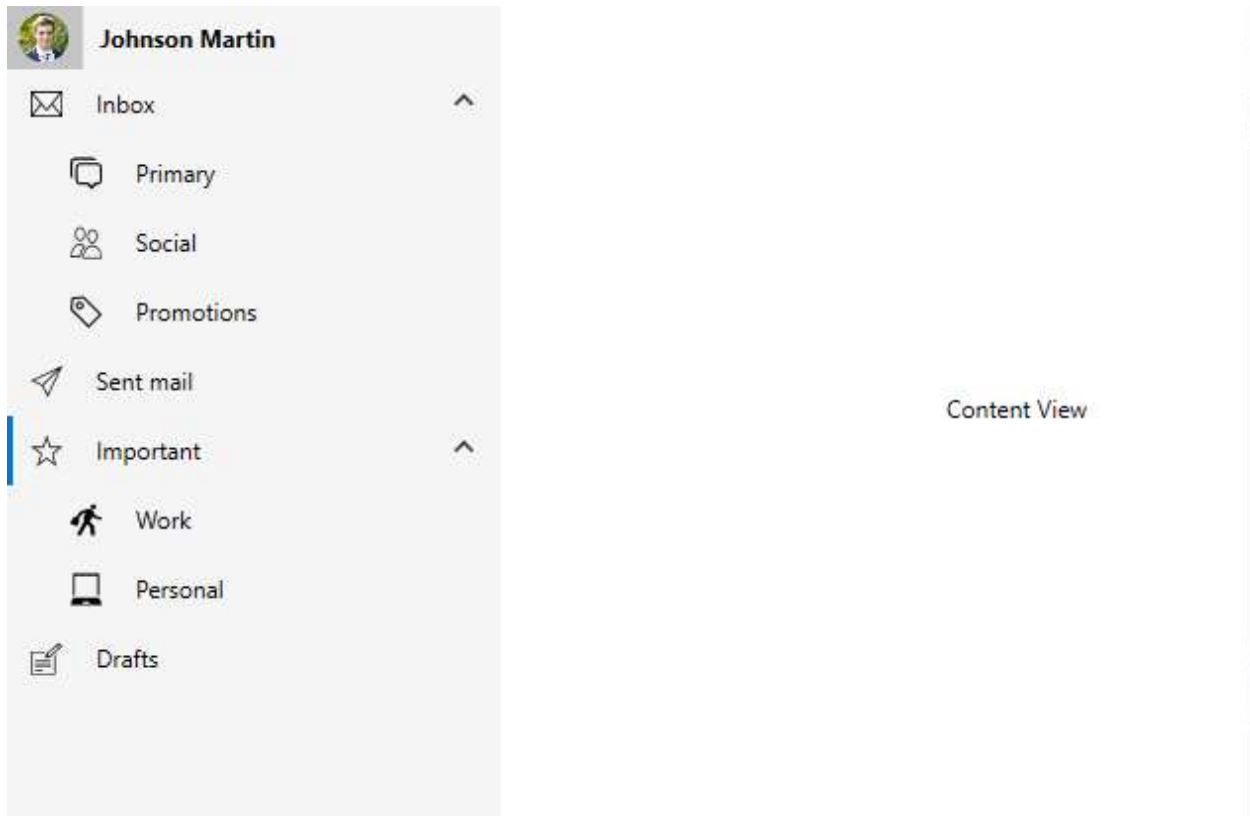
XML

```
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
    DisplayMode="Expanded">
    <syncfusion:NavigationItem Header="Inbox">
    <syncfusion:NavigationItem.Icon>
    <Path Data="M32.032381, 16.445429 L25.410999, 23 22.598995, 23 15.853724,
    16.561278 3.4150009, 29 44.586115, 29z M2.0000003, 3.3371553 L2.0000003,
    27.587 14.406845, 15.180154z M46.000002, 2.6187388 L33.45355, 15.038597
    46.000002, 27.585888z M3.4950623, 2.0000003 L23.399998, 21 24.589001, 21
    43.782564, 2.0000003z M0, 0 L48.000002, 0 48.000002, 31 0, 31z" ..... />
    </syncfusion:NavigationItem.Icon>
    <syncfusion:NavigationItem Header="Primary">
    <syncfusion:NavigationItem.Icon>
    <Path Data="M9.5189972,7.3780194C8.3389893,.../>
    </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Social">
    <syncfusion:NavigationItem.Icon>
    <Path Data="M22.133972,14.194015C17.582977,14.194015,13.821991,.../>
    </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Promotions">
    <syncfusion:NavigationItem.Icon>
    <Path Data="M9.4614787,7.2521966C8.897512,7.2521966 8.3335462,.../>
    </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Sent mail">
    <syncfusion:NavigationItem.Icon>
    <Path Data="M42.128046,6.7269681 L18.53705,30.317964 25.278003,.../>
    </syncfusion:NavigationItem.Icon>
```

```

</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Important">
<syncfusion:NavigationItem.Icon>
<Path Data="M25.085007,5.9780004 L20.577011,18.675001 6.3710022,.../>
</syncfusion:NavigationItem.Icon>
<syncfusion:NavigationItem Header="Work">
<syncfusion:NavigationItem.Icon>
<Path Data="M16.151009,4.5999908C16.851023,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Personal">
<syncfusion:NavigationItem.Icon>
<Path Data="M12.200012,21.899996L13.099976,.../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Drafts">
<syncfusion:NavigationItem.Icon>
<Path Data="M6.9999996,48.353 L19,48.353 19,..../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.ToggleButtonIconTemplate>
<DataTemplate>
<Image
x:Name="image"
Width="25"
Height="25"
HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"
Source="People.png" />
</DataTemplate>
</syncfusion:SfNavigationDrawer.ToggleButtonIconTemplate>
<syncfusion:SfNavigationDrawer.ToggleButtonContentTemplate>
<DataTemplate>
<Label
HorizontalAlignment="Left"
VerticalAlignment="Center"
Content="Johnson Martin" />
</DataTemplate>
</syncfusion:SfNavigationDrawer.ToggleButtonContentTemplate>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```



Customizing the Footer

The Navigation Drawer's Footer can be customized using the [FooterItems](#) property. `FooterItems` can be loaded with `NavigationItem` collection.

XML

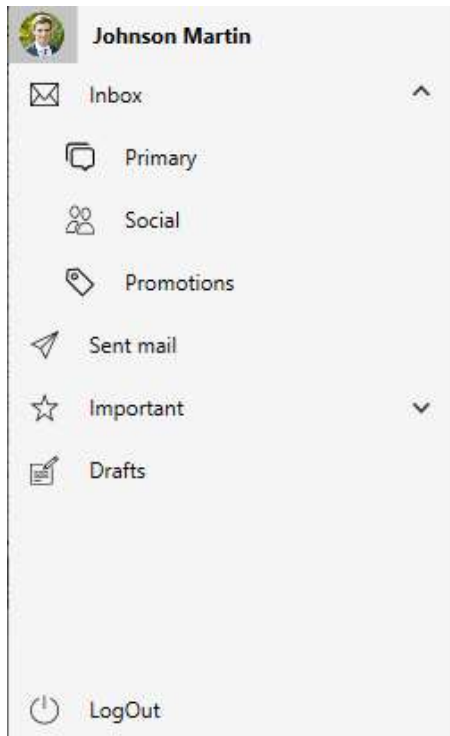
```
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
    DisplayMode="Expanded">
    <syncfusion:NavigationItem Header="Inbox">
    <syncfusion:NavigationItem.Icon>
    <Path
    Data="M32.032381, 16.445429 L25.410999, 23 22.598995,.../>
    </syncfusion:NavigationItem.Icon>
    <syncfusion:NavigationItem Header="Primary">
    <syncfusion:NavigationItem.Icon>
    <Path
    Data="M9.5189972,7.3780194C8.3389893,.../>
    </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Social">
    <syncfusion:NavigationItem.Icon>
    <Path
    Data="M22.133972,14.194015C17.582977,14.194015,.../>
    </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
    <syncfusion:NavigationItem Header="Promotions">
    <syncfusion:NavigationItem.Icon>
    <Path Data="M9.4614787,7.2521966C8.897512,.../>
    </syncfusion:NavigationItem.Icon>
```

```

</syncfusion:NavigationItem>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Sent mail">
  <syncfusion:NavigationItem.Icon>
    <Path Data="M42.128046,6.7269681 L18.53705,.../>
  </syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Important">
  <syncfusion:NavigationItem.Icon>
    <Path Data="M25.085007,5.9780004 L20.577011,.../>
  </syncfusion:NavigationItem.Icon>
<syncfusion:NavigationItem Header="Work">
  <syncfusion:NavigationItem.Icon>
    <Path Data="M16.151009,4.5999908C16.851023,.../>
  </syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Personal">
  <syncfusion:NavigationItem.Icon>
    <Path Data="M12.200012,21.899996L13.099976,.../>
  </syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Drafts">
  <syncfusion:NavigationItem.Icon>
    <Path Data="M6.9999996,48.353 L19,48.353 19,.../>
  </syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.ToggleButtonIconTemplate>
  <DataTemplate>
    <Image
      x:Name="image"
      Width="25"
      Height="25"
      HorizontalAlignment="Stretch"
      VerticalAlignment="Stretch"
      Source="People.png" />
    </DataTemplate>
  </syncfusion:SfNavigationDrawer.ToggleButtonIconTemplate>
  <syncfusion:SfNavigationDrawer.ToggleButtonContentTemplate>
    <DataTemplate>
      <Label
        HorizontalAlignment="Left"
        VerticalAlignment="Center"
        Content="Johnson Martin"
        FontWeight="Bold" />
    </DataTemplate>
  </syncfusion:SfNavigationDrawer.ToggleButtonContentTemplate>
  <syncfusion:SfNavigationDrawer.FooterItems>
    <syncfusion:NavigationItem Header="LogOut"
      SelectionBackground="Transparent">
      <syncfusion:NavigationItem.Icon>
        <Path Data="M13.999999,3.9500002 L13.999999,.../>
      </syncfusion:NavigationItem.Icon>
    </syncfusion:NavigationItem>
  </syncfusion:SfNavigationDrawer.FooterItems>
</syncfusion:SfNavigationDrawer.ContentView>
<Label

```

```
Width="150"  
Height="30"  
HorizontalAlignment="Center"  
VerticalAlignment="Center"  
Content="Content View" />  
</syncfusion:SfNavigationDrawer.ContentView>  
</syncfusion:SfNavigationDrawer>
```



Creating a custom header and footer

The header and footer part can also be customized by loading a custom view. This can be achieved in **Default** display mode. Using the [DrawerFooterView](#) and [DrawerFooterView](#) properties, custom views can be loaded to the header and footer.

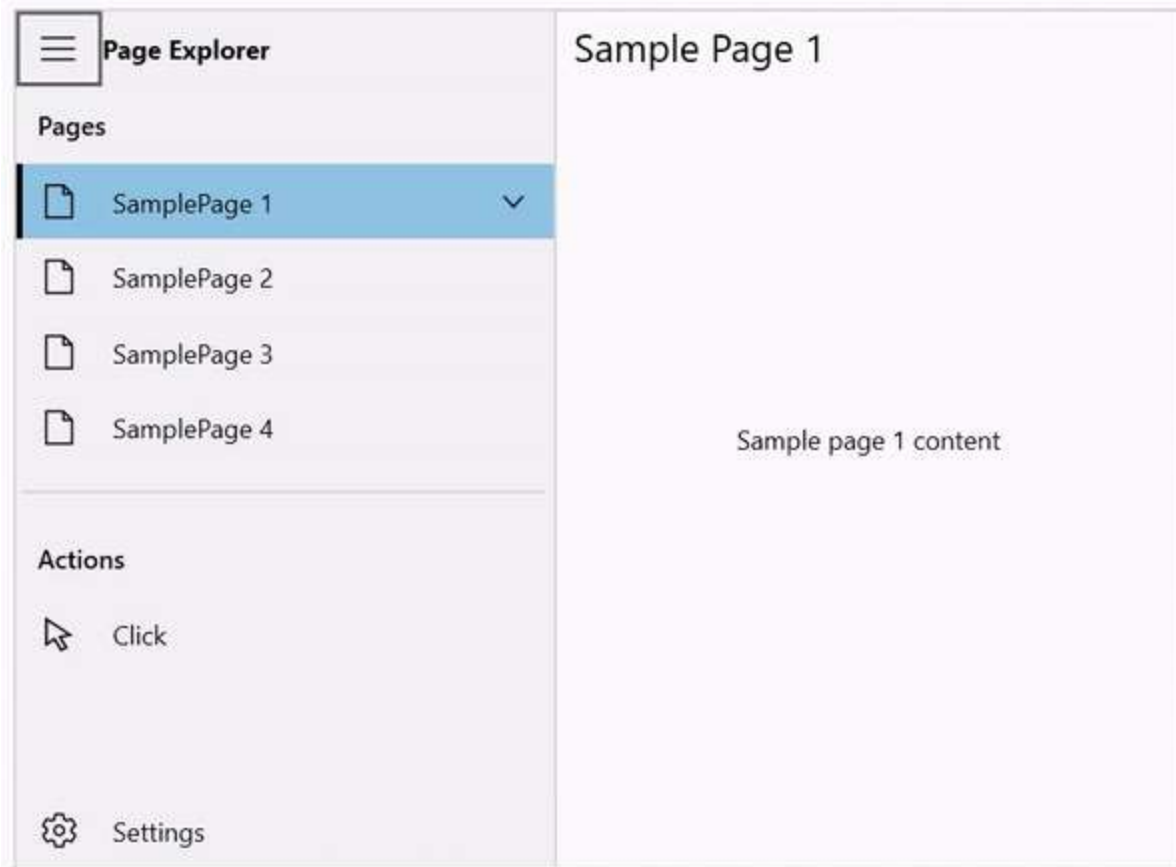
See also [Custom views](#) section for creating a custom navigation menu.

Note: View [sample](#) in GitHub

Keyboard Support in WPF NavigationDrawer (SfNavigationDrawer)

This section describes the keyboard support in SfNavigationDrawer.

- **Tab** - This key can be used to navigate between the toggle button, navigation items and footer items and the main content view.
- **Up** - This key is used to Navigate to the previous NavigationItem.
- **Down** - This key is used to Navigate to the next NavigationItem.
- **Enter and Space** - This key is used to select the currently focused item.



Commands and events in SfNavigationDrawer

This section describes the events and command support available in the Navigation Drawer sidebar.

Opening event

This [Opening](#) event gets triggered when the drawer menu is opening.

Using the `Cancel` property the opening of the drawer menu can be handled.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
Opening="NavigationDrawer_Opening">
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
```

```

</syncfusion:SfNavigationDrawer.DrawerHeaderView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                IsOpen = true
            };
            navigationDrawer.Opening += NavigationDrawer_Opening;
            Grid grid = new Grid()
            {
                Background = new
                SolidColorBrush((Color)ColorConverter.ConvertFromString("#31ade9"))
            };
            Label label = new Label()
            {
                Content = "Header View",
                HorizontalContentAlignment = HorizontalAlignment.Center,
                VerticalContentAlignment = VerticalAlignment.Center
            };
            grid.Children.Add(label);
            navigationDrawer.DrawerHeaderView = grid;
            this.Content = navigationDrawer;
        }

        private void NavigationDrawer_Opening(object sender,
            System.ComponentModel.CancelEventArgs e)
        {
            // To restrict drawer opening, by setting the e.Cancel value true.
        }
    }
}

```

Opened event

This [Opened](#) event gets triggered once the drawer menu is fully animated and opened.

XML

```

<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
Opened="NavigationDrawer_Opened">
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                IsOpen = true
            };
            navigationDrawer.Opened += NavigationDrawer_Opened;
            Grid grid = new Grid()
            {
                Background = new
                SolidColorBrush((Color)ColorConverter.ConvertFromString("#31ade9"))
            };
            Label label = new Label()
            {
                Content = "Header View",
                HorizontalContentAlignment = HorizontalAlignment.Center,
                VerticalContentAlignment = VerticalAlignment.Center
            };
            grid.Children.Add(label);
            navigationDrawer.DrawerHeaderView = grid;
            this.Content = navigationDrawer;
        }
        private void NavigationDrawer_Opened(object sender, System.EventArgs e)
        {

```

```
//Necessary action perform here
}
}
}
```

Closing event

This [Closing](#) event gets triggered when the drawer is about to close.

Closing of the drawer menu can be handled using the `Cancel` property.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
Closing="NavigationDrawer_Closing">
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
</syncfusion:SfNavigationDrawer>
</Window>
```

C#

```
using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                IsOpen = true
            };
            navigationDrawer.Closing += NavigationDrawer_Closing;
            Grid grid = new Grid()
            {
```

```

Background = new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#31ade9"))
};
Label label = new Label()
{
    Content = "Header View",
    HorizontalContentAlignment = HorizontalAlignment.Center,
    VerticalContentAlignment = VerticalAlignment.Center
};
grid.Children.Add(label);
navigationDrawer.DrawerHeaderView = grid;
this.Content = navigationDrawer;
}
private void NavigationDrawer_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
{
    // To restrict drawer closing, by setting the e.Cancel value to true.
}
}
}

```

Closed event

This [Closed](#) event gets triggered once the drawer menu is Closed.

XML

```

<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
Closed="NavigationDrawer_Closed">
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>

```

```

/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
        {
            IsOpen = true
        };
        navigationDrawer.Closed += NavigationDrawer_Closed;
        Grid grid = new Grid()
        {
            Background = new
            SolidColorBrush((Color)ColorConverter.ConvertFromString("#31ade9"))
        };
        Label label = new Label()
        {
            Content = "Header View",
            HorizontalContentAlignment = HorizontalAlignment.Center,
            VerticalContentAlignment = VerticalAlignment.Center
        };
        grid.Children.Add(label);
        navigationDrawer.DrawerHeaderView = grid;
        this.Content = navigationDrawer;
    }
    private void NavigationDrawer_Closed(object sender, System.EventArgs e)
    {
        // Necessary action perform here
    }
}

```

ItemClicked event

The [ItemClicked](#) event gets triggered when the item in the navigation menu is clicked.

Note: This event will get triggered only when the display mode is either compact or expanded.

XML

```

<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
DisplayMode="Compact" ItemClicked="NavigationDrawer_ItemClicked">
  <syncfusion:NavigationItem Header="Inbox">
    <syncfusion:NavigationItem.Icon>
      <Path
Data="M32.032381, 16.445429 L25.410999,..... />
    </syncfusion:NavigationItem.Icon>
  </syncfusion:NavigationItem>
  <syncfusion:NavigationItem Header="Sent mail">
    <syncfusion:NavigationItem.Icon>
      <Path Data="M42.128046,6.7269681 L18.53705,..../>
    </syncfusion:NavigationItem.Icon>
  </syncfusion:NavigationItem>
  <syncfusion:NavigationItem Header="Drafts">
    <syncfusion:NavigationItem.Icon>

```

```

<Path Data="M6.9999996,48.353 L19,48.353 19,....."/>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```

C#

```

SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Compact;
navigationDrawer.ItemClicked += NavigationDrawer_ItemClicked;
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Inbox",
    Icon = new Path()
    {
        Data = Geometry.Parse("M32.032381, 16.445429 L25.410999,.....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Sent mail",
    Icon = new Path()
    {
        Data = Geometry.Parse("M42.128046,6.7269681 L18.53705,.....),
        .....
    }
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Drafts",
    Icon = new Path()
    {
        Data = Geometry.Parse("M6.9999996,48.353 L19,48.353 19,.....),
        .....
    }
});
Label label = new Label();
label.Content = "Content View";
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;

```

C#

```
private void NavigationDrawer_ItemClicked(object sender,
NavigationItemClickedEventArgs e)
{
    var clickedItem = e.Item;
}
```

ItemCollapsed event

The [ItemCollapsed](#) event gets triggered when the sub items gets collapsed.

Note: This event will get triggered only when the display mode is either compact or expanded.

XML

```
<syncfusion:SfNavigationDrawer
x:Name="navigationDrawer"
DisplayMode="Compact"
ItemCollapsed="NavigationDrawer_ItemCollapsed">
    <syncfusion:NavigationItem Header="Inbox">
        <syncfusion:NavigationItem.Icon>
            <Path
Data="M32.032381, 16.445429 L25.410999, ...."/>
        </syncfusion:NavigationItem.Icon>
        <syncfusion:NavigationItem Header="Primary">
            <syncfusion:NavigationItem.Icon>
                <Path
Data="M9.5189972,7.3780194C8.3389893, ..... />
            </syncfusion:NavigationItem.Icon>
        </syncfusion:NavigationItem>
        <syncfusion:NavigationItem Header="Social">
            <syncfusion:NavigationItem.Icon>
                <Path
Data="M22.133972,14.194015C17.582977, ...../>
            </syncfusion:NavigationItem.Icon>
        </syncfusion:NavigationItem>
        <syncfusion:NavigationItem Header="Promotions">
            <syncfusion:NavigationItem.Icon>
                <Path
Data="M9.4614787,7.2521966C8.897512, ...../>
            </syncfusion:NavigationItem.Icon>
        </syncfusion:NavigationItem>
        <syncfusion:NavigationItem Header="Sent mail">
            <syncfusion:NavigationItem.Icon>
                <Path
Data="M42.128046,6.7269681 L18.53705, ...../>
            </syncfusion:NavigationItem.Icon>
        </syncfusion:NavigationItem>
        <syncfusion:NavigationItem Header="Drafts">
            <syncfusion:NavigationItem.Icon>
                <Path
Data="M6.9999996,48.353 L19, ...../>
            </syncfusion:NavigationItem.Icon>
        </syncfusion:NavigationItem>
    </syncfusion:SfNavigationDrawer.ContentView>
</Label
Width="150"
```

```

Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```

C#

```

SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Compact;
navigationDrawer.ItemCollapsed += NavigationDrawer_ItemCollapsed;
NavigationItemsCollection navigationSubItems = new
NavigationItemsCollection();
navigationSubItems.Add(new NavigationItem()
{
    Header = "Primary",
    Icon = new Path()
    {
        Data = Geometry.Parse("M9.5189972,7.3780194C8.3389893,.....),
        .....
    },
});
navigationSubItems.Add(new NavigationItem()
{
    Header = "Social",
    Icon = new Path()
    {
        Data = Geometry.Parse("M22.133972,14.194015C17.582977,.....),
        .....
    },
});
navigationSubItems.Add(new NavigationItem()
{
    Header = "Promotions",
    Icon = new Path()
    {
        Data = Geometry.Parse("M9.4614787,7.2521966C8.897512,.....),
        .....
    },
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Inbox",
    Icon = new Path()
    {
        Data = Geometry.Parse("M32.032381, 16.445429 L25.410999, .....),
        .....
    },
    Items= navigationSubItems
});
navigationDrawer.Items.Add(new NavigationItem()
{
    Header = "Sent mail",
    Icon = new Path()
    {

```

```

Data = Geometry.Parse("M42.128046,6.7269681 L18.53705,.....),
.....
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "Drafts",
Icon = new Path()
{
Data = Geometry.Parse("M6.9999996,48.353 L19,.....),
.....
}
});
Label label = new Label();
label.Content = "Content View";
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;

```

C#

```

private void NavigationDrawer_ItemCollapsed(object sender,
NavigationItemCollapsedEventArgs e)
{
var collapsedItem = e.Item;
}

```

ItemExpanded event

The [ItemExpanded](#) event gets triggered when the sub items gets expanded.

Note: This event will get triggered only when the display mode is either compact or expanded.

XML

```

<syncfusion:SfNavigationDrawer
x:Name="navigationDrawer"
DisplayMode="Compact"
ItemExpanded="NavigationDrawer_ItemExpanded">
<syncfusion:NavigationItem Header="Inbox">
<syncfusion:NavigationItem.Icon>
<Path Data="M32.032381, 16.445429 L25.410999,...../>
</syncfusion:NavigationItem.Icon>
<syncfusion:NavigationItem Header="Primary">
<syncfusion:NavigationItem.Icon>
<Path Data="M9.5189972,7.3780194C8.3389893,...../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Social">
<syncfusion:NavigationItem.Icon>
<Path Data="M22.133972,14.194015C17.582977,...../>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Promotions">

```



```

<syncfusion:NavigationItem.Icon>
<Path Data="M9.4614787,7.2521966C8.897512,...."/>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Sent mail">
<syncfusion:NavigationItem.Icon>
<Path Data="M42.128046,6.7269681 L18.53705,30.3....."/>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Drafts">
<syncfusion:NavigationItem.Icon>
<Path Data="M6.9999996,48.353 L19,48.353 19,....."/>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```

C#

```

SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
navigationDrawer.DisplayMode = DisplayMode.Compact;
navigationDrawer.ItemExpanded += NavigationDrawer_ItemExpanded;
NavigationItemsCollection navigationSubItems = new
NavigationItemsCollection();
navigationSubItems.Add(new NavigationItem()
{
Header = "Primary",
Icon = new Path()
{
Data = Geometry.Parse("M9.5189972,7.3780194C8.3389893,.....),
.....
},
});
navigationSubItems.Add(new NavigationItem()
{
Header = "Social",
Icon = new Path()
{
Data = Geometry.Parse("M22.133972,14.194015C17.582977,.....),
.....
},
});
navigationSubItems.Add(new NavigationItem()
{
Header = "Promotions",
Icon = new Path()
{
Data = Geometry.Parse("M9.4614787,7.2521966C8.897512,.....),
.....
},
});

```

```

.....
},
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "Inbox",
Icon = new Path()
{
Data = Geometry.Parse("M32.032381, 16.445429 L25.410999, .....),
.....
},
Items= navigationSubItems
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "Sent mail",
Icon = new Path()
{
Data = Geometry.Parse("M42.128046, 6.7269681 L18.53705, .....),
.....
},
});
navigationDrawer.Items.Add(new NavigationItem()
{
Header = "Drafts",
Icon = new Path()
{
Data = Geometry.Parse("M6.9999996, 48.353 L19, 48.353 19, .....),
.....
},
});
Label label = new Label();
label.Content = "Content View";
label.HorizontalAlignment = HorizontalAlignment.Center;
label.VerticalAlignment = VerticalAlignment.Center;
label.Height = 30;
label.Width = 150;
navigationDrawer.ContentView = label;
this.Content = navigationDrawer;

```

C#

```

private void NavigationDrawer_ItemExpanded(object sender,
NavigationItemExpandedEventArgs e)
{
var expandedItem = e.Item;
}

```

Commands

The command binds to the **NavigationItem**, and will be executed while the items gets clicked.

The **Tab** and **Button** navigation item types alone executes the [Command](#). [CommandParameter](#) is user defined data value that can be passed to the **Command** when it is executed.

*DelegateCommand class***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
}

```

ViewModel

C#

```

public class ViewModel:INotifyPropertyChanged
{
    private bool _canperformaction = true;
    public event PropertyChangedEventHandler PropertyChanged;
    public void OnPropertyChanged(string name)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(name));
        }
    }
    public ViewModel()
    {
        ClickCommand = new DelegateCommand<object>(ClickAction,
        CanPerformClickAction);
    }
    public bool CanPerformAction
    {
        get
        {
            return _canperformaction;
        }
        set
        {
            _canperformaction = value;
            this.ClickCommand.RaiseCanExecuteChanged();
            this.OnPropertyChanged("CanPerformAction");
        }
    }
    private bool CanPerformClickAction(object parameter)
    {
        return CanPerformAction;
    }
}

```

```

public DelegateCommand<object> ClickCommand { get; set; }
private void ClickAction(object parameter)
{
    MessageBox.Show(parameter.ToString() + " item has been clicked");
}
}

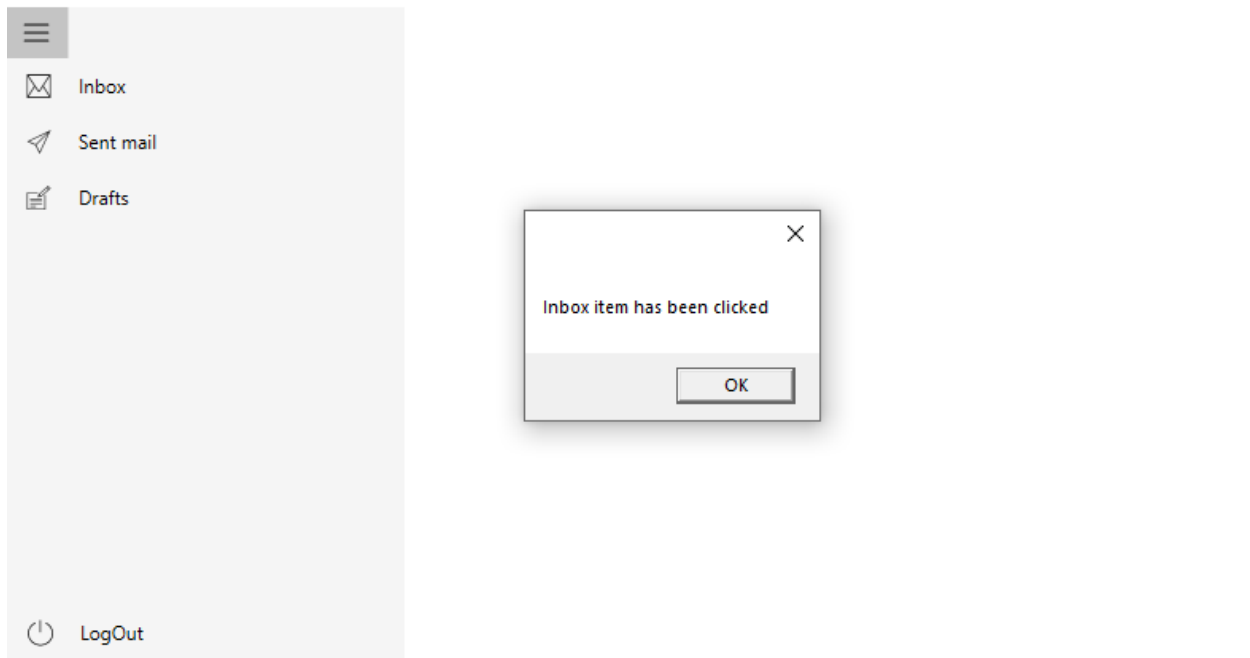
```

XML

```

<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<syncfusion:SfNavigationDrawer
x:Name="navigationDrawer"
DisplayMode="Compact">
<syncfusion:NavigationItem
Command="{Binding ClickCommand}"
CommandParameter="Inbox"
Header="Inbox">
<syncfusion:NavigationItem.Icon>
<Path Data="M32.032381, 16.445429 L25.410999, 23 22.598995,....."/>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Sent mail">
<syncfusion:NavigationItem.Icon>
<Path Data="M42.128046, 6.7269681 L18.53705,....."/>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:NavigationItem Header="Drafts">
<syncfusion:NavigationItem.Icon>
<Path Data="M6.9999996, 48.353 L19,....."/>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
<syncfusion:SfNavigationDrawer.FooterItems>
<syncfusion:NavigationItem
Command="{Binding ClickCommand}"
CommandParameter="LogOut"
Header="LogOut"
ItemType="Button">
<syncfusion:NavigationItem.Icon>
<Path
Data="M13.999999, 3.9500002 L13.999999,....."/>
</syncfusion:NavigationItem.Icon>
</syncfusion:NavigationItem>
</syncfusion:SfNavigationDrawer.FooterItems>
<syncfusion:SfNavigationDrawer.ContentView>
<Label
Width="150"
Height="30"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Content="Content View" />
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>

```



Note: The `Command` execution is only applicable for compact and expanded display mode.

Note: View [sample](#) in GitHub

Custom views in WPF Navigation Drawer (SfNavigationDrawer)

Populating the with custom views

The WPF Navigation Drawer sidebar can be populated with custom views by setting the [DisplayMode](#) as [Default](#). In this mode the sidebar can be customized as three parts.

Basically the sidebar will be in hidden state and can be displayed by swipe gestures.

- `DrawerHeaderView` - A custom view can be provided to be displayed as a header part of the navigation menu. This can be set to the `[DrawerHeaderView]()` property.
- `DrawerContentView` - A custom view can be provided to be displayed in the body of the navigation menu. This can be set to the `[DrawerContentView]()` property.
- `DrawerFooterView` - A custom view can be provided to be displayed as a footer part of the navigation menu. This can be set to the `[DrawerFooterView]()` property.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
```

```

<syncfusion:SfNavigationDrawer x:Name="navigationDrawer" DrawerWidth="300">
  <syncfusion:SfNavigationDrawer.ContentView>
    <Grid x:Name="mainContentView"
      Background="White">
      <Grid.RowDefinitions>
        <RowDefinition Height="auto"/>
        <RowDefinition/>
      </Grid.RowDefinitions>
      <StackPanel Background="#1aald6"
        Orientation="Horizontal">
        <Button x:Name="hamburgerButton"
          BorderBrush="Transparent"
          Height="50" Width="50"
          Background="#1aald6" Foreground="White"
          Click="HamburgerButton_Click">
          <Image Source="hamburger_icon.png"
            Height="20"
            Width="20"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"/>
          </Button>
          <Label x:Name="headerLabel"
            Height="50"
            HorizontalContentAlignment="Center"
            VerticalContentAlignment="Center"
            Content="Home"
            Foreground="White"
            Background="#1aald6"/>
          </StackPanel>
          <Label Grid.Row="1"
            x:Name="contentLabel"
            VerticalContentAlignment="Center"
            HorizontalContentAlignment="Center"
            Content="Content View"
            Foreground="Black"/>
          </Grid>
        </syncfusion:SfNavigationDrawer.ContentView>
        <syncfusion:SfNavigationDrawer.DrawerHeaderView>
          <Grid Background="#31ade9">
            <Label Content="Header View"
              HorizontalContentAlignment="Center"
              VerticalContentAlignment="Center"/>
            </Grid>
          </syncfusion:SfNavigationDrawer.DrawerHeaderView>
          <syncfusion:SfNavigationDrawer.DrawerContentView>
            <Grid>
              <ListBox x:Name="list"
                ItemsSource="{Binding Contents}">
                <ListBox.ItemTemplate>
                  <DataTemplate>
                    <TextBlock Text="{Binding Name}"
                      Foreground="Black"/>
                  </DataTemplate>
                </ListBox.ItemTemplate>
              </ListBox>
            </Grid>
          </syncfusion:SfNavigationDrawer.DrawerContentView>

```

```

<syncfusion:SfNavigationDrawer.DrawerFooterView>
<Grid Background="#31ade9">
<Label Content="Footer View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerFooterView>
</syncfusion:SfNavigationDrawer>
</Window>

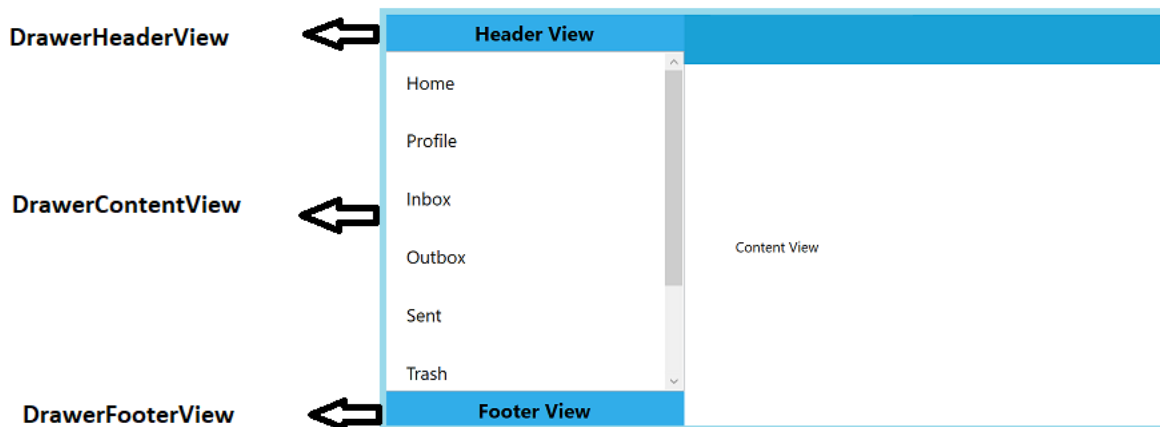
```

C#

```

using System.Collections.Generic;
using System.Windows;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void HamburgerButton_Click(object sender, RoutedEventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
        public class ViewModel
        {
            public ViewModel()
            {
                Contents = new List<Model>();
                Contents.Add(new Model() { Name = "Home" });
                Contents.Add(new Model() { Name = "Profile" });
                Contents.Add(new Model() { Name = "Inbox" });
                Contents.Add(new Model() { Name = "Outbox" });
                Contents.Add(new Model() { Name = "Sent" });
                Contents.Add(new Model() { Name = "Trash" });
                Contents.Add(new Model() { Name = "Sign Out" });
            }
            public List<Model> Contents { get; set; }
        }
        public class Model
        {
            public string Name { get; set; }
        }
    }
}

```

Customizing the drawer

Customize Panel size

The size of sidebar can be adjusted using the `DrawerHeight` and `DrawerWidth` properties.

DrawerHeight

The `DrawerHeight` property is used to change the height of side pane when the Position is Top or Bottom.

Note: The `DrawerHeight` property is applicable only when the `DisplayMode` is Default.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
DrawerHeight="200"
Position="Top">
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="This is a very short content used to demonstrate the
DrawerHeight property " />
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
</syncfusion:SfNavigationDrawer>
</Window>
```

C#

```
using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
```

```

namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                DrawerHeight = 200,
                Position = Position.Top
            };
            Grid grid = new Grid()
            {
                Background = new
                SolidColorBrush((Color)ColorConverter.ConvertFromString("#31ade9"))
            };
            Label label = new Label()
            {
                Content = "This is a very short content used to demonstrate the DrawerHeight property "
            };
            grid.Children.Add(label);
            navigationDrawer.DrawerHeaderView = grid;
            this.Content = navigationDrawer;
        }
    }
}

```

DrawerWidth

The **DrawerWidth** property is used to change the width of side pane when the Position is Left or Right.

Note: The DrawerWidth property is applicable only when the DisplayMode is Default.

XML

```

<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
    <syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
DrawerWidth="200"
Position="Left">
        <syncfusion:SfNavigationDrawer.DrawerHeaderView>
            <Grid Background="#31ade9">
                <TextBlock Text="This is a very short content used to demonstrate the
DrawerHeight property "
TextWrapping="Wrap"/>
            </Grid>
        </syncfusion:SfNavigationDrawer.DrawerHeaderView>
    </syncfusion:SfNavigationDrawer>
</Window>

```

```

</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                DrawerWidth = 200,
                Position = Position.Left,
                IsOpen = true
            };
            Grid grid = new Grid()
            {
                Background = new
                SolidColorBrush((Color)ColorConverter.ConvertFromString("#31ade9"))
            };
            TextBlock label = new TextBlock()
            {
                Text = "This is a very short content used to demonstrate the DrawerHeight
                property",
                TextWrapping = TextWrapping.Wrap
            };
            grid.Children.Add(label);
            navigationDrawer.DrawerHeaderView = grid;
            this.Content = navigationDrawer;
        }
    }
}

```

Navigation Pane Sides

The supplemental pane can be drawn in and out from all the four sides. The **Position** property is used to change the side of pane and the values are:

- Left
- Right
- Top
- Bottom

Note: The Position property is applicable only when the DisplayMode is Default.

The default position is Left.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
DrawerWidth="300"
Position="Left">
<syncfusion:SfNavigationDrawer.ContentView>
<Grid x:Name="mainContentView"
Background="White">
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
<RowDefinition/>
</Grid.RowDefinitions>
<StackPanel Background="#1aa1d6"
Orientation="Horizontal">
<Button x:Name="hamburgerButton"
BorderBrush="Transparent"
Height="50" Width="50"
HorizontalAlignment="Left"
Background="#1aa1d6" Foreground="White"
Click="HamburgerButton_Click">
<Image Source="hamburger_icon.png"
Height="20"
Width="20"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Button>
<Label x:Name="headerLabel"
Height="50"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"
Content="Home"
Foreground="White"
Background="#1aa1d6"/>
</StackPanel>
<Label Grid.Row="1"
x:Name="contentLabel"
VerticalContentAlignment="Center"
HorizontalContentAlignment="Center"
Content="Content View"
Foreground="Black"/>
</Grid>
```

```

</syncfusion:SfNavigationDrawer.ContentView>
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
<syncfusion:SfNavigationDrawer.DrawerContentView>
<Grid>
<ListBox x:Name="list"
ItemsSource="{Binding Contents}">
<ListBox.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Black"/>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerContentView>
<syncfusion:SfNavigationDrawer.DrawerFooterView>
<Grid Background="#31ade9">
<Label Content="Footer View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerFooterView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

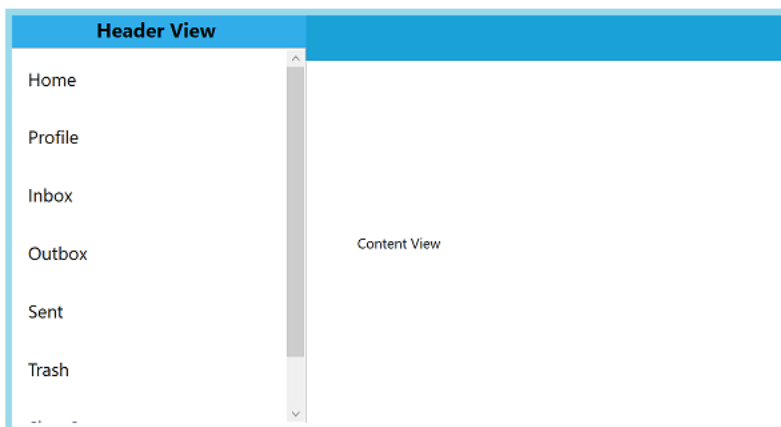
using System.Collections.Generic;
using System.Windows;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            navigationDrawer.Position =
            Syncfusion.UI.Xaml.NavigationDrawer.Position.Left;
        }
        private void HamburgerButton_Click(object sender, RoutedEventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
    public class ViewModel
    {
        public ViewModel()

```

```

{
    Contents = new List<Model>();
    Contents.Add(new Model() { Name = "Home" });
    Contents.Add(new Model() { Name = "Profile" });
    Contents.Add(new Model() { Name = "Inbox" });
    Contents.Add(new Model() { Name = "Outbox" });
    Contents.Add(new Model() { Name = "Sent" });
    Contents.Add(new Model() { Name = "Trash" });
    Contents.Add(new Model() { Name = "Sign Out" });
}
public List<Model> Contents { get; set; }
}
public class Model
{
    public string Name { get; set; }
}
}

```



The following code example shows how to set the SfNavigationDrawer to the right.

XML

```

<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
    xmlns:local="clr-namespace:NavigationDrawerWPF"
    mc:Ignorable="d"
    Title="MainWindow" Height="450" Width="800">
    <Window.DataContext>
    <local:ViewModel/>
    </Window.DataContext>
    <syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
    DrawerWidth="300"
    Position="Right">
    <syncfusion:SfNavigationDrawer.ContentView>
    <Grid x:Name="mainContentView"
    Background="White">
    <Grid.RowDefinitions>

```

```

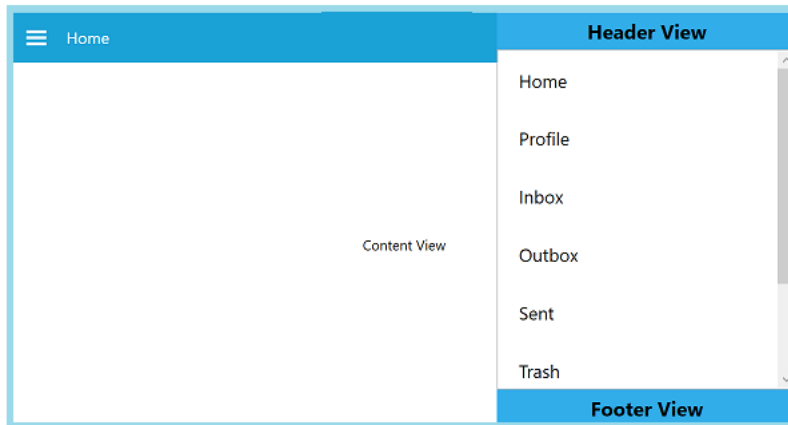
<RowDefinition Height="auto"/>
</RowDefinition>
</Grid.RowDefinitions>
<StackPanel Background="#1aa1d6"
Orientation="Horizontal">
<Button x:Name="hamburgerButton"
BorderBrush="Transparent"
Height="50" Width="50"
HorizontalAlignment="Left"
Background="#1aa1d6" Foreground="White"
Click="HamburgerButton_Click">
<Image Source="hamburger_icon.png"
Height="20"
Width="20"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Button>
<Label x:Name="headerLabel"
Height="50"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"
Content="Home"
Foreground="White"
Background="#1aa1d6"/>
</StackPanel>
<Label Grid.Row="1"
x:Name="contentLabel"
VerticalContentAlignment="Center"
HorizontalContentAlignment="Center"
Content="Content View"
Foreground="Black"/>
</Grid>
</syncfusion:SfNavigationDrawer.ContentView>
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
<syncfusion:SfNavigationDrawer.DrawerContentView>
<Grid>
<ListBox x:Name="list"
ItemsSource="{Binding Contents}">
<ListBox.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Black"/>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerContentView>
<syncfusion:SfNavigationDrawer.DrawerFooterView>
<Grid Background="#31ade9">
<Label Content="Footer View"
HorizontalContentAlignment="Center"

```

```
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerFooterView>
</syncfusion:SfNavigationDrawer>
</Window>
```

C#

```
using System.Collections.Generic;
using System.Windows;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            navigationDrawer.Position =
            Syncfusion.UI.Xaml.NavigationDrawer.Position.Right;
        }
        private void HamburgerButton_Click(object sender, RoutedEventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
    public class ViewModel
    {
        public ViewModel()
        {
            Contents = new List<Model>();
            Contents.Add(new Model() { Name = "Home" });
            Contents.Add(new Model() { Name = "Profile" });
            Contents.Add(new Model() { Name = "Inbox" });
            Contents.Add(new Model() { Name = "Outbox" });
            Contents.Add(new Model() { Name = "Sent" });
            Contents.Add(new Model() { Name = "Trash" });
            Contents.Add(new Model() { Name = "Sign Out" });
        }
        public List<Model> Contents { get; set; }
    }
    public class Model
    {
        public string Name { get; set; }
    }
}
```

The following code example shows how to set the SfNavigationDrawer at the top.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
  <Window.DataContext>
    <local:ViewModel/>
  </Window.DataContext>
  <syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
  DrawerHeight="200"
  Position="Top">
    <syncfusion:SfNavigationDrawer.ContentView>
      <Grid x:Name="mainContentView"
      Background="White">
        <Grid.RowDefinitions>
          <RowDefinition Height="auto"/>
        </Grid.RowDefinitions>
        <StackPanel Background="#1aa1d6"
        Orientation="Horizontal">
          <Button x:Name="hamburgerButton"
          BorderBrush="Transparent"
          Height="50" Width="50"
          HorizontalAlignment="Left"
          Background="#1aa1d6" Foreground="White"
          Click="HamburgerButton_Click">
            <Image Source="hamburger_icon.png"
            Height="20"
            Width="20"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"/>
          </Button>
          <Label x:Name="headerLabel"
```

```

Height="50"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"
Content="Home"
Foreground="White"
Background="#1aa1d6"/>
</StackPanel>
<Label Grid.Row="1"
x:Name="contentLabel"
VerticalContentAlignment="Center"
HorizontalContentAlignment="Center"
Content="Content View"
Foreground="Black"/>
</Grid>
</syncfusion:SfNavigationDrawer.ContentView>
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
<syncfusion:SfNavigationDrawer.DrawerContentView>
<Grid>
<ListBox x:Name="list"
ItemsSource="{Binding Contents}">
<ListBox.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Black"/>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerContentView>
<syncfusion:SfNavigationDrawer.DrawerFooterView>
<Grid Background="#31ade9">
<Label Content="Footer View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerFooterView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

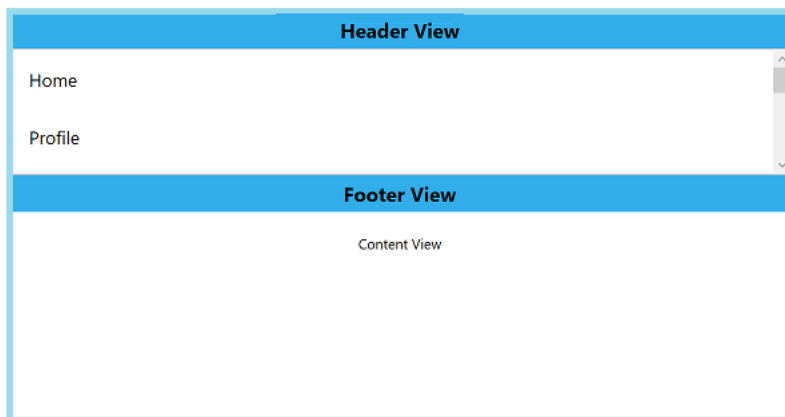
using System.Collections.Generic;
using System.Windows;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

```

```

public MainWindow()
{
    InitializeComponent();
    navigationDrawer.Position =
    Syncfusion.UI.Xaml.NavigationDrawer.Position.Top;
}
private void HamburgerButton_Click(object sender, RoutedEventArgs e)
{
    navigationDrawer.ToggleDrawer();
}
}
public class ViewModel
{
    public ViewModel()
    {
        Contents = new List<Model>();
        Contents.Add(new Model() { Name = "Home" });
        Contents.Add(new Model() { Name = "Profile" });
        Contents.Add(new Model() { Name = "Inbox" });
        Contents.Add(new Model() { Name = "Outbox" });
        Contents.Add(new Model() { Name = "Sent" });
        Contents.Add(new Model() { Name = "Trash" });
        Contents.Add(new Model() { Name = "Sign Out" });
    }
    public List<Model> Contents { get; set; }
}
public class Model
{
    public string Name { get; set; }
}
}

```



The following code example shows how to set the SfNavigationDrawer at the bottom.

XML

```

<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
DrawerHeight="150"
Position="Bottom">
<syncfusion:SfNavigationDrawer.ContentView>
<Grid x:Name="mainContentView"
Background="White">
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
<RowDefinition/>
</Grid.RowDefinitions>
<StackPanel Background="#1aa1d6"
Orientation="Horizontal">
<Button x:Name="hamburgerButton"
BorderBrush="Transparent"
Height="50" Width="50"
HorizontalAlignment="Left"
Background="#1aa1d6" Foreground="White"
Click="HamburgerButton_Click">
<Image Source="hamburger_icon.png"
Height="20"
Width="20"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Button>
<Label x:Name="headerLabel"
Height="50"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"
Content="Home"
Foreground="White"
Background="#1aa1d6"/>
</StackPanel>
<Label Grid.Row="1"
x:Name="contentLabel"
VerticalContentAlignment="Center"
HorizontalContentAlignment="Center"
Content="Content View"
Foreground="Black"/>
</Grid>
</syncfusion:SfNavigationDrawer.ContentView>
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
<syncfusion:SfNavigationDrawer.DrawerContentView>
<Grid>
<ListBox x:Name="list"

```

```

ItemsSource="{Binding Contents}">
<ListBox.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Black"/>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerContentView>
<syncfusion:SfNavigationDrawer.DrawerFooterView>
<Grid Background="#31ade9">
<Label Content="Footer View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerFooterView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

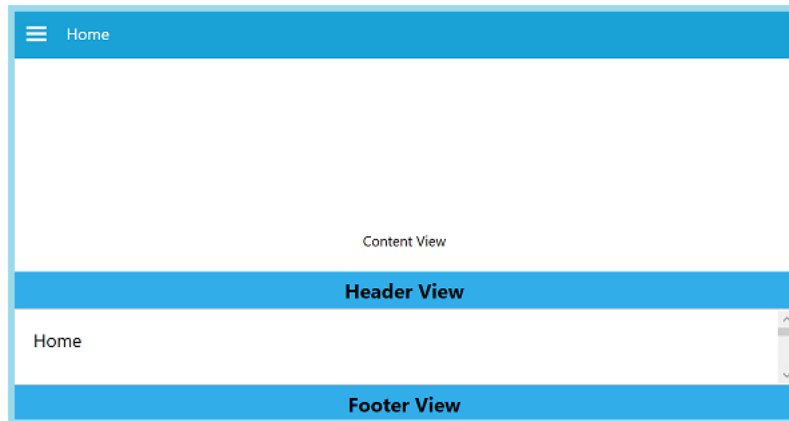
using System.Collections.Generic;
using System.Windows;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            navigationDrawer.Position =
            Syncfusion.UI.Xaml.NavigationDrawer.Position.Bottom;
        }
        private void HamburgerButton_Click(object sender, RoutedEventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
    public class ViewModel
    {
        public ViewModel()
        {
            Contents = new List<Model>();
            Contents.Add(new Model() { Name = "Home" });
            Contents.Add(new Model() { Name = "Profile" });
            Contents.Add(new Model() { Name = "Inbox" });
            Contents.Add(new Model() { Name = "Outbox" });
            Contents.Add(new Model() { Name = "Sent" });
            Contents.Add(new Model() { Name = "Trash" });
            Contents.Add(new Model() { Name = "Sign Out" });
        }
        public List<Model> Contents { get; set; }
    }
}

```

```

}
public class Model
{
public string Name { get; set; }
}
}

```



Swipe gesture and sensitivity

The NavigationDrawer supports swipe gesture for opening and closing the drawer.

Enabling swipe gesture

It can be enabled or disabled using the `EnableSwipeGesture` property.

Note: This property is applicable only when the DisplayMode is Default.

XML

```

<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
EnableSwipeGesture="True">
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

using Syncfusion.UI.Xaml.NavigationDrawer;

```

```

using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                EnableSwipeGesture = true
            };
            Grid grid = new Grid()
            {
                Background = new
                SolidColorBrush((Color)ColorConverter.ConvertFromString("#31ade9"))
            };
            Label label = new Label()
            {
                Content = "Header View",
                HorizontalContentAlignment = HorizontalAlignment.Center,
                VerticalContentAlignment = VerticalAlignment.Center
            };
            grid.Children.Add(label);
            navigationDrawer.DrawerHeaderView = grid;
            this.Content = navigationDrawer;
        }
    }
}

```

Swipe sensitivity

In smaller screens, may find it difficult to swipe the drawer from the edge, in such cases it can increase the swipe region using the `TouchThreshold` property. It can be set as below:

Note: The `TouchThreshold` property is applicable only when the `DisplayMode` is Default.

XML

```

<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
EnableSwipeGesture="True"
TouchThreshold="70">
<syncfusion:SfNavigationDrawer.DrawerHeaderView>

```

```

<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                EnableSwipeGesture = true,
                TouchThreshold = 70
            };
            Grid grid = new Grid()
            {
                Background = new
                SolidColorBrush((Color)ColorConverter.ConvertFromString("#31ade9"))
            };
            Label label = new Label()
            {
                Content = "Header View",
                HorizontalContentAlignment = HorizontalAlignment.Center,
                VerticalContentAlignment = VerticalAlignment.Center
            };
            grid.Children.Add(label);
            navigationDrawer.DrawerHeaderView = grid;
            this.Content = navigationDrawer;
        }
    }
}

```

Customize animations

The Transition property specifies the animations for the DrawerView panel. The [Transition](#) property has the following options:

- SlideOnTop
- Push

- Reveal

The default transition is [SlideOnTop](#) that draws the DrawerContent on top of the main content.

Note: The Transition property is applicable only when the DisplayMode is Default.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
Transition="SlideOnTop"
DrawerWidth="300"
IsOpen="True">
<syncfusion:SfNavigationDrawer.ContentView>
<Grid x:Name="mainContentView"
Background="White">
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
<RowDefinition/>
</Grid.RowDefinitions>
<StackPanel Background="#1a1a1d6"
Orientation="Horizontal">
<Button x:Name="hamburgerButton"
BorderBrush="Transparent"
Height="50" Width="50"
HorizontalAlignment="Left"
Background="#1a1a1d6" Foreground="White"
Click="HamburgerButton_Click">
<Image Source="hamburger_icon.png"
Height="20"
Width="20"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Button>
<Label x:Name="headerLabel"
Height="50"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"
Content="Home"
Foreground="White"
Background="#1a1a1d6"/>
</StackPanel>
<Label Grid.Row="1"
x:Name="contentLabel"
VerticalContentAlignment="Center"
HorizontalContentAlignment="Center"
```

```

Content="Content View"
Foreground="Black"/>
</Grid>
</syncfusion:SfNavigationDrawer.ContentView>
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
<syncfusion:SfNavigationDrawer.DrawerContentView>
<Grid>
<ListBox x:Name="list"
ItemsSource="{Binding Contents}">
<ListBox.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Black"/>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerContentView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

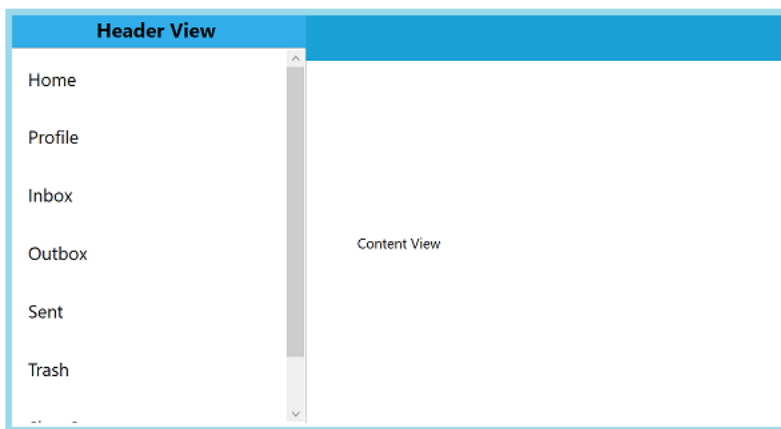
using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            navigationDrawer.Transition = Transition.SlideOnTop;
        }
        private void HamburgerButton_Click(object sender, RoutedEventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
    public class ViewModel
    {
        public ViewModel()
        {
            Contents = new List<Model>();
        }
    }
}

```

```

Contents.Add(new Model() { Name = "Home" });
Contents.Add(new Model() { Name = "Profile" });
Contents.Add(new Model() { Name = "Inbox" });
Contents.Add(new Model() { Name = "Outbox" });
Contents.Add(new Model() { Name = "Sent" });
Contents.Add(new Model() { Name = "Trash" });
Contents.Add(new Model() { Name = "Sign Out" });
}
public List<Model> Contents { get; set; }
}
public class Model
{
public string Name { get; set; }
}
}

```



The following code example shows how to set [Transition](#) as [Push](#) to SfNavigationDrawer. This transition moves the Drawer and main content simultaneously.

XML

```

<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="Main Window" Height="450" Width="800">
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
Transition="Push"
DrawerWidth="300"
IsOpen="True">
<syncfusion:SfNavigationDrawer.ContentView>
<Grid x:Name="mainContentView"
Background="White">
<Grid.RowDefinitions>

```

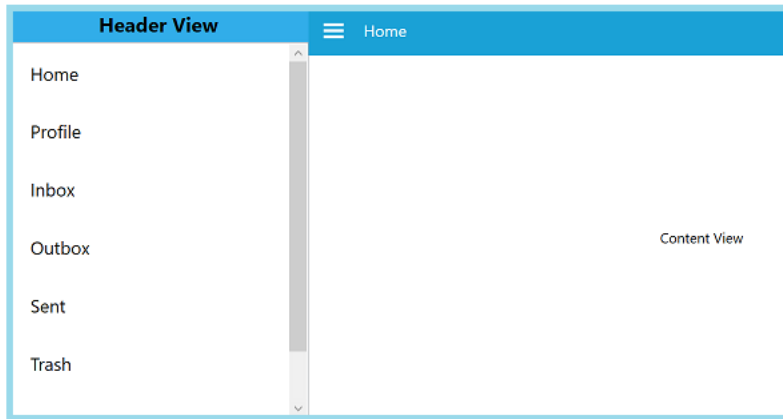
```

<RowDefinition Height="auto"/>
</RowDefinition>
</Grid.RowDefinitions>
<StackPanel Background="#1aa1d6"
Orientation="Horizontal">
<Button x:Name="hamburgerButton"
BorderBrush="Transparent"
Height="50" Width="50"
HorizontalAlignment="Left"
Background="#1aa1d6" Foreground="White"
Click="HamburgerButton_Click">
<Image Source="hamburger_icon.png"
Height="20"
Width="20"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Button>
<Label x:Name="headerLabel"
Height="50"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"
Content="Home"
Foreground="White"
Background="#1aa1d6"/>
</StackPanel>
<Label Grid.Row="1"
x:Name="contentLabel"
VerticalContentAlignment="Center"
HorizontalContentAlignment="Center"
Content="Content View"
Foreground="Black"/>
</Grid>
</syncfusion:SfNavigationDrawer.ContentView>
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
<syncfusion:SfNavigationDrawer.DrawerContentView>
<Grid>
<ListBox x:Name="list"
ItemsSource="{Binding Contents}">
<ListBox.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Black"/>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerContentView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```
using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            navigationDrawer.Transition = Transition.Push;
        }
        private void HamburgerButton_Click(object sender, RoutedEventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
    public class ViewModel
    {
        public ViewModel()
        {
            Contents = new List<Model>();
            Contents.Add(new Model() { Name = "Home" });
            Contents.Add(new Model() { Name = "Profile" });
            Contents.Add(new Model() { Name = "Inbox" });
            Contents.Add(new Model() { Name = "Outbox" });
            Contents.Add(new Model() { Name = "Sent" });
            Contents.Add(new Model() { Name = "Trash" });
            Contents.Add(new Model() { Name = "Sign Out" });
        }
        public List<Model> Contents { get; set; }
    }
    public class Model
    {
        public string Name { get; set; }
    }
}
```



The following code example shows how to set [Transition](#) as [Reveal](#) to SfNavigationDrawer. In this transition, the Drawer content is stable and the main content is moved to reveal the drawer content.

XML

```
<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
Transition="Reveal"
DrawerWidth="300"
IsOpen="True">
<syncfusion:SfNavigationDrawer.ContentView>
<Grid x:Name="mainContentView"
Background="White">
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
</Grid.RowDefinitions>
<StackPanel Background="#1aa1d6"
Orientation="Horizontal">
<Button x:Name="hamburgerButton"
BorderBrush="Transparent"
Height="50" Width="50"
HorizontalAlignment="Left"
Background="#1aa1d6" Foreground="White"
Click="HamburgerButton_Click">
<Image Source="hamburger_icon.png"
Height="20"
Width="20"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Image>
</StackPanel>
</Grid>
</syncfusion:SfNavigationDrawer.ContentView>
</syncfusion:SfNavigationDrawer>
</Window>
```

```

</Button>
<Label x:Name="headerLabel"
Height="50"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"
Content="Home"
Foreground="White"
Background="#1aa1d6"/>
</StackPanel>
<Label Grid.Row="1"
x:Name="contentLabel"
VerticalContentAlignment="Center"
HorizontalContentAlignment="Center"
Content="Content View"
Foreground="Black"/>
</Grid>
</syncfusion:SfNavigationDrawer.ContentView>
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
<syncfusion:SfNavigationDrawer.DrawerContentView>
<Grid>
<ListBox x:Name="list"
ItemsSource="{Binding Contents}">
<ListBox.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Black"/>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerContentView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

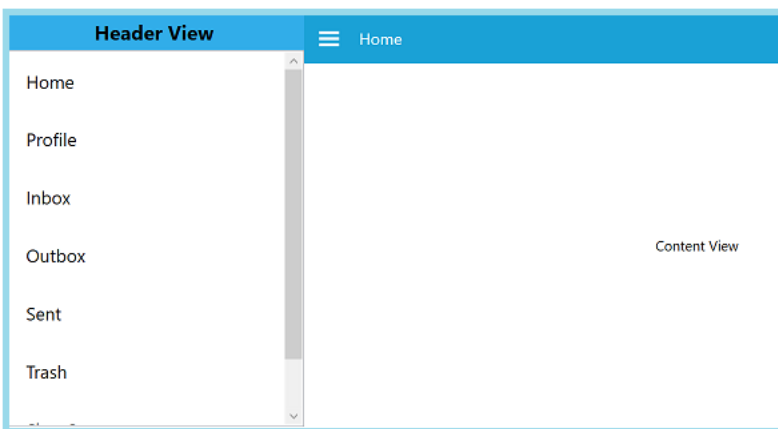
using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {

```

```

InitializeComponent();
navigationDrawer.Transition = Transition.Reveal;
}
private void HamburgerButton_Click(object sender, RoutedEventArgs e)
{
    navigationDrawer.ToggleDrawer();
}
}
public class ViewModel
{
    public ViewModel()
    {
        Contents = new List<Model>();
        Contents.Add(new Model() { Name = "Home" });
        Contents.Add(new Model() { Name = "Profile" });
        Contents.Add(new Model() { Name = "Inbox" });
        Contents.Add(new Model() { Name = "Outbox" });
        Contents.Add(new Model() { Name = "Sent" });
        Contents.Add(new Model() { Name = "Trash" });
        Contents.Add(new Model() { Name = "Sign Out" });
    }
    public List<Model> Contents { get; set; }
}
public class Model
{
    public string Name { get; set; }
}
}

```



Animation duration

The [AnimationDuration](#) property sets the time span value, by which the DrawerContent can be brought to view.

Note: The AnimationDuration property is applicable only when the DisplayMode is Default.

XML

```

<Window x:Class="NavigationDrawerWPF.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="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:NavigationDrawerWPF"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:SfNavigationDrawer x:Name="navigationDrawer"
AnimationDuration="100">
<syncfusion:SfNavigationDrawer.DrawerHeaderView>
<Grid Background="#31ade9">
<Label Content="Header View"
HorizontalContentAlignment="Center"
VerticalContentAlignment="Center"/>
</Grid>
</syncfusion:SfNavigationDrawer.DrawerHeaderView>
</syncfusion:SfNavigationDrawer>
</Window>

```

C#

```

using Syncfusion.UI.Xaml.NavigationDrawer;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
namespace NavigationDrawerWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                AnimationDuration = new System.TimeSpan(100)
            };
            Grid grid = new Grid()
            {
                Background = new
                SolidColorBrush((Color)ColorConverter.ConvertFromString("#31ade9"))
            };
            Label label = new Label()
            {
                Content = "Header View",
                HorizontalContentAlignment = HorizontalAlignment.Center,
                VerticalContentAlignment = VerticalAlignment.Center
            };
            grid.Children.Add(label);
            navigationDrawer.DrawerHeaderView = grid;
            this.Content = navigationDrawer;
        }
    }
}

```

Note: View [sample](#) in GitHub

NotifyIcon

WPF Notify Icon Overview

Notify Icon control is implemented with different animations and shape support. It allows you to add a icon in the notification tray of the system.

Features

- Different animation support.
- Different shape support.

Getting Started with WPF Notify Icon

This section provides you an overview of working with [NotifyIcon](#) for WPF and provides a walkthrough to configure the NotifyIcon control in a real-time scenario.

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.

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 to the NotifyIcon's dependent assemblies.

Creating an application with NotifyIcon control

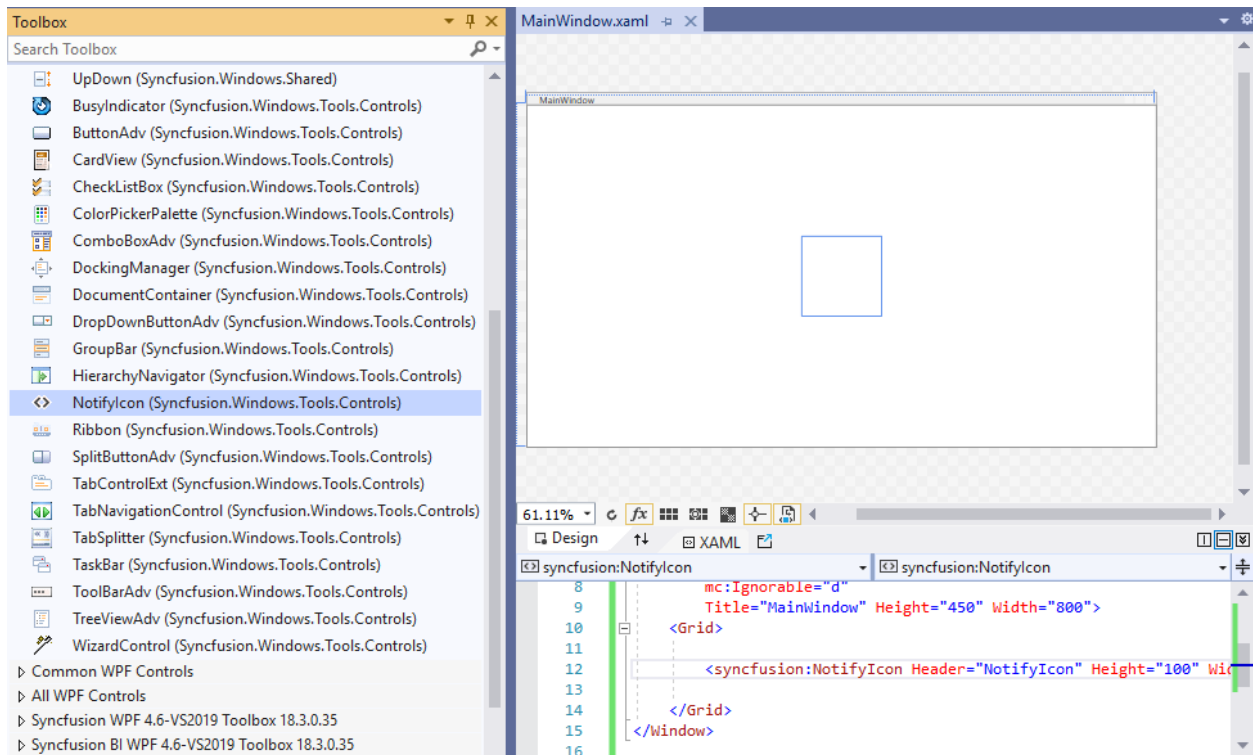
In this walkthrough, you will create a WPF application that contains the NotifyIcon control.

Creating a project

Create a new WPF project to show the [NotifyIcon](#) control in Visual Studio.

Adding control via designer

The WPF **NotifyIcon** 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 the control manually in XAML

To add the control manually to the 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 the XAML page or Syncfusion.Windows.Tools.Controls namespace.
3. Declare the **NotifyIcon** 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:NotifyIcon_GettingStarted"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="NotifyIcon_GettingStarted.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<syncfusion:NotifyIcon Height="80" Width="150" />
</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* and *Syncfusion.Tools.WPF*.
2. Import the `NotifyIcon` namespace **Syncfusion.Windows.Tools.Controls**.
3. Create the `NotifyIcon` control instance and add it to the page.

C#

```
using Syncfusion.Windows.Tools.Controls;
using System.Windows;
namespace NotifyIcon_GettingStarted
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            NotifyIcon notifyIcon = new NotifyIcon();
            notifyIcon.Height = 80;
            notifyIcon.Width = 150;
            this.Content = notifyIcon;
        }
    }
}
```

Show the notify icon

You can set the icon to the `NotifyIcon` control using the `Icon` property and then display the notify icon by setting the `ShowInTaskBar` property to `true`.

XML

```
<syncfusion:NotifyIcon Name="notify" Header="NotifyIcon" Height="80"
ShowInTaskBar="True" Width="150" Icon="images\notifyicon.png" />
```



Text

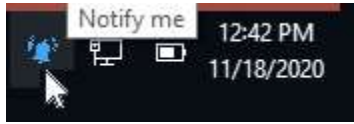
You can use the `Text` property to add text to the `NotifyIcon` control. It will be shown on the notify icon while hovering.

XML

```
<syncfusion:NotifyIcon Name="notify" Header="NotifyIcon" Height="80"
ShowInTaskBar="True" Width="150" Icon="images\notifyicon.png" Text="Notify
me" />
```

C#

```
notify.Text = "Notify me";
```



Tooltip

You can show the tooltip for `Notifylcon` control by using the [ShowBalloonTip](#) method. You can also add a title and text by using the [BalloonTipTitle](#) and [BalloonTipText](#) properties. You can enable the title by setting the [BalloonTipHeaderVisibility](#) property to `true`.

XML

```
<syncfusion:Notifylcon Name="notify" Header="NotifyIcon" Height="80"
ShowInTaskBar="True" Width="150"
Icon="images\notifyicon.png" Text="Notify me"
BalloonTipTitle="Default NotifyIcon" BalloonTipText="Custom NotifyIcon"
BalloonTipHeaderVisibility="Visible" />
```

C#

```
//Show the balloontip.
notify.ShowBalloonTip(3000);
```

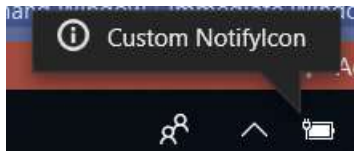


Note: [View sample in GitHub.](#)

Theme

Notifylcon supports various built-in themes. Refer to the below links to apply themes for the Notifylcon,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Interactive Features in WPF Notify Icon

This section illustrates the following interactive features of Notifylcon control.

BalloonTip

This section illustrates the following parts of a BalloonTip.

Header of BalloonTip

Using the [BalloonTipHeaderVisibility](#) property, user can collapse or show the header of the Notifylcon control.

The following code snippet is used to set this property.

XML

```
<!--BalloonTipHeaderVisibility="Visible" -->
<syncfusion:NotifyIcon Name="notifyIcon" BalloonTipText="Custom Notify
Icon is Available" BalloonTipTitle="Default NotifyIcon"
BalloonTipHeaderVisibility="Visible"
ShowBalloonTipTime="1000" HideBalloonTipTime="1000"/>
<!--BalloonTipHeaderVisibility="Collapsed" -->
<syncfusion:NotifyIcon Name="notifyIcon" BalloonTipText="Custom Notify
Icon is Available" BalloonTipTitle="Default NotifyIcon"
BalloonTipHeaderVisibility="Collapsed"
ShowBalloonTipTime="1000" HideBalloonTipTime="1000"/>
```

C#

```
//BalloonTipHeaderVisibility="Visible"
notifyIcon.BalloonTipHeaderVisibility = Visibility.Visible;
//BalloonTipHeaderVisibility="Collapsed"
notifyIcon.BalloonTipHeaderVisibility = Visibility.Collapsed;
```



Text and Title of BalloonTip

BalloonTipText property is used to set the text that should be displayed in the Notify icon.

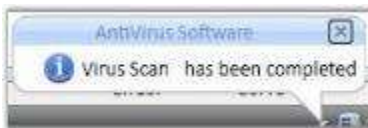
BalloonTipTitle is used to set the title for NotifyIcon. The following code snippet is used to set these properties.

XML

```
<syncfusion:NotifyIcon Name="notifyIcon" BalloonTipText="Virus Scan has been
completed" BalloonTipTitle="AntiVirus Software" ShowBalloonTipTime="1000"
HideBalloonTipTime="1000"/>
```

C#

```
notifyIcon.BalloonTipText = "Virus Scan has been completed";
notifyIcon.BalloonTipTitle = "AntiVirus Software";
```



BalloonTipTitle = "AntiVirus Software"

Icon of BalloonTip

You can display the following five different icons in the NotifylIcon control.

- Custom
- Info
- Error
- None
- Warning

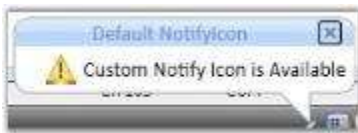
The following code snippet is used to set this property.

XML

```
<!-- BalloonTipIcon="Warning" -->
<syncfusion:NotifylIcon Name="notifyIcon" BalloonTipText="Custom Notify
Icon is Available" BalloonTipTitle="Default NotifyIcon"
BalloonTipIcon="Warning"
ShowBalloonTipTime="1000" HideBalloonTipTime="1000"/>
```

C#

```
//BalloonTipIcon="Warning"
notifyIcon.BalloonTipIcon = BalloonTipIcon.Warning;
```



Animation

This topic illustrates the animation type and animation time for NotifylIcon control.

Animation Type

You can set different animation effects for the NotifylIcon by using the BalloonTipAnimationEffect property. They are listed below.

- Custom
- Fade
- Slide
- Scale

Animation Time

NotifylIcon display time is set by using the property called ShowBalloonTipTime. The time during which the NotifylIcon is hidden is set by using the property called HideBalloonTipTime.

The following code snippet is used to set these properties.

XML

```
<!-- For Fade -->
<syncfusion:NotifylIcon Name="notifyIcon" BalloonTipText="Custom Notify
Icon is Available" BalloonTipTitle="Default NotifyIcon"
BalloonTipAnimationEffect="Fade"
```

```

ShowBalloonTipTime="1000" HideBalloonTipTime="1000"/>
<!-- For Scale -->
<syncfusion:NotifyIcon Name="notifyIcon" BalloonTipText="Custom Notify
Icon is Available" BalloonTipTitle="Default NotifyIcon"
BalloonTipAnimationEffect="Scale"
ShowBalloonTipTime="1000" HideBalloonTipTime="1000"/>

```

C#

```

NotifyIcon notifyIcon = new NotifyIcon();
notifyIcon.BalloonTipText = "Custom Notify Icon is Available";
notifyIcon.BalloonTipTitle = "Default NotifyIcon";
//For Fade
notifyIcon.BalloonTipAnimationEffect = BalloonTipAnimationEffects.Fade;
//For Scale
notifyIcon.BalloonTipAnimationEffect = BalloonTipAnimationEffects.Scale;
// Duration for Showing NotifyIcon
notifyIcon.ShowBalloonTipTime = 1000;
// Duration for Hiding NotifyIcon
notifyIcon.HideBalloonTipTime = 1000;

```

Setting the position of the Notify Icon

The position where the NotifyIcon is to be displayed is specified using the property called BalloonTipLocation. To set this property, refer the below code.

XML

```

<syncfusion:NotifyIcon Name="notifyIcon" BalloonTipLocation="500,500"
BalloonTipText="Custom Notify
Icon is Available" BalloonTipTitle="Default NotifyIcon"
ShowBalloonTipTime="1000" HideBalloonTipTime="1000"/>

```

C#

```

defaults.BalloonTipLocation = new Point(500, 500);

```

Setting the Shape of the Notify Icon

NotifyIcon can take the following three different shapes.

- Balloon
- Rectangle
- RoundedRectangle

The shape for the NotifyIcon is set by using the BalloonTipShape property. The following code snippet is used to change the Shape of the NotifyIcon control.

XML

```

<!-- Rectangle Shape -->
<syncfusion:NotifyIcon Name="notifyIcon" BalloonTipText="Custom Notify
Icon is Available" BalloonTipTitle="Default NotifyIcon"
BalloonTipShape="Rectangle"
ShowBalloonTipTime="1000" HideBalloonTipTime="1000"/>

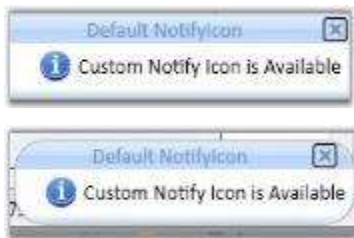
```



```
<!-- RoundedRectangle Shape -->
<syncfusion:NotifylIcon Name="notifyIcon" BalloonTipText="Custom Notify
Icon is Available" BalloonTipTitle="Default NotifyIcon"
BalloonTipShape="RoundedRectangle"
ShowBalloonTipTime="1000" HideBalloonTipTime="1000"/>
<!-- Balloon Shape -->
<syncfusion:NotifylIcon Name="notifyIcon" BalloonTipText="Custom Notify
Icon is Available" BalloonTipTitle="Default NotifyIcon"
BalloonTipShape="Balloon"
ShowBalloonTipTime="1000" HideBalloonTipTime="1000"/>
```

C#

```
// Rectangle Shape
notifyIcon.BalloonTipShape = BalloonTipShapes.Rectangle;
//RoundedRectangle Shape
notifyIcon.BalloonTipShape = BalloonTipShapes.RoundedRectangle;
//Balloon Shape
notifyIcon.BalloonTipShape = BalloonTipShapes.RoundedRectangle;
```



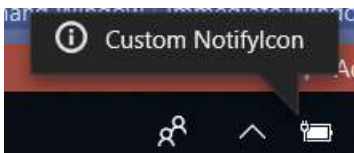
Layout Related Features in WPF Notify Icon

This section illustrates the following Layout-related feature of NotifylIcon control.

Theme

NotifylIcon supports various built-in themes. Refer to the below links to apply themes for the NotifylIcon,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Customizing the Header of the NotifylIcon

You can customize the background and foreground for the BalloonTipHeader by using the [HeaderBackground](#) and [HeaderForeground](#) properties.

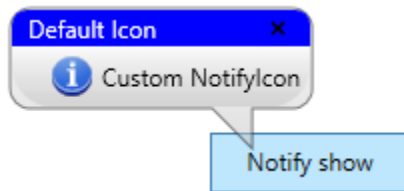
XML

```
<Button x:Name="button" Height="30" Width="100" Click="Button_Click"
Content="Notify show">
</Button>
<syncfusion:NotifylIcon Name="notifyIcon" Header="NotifyIcon"
BalloonTipTitle="Default NotifyIcon"
```

```
BalloonTipText="Custom NotifyIcon" BalloonTipIcon="Info"
ShowBalloonTipTime="1000" HideBalloonTipTime="1000"
HeaderBackground="Blue" HeaderForeground="White">
</syncfusion:NotifyIcon>
```

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    notifyIcon.HeaderBackground = Brushes.Blue;
    notifyIcon.HeaderForeground = Brushes.White;
}
```



Events

The [NotifyIcon](#) includes several pre-defined events to perform any required action as follows:

- BalloonTipOpening
- BalloonTipOpened
- BalloonTipHiding
- BalloonTipHidden
- CloseButtonClick
- Click

BalloonTipOpening event

The [BalloonTipOpening](#) event occurs before opening the balloon tip and action can be handled in the respective event handler.

The [CancelEventArgs](#) object contains the following property.

- Cancel : Canceling the action of the balloon tip show.

XML

```
<syncfusion:NotifyIcon Name="notifyIcon" Header="NotifyIcon"
BalloonTipOpening="NotifyIcon_BalloonTipOpening">
</syncfusion:NotifyIcon>
```

C#

```
notifyIcon.BalloonTipOpening += NotifyIcon_BalloonTipOpening;
private void NotifyIcon_BalloonTipOpening(object sender,
System.ComponentModel.CancelEventArgs e)
{
    //Cancel the balloontip action.
    e.Cancel = true;
}
```

```
}

```

BalloonTipOpened event

The [BalloonTipOpened](#) event occurs after the balloon tip is opened and action can be handled in the respective event handler.

The [DependencyPropertyChangedEventArgs](#) object contains the following properties:

- NewValue : gets an new value.
- OldValue : gets an old value.
- Property : Identify the value.

XML

```
<syncfusion:NotifyIcon Name="notifyIcon" Header="NotifyIcon"
BalloonTipOpened="NotifyIcon_BalloonTipOpened">
</syncfusion:NotifyIcon>

```

C#

```
notifyIcon.BalloonTipOpened += NotifyIcon_BalloonTipOpened;
private void NotifyIcon_BalloonTipOpened(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    if(e.Property.Name == "IsOpen")
    {
        //Change the balloontip text.
        notifyIcon.BalloonTipText = "Welcome";
    }
}

```

BalloonTipHiding event

The [BalloonTipHiding](#) event occurs before hiding the balloon tip and action can be handled in the respective event handler.

The [CancelEventArgs](#) object contains the following property:

- Cancel : Canceling the hiding action of the balloon tip.

XML

```
<syncfusion:NotifyIcon Name="notifyIcon" Header="NotifyIcon"
BalloonTipHiding="NotifyIcon_BalloonTipHiding">
</syncfusion:NotifyIcon>

```

C#

```
notifyIcon.BalloonTipHiding += NotifyIcon_BalloonTipHiding;
private void NotifyIcon_BalloonTipHiding(object sender,
System.ComponentModel.CancelEventArgs e)
{
    //Cancel the hiding action.
}

```

```
e.Cancel = true;
}
```

BalloonTipHidden event

The [BalloonTipHidden](#) event occurs after hiding the balloon tip and action can be handled in the respective event handler.

The [DependencyPropertyChangedEventArgs](#) object contains the following properties:

- **newValue** : gets an new value.
- **OldValue** : gets an old value.
- **Property** : Identify the value.

XML

```
<syncfusion:NotifyIcon Name="notifyIcon" Header="NotifyIcon"
BalloonTipHidden="NotifyIcon_BalloonTipHidden">
</syncfusion:NotifyIcon>
```

C#

```
notifyIcon.BalloonTipHidden += NotifyIcon_BalloonTipHidden;
private void NotifyIcon_BalloonTipHidden(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
}
```

CloseButtonClick event

The [CloseButtonClick](#) event occurs while clicking the close button in the balloon tip.

XML

```
<syncfusion:NotifyIcon Name="notifyIcon" Header="NotifyIcon"
CloseButtonClick="NotifyIcon_CloseButtonClick">
</syncfusion:NotifyIcon>
```

C#

```
notifyIcon.CloseButtonClick += NotifyIcon_CloseButtonClick;
private void NotifyIcon_CloseButtonClick(object sender, EventArgs e)
{
}
```

Click event

The [Click](#) event occurs while clicking the icon inside the balloon tip.

XML

```
<syncfusion:NotifyIcon Name="notifyIcon" Header="NotifyIcon"
Click="NotifyIcon_Click">
</syncfusion:NotifyIcon>
```

C#

```
notifyIcon.Click += NotifyIcon_Click;  
private void NotifyIcon_Click(object sender, EventArgs e)  
{  
}
```

OlapChart

WPF Olap Chart Overview

An OLAP chart control is a lightweight control that allows you to efficiently visualize multi-dimensional data from the OLAP data source with the help of a report bound to it.

Key features

- **Multi-level drill up or down:** Enables you to visualize a multi-level data in the chart. You can drill up or drill down the hierarchies natively.
- **KPI:** Allows you to visualize the KPI status, trend, goal, and value in different chart types.
- **Appearance:** Enables you to customize the area, axes, series, legends, and chart types easily.
- **Zooming and Scrolling:** Allows you to zoom and scroll across the chart.
- **Tooltip:** Allows you to visualize the data point-related values in a form of tooltips.
- **Exporting:** Allows you to export the OLAP chart into various formats such as PDF, Word, and image formats.
- **Printing:** Allows you to customize and print in color mode or black and white.

Getting Started with WPF Olap Chart

Important

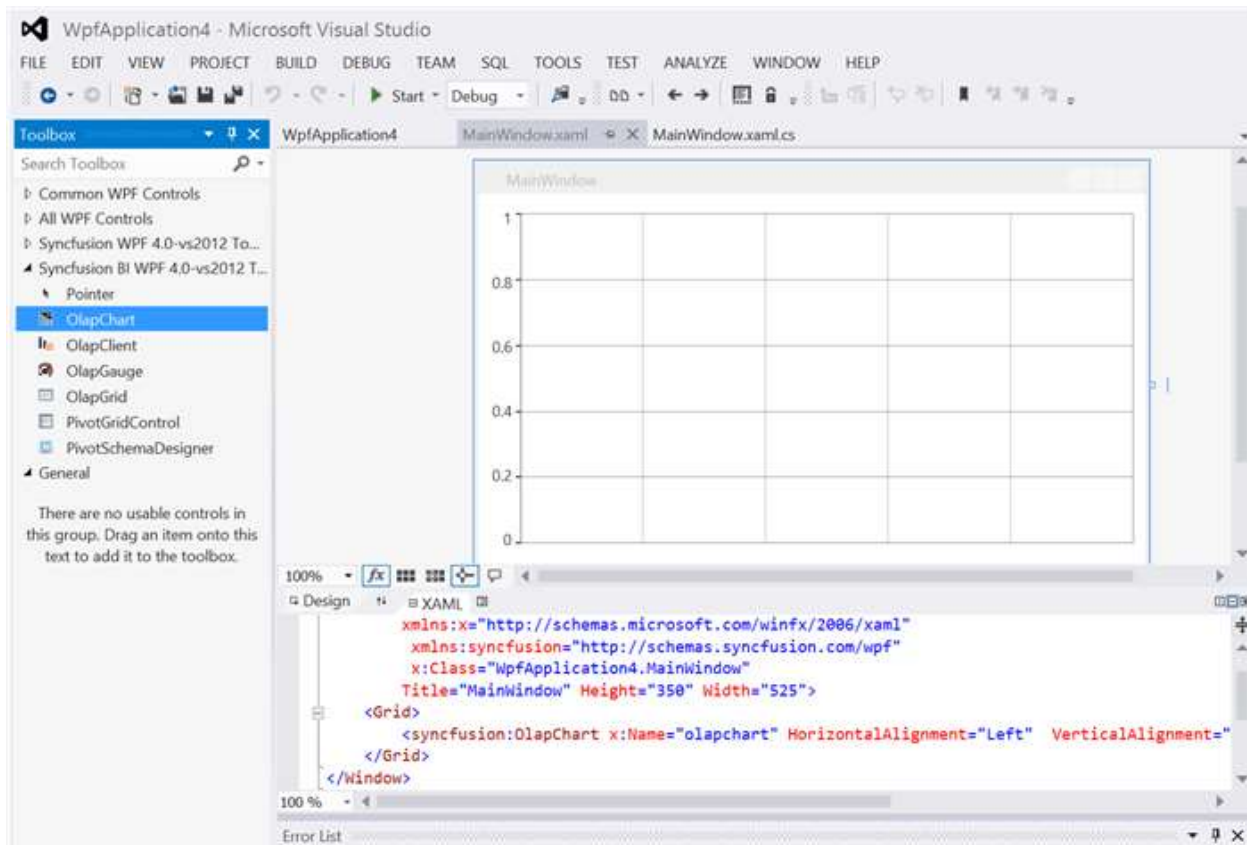
Starting with v16.2.0.x, if you refer to Syncfusion assemblies from trial setup or from the NuGet feed, include a license key in your projects. Refer to this [link](#) to learn about registering Syncfusion license key in your WPF application to use the components.

This section covers the information required to create a simple OLAP chart bound to the OLAP data source.

Through Visual Studio

Open the Visual Studio IDE and navigate to File > New > Project > WPF Application (inside Visual C# Templates) to create a new WPF application.

After the WPF application is created, go to View menu and select Toolbox option. Now, the toolbox will appear inside the Visual Studio IDE. From the Visual Studio Toolbox, drag the OLAP chart under the **Syncfusion BI WPF** tag. It will automatically add the required assemblies to the application.



Add a **Name** to the OLAP chart component for accessing it through code-behind as shown in the following code sample.

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="SampleApplication.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid>
<syncfusion:OlapChart x:Name="olapChart" HorizontalAlignment="Left"
VerticalAlignment="Top" Height="319" Width="517"/>
</Grid>
</Window>
```

Include the following namespaces in the code-behind for using OlapReport and OlapDataManager in the application.

- Syncfusion.Olap.Reports
- Syncfusion.Olap.Manager

C#

```
using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
```

```

namespace SampleApplication
{
    public partial class MainWindow : SampleWindow
    {
        private string _connectionString;
        private OlapDataManager _olapDataManager;
        public MainWindow()
        {
            InitializeComponent();
            _connectionString = " Enter a valid connection string ";
            //Connection string is passed to OlapDataManager as an argument
            _olapDataManager = new OlapDataManager(_connectionString);
            //A default OlapReport is set to OlapDataManager
            _olapDataManager.SetCurrentReport(CreateOlapReport());
            //Finally OlapChart gets the information from the OlapDataManager
            this.olapChart.OlapDataManager = _olapDataManager;
            this.olapChart.DataBind();
        }
        /// <summary>
        /// Defining OlapReport with Dimension and Measure
        /// </summary>
        private OlapReport CreateOlapReport()
        {
            OlapReport olapReport = new OlapReport();
            // Setting the Cube name
            olapReport.CurrentCubeName = "Adventure Works";
            DimensionElement dimensionElementColumn = new DimensionElement();
            // Specifying the name of the Dimension
            dimensionElementColumn.Name = "Customer";
            // Specifying the Hierarchy and Level name
            dimensionElementColumn.AddLevel("Customer Geography", "Country");
            MeasureElements measureElementColumn = new MeasureElements();
            //Specifying the Measure name
            measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet Sales Amount" });
            DimensionElement dimensionElementRow = new DimensionElement();
            // Specifying the name of the Dimension
            dimensionElementRow.Name = "Date";
            // Specifying the Hierarchy and Level name
            dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
            ///Adding Dimension in column axis
            olapReport.CategoricalElements.Add(dimensionElementColumn);
            ///Adding Measure in column axis
            olapReport.CategoricalElements.Add(measureElementColumn);
            ///Adding Dimension in row axis
            olapReport.SeriesElements.Add(dimensionElementRow);
            return olapReport;
        }
    }
}

```

VB.NET

```

Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Namespace SampleApplication

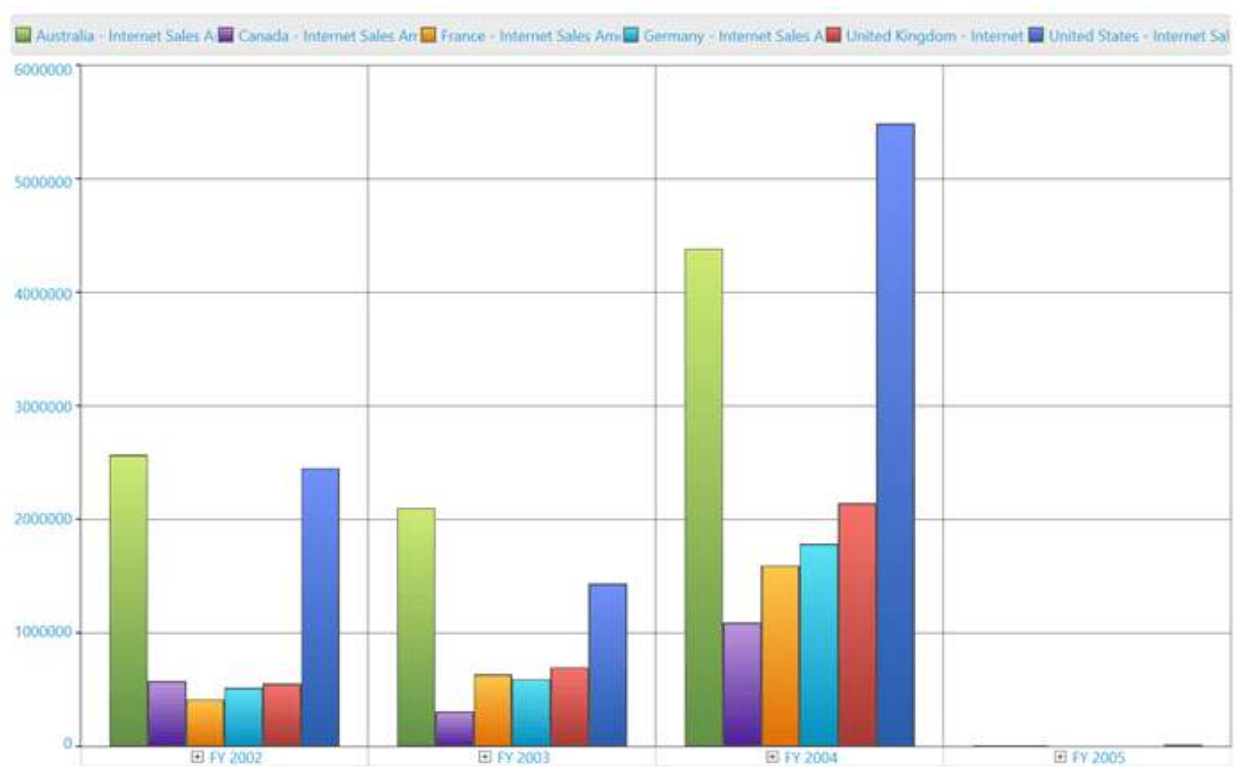
```

```

Partial Public Class MainWindow Inherits SampleWindow
Private _connectionString As String
Private _olapDataManager As OlapDataManager
Public Sub New()
InitializeComponent()
_connectionString = "Enter a valid connection string"
' Connection string is passed to OlapDataManager as an argument
_olapDataManager = New OlapDataManager(_connectionString)
' A default OlapReport is set to OlapDataManager
_olapDataManager.SetCurrentReport(CreateOlapReport())
' Finally OlapChart gets the information from the OlapDataManager
Me.olapChart.OlapDataManager = _olapDataManager
Me.olapChart.DataBind()
End Sub
''' <summary>
''' Defining OlapReport with Dimension and Measure
''' </summary>
Private Function CreateOlapReport() As OlapReport
Dim olapReport As OlapReport = New OlapReport()
' Setting the Cube name
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementColumn.Name = "Customer"
' Specifying the Hierarchy and Level name
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
' Specifying the Measure name
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementRow.Name = "Date"
' Specifying the Hierarchy and Level name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
''' Adding Dimension in column axis
olapReport.CategoricalElements.Add(dimensionElementColumn)
''' Adding Measure in column axis
olapReport.CategoricalElements.Add(measureElementColumn)
''' Adding Dimension in row axis
olapReport.SeriesElements.Add(dimensionElementRow)
Return olapReport
End Function
End Class
End Namespace

```

Run the application. The following output will be generated.



Through expression blend

Open Blend for Visual Studio and navigate to File > New project > WPF > WPF Application to create a new WPF application.

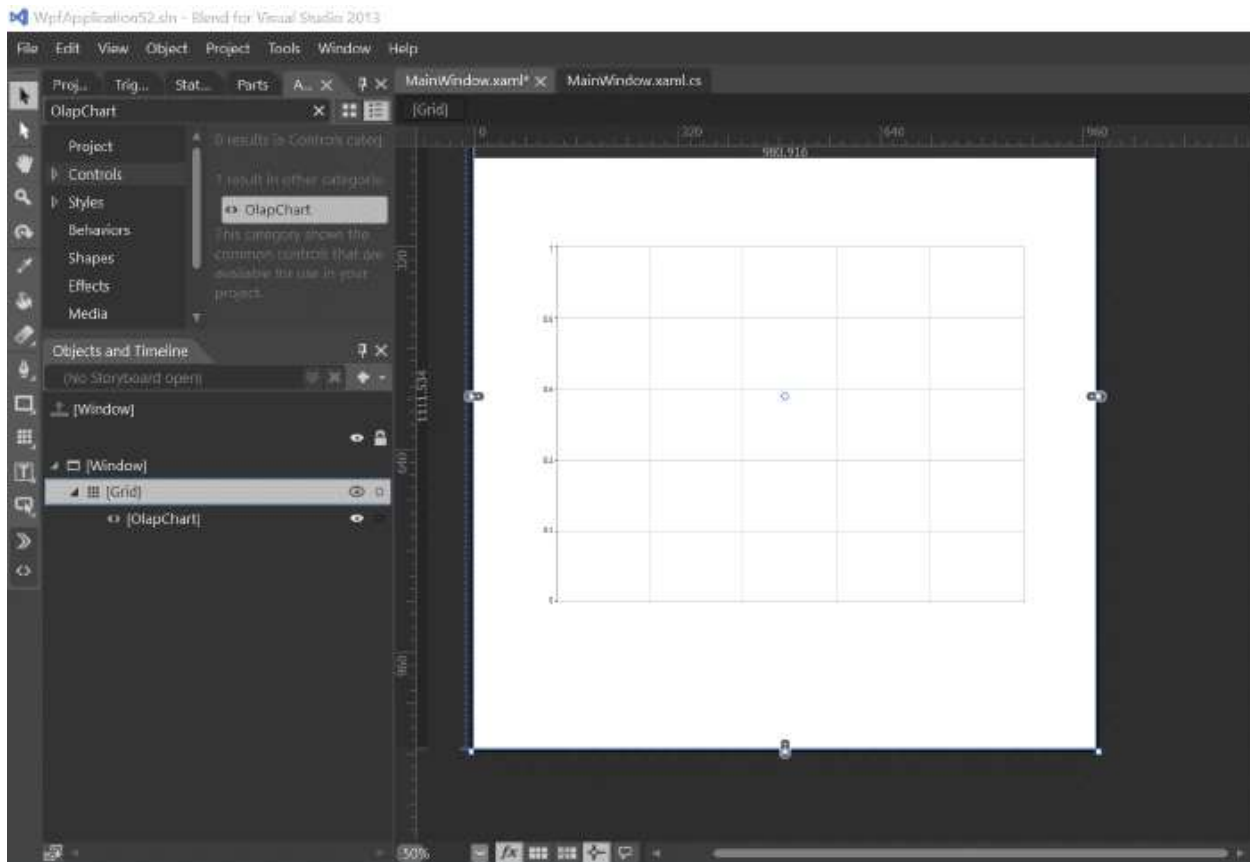
Select the **Project** tab available in the left-corner of the Blend IDE. Right-click the **References** and select **Add Reference**. Now, browse and add the following assemblies to the project.

- Syncfusion.Chart.WPF
- Syncfusion.Olap.Base
- Syncfusion.OlapChart.WPF
- Syncfusion.OlapChartConverter.WPF
- Syncfusion.OlapShared.WPF

Note: You can also get the assemblies by browsing to the default assembly location:

{System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<framework version>\

On adding the above assemblies, the OLAP chart control will be added under the **Assets** tab automatically. Now, choose the **Assets** tab and drag the OLAP chart to the designer.



Add a **Name** to the OLAP chart component for accessing it through the code-behind as shown in the following code sample.

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="SampleApplication.MainWindow"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <syncfusion:OlapChart x:Name="olapChart" HorizontalAlignment="Left"
      VerticalAlignment="Top" Height="319" Width="517"/>
  </Grid>
</Window>
```

Include the following namespace in the code-behind for using OlapReport and OlapDataManager in the application.

- Syncfusion.Olap.Reports
- Syncfusion.Olap.Manager

C#

```
using Syncfusion.Olap.Manager;
```

```

using Syncfusion.Olap.Reports;
namespace SampleApplication
{
    public partial class MainWindow : SampleWindow
    {
        private string _connectionString;
        private OlapDataManager _olapDataManager;
        public MainWindow()
        {
            InitializeComponent();
            _connectionString = " Enter a valid connection string ";
            //Connection string is passed to OlapDataManager as an argument
            _olapDataManager = new OlapDataManager(_connectionString);
            //A default OlapReport is set to OlapDataManager
            _olapDataManager.SetCurrentReport(CreateOlapReport());
            // Finally OlapChart gets the information from the OlapDataManager
            this.olapChart.OlapDataManager = _olapDataManager;
            this.olapChart.DataBind();
        }
        /// <summary>
        /// Defining OlapReport with Dimension and Measure
        /// </summary>
        private OlapReport CreateOlapReport()
        {
            OlapReport olapReport = new OlapReport();
            // Setting the Cube name
            olapReport.CurrentCubeName = "Adventure Works";
            DimensionElement dimensionElementColumn = new DimensionElement();
            // Specifying the name of the Dimension
            dimensionElementColumn.Name = "Customer";
            // Specifying the Hierarchy and Level name
            dimensionElementColumn.AddLevel("Customer Geography", "Country");
            MeasureElements measureElementColumn = new MeasureElements();
            //Specifying the Measure name
            measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet Sales Amount" });
            DimensionElement dimensionElementRow = new DimensionElement();
            // Specifying the name of the Dimension
            dimensionElementRow.Name = "Date";
            // Specifying the Hierarchy and Level name
            dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
            ///Adding Dimension in column axis
            olapReport.CategoricalElements.Add(dimensionElementColumn);
            ///Adding Measure in column axis
            olapReport.CategoricalElements.Add(measureElementColumn);
            ///Adding Dimension in row axis
            olapReport.SeriesElements.Add(dimensionElementRow);
            return olapReport;
        }
    }
}

```

VB.NET

```

Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports

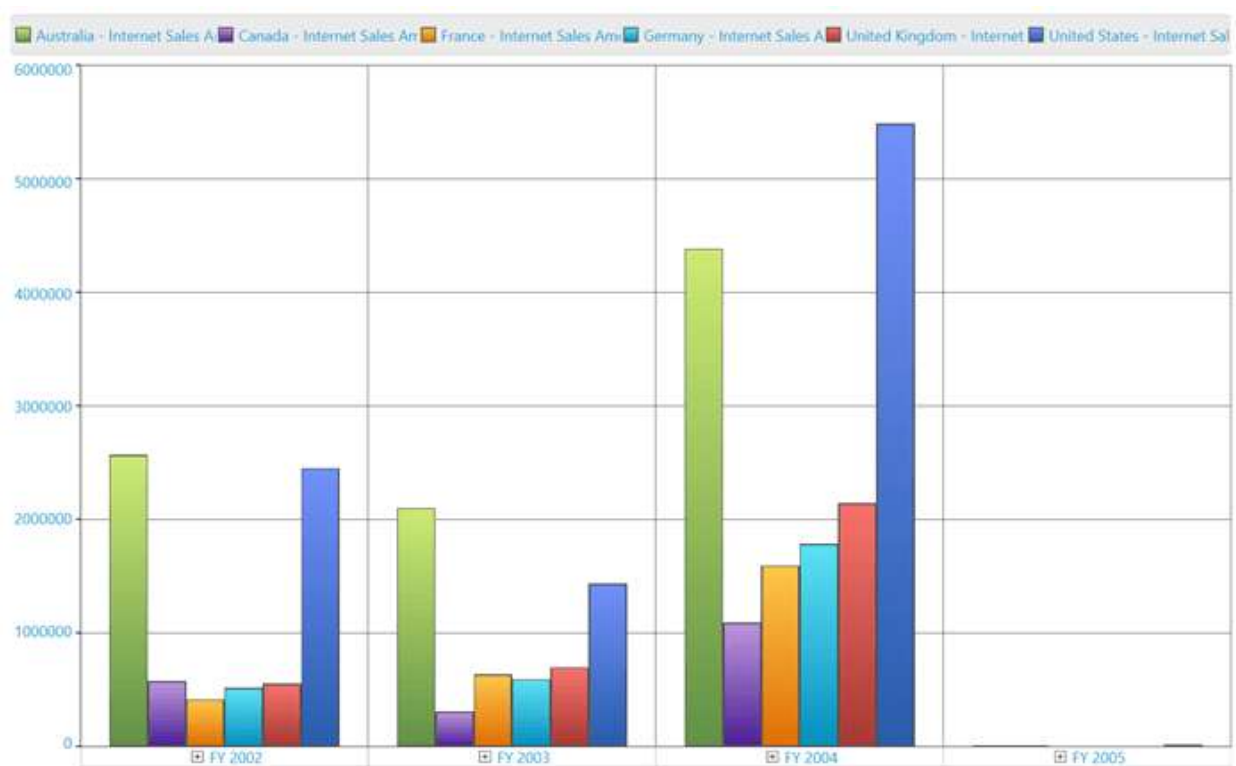
```

```

Namespace SampleApplication
Partial Public Class MainWindow Inherits SampleWindow
Private _connectionString As String
Private _olapDataManager As OlapDataManager
Public Sub New()
InitializeComponent()
_connectionString = " Enter a valid connection string "
' Connection string is passed to OlapDataManager as an argument
_olapDataManager = New OlapDataManager(_connectionString)
' A default OlapReport is set to OlapDataManager
_olapDataManager.SetCurrentReport(CreateOlapReport())
' Finally OlapChart gets the information from the OlapDataManager
Me.olapChart.OlapDataManager = _olapDataManager
Me.olapChart.DataBind()
End Sub
''' <summary>
''' Defining OlapReport with Dimension and Measure
''' </summary>
Private Function CreateOlapReport() As OlapReport
Dim olapReport As OlapReport = New OlapReport()
' Setting the Cube name
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementColumn.Name = "Customer"
' Specifying the Hierarchy and Level name
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
' Specifying the Measure name
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementRow.Name = "Date"
' Specifying the Hierarchy and Level name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
''' Adding Dimension in column axis
olapReport.CategoricalElements.Add(dimensionElementColumn)
''' Adding Measure in column axis
olapReport.CategoricalElements.Add(measureElementColumn)
''' Adding Dimension in row axis
olapReport.SeriesElements.Add(dimensionElementRow)
Return olapReport
End Function
End Class
End Namespace

```

Run the application. The following output will be generated.



Through code-behind

Open the Visual Studio IDE and navigate to File > New > Project > WPF Application (inside Visual C# Templates) to create a new WPF application.

To add the dependency assemblies within the application, right-click **References** in the solution explorer and select **Add Reference**. Then, add the following Syncfusion assemblies manually to the project from the installed location.

- Syncfusion.Chart.WPF
- Syncfusion.Core
- Syncfusion.Olap.Base
- Syncfusion.OlapChart.WPF
- Syncfusion.OlapChartConverter.WPF
- Syncfusion.OlapShared.WPF
- Syncfusion.Shared.WPF
- Syncfusion.Tools.WPF

Note: You can also get the assemblies by browsing to the default assembly location

{System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<framework version>\

Include the following namespaces in code-behind for using OlapChart, OlapReport, and OlapDataManager in the application.

- Syncfusion.Olap.Reports
- Syncfusion.Olap.Manager

- Syncfusion.Windows.Chart.Olap

C#

```

using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
using Syncfusion.Windows.Chart.Olap;
namespace SampleApplication
{
    public partial class MainWindow : SampleWindow
    {
        private string _connectionString;
        private OlapDataManager _olapDataManager;
        public MainWindow()
        {
            InitializeComponent();
            OlapChart olapChart = new OlapChart();
            _connectionString = " Enter a valid connection string ";
            //Connection string is passed to OlapDataManager as an argument
            _olapDataManager = new OlapDataManager(_connectionString);
            //A default OlapReport is set to OlapDataManager
            _olapDataManager.SetCurrentReport(CreateOlapReport());
            // Finally OlapChart gets the information from the OlapDataManager
            olapChart.OlapDataManager = _olapDataManager;
            olapChart.DataBind();
            // OlapChart added to the Main Window Grid region
            chart.Children.Add(olapChart);
        }
        /// <summary>
        /// Defining OlapReport with Dimension and Measure
        /// </summary>
        private OlapReport CreateOlapReport()
        {
            OlapReport olapReport = new OlapReport();
            // Setting the Cube name
            olapReport.CurrentCubeName = "Adventure Works";
            DimensionElement dimensionElementColumn = new DimensionElement();
            // Specifying the name of the Dimension
            dimensionElementColumn.Name = "Customer";
            // Specifying the Hierarchy and Level name
            dimensionElementColumn.AddLevel("Customer Geography", "Country");
            MeasureElements measureElementColumn = new MeasureElements();
            //Specifying the Measure name
            measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet Sales Amount" });
            DimensionElement dimensionElementRow = new DimensionElement();
            // Specifying the name of the Dimension
            dimensionElementRow.Name = "Date";
            // Specifying the Hierarchy and Level name
            dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
            ///Adding Dimension in column axis
            olapReport.CategoricalElements.Add(dimensionElementColumn);
            ///Adding Measure in column axis
            olapReport.CategoricalElements.Add(measureElementColumn);
            ///Adding Dimension in row axis
            olapReport.SeriesElements.Add(dimensionElementRow);
        }
    }
}

```

```

return olapReport;
}
}
}

```

VB.NET

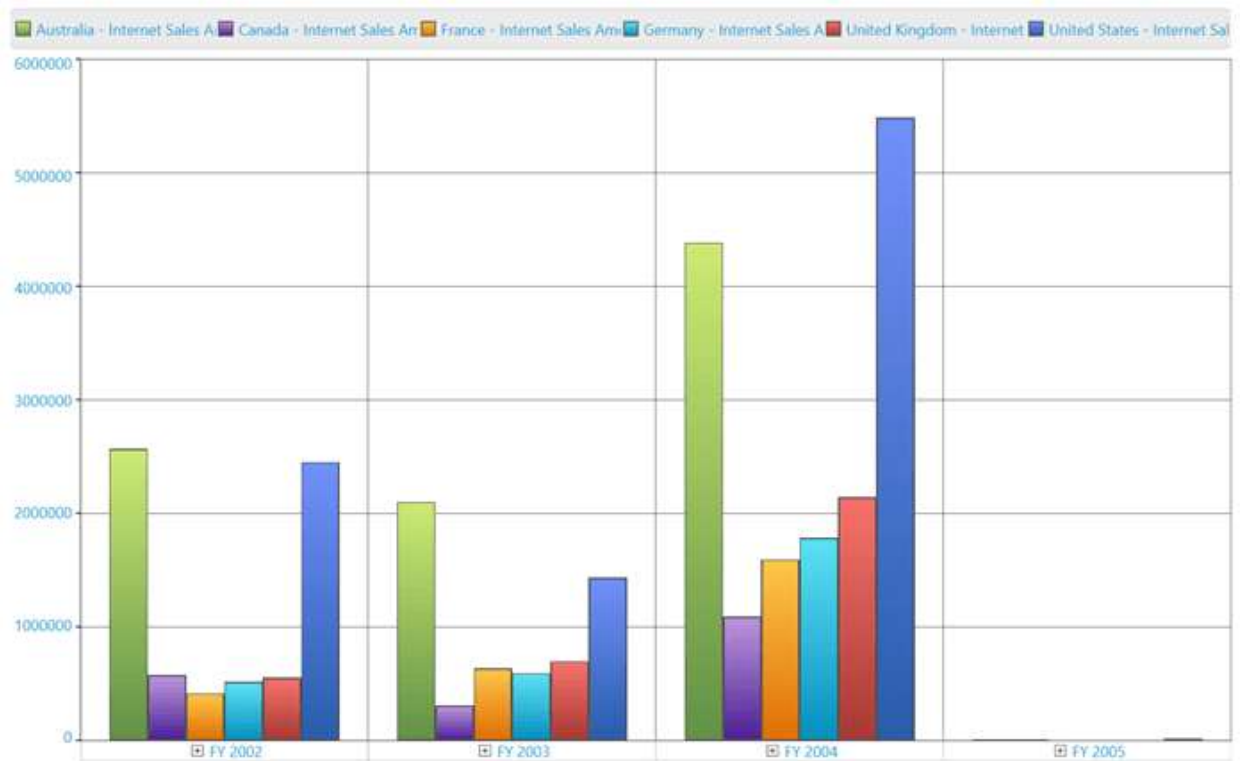
```

Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Imports Syncfusion.Windows.Chart.Olap
Namespace SampleApplication
Partial Public Class MainWindow Inherits SampleWindow
Private _connectionString As String
Private _olapDataManager As OlapDataManager
Public Sub New()
InitializeComponent()
Dim olapChart As New OlapChart()
_connectionString = " Enter a valid connection string "
' Connection string is passed to OlapDataManager as an argument
_olapDataManager = New OlapDataManager(_connectionString)
' A default OlapReport is set to OlapDataManager
_olapDataManager.SetCurrentReport(CreateOlapReport())
' Finally OlapChart gets the information from the OlapDataManager
olapChart.OlapDataManager = _olapDataManager
olapChart.DataBind()
// OlapChart added to the Main Window Grid region
chart.Children.Add(olapChart)
End Sub
''' <summary>
''' Defining OlapReport with Dimension and Measure
''' </summary>
Private Function CreateOlapReport() As OlapReport
Dim olapReport As OlapReport = New OlapReport()
' Setting the Cube name
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementColumn.Name = "Customer"
' Specifying the Hierarchy and Level name
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
' Specifying the Measure name
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementRow.Name = "Date"
' Specifying the Hierarchy and Level name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
''' Adding Dimension in column axis
olapReport.CategoricalElements.Add(dimensionElementColumn)
''' Adding Measure in column axis
olapReport.CategoricalElements.Add(measureElementColumn)
''' Adding Dimension in row axis
olapReport.SeriesElements.Add(dimensionElementRow)
Return olapReport

```

```
End Function  
End Class  
End Namespace
```

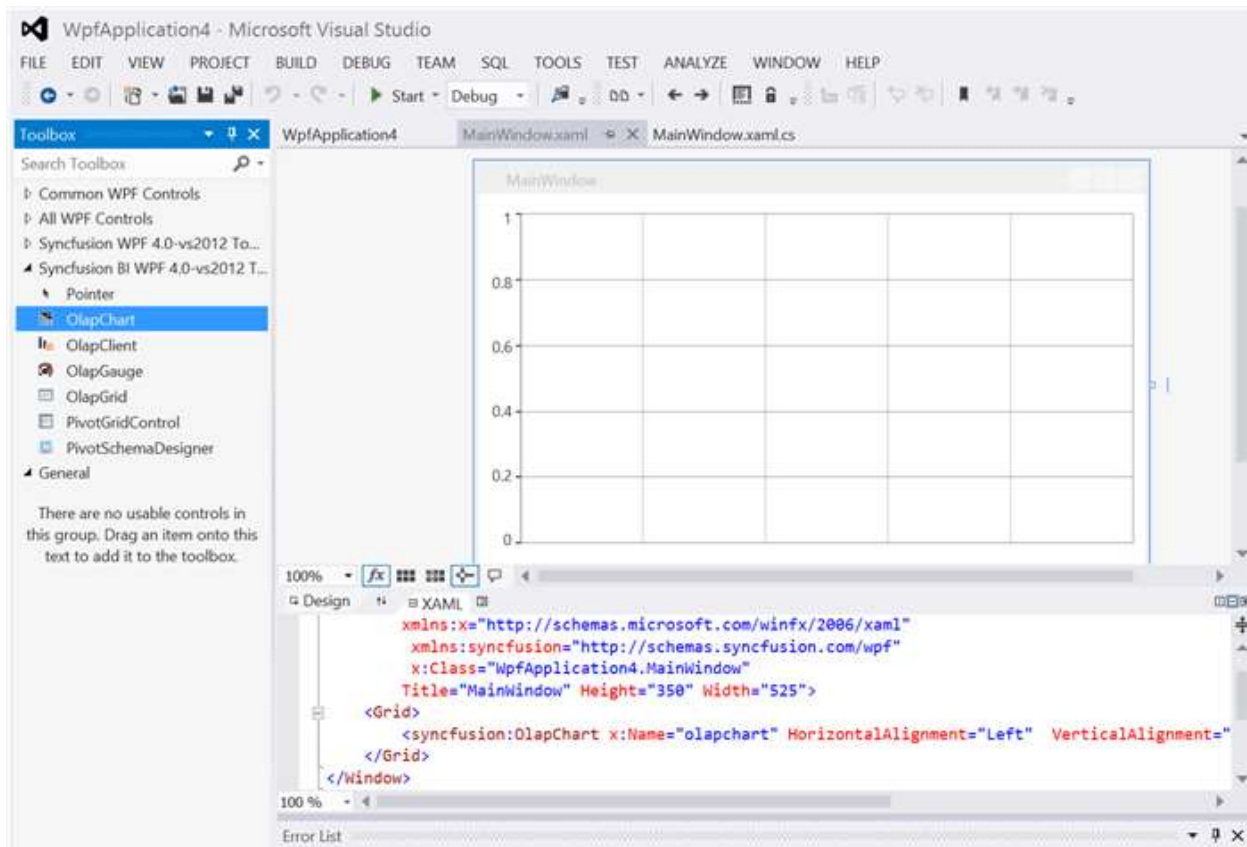
Run the application. The following output will be generated.



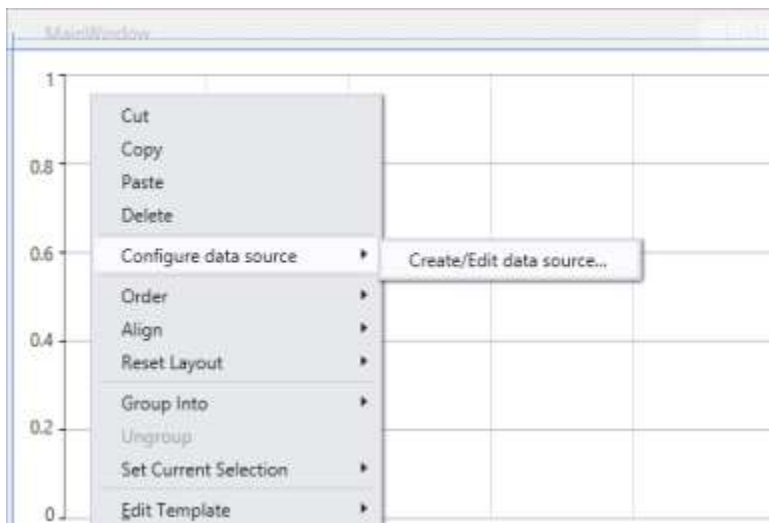
Design-time binding

Design-time binding support for OLAP chart allows you to reduce the time spent on creating and customizing the report. Normally, it takes 5 to 10 minutes for creating a report and in case of unfamiliar cubes it may extend further. But by using design-time support, you can create a report in a couple of minutes. The following section explains how to create a report during design-time.

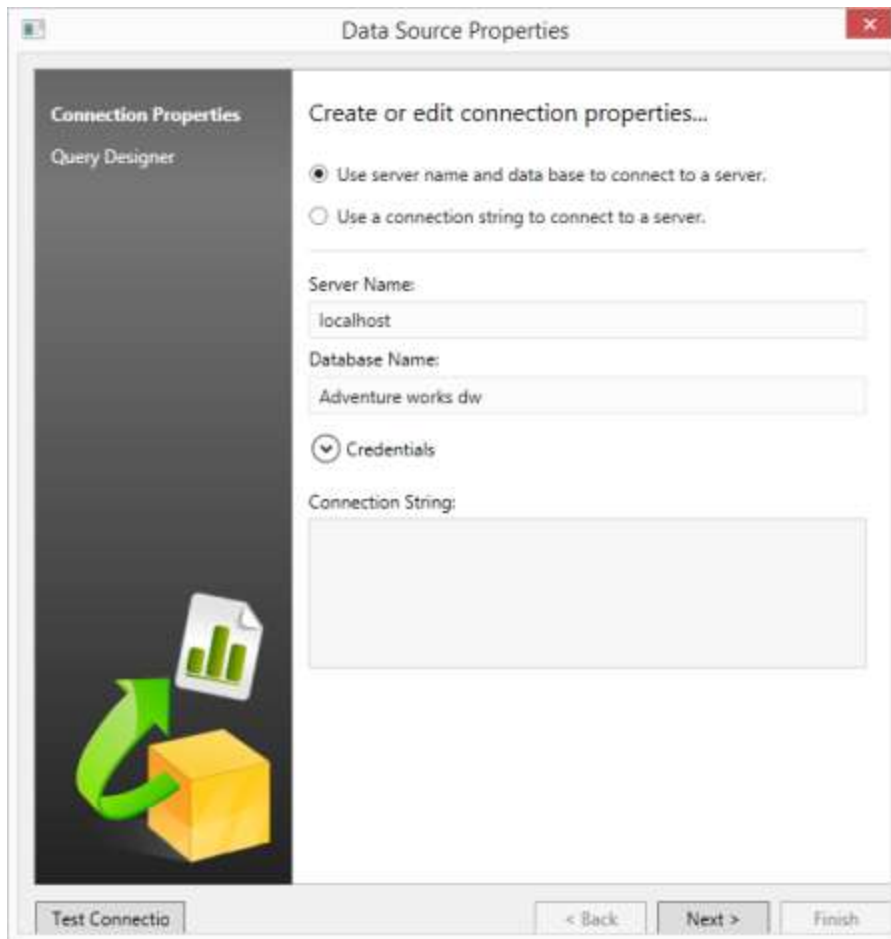
First drag the OLAP chart control from the toolbox to the Visual Studio designer surface.



Right-click the OLAP chart available in the designer and go to **Configure data source > Create/Edit data source...** option in the context menu. Now, the **Data Source Properties** wizard opens.



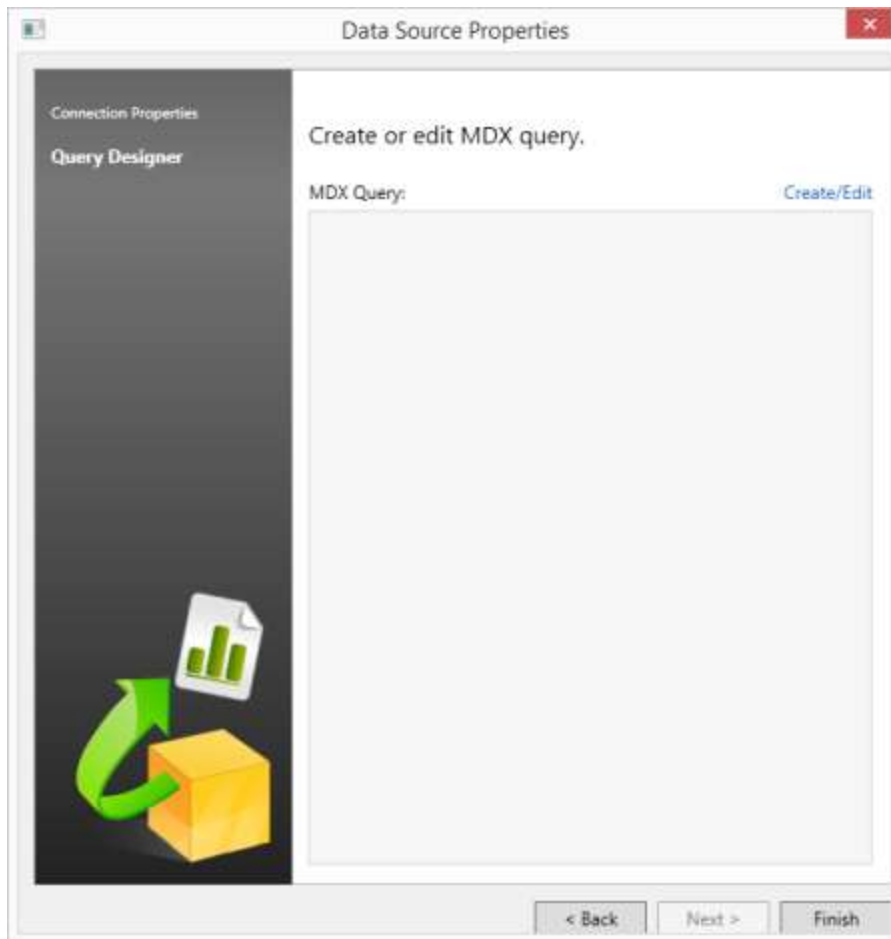
From the **Data Source Properties** wizard, select the connection type. If you want to connect to SSAS, select **Use server name and database to connect to a server** and specify the necessary information to connect to the server. If you want to connect to an Offline cube, select **Use a connection string to connect to a server** and enter your connection string path.



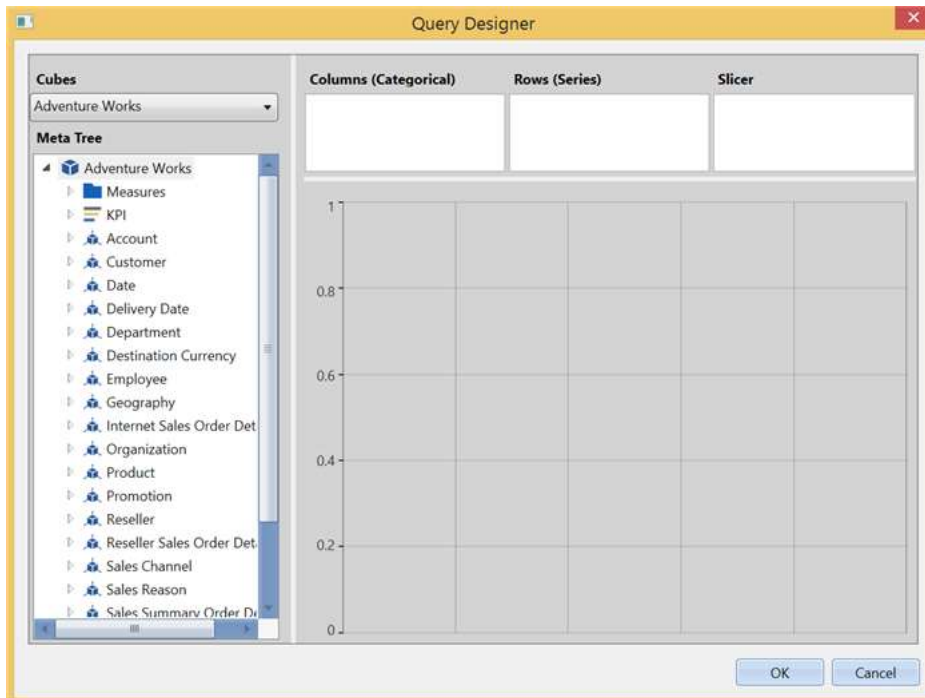
If you want to test the connection, click the **Test Connection** button that is displayed on the bottom-left corner of the window. Click **Next** to proceed.

Note: The Next button is enabled only when any one of the connection option is filled properly.

When the connection is valid, it displays the summary page of the **Data Source Properties** wizard. When you create a query for the first time, the MDX query text box in the summary page is empty. When you edit an existing query, it displays the current query in the text box.



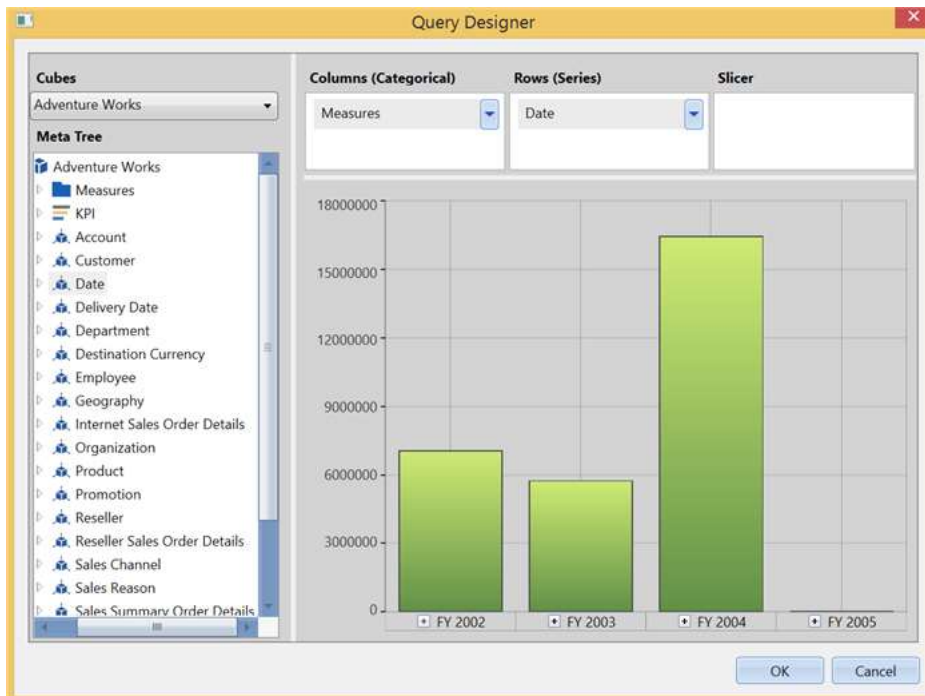
For creating or editing a query, click the **Create/Edit** link that is displayed on the top-right side of the MDX query text box. This opens a **Query Designer** dialog.



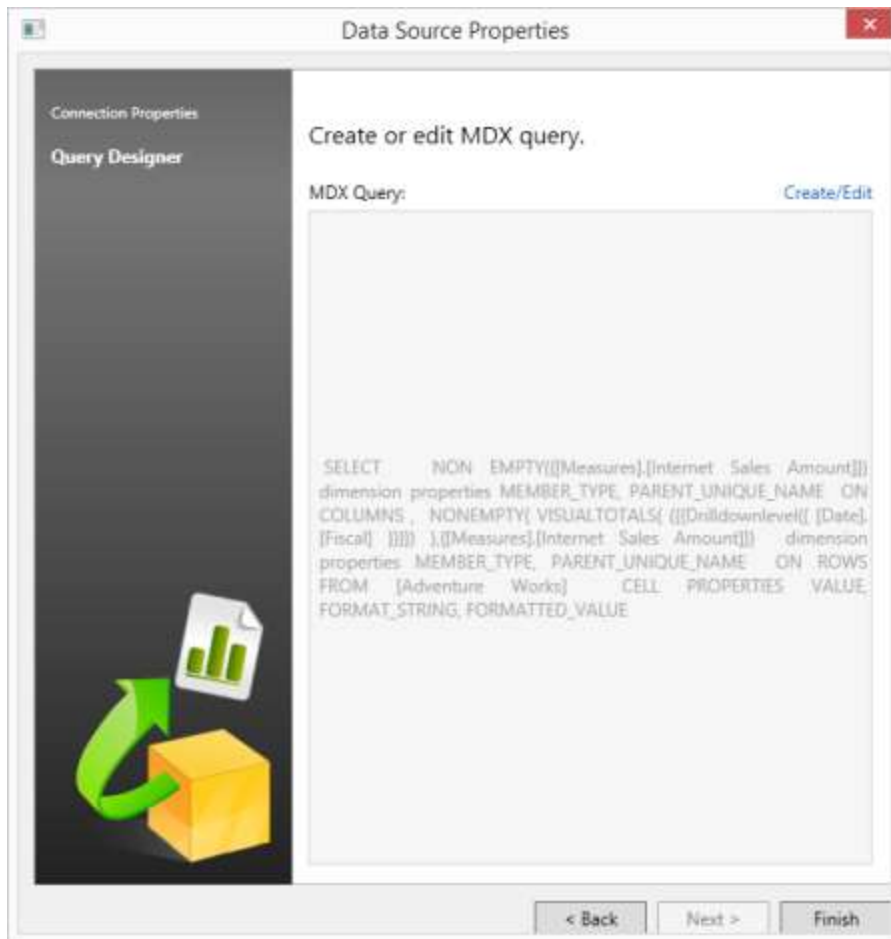
When you edit an existing query, it displays the required dimensions in the specific axis of the query designer and the preview of that query is displayed in the chart control.

Note: It does not display any style/formatting applied to the chart. It only displays the result of the query.

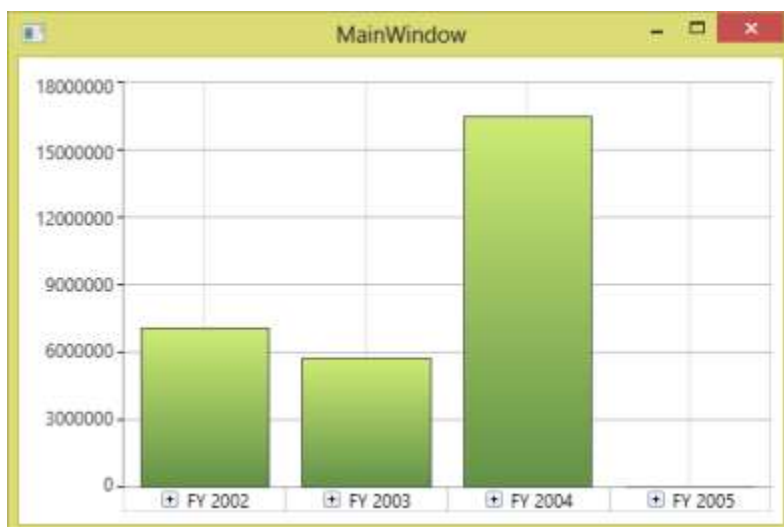
Drag the dimensions to frame a new query. Then, click OK to save the query or click Cancel to revert the changes made during this session.



The summary page of the **Data Source Properties** wizard displays the resultant MDX query.



Click **Finish** and run the application.



Data Binding in WPF Olap Chart

Binding OLAP chart to offline cube

To connect to an OLAP cube available in the local machine, set the physical path of the cube set in the connection string. The following code sample illustrates the same.

C#

```
string connectionString = @"DataSource = system
drive:\OfflineCube\Adventure_Works_Ext.cub; Provider = MSOLAP;";
OlapDataManager DataManager = new OlapDataManager(connectionString);
```

Binding OLAP chart to cube in local SQL Server

To connect to the OLAP cube available in SQL Server Analysis Service in the local machine, set the server name and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code sample illustrates the same.

C#

```
string connectionString = "Data source=localhost; Initial Catalog=Adventure
Works DW;";
OlapDataManager DataManager = new OlapDataManager(connectionString);
```

Binding OLAP chart to cube in online SQL Server

To connect to the OLAP cube available in SQL Server Analysis Service in the online server through **XML/A**, set the host server link and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code sample illustrates the same.

C#

```
string connectionString = "Data
Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure
Works DW 2008 SE;";
OlapDataManager DataManager = new OlapDataManager(connectionString);
```

Binding OLAP chart to cube in online Mondrian Server

To connect to the OLAP cube available in the Mondrian Server through **XML/A**, set the host server link and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code sample illustrates the same.

C#

```
string connectionString = @"Data Source =
http://localhost:8080/mondrian/xmla; Initial Catalog =FoodMart;";
OlapDataManager DataManager = new OlapDataManager(connectionString);
DataManager.DataProvider.ProviderName =
Syncfusion.Olap.DataProvider.Providers.Mondrian;
```

Binding OLAP chart to cube in online ActivePivot Server

To connect to the OLAP cube available in the ActivePivot Server through **XML/A**, set the host server link and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code sample illustrates the same.

C#

```
string connectionString = @"Data Source = http://localhost:8080/cva_s/xmla;
Initial Catalog = CVAS;";
```

```
OlapDataManager DataManager = new OlapDataManager(connectionString);
DataManager.DataProvider.ProviderName=Syncfusion.Olap.DataProvider.Providers
.ActivePivot;
```

XAML Configuration in WPF Olap Chart

XAML configuration is one of the important features of the OLAP chart, as it helps users to configure the control entirely through XAML by eliminating the required code in code behind.

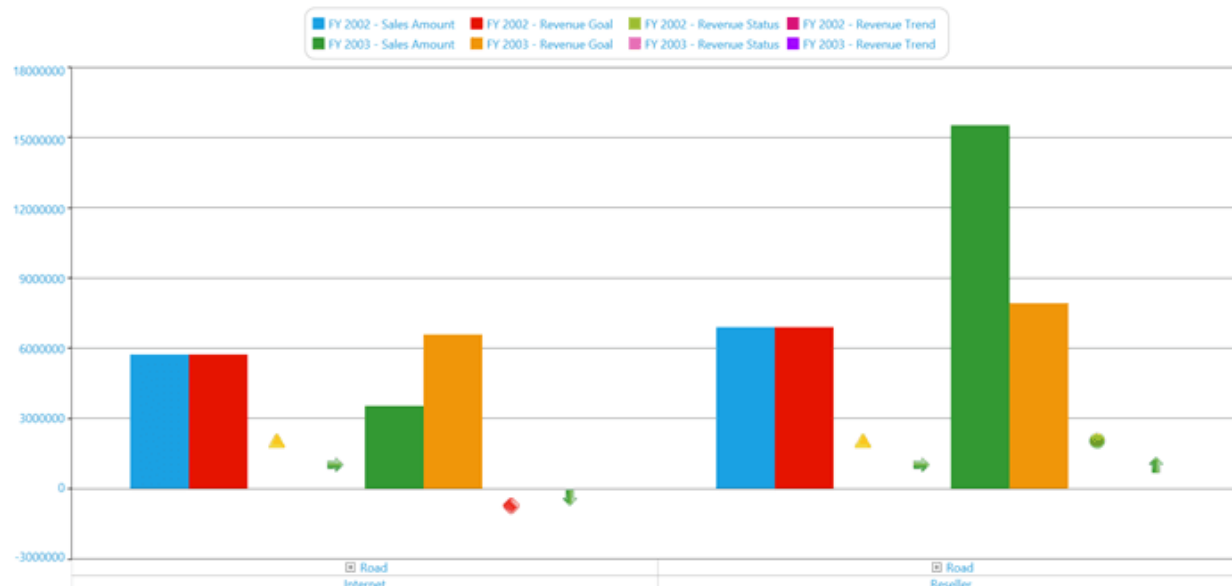
Properties

- **DataSource.ConnectionString:** Specifies the connection string of the data manager.
- **DataSource.ConnectionName:** Specifies the connection name, which is available in the App.Config file of the application.
- **DataSource.DataManagerName:** Specifies the data manager name.
- **SharedDataManagerName:** Specifies the data manager name, which is available in the shared data manager collection.
- **ReportName:** Specifies the OLAP report name.
- **CurrentCubeName:** Specifies the current cube name of an OLAP report.
- **CategoricalAxis:** Specifies the categorical axis of the OLAP report.
- **SeriesAxis:** Specifies the series axis of the OLAP report.
- **SlicerAxis:** Specifies the slicer axis of the OLAP report.
- **CalculatedMembers:** Specifies the calculated members of the OLAP report.

Adding an OLAP report to the OLAP chart in design time is described in the following code sample.

XML

```
<syncfusion:OlapChart x:Name="olapChart" HorizontalAlignment="Stretch"
ReportName="SalesReport"
CurrentCubeName="Adventure Works" SharedDataManagerName="localManager"
olapshared:DataSource.DataManagerName="localManager"
olapshared:DataSource.ConnectionString="datasource=localhost;
initial catalog=adventure works dw">
<!-- Adding Elements to Categorical Axis -->
<syncfusion:OlapChart.CategoricalAxis>
<syncfusion:Dimension Name="Date" HierarchyName="Fiscal" LevelName="Fiscal Y
ear" IncludeMembers="FY 2002, FY 2003" /> <!-- Multiple Members where
specified by comma separate -->
<syncfusion:Kpi Name="Revenue" ShowGoal="True" ShowStatus="True" ShowValue="
True" ShowTrend="True" />
</syncfusion:OlapChart.CategoricalAxis>
<!-- Adding Elements to Series Axis -->
<syncfusion:OlapChart.SeriesAxis>
<syncfusion:Dimension Name="Sales Channel" HierarchyName="Sales Channel" Lev
elName="Sales Channel" />
<syncfusion:Dimension Name="Product" HierarchyName="Product Model Lines" Lev
elName="Product Line" IncludeMembers="Road" />
</syncfusion:OlapChart.SeriesAxis>
</syncfusion:OlapChart>
```



A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Defining Reports\ XAML Configuration Demo

KPI in WPF Olap Chart

KPI is a collection of calculations that are associated with a measure group in a cube, which is used to evaluate business success. Typically, these calculations are a combination of multi-dimensional expressions (MDX) or calculated members. KPIs also have additional metadata that provides information about how Grid applications should display the results of KPIs calculations.

The following are the different types of indicators:

- KPI goal
- KPI status
- KPI trend
- KPI value

The KPI elements can be defined in the OLAP report in the following way.

C#

```

/// <summary>
/// OlapReport with KPI Elements
/// </summary>
/// <returns></returns>
private OlapReport LoadBasicKPI()
{
    OlapReport olapReport = new OlapReport();
    // Selecting the Cube
    olapReport.CurrentCubeName = "Adventure Works";
    KpiElements kpiElement = new KpiElements();
    // Specifying the KPI Element name and configuring its Indicators
    kpiElement.Elements.Add(new KpiElement
    {

```



```

Name = "Internet Revenue",
ShowKPIGoal = true,
ShowKPIStatus = true,
ShowKPIValue = true,
ShowKPITrend = true
});
DimensionElement dimensionElementRow = new DimensionElement();
// Specifying the Name for Row Dimension Element
dimensionElementRow.Name = "Date";
// Specifying the Level element
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
// Adding Row Elements
olapReport.SeriesElements.Add(dimensionElementRow);
// Adding Column Elements
olapReport.CategoricalElements.Add(kpiElement);
return olapReport;
}

```

VB.NET

```

''' <summary>
''' OlapReport with KPI Elements
''' </summary>
''' <returns></returns>
Private Function LoadBasicKPI() As OlapReport
Dim olapReport As New OlapReport()
' Selecting the Cube
olapReport.CurrentCubeName = "Adventure Works"
Dim kpiElement As New KpiElements()
' Specifying the KPI Element name and configuring its Indicators
kpiElement.Elements.Add(New KpiElement() With { _
Key .Name = "Internet Revenue", _
Key .ShowKPIGoal = True, _
Key .ShowKPIStatus = True, _
Key .ShowKPIValue = True, _
Key .ShowKPITrend = True _
})
Dim dimensionElementRow As New DimensionElement()
' Specifying the Name for Row Dimension Element
dimensionElementRow.Name = "Date"
' Specifying the Level element
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
' Adding Row Elements
olapReport.SeriesElements.Add(dimensionElementRow)
' Adding Column Elements
olapReport.CategoricalElements.Add(kpiElement)
Return olapReport
End Function

```



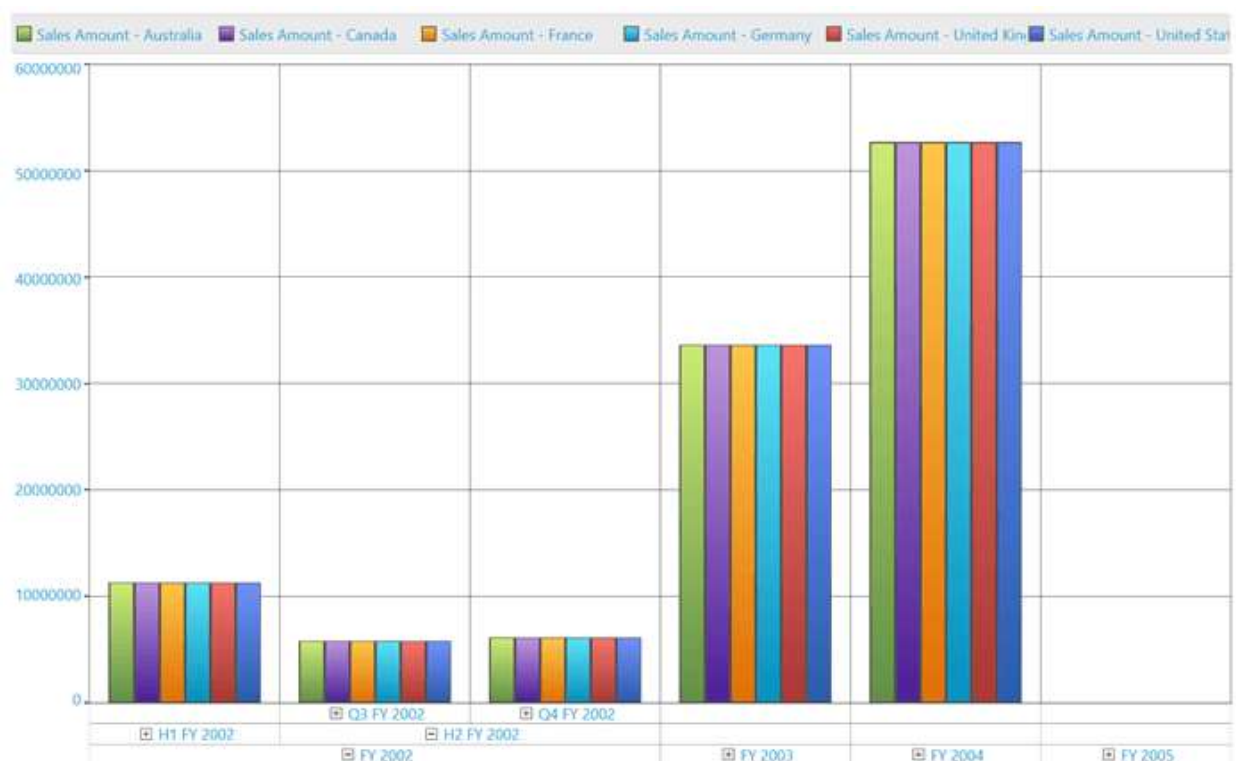
Drill Operation in WPF Olap Chart

This is a basic feature of OLAP chart through which the amount of information can be limited, for a better view. It allows you to drill down to access the detailed level of data or drill up to see the summarized data by using the context menu present in the OLAP chart.

Drill up, also called roll up, navigates from more detailed data to less detailed data by climbing up a concept hierarchy for a dimension.

Drill down, also called roll down, is the reverse of drill up. It navigates from less detailed data to more detailed data by climbing down a concept hierarchy for a dimension.

While binding hierarchical dimensions (for example, the time dimension includes three levels namely year, quarter, and month), the chart allows you to visualize the data for different levels by using the collapsible labels. This is illustrated in the following screenshot.



A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Creating Reports\Reports In Code

Drill position

Drill position allows users to drill only the current position of a selected member and it will exclude the drilled data of the selected member in other positions by using the MDX query. It can be enabled by setting the “**DrillType**” enumeration to “**DrillPosition**” in the OLAP report.

C#

```
dataManager.CurrentReport.DrillType = DrillType.DrillPosition;
```

VB.NET

```
dataManager.CurrentReport.DrillType = DrillType.DrillPosition
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Data Relation\Drill Types Demo

Show/hide expanders

The visibility of expanders in the OLAP chart can be toggled by using the **ShowExpanders** property available in the OLAP report.

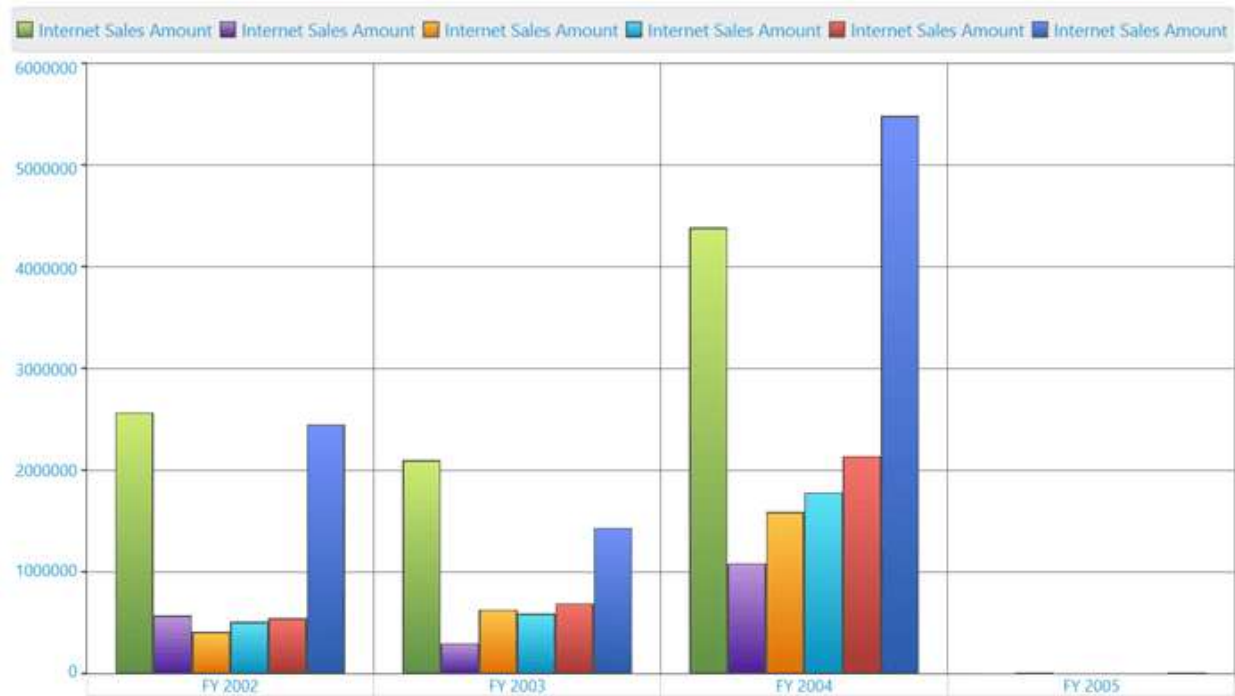
C#

```
this.olapchart1.OlapDataManager.CurrentReport.ShowExpanders = false;
```

VB.NET

```
Me.olapchart1.OlapDataManager.CurrentReport.ShowExpanders = False
```

The following image shows an OLAP chart without expanders.



Note: Since the `ShowExpanders` property interacts with the `OlapDataManager`, you need to assign this property before you call the `DataBind()` method in the OLAP chart.

Chart Types in WPF Olap Chart

The OLAP chart supports the following 16 types of charts:

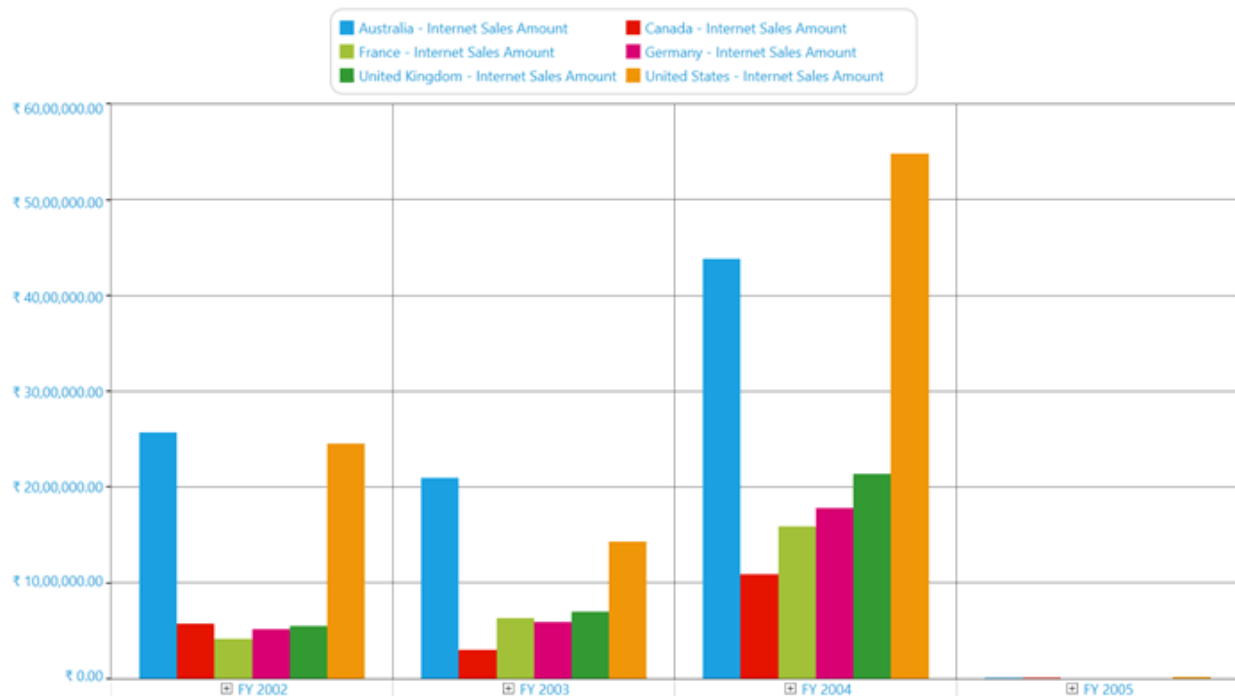
- Column
- Stacking column
- Stacking column 100
- Bar
- Stacking bar
- Stacking bar 100
- Area
- Stacking area
- Spline area
- Step area
- Line
- Spline
- Rotated spline
- Step line
- Scatter

- Pie

Note: Chart type must be set before invoking the `DataBind()` method. Whenever you change the chart type, you need to call the `DataBind()` method to get the changes reflected.

Column chart

Column chart is the most basic type of all charts. Column charts are widely used for comparison analysis.



The following code sample shows how to select a simple column chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="Column" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.Column;
```

VB.NET

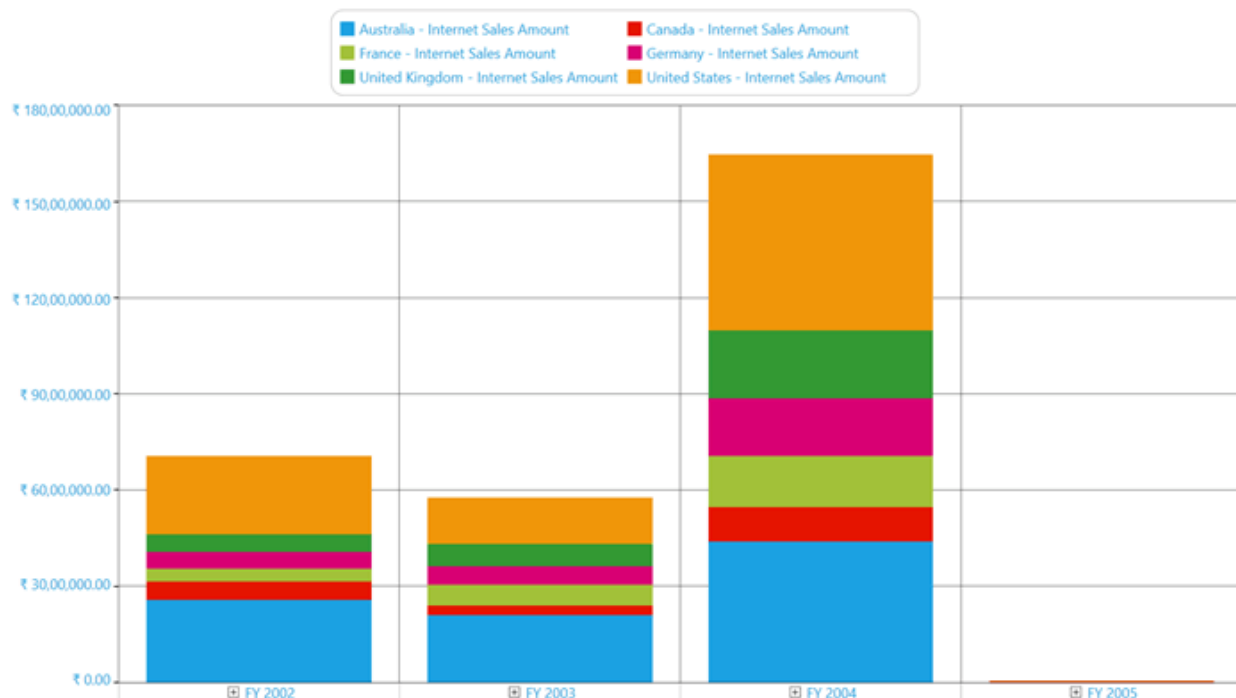
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.Column
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Column Chart Demo

Stacking column chart

The stacking column chart is a simple form of chart, which contains segments in each series. This chart type is widely used for proportional analysis over a particular period of time.



The following code sample shows how to select a stacking column chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="StackingColumn" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.StackingColumn;
```

VB.NET

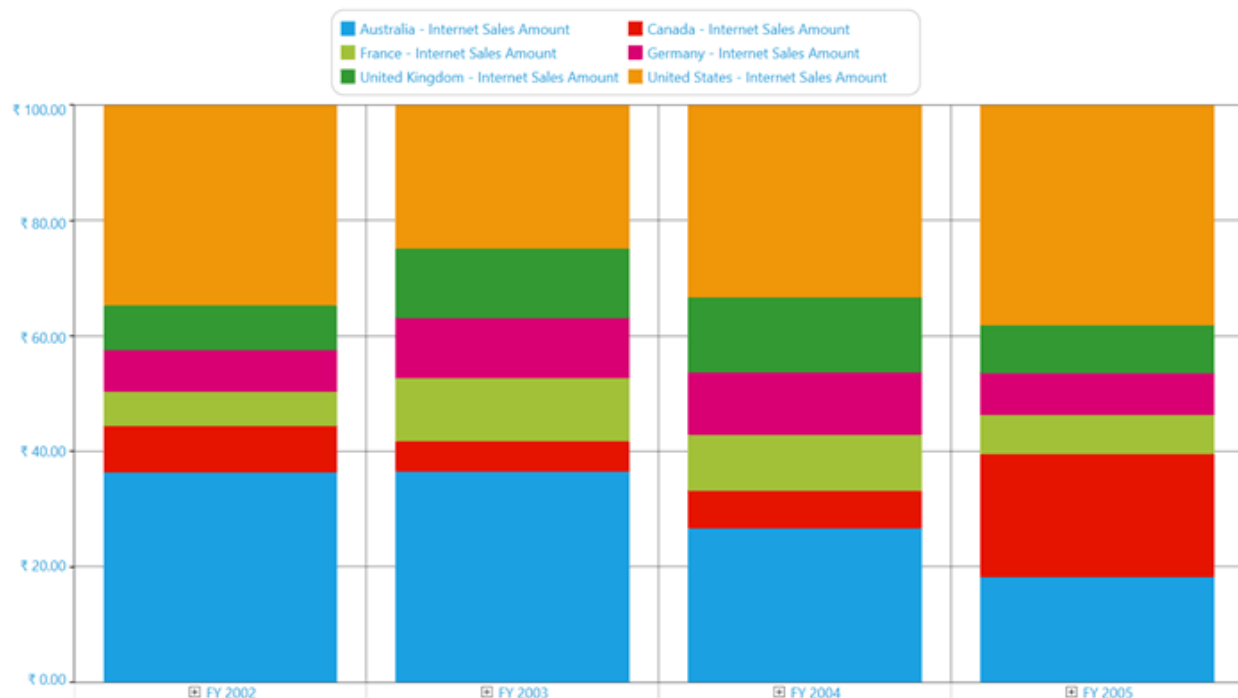
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.StackingColumn
```

A sample demo is available at the following location.

{system drive}\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Column Chart Demo

Stacking column 100 chart

Stacking column 100 chart is a simple form of chart. Like stacking column chart, the stacking column 100 chart also contains segments in each series, which is added to equate each series to 100%. This chart type is widely used for proportional analysis over a particular period of time.



The following code sample shows how to select a stacking column 100 chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="StackingColumn100" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.StackingColumn100;
```

VB.NET

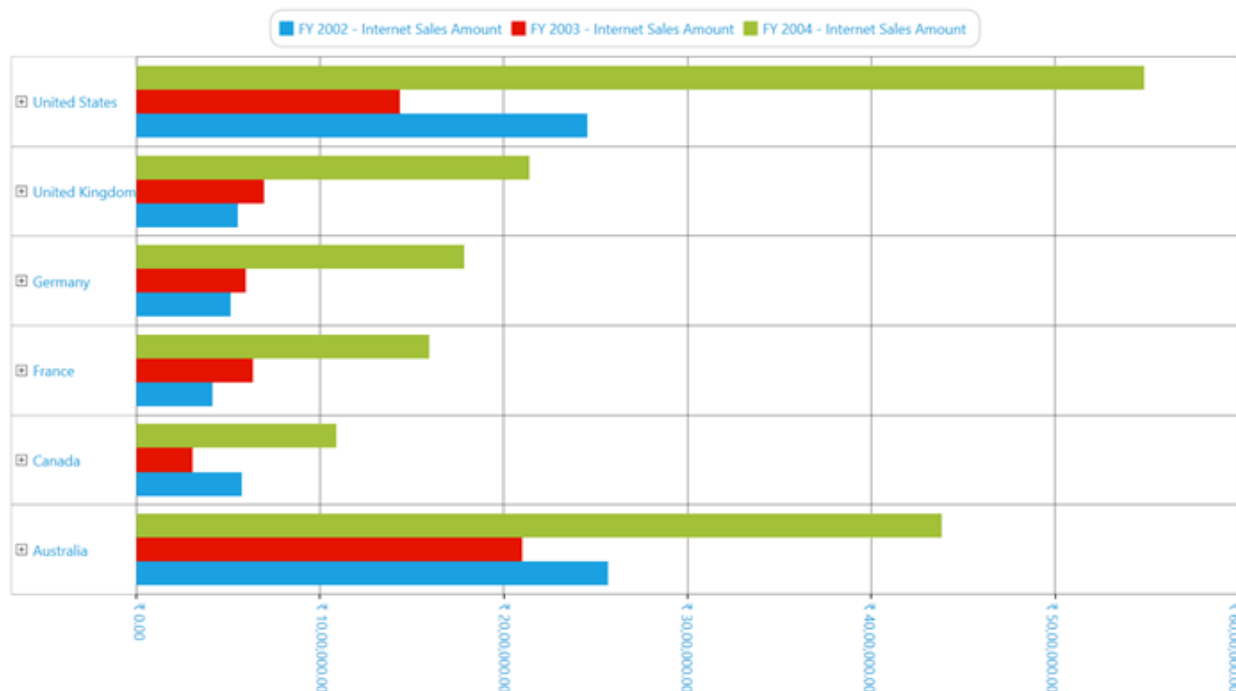
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.StackingColumn100
```

A sample demo is available at the following location.

{system drive}\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Column Chart Demo

Bar chart

The bar chart is same as the column chart, but it can be rotated to 90 degrees in the clockwise direction. This chart type is widely used for comparison analysis over a particular period of time.



The following code sample shows how to select a bar chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="Bar" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.Bar;
```

VB.NET

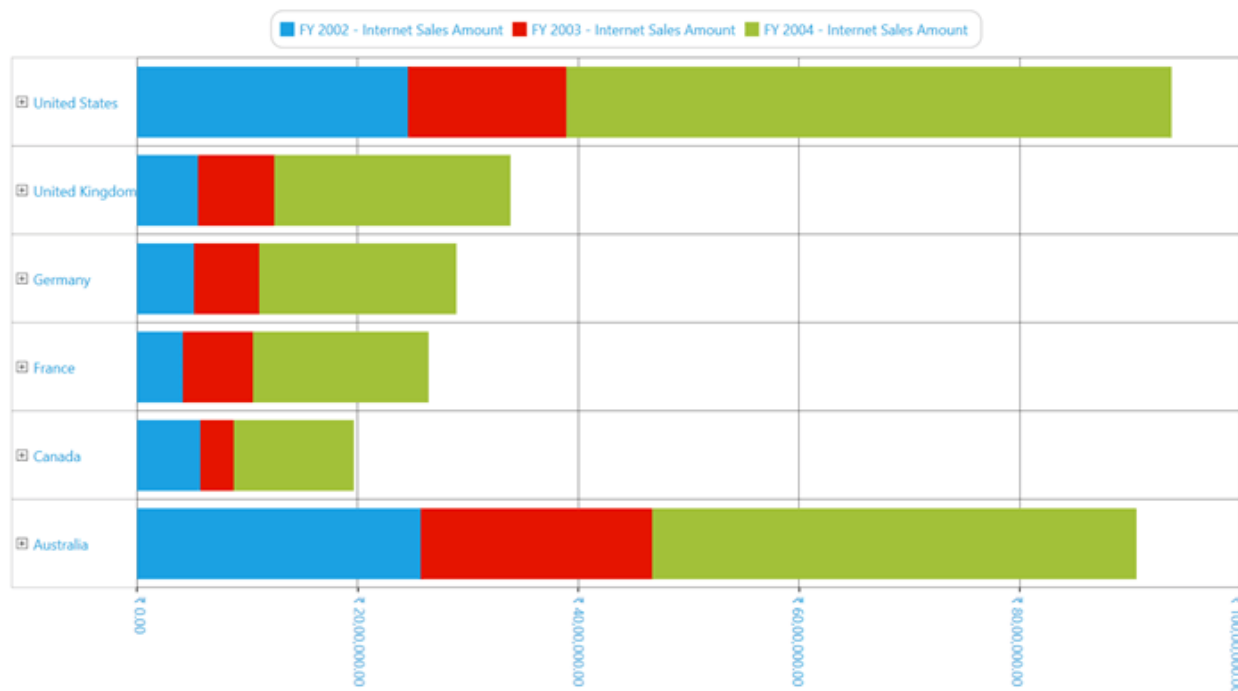
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.Bar
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Bar Chart Demo

Stacking bar chart

Stacking bar chart functions same as the stacking column chart, but it can be rotated to 90 degrees in the clockwise direction. This chart type is widely used for proportional analysis over a particular period of time.



The following code sample shows how to select a stacking bar chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="StackingBar" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.StackingBar;
```

VB.NET

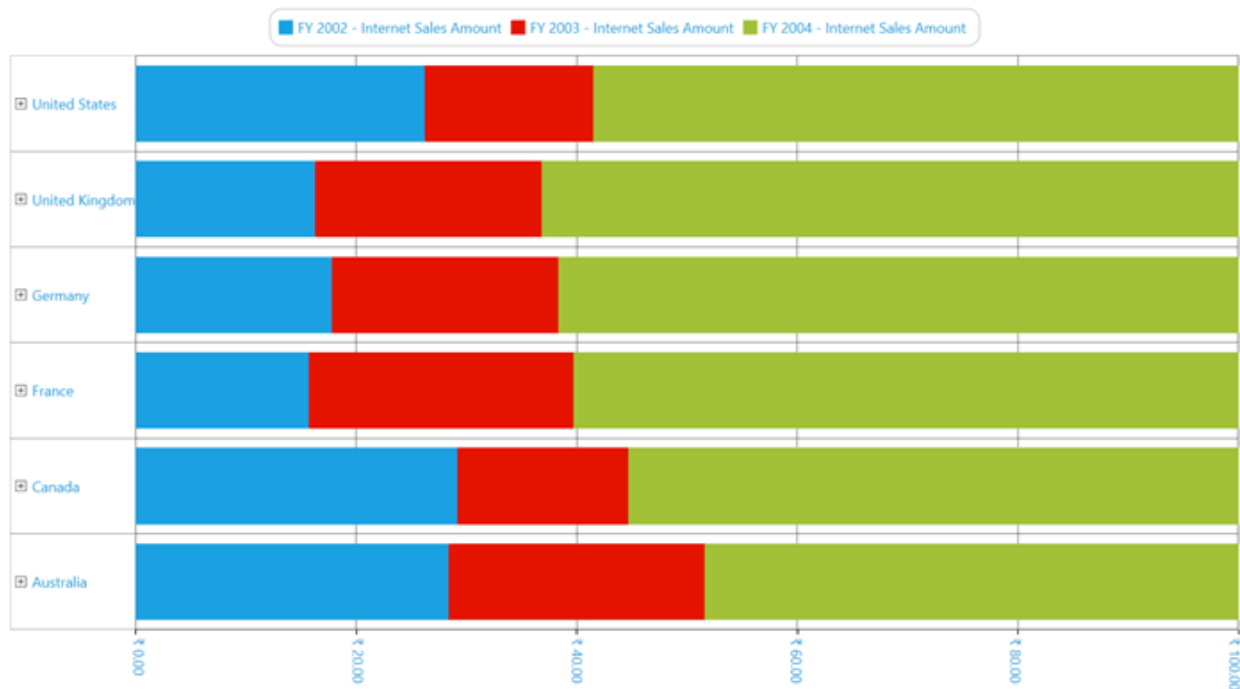
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.StackingBar
```

A sample demo is available at the following location.

{system drive}\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Bar Chart Demo

Stacking bar 100 chart

The stacking bar 100 chart functions same as the stacking column 100 chart, but it can be rotated to 90 degrees in the clockwise direction. This chart type is widely used for proportional analysis over a particular period of time.



The following code sample shows how to select a stacking bar 100 chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="StackingBar100" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.StackingBar100;
```

VB.NET

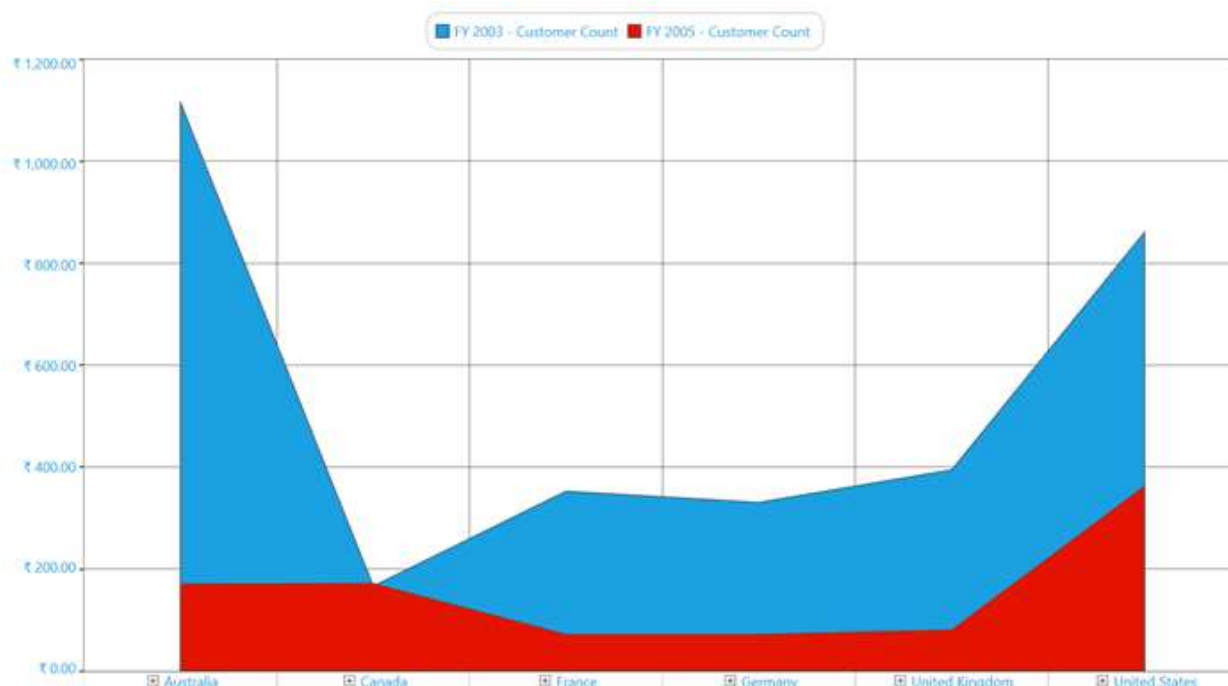
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.StackingBar100
```

A sample demo is available at the following location.

{system drive}\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Bar Chart Demo

Area chart

The area chart fills the quantitative data over a period of time. It is mainly used to compare the quantity plotted over two or more series.



The following code sample shows how to select an area chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="Area" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.Area;
```

VB.NET

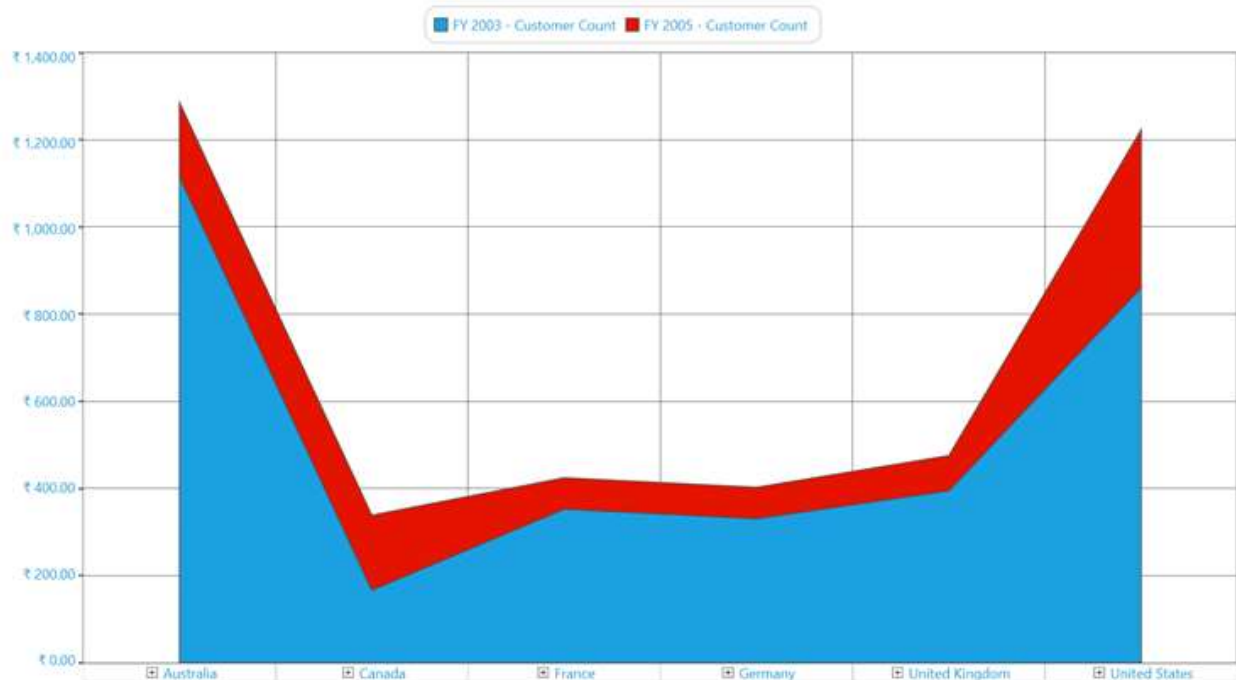
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.Area
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Area Chart Demo

Stacking area chart

The stacking area chart fills the quantitative data over a period of time just like the line area chart, but it differs by the plotting method. In the stacking area chart, each series is plotted on the top of the previous series rather than starting from 0 of the horizontal axis. It is mainly used to compare the quantity plotted over two or more series.



The following code sample shows how to select a stacking area chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="StackingArea" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.StackingArea;
```

VB.NET

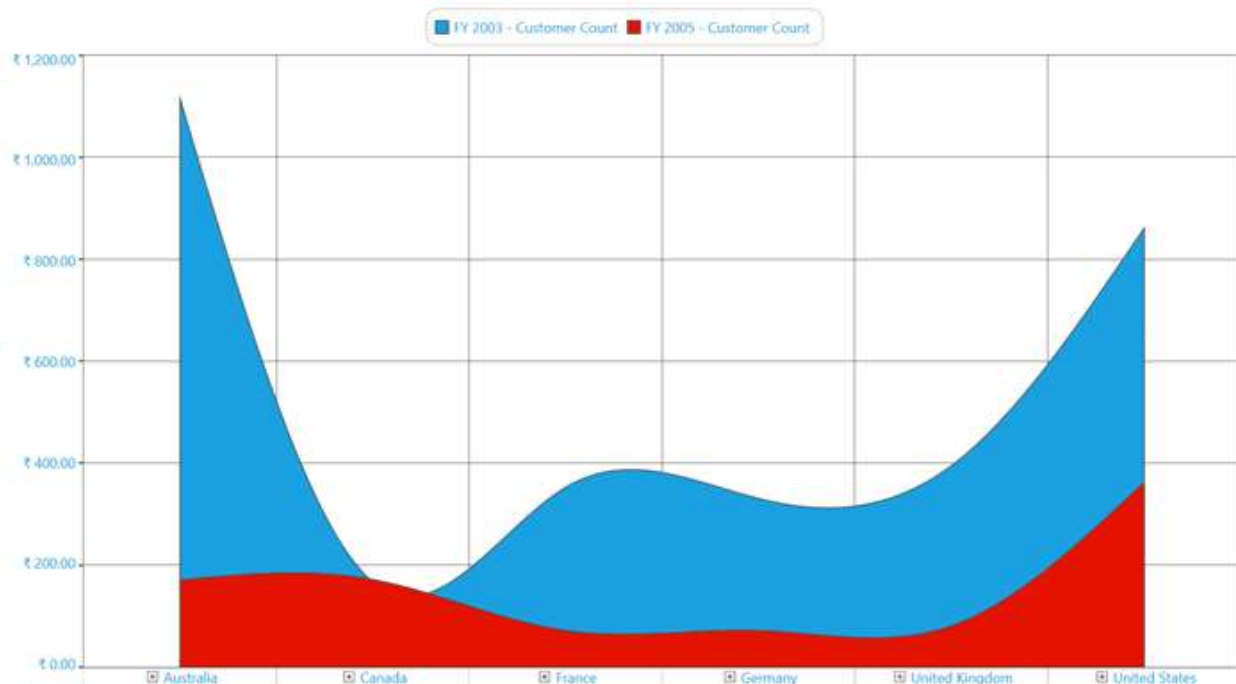
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.StackingArea
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Area Chart Demo

Spline area chart

The spline area chart is usually used in the case of approximating the intervals by using the spline curve. It is often used when data points are in limited number.



The following code sample shows how to select a spline area chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="SplineArea" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.SplineArea;
```

VB.NET

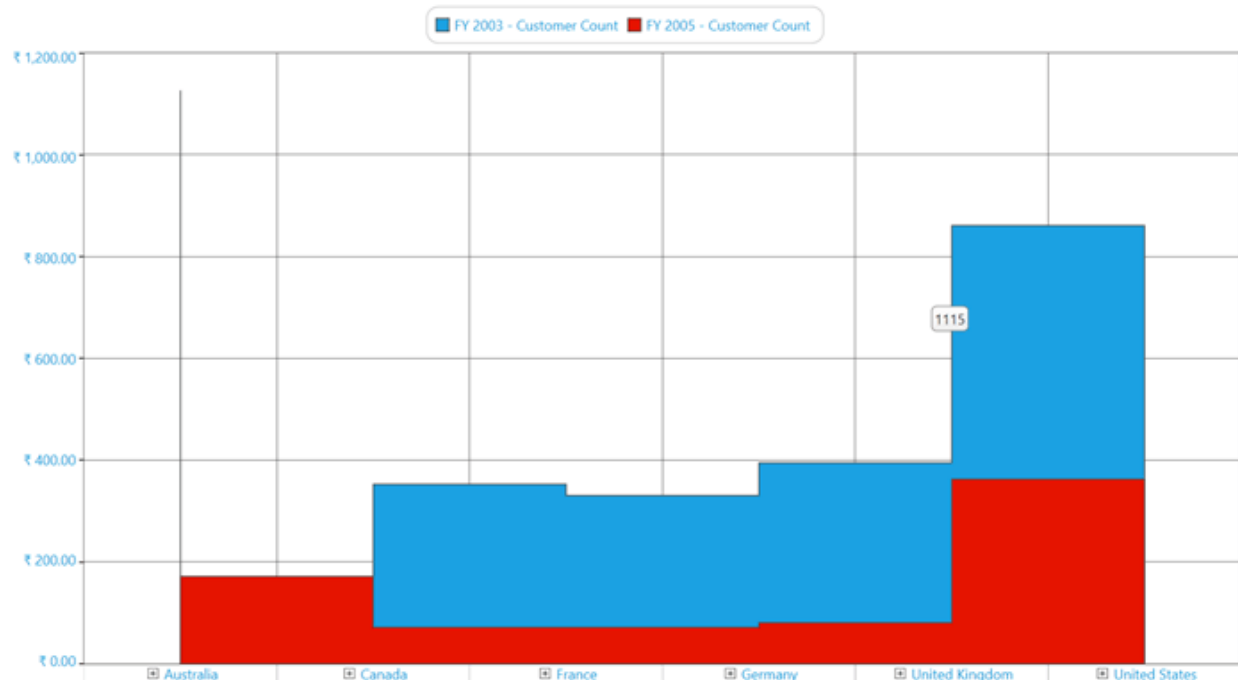
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.SplineArea
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Area Chart Demo

Step area chart

In the step area chart, the points are plotted instead of a straight line tracing the shortest path between points. The values are connected by continuous vertical and horizontal lines.



The following code sample shows how to select a step area chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="StepArea" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.StepArea;
```

VB.NET

```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.StepArea
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Area Chart Demo

Line chart

The line chart is a simple form of chart, which connects a series of data points. Usually, it is used for Trend analysis, Forecasting, or in the case of large data points.



The following code sample shows how to select a line chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="Line" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.Line;
```

VB.NET

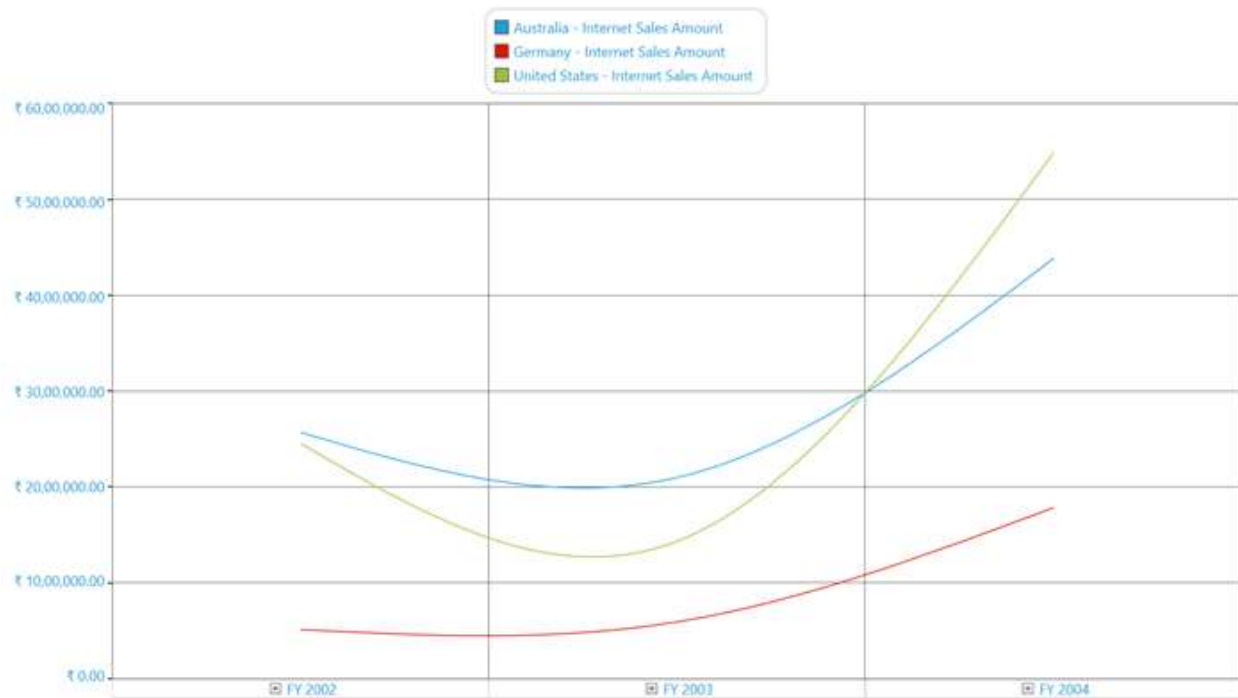
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.Line
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Line Chart Demo

Spline chart

The spline chart is a simple form of chart, which connects the series of data points with an arc rather than a straight line.



The following code sample shows how to select a spline chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="Spline" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.Spline;
```

VB.NET

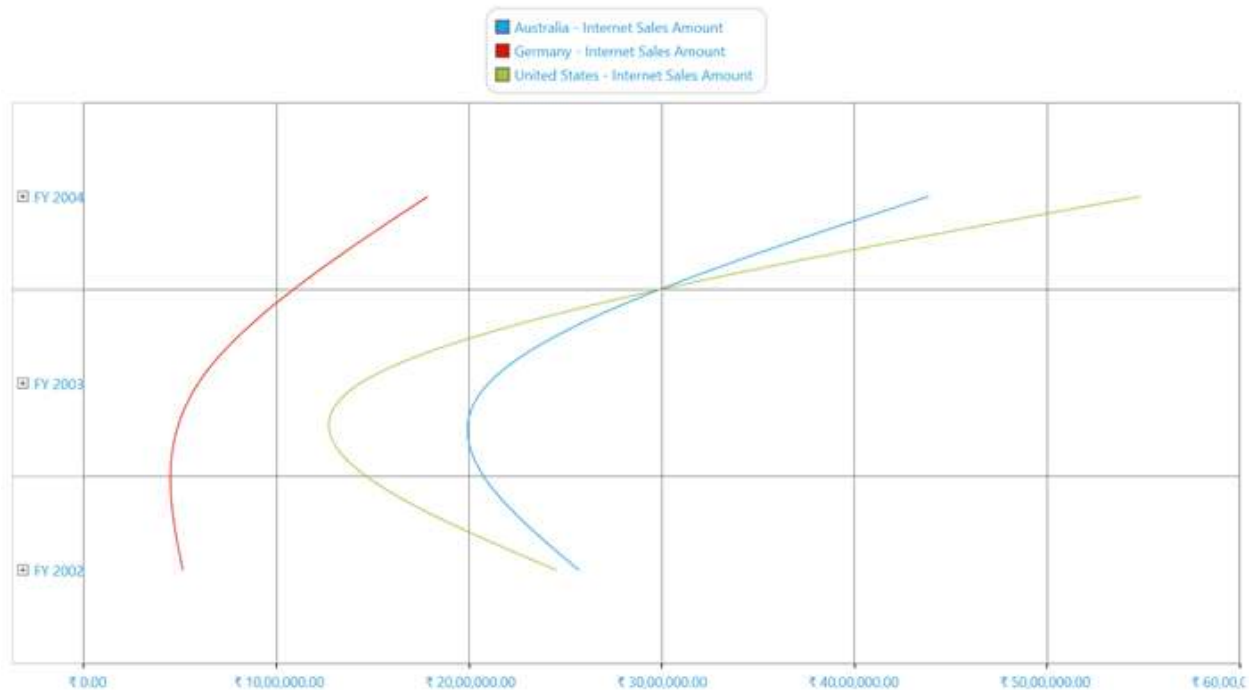
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.Spline
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Line Chart Demo

Rotated spline chart

The rotated spline chart is similar to the spline chart, but it can be rotated to 90 degrees in the clockwise direction.



The following code sample shows how to select a rotated spline chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="RotatedSpline" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.RotatedSpline;
```

VB.NET

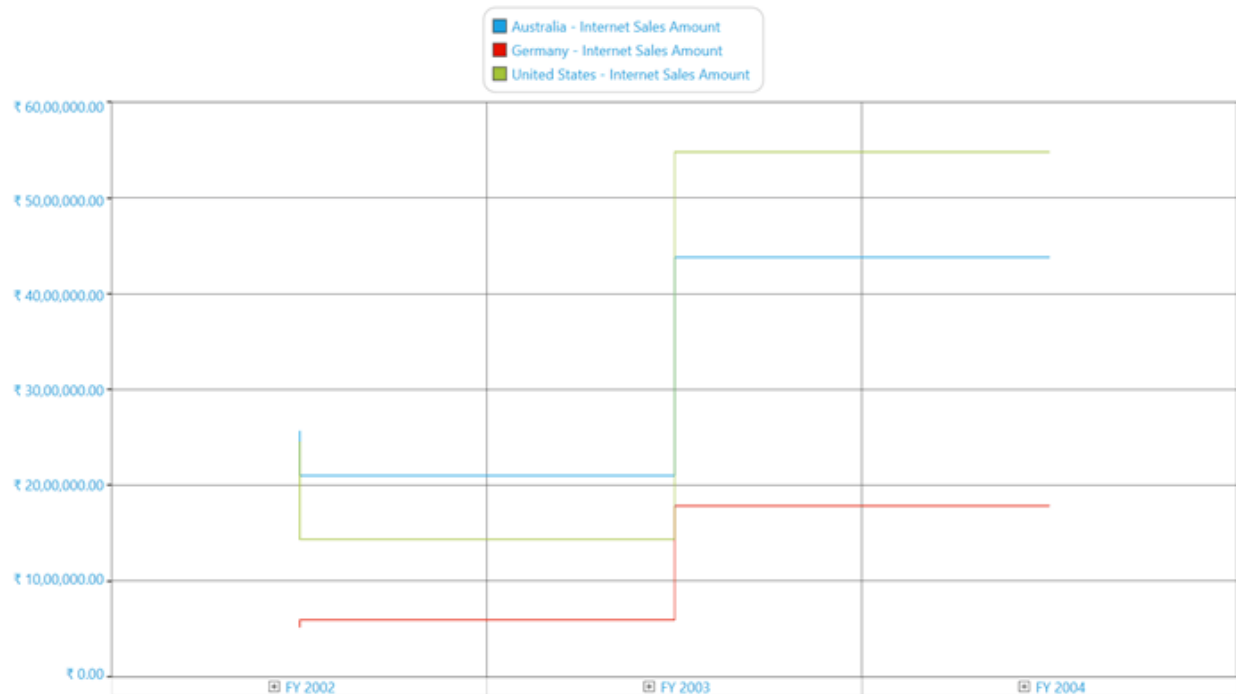
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.RotatedSpline
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Line Chart Demo

Step line chart

The step line chart is another form of chart, which connects the series of data points by using horizontal and vertical lines.



The following code sample shows how to select a step line chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="StepLine" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.StepLine;
```

VB.NET

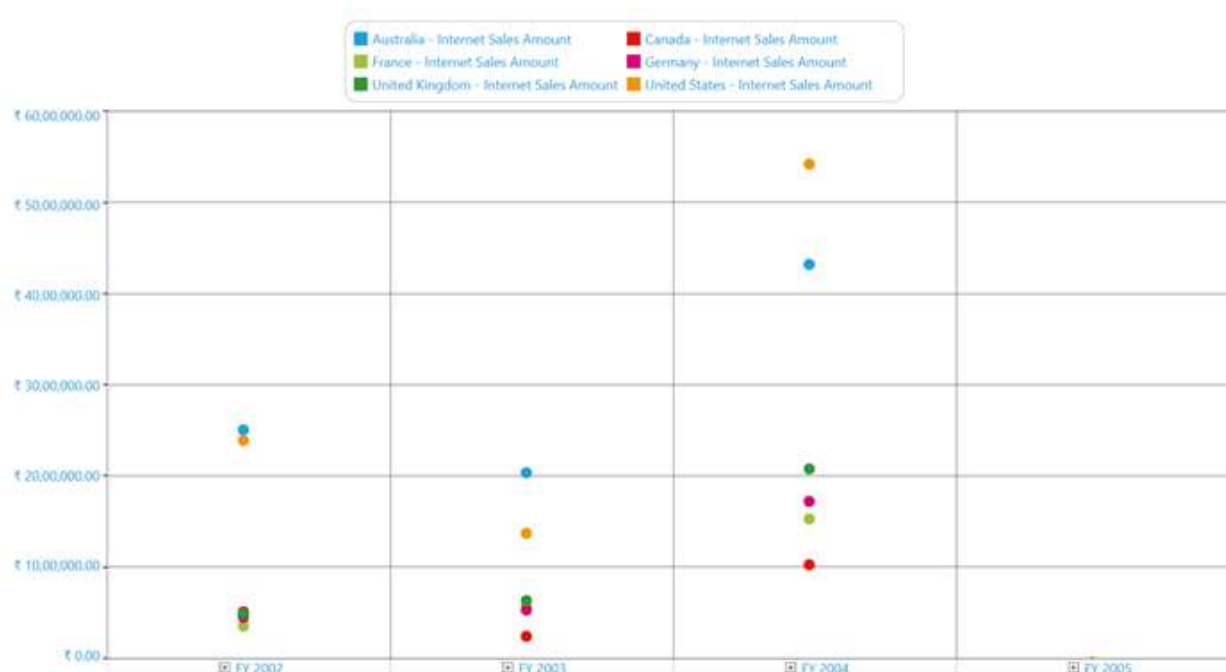
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.StepLine
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Line Chart Demo

Scatter chart

The scatter chart is a collection of points plotted in the rectangular co-ordinate system. It is often used in relationship analysis upto one independent variable.



The following code sample shows how to select a scatter chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="Scatter" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.Scatter;
```

VB.NET

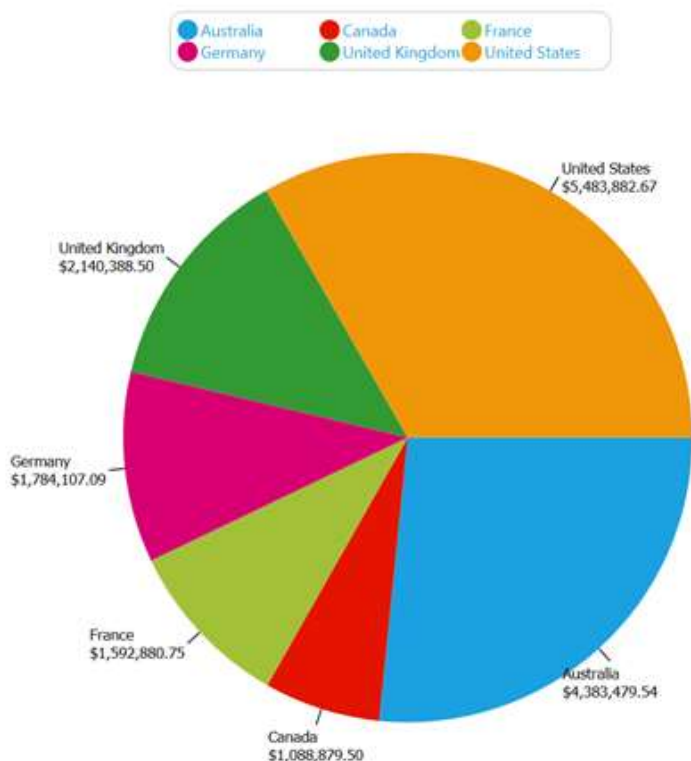
```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.Scatter
```

A sample demo is available at the following location.

{system drive}\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Scatter Chart Demo

Pie chart

The pie chart renders the data points in segments. It can render only one series at a time. Usually, it is used for proportional analysis for a small set of data points.



The following code sample shows how to select a pie chart.

XML

```
<syncfusion:OlapChart Name="olapChart" ChartType="Pie" />
```

C#

```
OlapChart olapChart = new OlapChart();
olapChart.ChartType = ChartTypes.Pie;
```

VB.NET

```
Dim olapChart As OlapChart = New OlapChart()
olapChart.ChartType = ChartTypes.Pie
```

Note: The pie chart should not be used for comparison analysis of large data points, because it is harder for people to estimate angles rather than distance.

A sample demo is available at the following location:

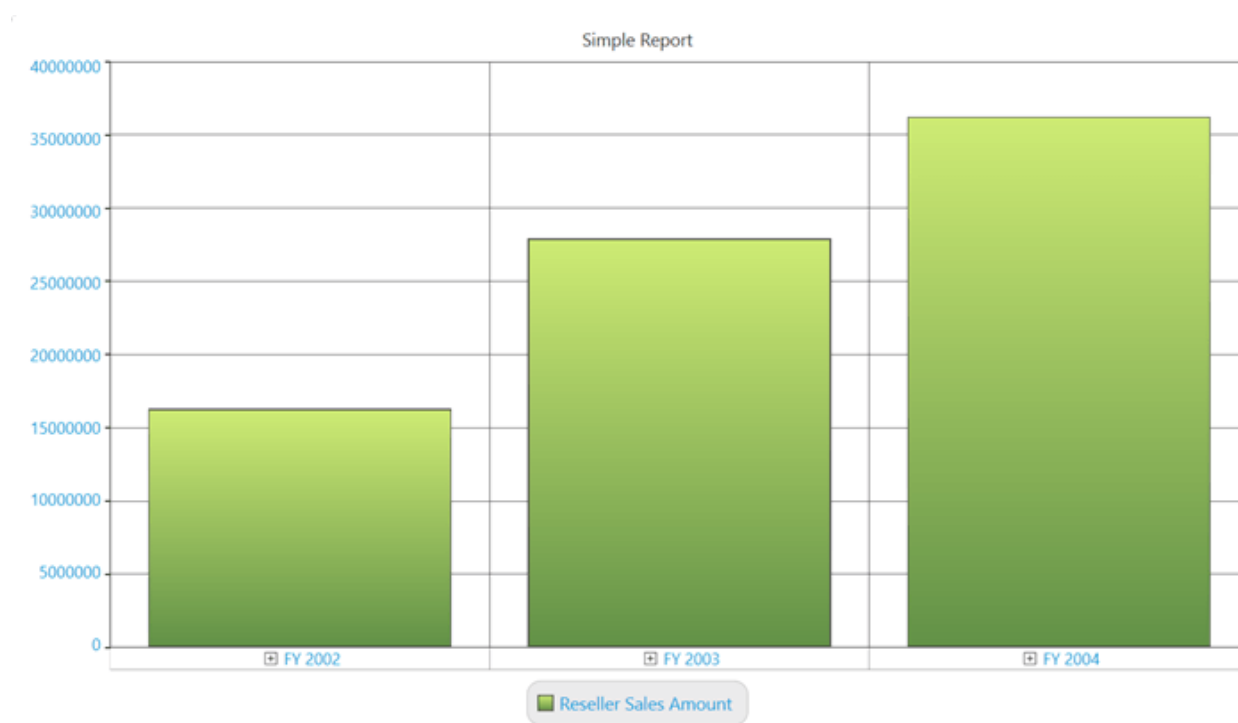
{system drive}\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Types\Pie Chart Demo

Area in WPF Olap Chart

The OLAP area is a chart area, which contains the chart series and chart axes.

Adding chart header

The chart header is the title of the chart, which is usually displayed at the top center of the chart area. The following illustration displays the chart header displayed in the chart area.



In general, chart series contains a chart area instance, which can be used to customize the OLAP area. The following code sample describes how to add a chart header to an OLAP chart.

C#

```
this.olapchart1.Series[0].Area.Header = "Simple Report";
```

VB.NET

```
Me.olapchart1.Series(0).Area.Header = "Simple Report"
```

Note: The series will be available only after data is bound to the control. In other words, you can access the series property of the OLAP chart only after the call to `DataBind()` is made.

Area customization

An OLAP area is basically derived from the `ChartArea` class belonging to the base chart WPF control. So, the customization options are available in the chart area. However, 3D charts and multiple chart areas are not supported in an OLAP chart. So, those customizations are not applicable.

The following are the frequently used customization options that are available in the chart area.

- Background
- GridBackground
- Foreground
- FontFamily

- FontSize
- FontStyle
- FontWeight
- BorderBrush
- BorderThickness
- CornerRadius

The OLAP area allows you to customize the border properties. The following code sample explains how these properties can be customized.

BorderBrush

C#

```
this.olapChart.Series[0].Area.BorderBrush = Brushes.Black;
```

VB.NET

```
Me.olapChart.Series(0).Area.BorderBrush = Brushes.Black
```

BorderThickness

C#

```
this.olapChart.Series[0].Area.BorderThickness = new Thickness(2);
```

VB.NET

```
Me.olapChart.Series(0).Area.BorderThickness = New Thickness(2)
```

CornerRadius

C#

```
this.olapChart.Series[0].Area.CornerRadius = new CornerRadius(5);
```

VB.NET

```
Me.olapChart.Series(0).Area.CornerRadius = New CornerRadius(5)
```

Background

C#

```
this.olapChart.Series[0].Area.Background = Brushes.SkyBlue;
```

VB.NET

```
Me.olapChart.Series(0).Area.Background = Brushes.SkyBlue
```

GridBackground

C#

```
this.olapChart.Series[0].Area.GridBackground = Brushes.LightBlue;
```

VB.NET

```
Me.olapChart.Series(0).Area.GridBackground = Brushes.LightBlue
```

*FontStyle***C#**

```
this.olapChart.Series[0].Area.FontStyle = FontStyles.Italic;
```

VB.NET

```
Me.olapChart.Series(0).Area.FontStyle = FontStyles.Italic
```

Legend in WPF Olap Chart

Legends are used to display the name of data series. The chart legend can be added to an OLAP chart by adding the chart legend of the chart WPF, which is found under the Syncfusion.Windows.Chart namespace. The following code sample explains how to add a legend to an OLAP chart.

XML

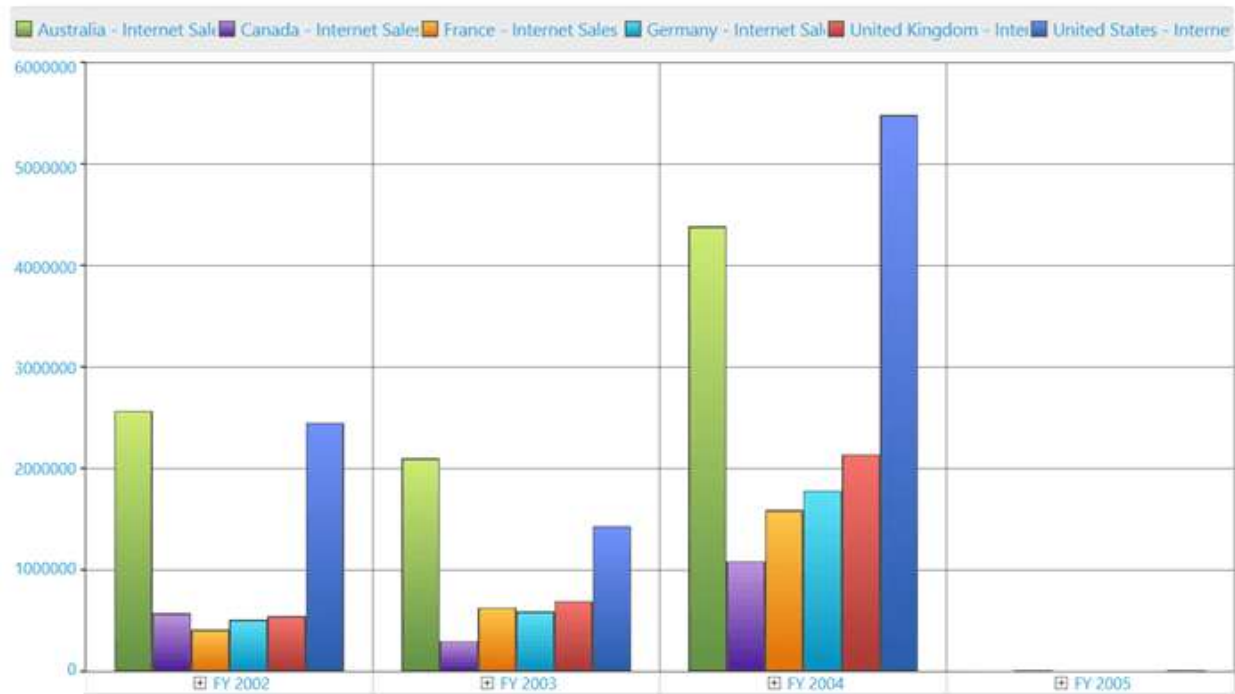
```
<syncfusion:OlapChart.Legend>  
<syncfusion:ChartLegend Background="Transparent"/>  
</syncfusion:OlapChart.Legend>
```

C#

```
this.olapChart.Legend = new ChartLegend();
```

VB.NET

```
Me.olapChart.Legend = New ChartLegend()
```



Show/hide legend

The chart legend has a **Visibility** property using which you can show or hide the chart legend in the OLAP chart. The following code sample shows how you can collapse the visibility of the chart legend.

XML

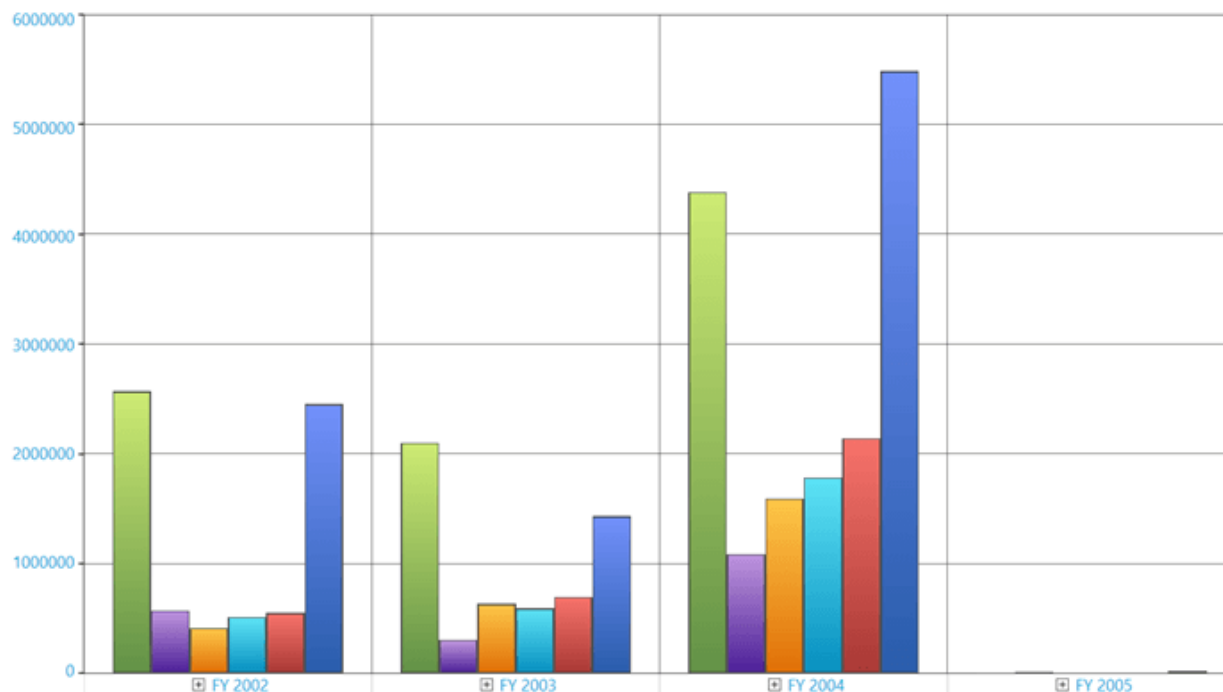
```
<syncfusion:OlapChart.Legend>
<syncfusion:ChartLegend Visibility="Collapsed" />
</syncfusion:OlapChart.Legend>
```

C#

```
this.olapChart.Legend.Visibility = System.Windows.Visibility.Collapsed;
```

VB.NET

```
Me.olapChart.Legend.Visibility = System.Windows.Visibility.Collapsed
```

Visibility customization

The visibility of the legend check box can be toggled by using the `CheckBoxVisibility` property in the chart legend. The following code sample shows how to toggle the visibility of check box in the legend of the OLAP chart.

XML

```
<syncfusion:OlapChart.Legend>
<syncfusion:ChartLegend CheckBoxVisibility="Collapsed" />
</syncfusion:OlapChart.Legend>
```

C#

```
this.olapChart.Legend.CheckBoxVisibility = System.Windows.Visibility.Collapsed;
```

VB.NET

```
Me.olapChart.Legend.CheckBoxVisibility = System.Windows.Visibility.Collapsed
```

The visibility of the legend icon can be toggled by using the `IconVisibility` property in the chart legend. The following code sample shows how to toggle the visibility of icons in the OLAP chart legend.

XML

```
<syncfusion:OlapChart.Legend>
<syncfusion:ChartLegend IconVisibility="Collapsed" />
</syncfusion:OlapChart.Legend>
```

C#

```
this.olapChart.Legend.IconVisibility = System.Windows.Visibility.Collapsed;
```

VB.NET

```
Me.olapChart.Legend.IconVisibility = System.Windows.Visibility.Collapsed
```

Dock position

The chart legend contains an enum property called **ChartDock**, which has the following values **Floating**, **Right**, **Left**, **Top**, and **Bottom**. You can choose the required docking position to dock the legend. The following code sample explains how to set the docking position for the OLAP chart legend.

C#

```
ChartDockPanel.SetDock(this.olapChart.Legend, ChartDock.Right);
```

VB.NET

```
ChartDockPanel.SetDock(Me.olapChart.Legend, ChartDock.Right)
```

Row/column setting

You can use the **RowCount** and **ColumnsCount** property to create the rows or columns of the OLAP chart legend. The **RowCount** and **ColumnsCount** will be used internally to create a grid layout control to place the legends. The following code sample shows how to set the number of rows or columns in an legend.

XML

```
<syncfusion:OlapChart.Legend>
<syncfusion:ChartLegend Background="Transparent"
RowCount="2" ColumnsCount="2" />
</syncfusion:OlapChart.Legend>
```

C#

```
this.olapChart.Legend.RowCount = 2;
this.olapChart.Legend.ColumnsCount = 2;
```

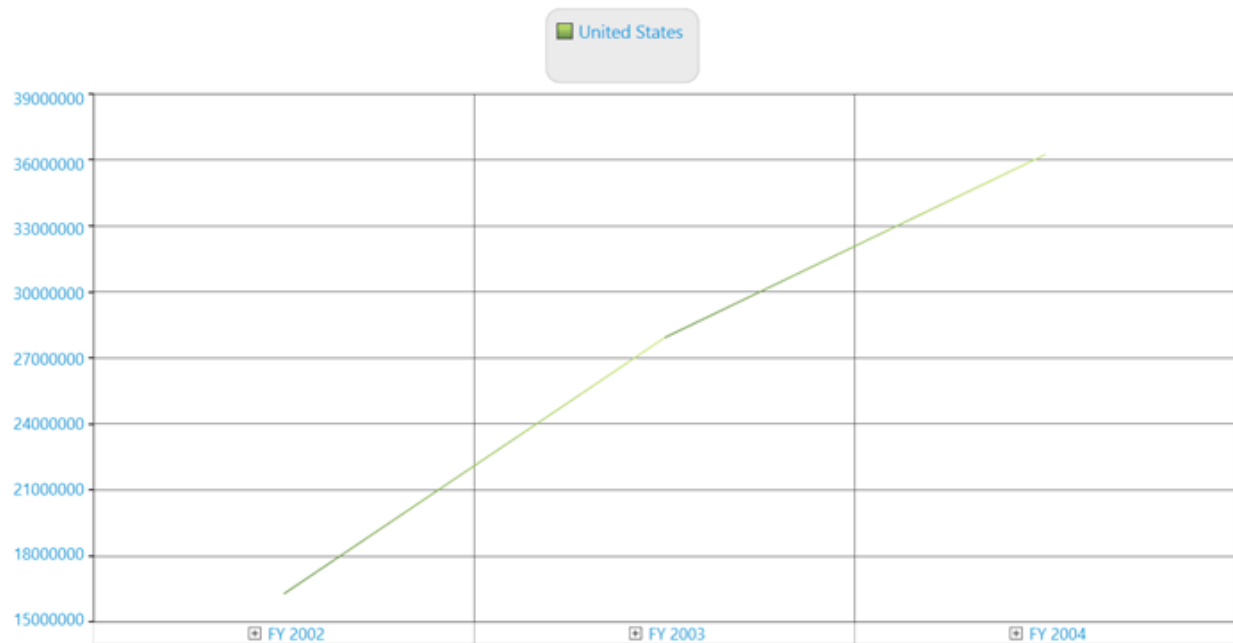
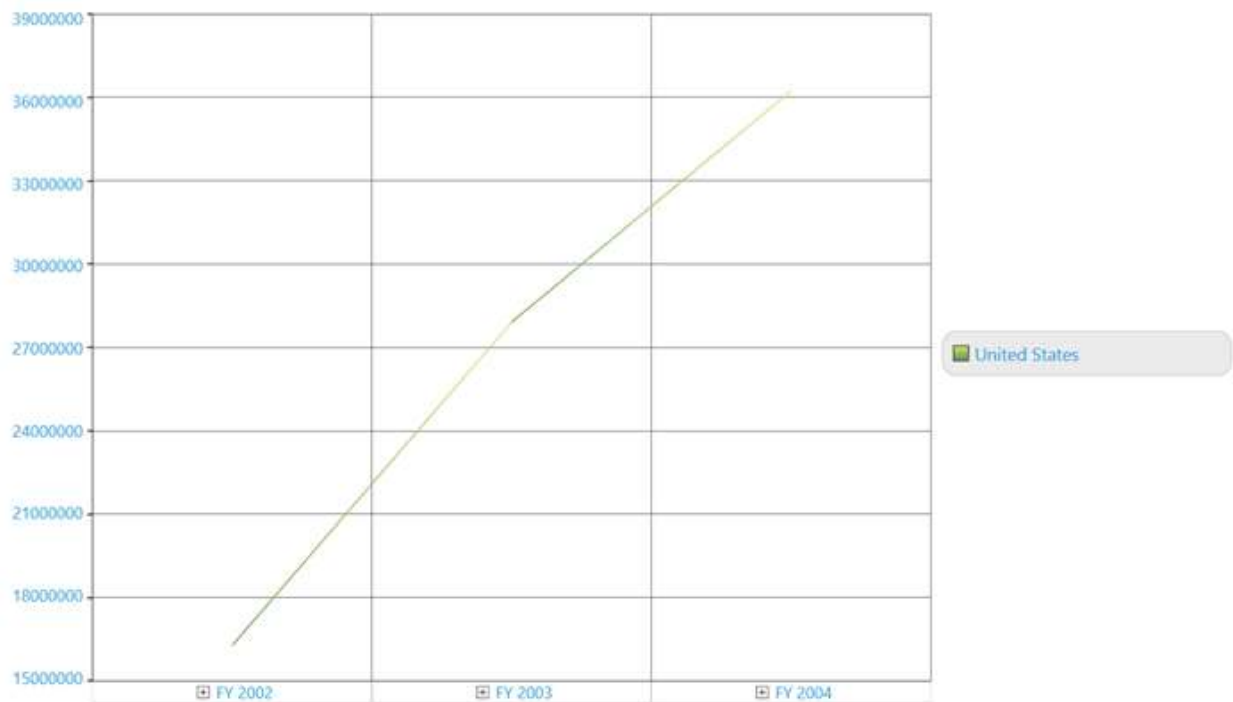
VB.NET

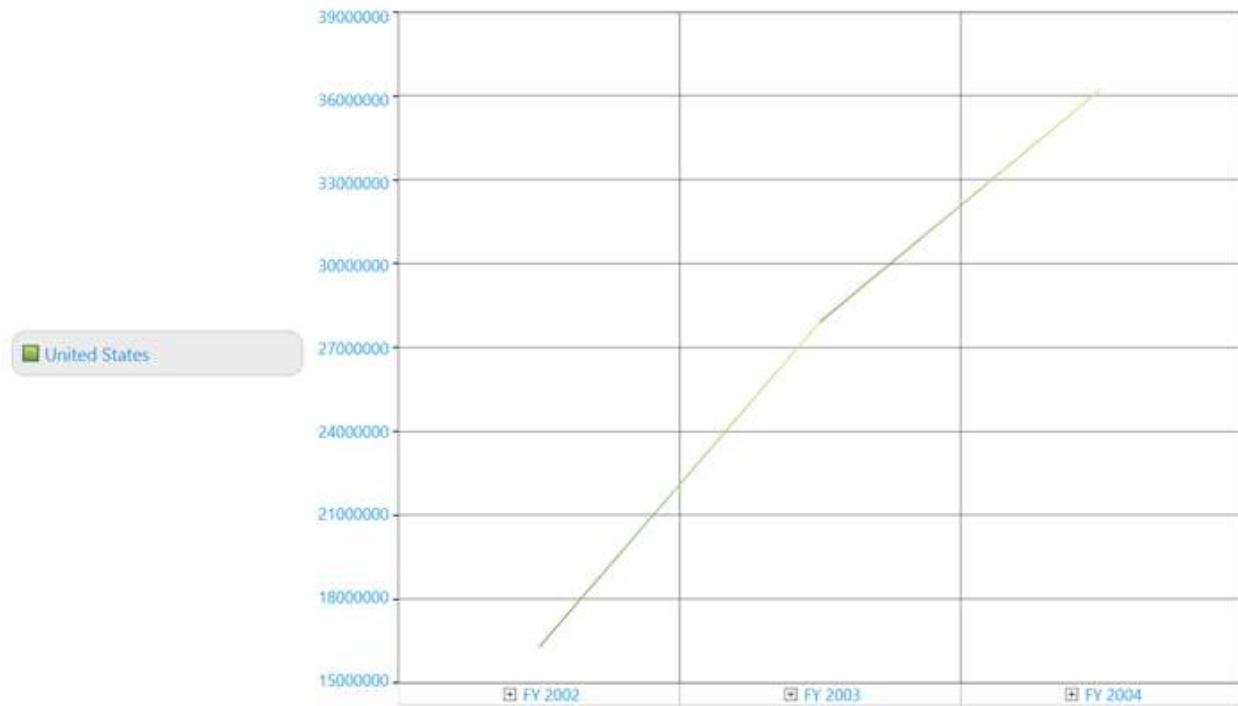
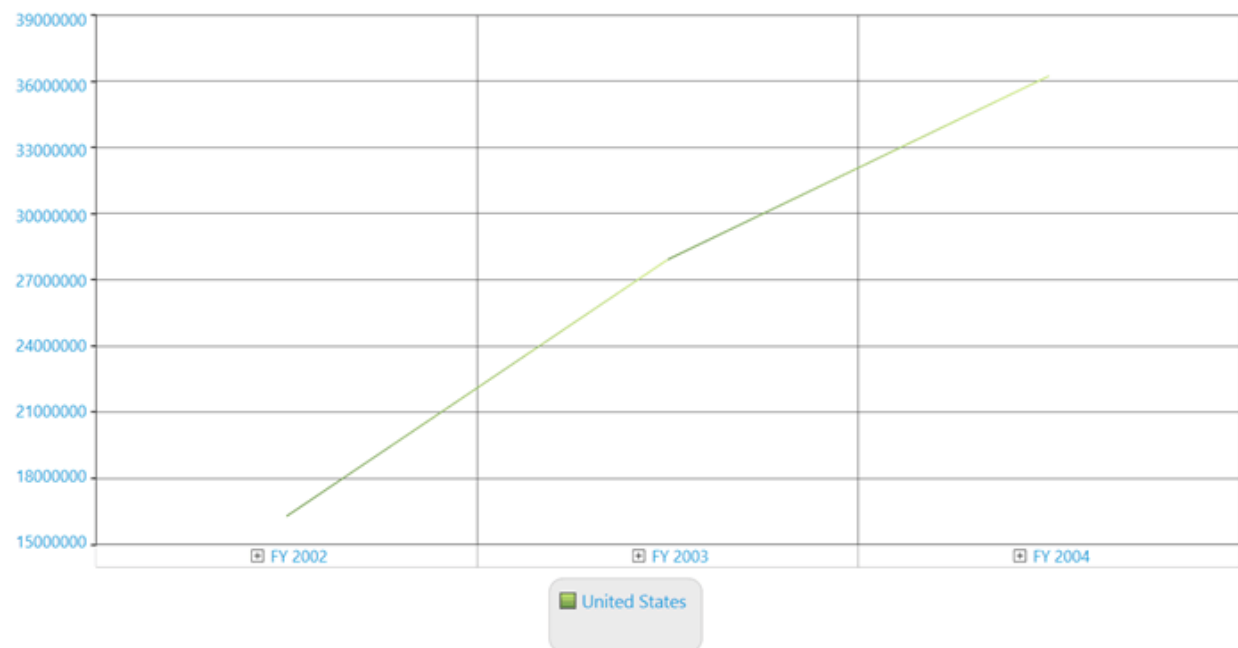
```
Me.olapChart.Legend.RowCount = 2
Me.olapChart.Legend.ColumnsCount = 2
```

Note: The **RowCount** and **ColumnsCount** are used to create the rows and columns in the grid layout control, which is used to place the legends. If you give extra row or column count than the legend availability, it will display empty spaces to fill the structure of the grid. The following illustration explains this in detail.

The following chart has only one legend, but the **RowCount** and **ColumnsCount** are set as 2. Therefore, the resultant legend will appear as follows:

Legend with RowCount=2, ColumnsCount=2 and ChartDock.Top

**Legend with RowsCount=2, ColumnsCount=2 and ChartDock.Right****Legend with RowsCount=2, ColumnsCount=2 and ChartDock.Left**

**Legend with RowsCount=2, ColumnsCount=2 and ChartDock.Bottom**

Series in WPF Olap Chart

Series are the data points plotted in the rectangular co-ordinate system.

Point label

Point label provides information about the data point. Data point can be added to a series by using the following code sample.

C#

```

for(int i=0; i< this.olapChart.Series.Count; i++)
{
    //// Setting the visibility of adornment.
    this.olapChart.Series[i].AdornmentsInfo.Visible = true;
    //// Setting horizontal alignment
    this.olapChart.Series[i].AdornmentsInfo.SegmentHorizontalAlignment = System.
Windows.HorizontalAlignment.Right;
    //// Makes the segment out from the series.
    this.olapChart.Series[i].AdornmentsInfo.SegmentIsOut = true;
    this.olapChart.Series[i].AdornmentsInfo.LabelContentPath = "DataPoint.Y";
    this.olapChart.Series[i].AdornmentsInfo.SegmentLabelFontSize = 12;
    this.olapChart.Series[i].AdornmentsInfo.SegmentLabelRotation = 325;
}

```

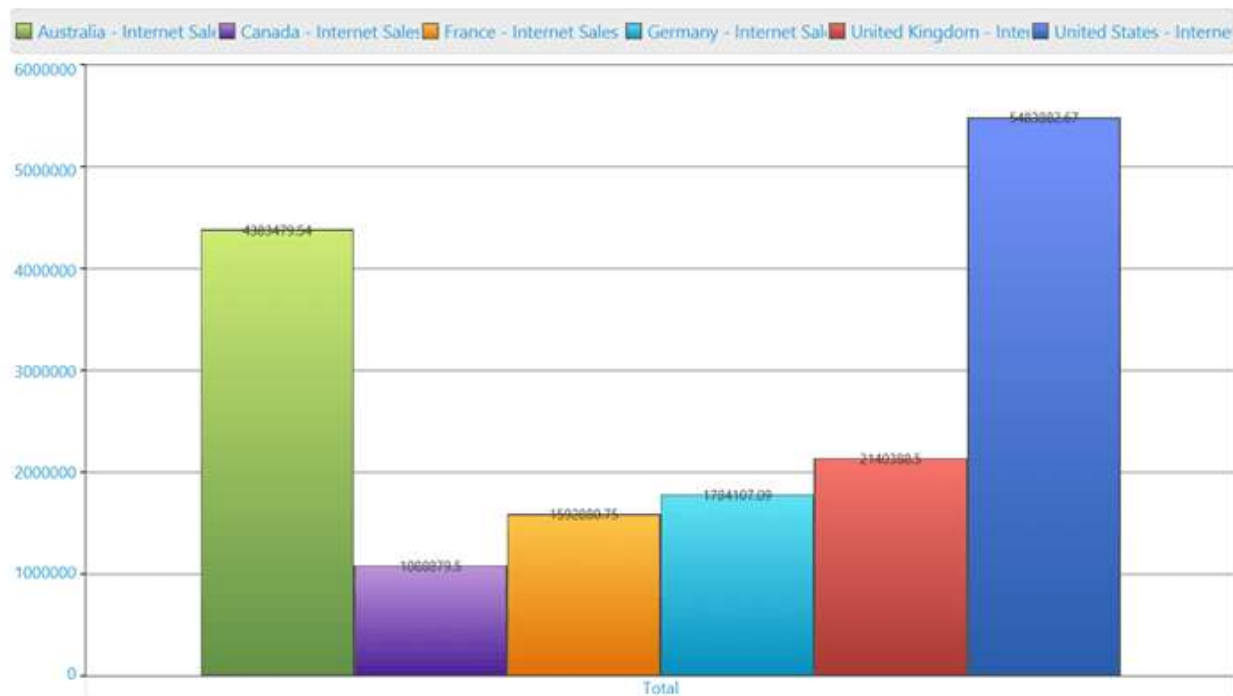
VB.NET

```

For i As Integer = 0 To Me.olapChart.Series.Count - 1
    ' Setting the visibility of adornment.
    Me.olapChart.Series(i).AdornmentsInfo.Visible = True
    ' Setting horizontal alignment
    Me.olapChart.Series(i).AdornmentsInfo.SegmentHorizontalAlignment =
System.Windows.HorizontalAlignment.Right
    ' Makes the segment out from the series.
    Me.olapChart.Series(i).AdornmentsInfo.SegmentIsOut = True
    Me.olapChart.Series(i).AdornmentsInfo.LabelContentPath = "DataPoint.Y"
    Me.olapChart.Series(i).AdornmentsInfo.SegmentLabelFontSize = 12
    Me.olapChart.Series(i).AdornmentsInfo.SegmentLabelRotation = 325
Next i

```

The following screenshot shows an OLAP chart with point labels enabled.



Color customization

You can set a custom color for each series in the OLAP chart. To apply different colors to different series iterate through the series, apply the custom brush to the series.

C#

```
this.olapChart.Series[0].Interior = Brushes.Orange;
```

VB.NET

```
Me.olapChart.Series(0).Interior = Brushes.Orange
```

Border customization

You can customize the thickness of series border of the OLAP chart by using the following code sample.

C#

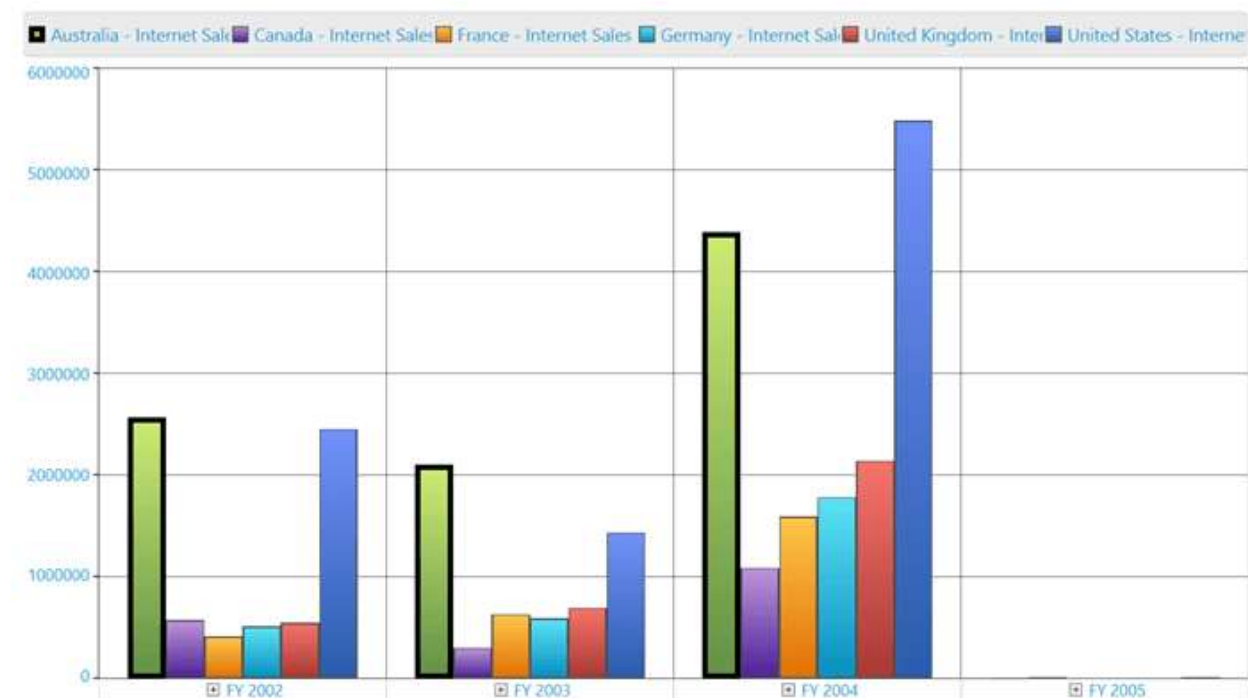
```
this.olapchart.Series[0].Stroke = Brushes.Black;
this.olapChart.Series[0].StrokeThickness = 4;
```

VB.NET

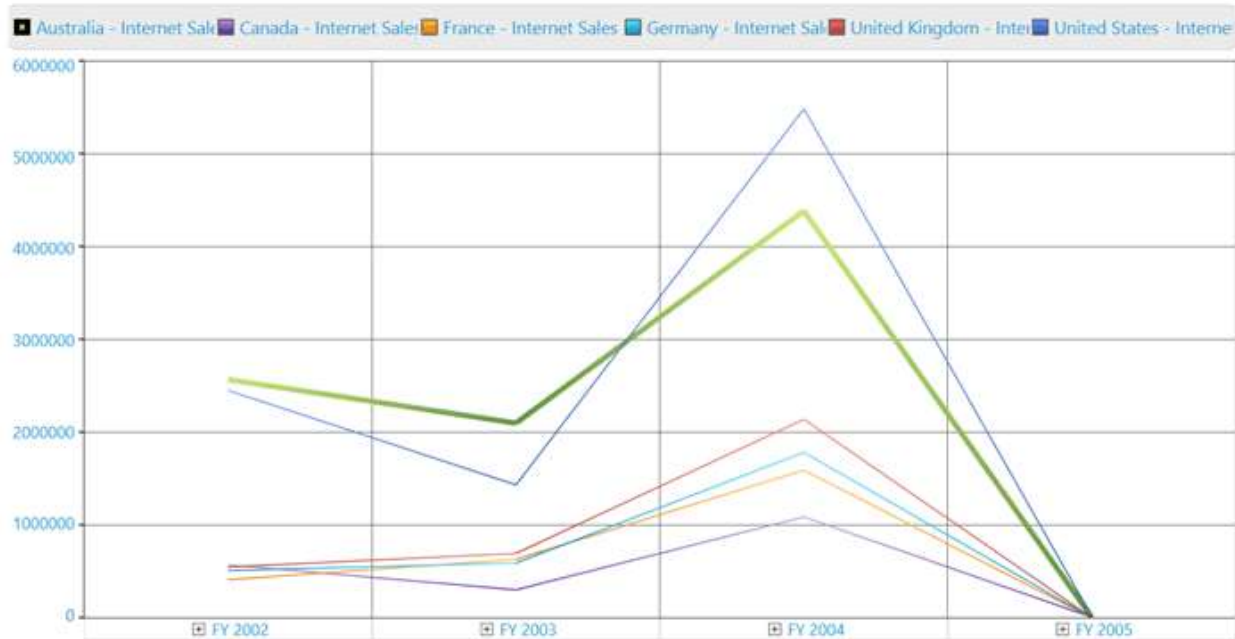
```
Me.olapChart.Series(0).Stroke = Brushes.Black
Me.olapChart.Series(0).StrokeThickness = 4
```

Note: The behavior of the series border varies for different chart types. The following illustration describes them in detail.

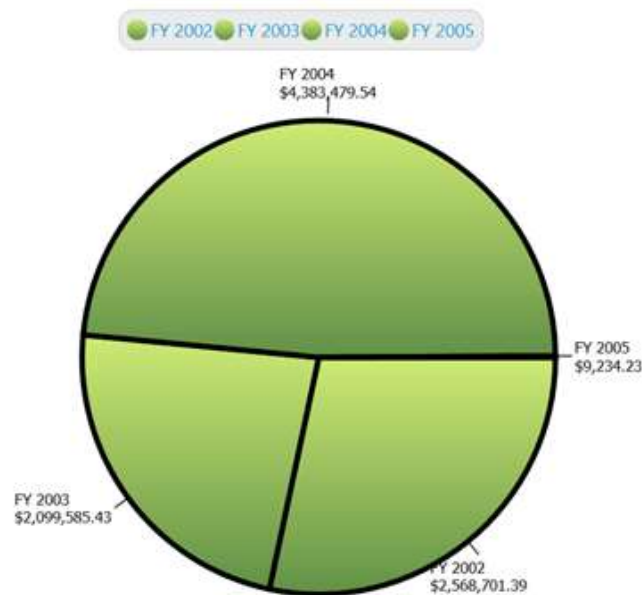
The series border is applied to the first series of a column chart by using the `StrokeThickness` property. Now, the first series element is surrounded with applied border.



Notice the variation. The same series border property is applied to a series in the line chart. Instead of creating a 4 pixel width border it increases the thickness of the particular series line.



The pie chart renders everything in a single series; each block in the pie chart known as segments. The border will be applied on each segments. This is illustrated in the following screenshot.



Custom data templates

Series can be customized with user-defined data templates. The following sample usage describes how to apply a data template to series in the OLAP chart.

The following data template can be used to customize the series.

XML

```
<DataTemplate x:Key="ColumnTemplate">
<Canvas Name="myCanvas">
<Grid Name="OuterGrid" Canvas.Left="{Binding X}" Width="{Binding Width}"
Height="{Binding ElementName=myCanvas, Path=ActualHeight}" >
<Border Name="ColumnRect" VerticalAlignment="Bottom" Width="{Binding Width}"
Height="{Binding Height}"
CornerRadius="8,8,0,0" Background="{Binding Interior}">
</Border>
</Grid>
</Canvas>
</DataTemplate>
```

C#

```
for (int i = 0; i < this.olapchart1.Series.Count; i++)
{
//Apply Series Template to display the series cylindrical style.
this.olapchart1.Series[i].Template = this.Resources["ColumnTemplate"] as DataTemplate;
//Apply Series Interior to display the series in different colors.
this.olapchart1.Series[i].Interior = App.Current.Resources["SeriesInterior"
+ i] as LinearGradientBrush;
}
```

VB.NET

```
For i As Integer = 0 To Me.olapchart1.Series.Count - 1
'Apply Series Template to display the series cylindrical style
Me.olapchart1.Series(i).Template = TryCast(Me.Resources("ColumnTemplate"),
DataTemplate)
'Apply Series Interior to display the series in different colors.
Me.olapchart1.Series(i).Interior =
TryCast(App.Current.Resources("SeriesInterior" & i), LinearGradientBrush)
Next i
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Customization\Series Customization Demo

Event

ChartMouseEventArgs are the arguments returned when the mouse events are triggered by the ChartSeries. **ChartMouseEventArgs** return the segment on which the mouse events are triggered along with default mouse event args. This event args can be used to customize a segment when the mouse event is encountered. The segment returns different values that are used to perform calculations or operations.

The following code sample demonstrates how the **ChartMouseEventArgs** can be used to retrieve information on the ChartSeries segment.

C#

```
///// Event Tagging
```



```

this.olapchart1.Series[0].MouseClicked += new ChartMouseEventHandler(series_MouseClick);
//// Mouse click event for a series.
void series_MouseClick(object sender, ChartMouseEventArgs e)
{
    ChartPoint point = (ChartPoint)e.Segment.CorrespondingPoints[0].DataPoint;
    MessageBox.Show("X = " + point.X.ToString() + "\n" + "Y = " + point.Y.ToString());
}

```

VB.NET

```

' Event Tagging
AddHandler olapchart1.Series(0).MouseClicked, AddressOf series_MouseClick
' Mouse click event for a series.
Private Sub series_MouseClick(ByVal sender As Object, ByVal e As
ChartMouseEventArgs)
    Dim point As ChartPoint = CType(e.Segment.CorrespondingPoints(0).DataPoint,
ChartPoint)
    MessageBox.Show("X = " & point.X.ToString() & Constants.vbLf & "Y = " &
point.Y.ToString())
End Sub

```

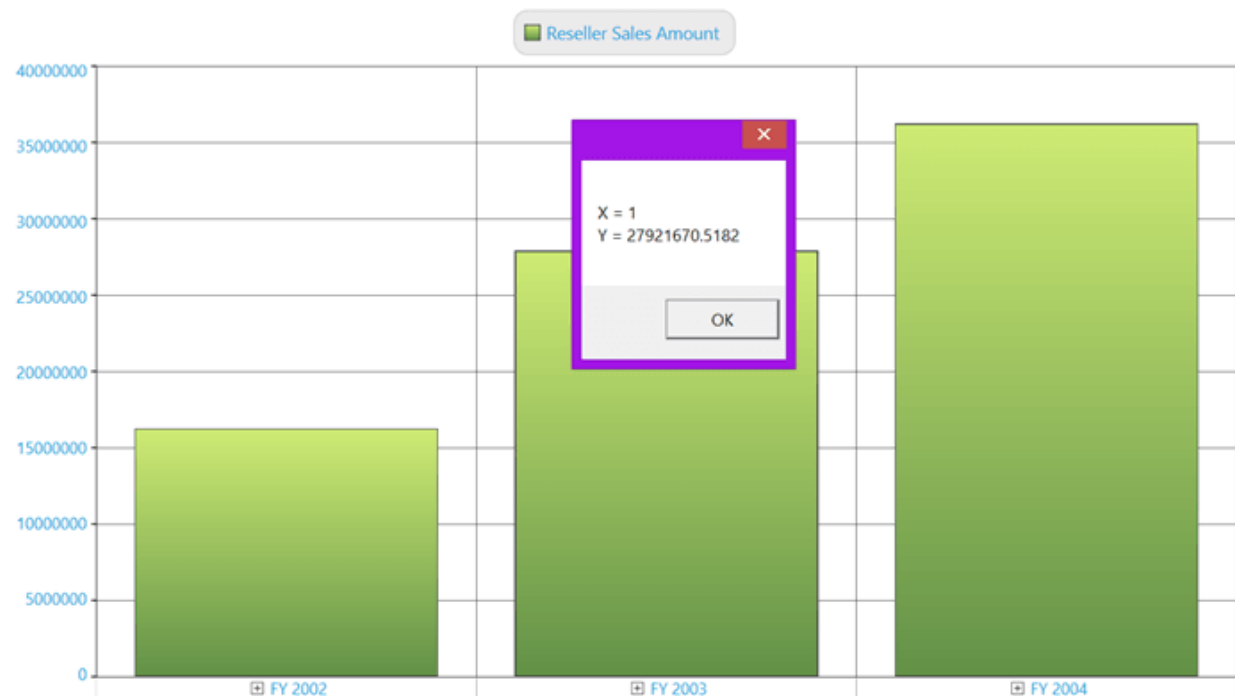


Chart Series

Chart animation

Animations can be added to the OLAP chart control. This can be achieved through the following properties:

- **SeriesAnimateOption:** Gets or sets animation for each series.
- **SeriesAnimateOneByOne:** Gets or sets whether each series animate one by one. By default, it is false.
- **EnableSeriesAnimation:** Gets or sets whether the animation for each series is enabled.
- **SeriesAnimationDuration:** Gets or sets the animation duration for each series.

The following code sample is used for enabling chart animations.

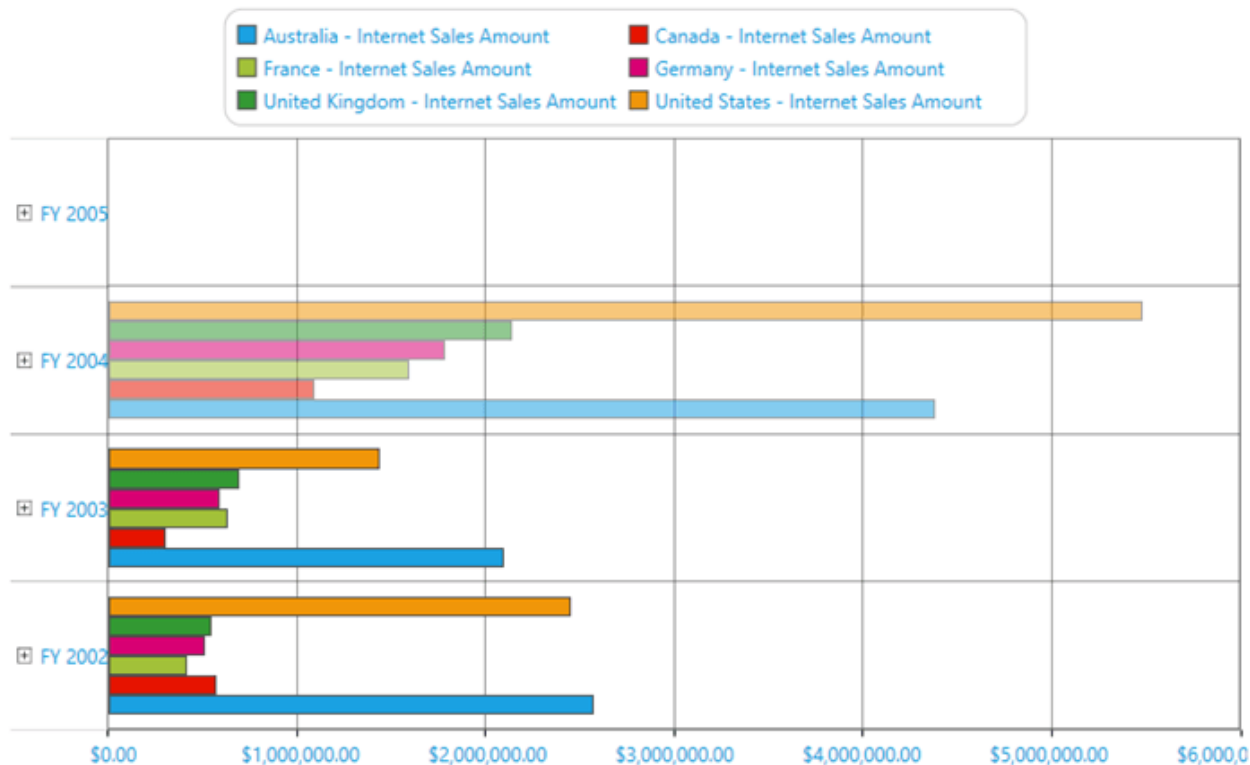
XML

```
<syn:OlapChart ChartType="{Binding OlapChartType}" Name="olapChart"
OlapDataManager="{Binding DataManager}" Grid.Row="1"
Background="Transparent" ColorPalette="Metro"
SeriesAnimateOption="Bottom"
SeriesAnimateOneByOne="True"
EnableSeriesAnimation="True"
SeriesAnimationDuration="00:00:3" />
```

C#

```
// To set the Series Animate Option to Bottom in OlapChart.
this.olapChart.SeriesAnimateOption =
Syncfusion.Windows.Chart.AnimationOptions.Bottom;
// To enable the Series Animate OneByOne in OlapChart.
this.olapChart.SeriesAnimateOneByOne = true;
// To disable Series Animate OneByOne in OlapChart.
this.olapChart.SeriesAnimateOneByOne = false;
// To disable Series Animation in OlapChart.
this.olapChart.EnableSeriesAnimation = false;
// To set the Series Animation Duration in OlapChart.
this.olapChart.SeriesAnimationDuration = new TimeSpan(1);
```

The following illustration shows the chart animations.



A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OLAPChart.WPF\Samples\Appearance\Chart Animations Demo\

Pie chart customization

You can set the explode index and explode radius and enable or disable the series effects for each series in the pie chart.

The following code sample demonstrates the customization of each series in the pie chart.

C#

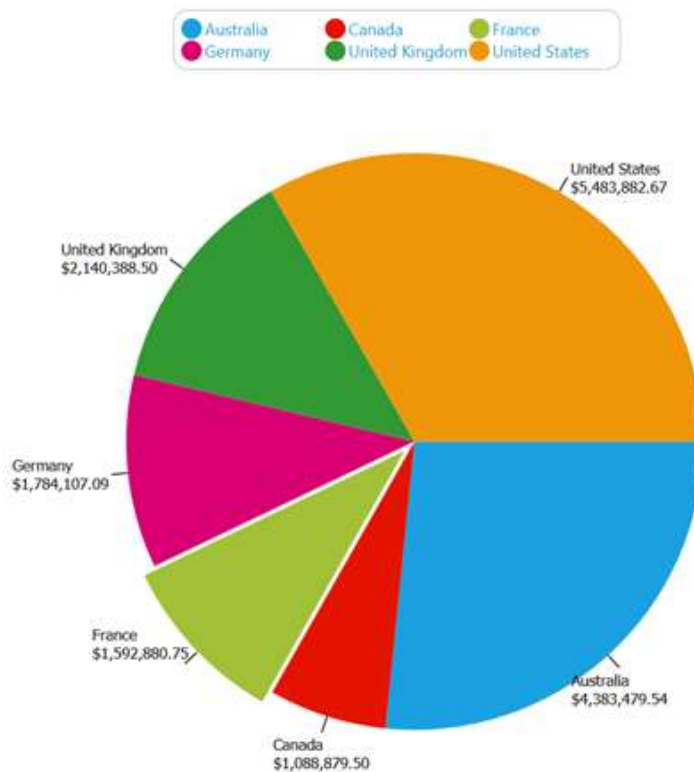
```
foreach (ChartSeries series in this.olapchart.Series)
{
    //To enable the chart to explode we have pass the ChartSeries and true as
    //parameter to SetExplodedAll method
    ChartPieType.SetExplodedAll(series, true);
}
foreach (ChartSeries series in this.olapchart.Series)
{
    //To enable the chart to explode we have pass the ChartSeries and false as
    //parameter to disable the SetExplodedAll method
    ChartPieType.SetExplodedAll(series, false);
}
foreach (ChartSeries series in this.olapchart.Series)
{
    //To enable the effects in Pie chart
    series.EnableEffects = true;
}
foreach (ChartSeries series in this.olapchart.Series)
```

```

{
    //To disable the effects in Pie chart
    series.EnableEffects = false;
}
foreach (ChartSeries series in this. olapchart.Series)
{
    //To set the explore index value we have to pass the ChartSeries and the
    //index value of which part of the Chart to explode
    ChartPieType.SetExplodedIndex(series, 2);
}
foreach (ChartSeries series in this. olapchart.Series)
{
    //To set the radius of the exploded chart we have to pass the ChartSeries
    //and the radius which is a double value
    ChartPieType.SetExplodeRadius(series, 8.0);
}

```

The following illustration shows the customization.



A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OLAPChart.WPF\Samples\Chart Types\Pie Chart Demo

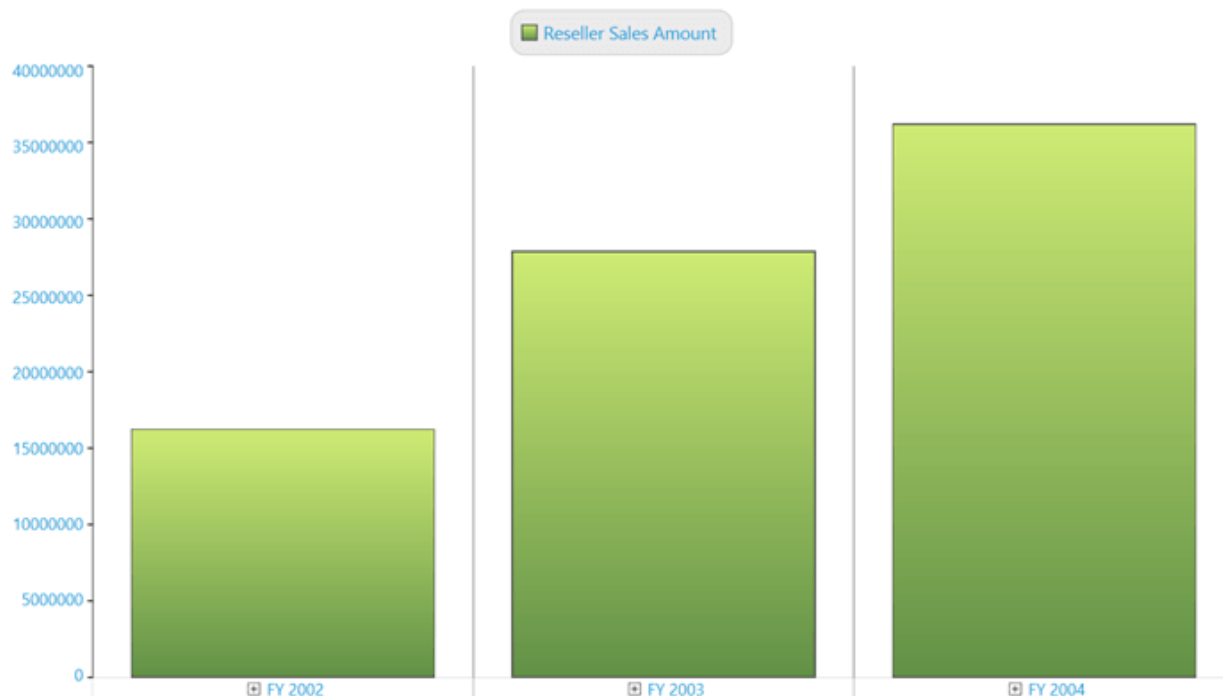
Axes in WPF Olap Chart

An OLAP area contains two axes, the primary axis and secondary axis, in an OLAP chart control. Values or data in the chart are plotted against these axes.

Grid lines customization

In general, for column type charts, the horizontal grid line belongs to the secondary axis. To disable the horizontal grid lines for these types of charts, you should use the `ShowGridLinesProperty` of the secondary axis.

The following illustration describes how the chart will look after the horizontal grid lines are disabled.



The following code sample describes how to disable the horizontal grid lines.

C#

```
this.olapChart.Series[0].Area.SecondaryAxis.SetValue (ChartArea.ShowGridLinesProperty, false);
```

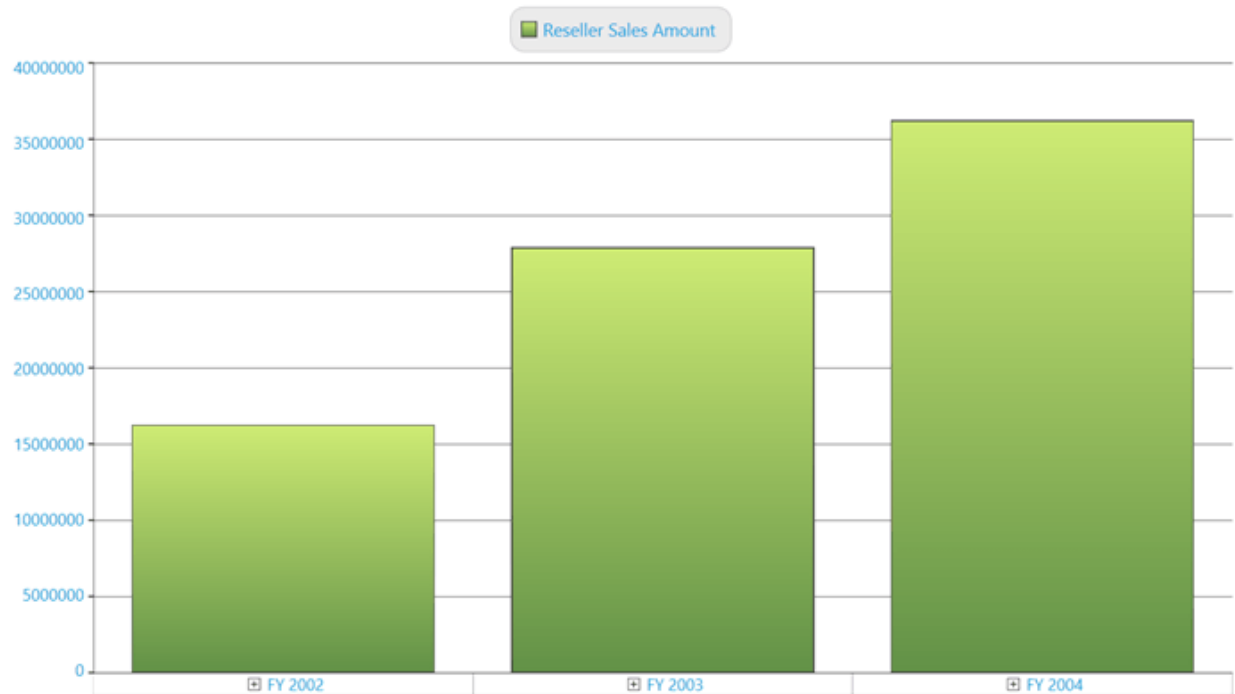
VB.NET

```
Me.olapChart.Series(0).Area.SecondaryAxis.SetValue (ChartArea.ShowGridLinesProperty, False)
```

Note: For bar type charts, such as bar, stacking bar, and stacking 100 bar, you can disable the horizontal grid lines by using the `ShowGridLinesProperty` of the primary axis.

In general, for column type charts, the vertical grid line belongs to the primary axis. To disable the vertical grid lines for these types of charts, you should use the `ShowGridLinesProperty` of the primary axis.

The following illustration describes how the chart will look after the vertical grid lines are disabled.



The following code sample describes how to disable the vertical grid lines.

C#

```
this.olapChart.Series[0].Area.PrimaryAxis.SetValue (ChartArea.ShowGridLinesProperty, false);
```

VB.NET

```
Me.olapChart.Series(0).Area.PrimaryAxis.SetValue (ChartArea.ShowGridLinesProperty, False)
```

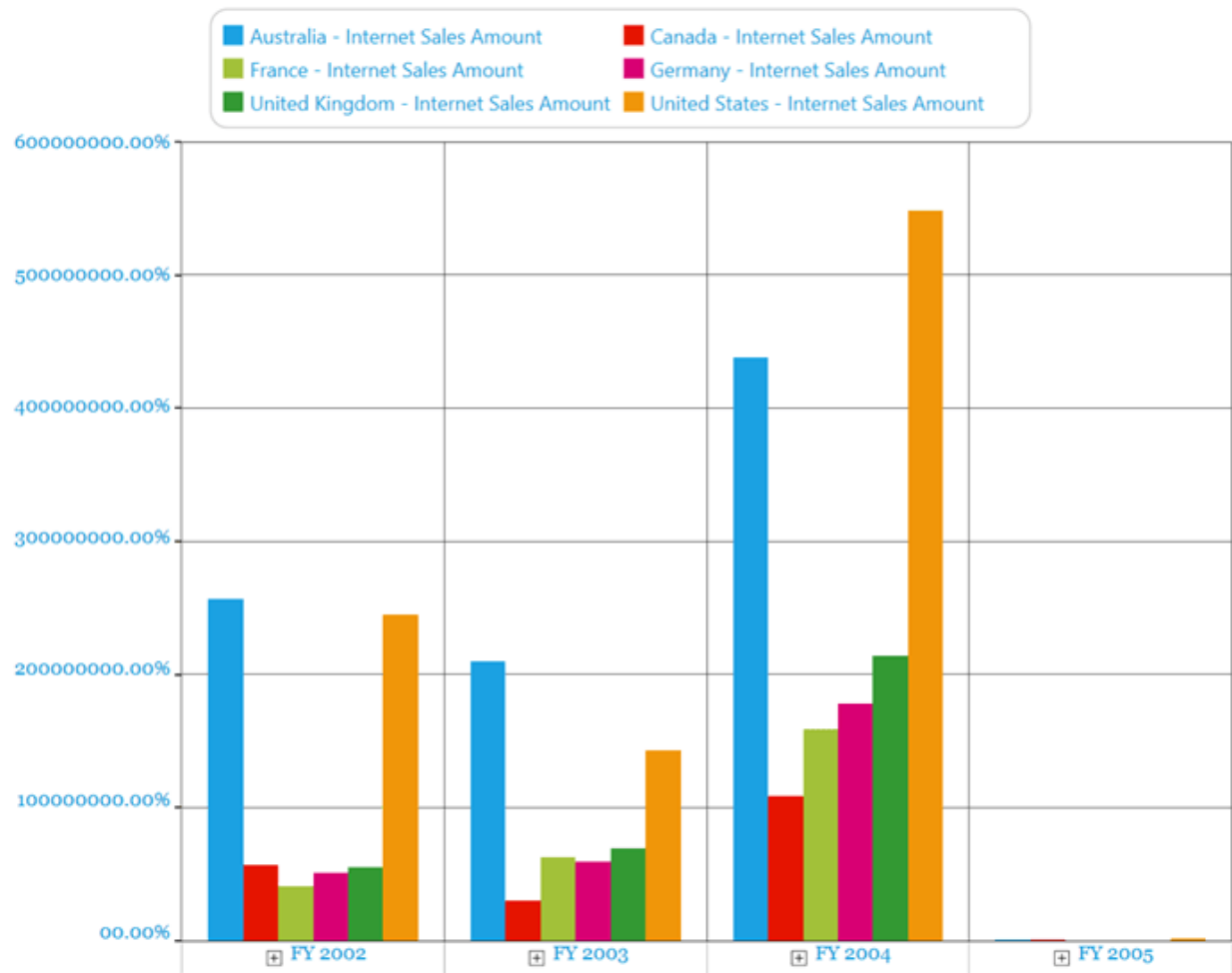
Note: For bar type charts such as bar, stacking bar, and stacking 100 bar, you can disable the vertical grid lines by using the `ShowGridLinesProperty` of the secondary axis.

Format settings

To display the '%' symbol in the secondary axis, you should set the secondary axis label format property. The following code sample describes the usage of '%' in the secondary axis label.

XML

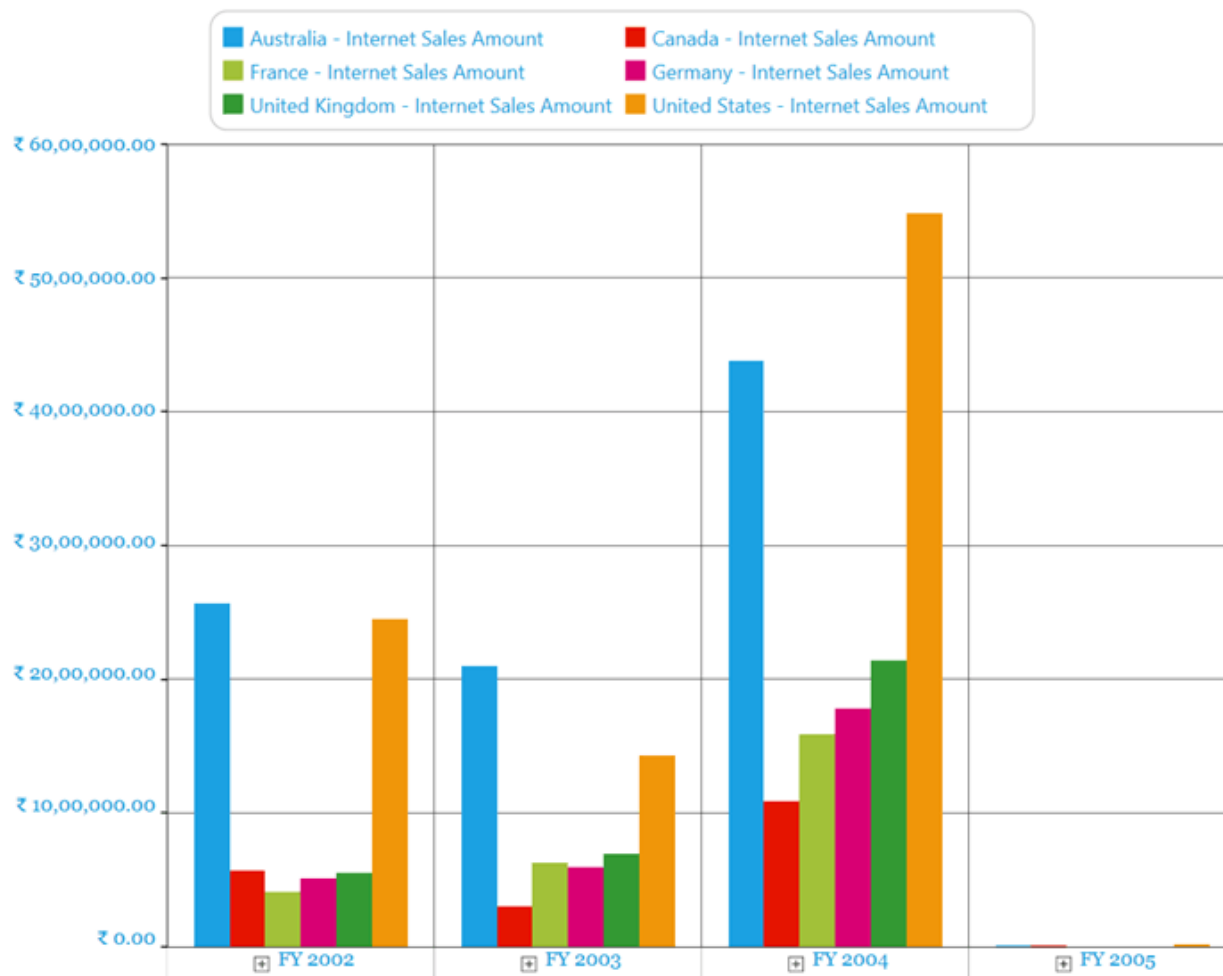
```
<syncfusion:OlapChart.SecondaryAxis>
<syncfusion:ChartAxis LabelFormat="00.00%" />
</syncfusion:OlapChart.SecondaryAxis>
```



To display the currency symbol in the secondary axis, you should set the secondary axis label format property. The following code sample describes the usage of currency in the secondary axis label.

XML

```
<syncfusion:OlapChart.SecondaryAxis>
  <syncfusion:ChartAxis LabelFormat="C" />
</syncfusion:OlapChart.SecondaryAxis>
```



Label font settings

The label font settings of the primary and secondary axes can be easily applied to an OLAP chart by specifying the label font properties, which are available under the primary and secondary axes of the OLAP chart.

XML

```
<syncfusion:OlapChart.PrimaryAxis>
<syncfusion:ChartAxis LabelFormat="C"
LabelFontFamily="Arial"
LabelFontSize="14"
LabelFontWeight="ExtraBold"
LabelForeground="DarkGray" />
</syncfusion:OlapChart.PrimaryAxis>
<syncfusion:OlapChart.SecondaryAxis>
<syncfusion:ChartAxis LabelFormat="C"
LabelFontFamily="Arial"
LabelFontSize="14"
LabelFontWeight="ExtraBold"
LabelForeground="DarkGray" />
</syncfusion:OlapChart.SecondaryAxis>
```


C#

```

this.olapChart.PrimaryAxis.LabelForeground = Brushes.DarkGray;
this.olapChart.PrimaryAxis.LabelFontFamily = new FontFamily("Arial");
this.olapChart.PrimaryAxis.LabelFontSize = 14d;
this.olapChart.PrimaryAxis.LabelFontWeight = FontWeights.ExtraBold;
this.olapChart.SecondaryAxis.LabelForeground = Brushes.DarkGray;
this.olapChart.SecondaryAxis.LabelFontFamily = new FontFamily("Arial");
this.olapChart.SecondaryAxis.LabelFontSize = 14d;
this.olapChart.SecondaryAxis.LabelFontWeight = FontWeights.ExtraBold;

```

VB.NET

```

Me.olapChart.PrimaryAxis.LabelForeground = Brushes.DarkGray
Me.olapChart.PrimaryAxis.LabelFontFamily = New FontFamily("Arial")
Me.olapChart.PrimaryAxis.LabelFontSize = 14R
Me.olapChart.PrimaryAxis.LabelFontWeight = FontWeights.ExtraBold
Me.olapChart.SecondaryAxis.LabelForeground = Brushes.DarkGray
Me.olapChart.SecondaryAxis.LabelFontFamily = New FontFamily("Arial")
Me.olapChart.SecondaryAxis.LabelFontSize = 14R
Me.olapChart.SecondaryAxis.LabelFontWeight = FontWeights.ExtraBold

```

Primary axis label visibility

The primary axis label panel visibility can be toggled by setting the **PrimaryAxisLabelVisibility** property.

XML

```

<syncfusion:OlapChart Name="olapChart" PrimaryAxisLabelVisibility="Collapsed" />

```

C#

```

this.olapChart.PrimaryAxisLabelVisibility = System.Windows.Visibility.Collapsed;

```

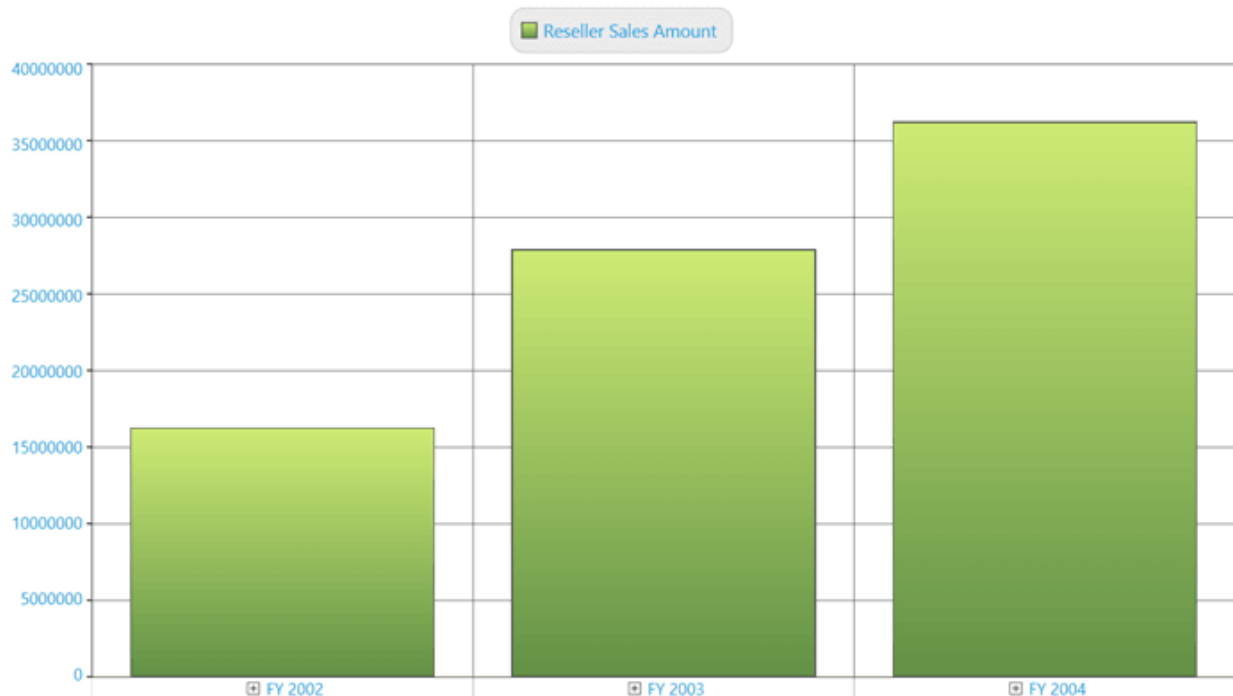
VB.NET

```

Me.olapChart.PrimaryAxisLabelVisibility =
System.Windows.Visibility.Collapsed

```

The following illustration shows how the OLAP chart will look before collapsing the primary axis label.



The following illustration shows how the OLAP chart will look after collapsing the primary axis label.



Paging in WPF Olap Chart

Paging in the OLAP chart supports you to load and render the large amount of data without any performance constraint.

A OLAP pager (user control) is included and bound with the `OlapDataManager` object of the OLAP chart. To enable paging, set the `EnablePaging` property to true.

When you process the large `CellSet`, it is split into several number of segments and each segment is assigned and rendered in a separate page. You can navigate back and forth in all possible ways by using

the UI options in the OLAP pager. You can also change the page size and other pager settings at runtime by using the **PageSetting** window.

Include the following Syncfusion assembly from the installed location to add the OLAP pager (User Control) with OLAP chart.

- Syncfusion.OlapShared.Wpf

Note: You can also get the assemblies by browsing to the default assembly location: {System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<framework version>\

Enable paging through XAML

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:pager="clr-
namespace:Syncfusion.Windows.Shared.Olap;assembly=Syncfusion.OlapShared.WPF"
x:Class="SampleApplication.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<GroupBox Header="OlapChart" Grid.Row="0">
<syncfusion:OlapChart Name="olapChart" Background="Transparent"
SeriesStrokeThickness="0"></syncfusion:OlapChart>
</GroupBox>
<GroupBox Grid.Row="1" Header="OlapPager" Margin="5" >
<pager:OlapPager x:Name="olapPager" ></pager:OlapPager>
</GroupBox>
</Grid>
</Window>
```

Enable paging through report

C#

```
using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
namespace SampleApplication
{
public partial class MainWindow : SampleWindow
{
private string _connectionString;
private OlapDataManager _olapDataManager;
public MainWindow()
{
InitializeComponent();
_connectionString = "Enter a valid connection string";
//Created connection string is passed to OlapDataManager as argument
_olapDataManager = new OlapDataManager(_connectionString);
```

```

//Created OlapReport is set as a current report to OlapDataManager
_olapDataManager.SetCurrentReport(SimpleDimensions());
//Finally OlapChart control gets the data from the created OlapDataManager
this.olapChart.OlapDataManager = _olapDataManager;
this.olapChart.DataBind();
}
private OlapReport SimpleDimensions()
{
    OlapReport olapReport = new OlapReport();
    olapReport.CurrentCubeName = "Adventure Works";
    olapReport.EnablePaging = true;
    olapReport.PagerOptions.CategoricalPageSize = 10;
    olapReport.PagerOptions.SeriesPageSize = 10;
    DimensionElement dimensionElement = new DimensionElement() { Name =
    "Customer", HierarchyName = "Customer" };
    dimensionElement.AddLevel("Customer Geography", "Country");
    olapReport.CategoricalElements.Add(dimensionElement);
    MeasureElements measureElements = new MeasureElements();
    measureElements.Add(new MeasureElement { Name = "Internet Sales Amount" });
    olapReport.SeriesElements.Add(measureElements);
    dimensionElement = new DimensionElement() { Name = "Geography",
    HierarchyName = "Geography" };
    dimensionElement.AddLevel("Geography", "Country");
    olapReport.CategoricalElements.Add(dimensionElement);
    dimensionElement = new DimensionElement() { Name = "Date" };
    dimensionElement.AddLevel("Fiscal", "Fiscal Year");
    olapReport.SeriesElements.Add(dimensionElement);
    return olapReport;
}
}
}

```

VB.NET

```

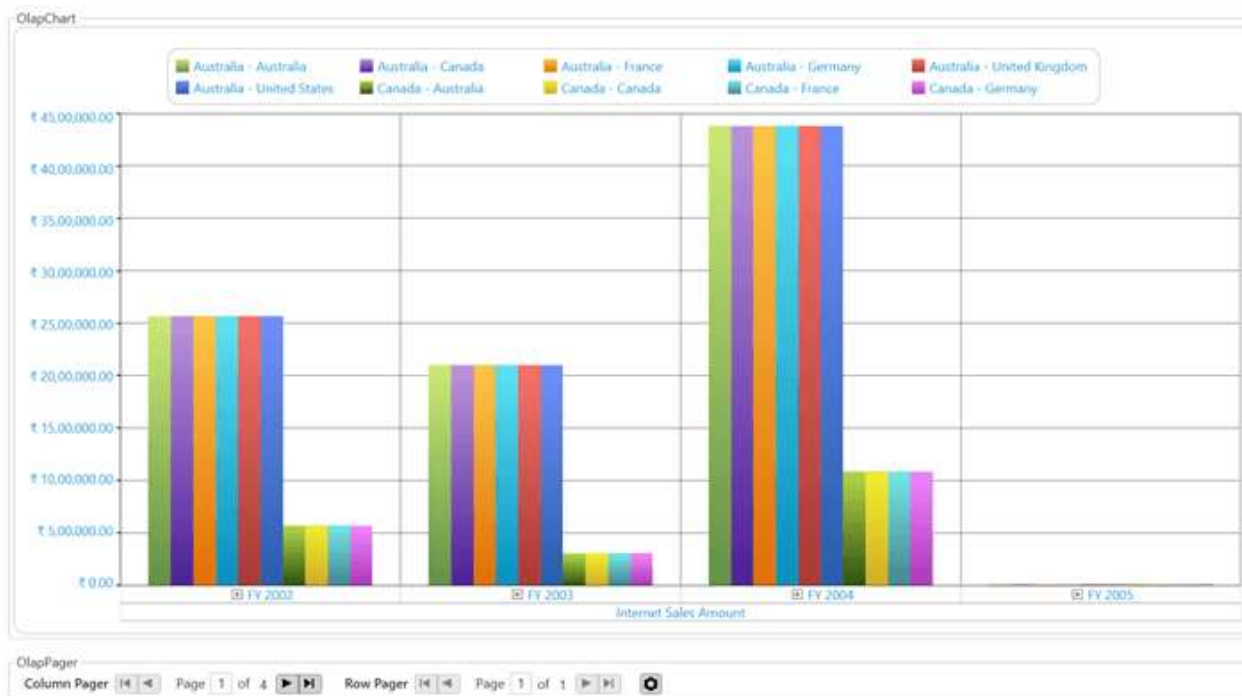
Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Namespace SampleApplication
Partial Public Class MainWindow
Inherits SampleWindow
Private _connectionString As String
Private _olapDataManager As OlapDataManager
Public Sub New()
InitializeComponent()
_connectionString = "Enter a valid connection string"
'Created connection string is passed to OlapDataManager as argument
_olapDataManager = New OlapDataManager(_connectionString)
'Created OlapReport is set as a current report to OlapDataManager
_olapDataManager.SetCurrentReport(SimpleDimensions())
'Finally OlapChart control gets the data from the created OlapDataManager
Me.olapChart.OlapDataManager = _olapDataManager
Me.olapChart.DataBind()
End Sub
Private Function SimpleDimensions() As OlapReport
Dim olapReport As New OlapReport()
olapReport.CurrentCubeName = "Adventure Works"
olapReport.EnablePaging = True

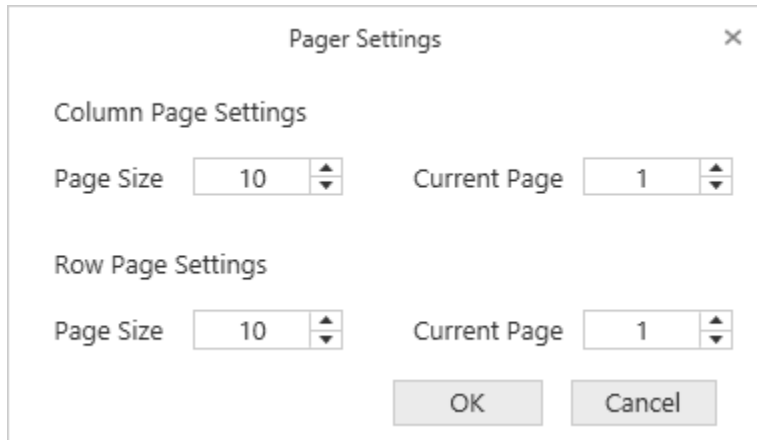
```

```

olapReport.PagerOptions.CategoricalPageSize = 10
olapReport.PagerOptions.SeriesPageSize = 10
Dim dimensionElement As New DimensionElement() With {.Name = "Customer",
.HierarchyName = "Customer"}
dimensionElement.AddLevel("Customer Geography", "Country")
olapReport.CategoricalElements.Add(dimensionElement)
Dim measureElements As New MeasureElements()
measureElements.Add(New MeasureElement With {.Name = "Internet Sales
Amount"})
olapReport.SeriesElements.Add(measureElements)
dimensionElement = New DimensionElement() With {.Name = "Geography",
.HierarchyName = "Geography"}
dimensionElement.AddLevel("Geography", "Country")
olapReport.CategoricalElements.Add(dimensionElement)
dimensionElement = New DimensionElement() With {.Name = "Date"}
dimensionElement.AddLevel("Fiscal", "Fiscal Year")
olapReport.SeriesElements.Add(dimensionElement)
Return olapReport
End Function
End Class
End Namespace

```





The image shows a 'Pager Settings' dialog box with a close button (X) in the top right corner. It is divided into two sections: 'Column Page Settings' and 'Row Page Settings'. Each section contains a 'Page Size' spinner set to 10 and a 'Current Page' spinner set to 1. At the bottom are 'OK' and 'Cancel' buttons.

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Paging

Zooming and Scrolling in WPF Olap Chart

The OLAP chart for WPF allows you to zoom in to a narrower range within the OLAP chart.

In the zooming mode, a zooming toolkit is displayed at the top-left corner of the OLAP chart. Using the buttons in the zooming toolkit, ChartSeries can be zoomed in, out, reset, or closed.



The visibility of the zooming toolkit or individual buttons in the toolkit can be controlled by using the following properties:

Property

- **ZoomInButtonVisibility:** Gets or sets the zoom in button visibility.
- **ZoomOutButtonVisibility:** Gets or sets the zoom out button visibility.
- **ZoomCloseButtonVisibility:** Gets or sets the zoom close button visibility.
- **ZoomResetButtonVisibility:** Gets or sets the zoom reset button visibility.

The following code sample illustrates the above settings.

XML

```
<syncfusion:OlapChart Name="olapChart"
syncfusion:ChartZoomingToolkit.ZoomInButtonVisibility="{Binding IsChecked,
ElementName=cbxZoomIn, Converter={StaticResource boolToVisibilityConverter}}"
"
syncfusion:ChartZoomingToolkit.ZoomOutButtonVisibility="{Binding IsChecked,
ElementName=cbxZoomOut, Converter={StaticResource boolToVisibilityConverter
}}"
syncfusion:ChartZoomingToolkit.ZoomCloseButtonVisibility="{Binding IsChecked
,
ElementName=cbxZoomClose, Converter={StaticResource boolToVisibilityConverte
r }}"
syncfusion:ChartZoomingToolkit.ZoomResetButtonVisibility="{Binding IsChecked
,
ElementName=cbxZoomReset, Converter={StaticResource boolToVisibilityConverte
r }}">
</syncfusion:OlapChart>
```

C#

```
ChartZoomingToolkit.SetZoomInButtonVisibility(olapChart, Visibility.Collapse
d);
ChartZoomingToolkit.SetZoomOutButtonVisibility(olapChart, Visibility.Hidden)
;
ChartZoomingToolkit.SetZoomResetButtonVisibility(olapChart, Visibility.Colla
psed);
ChartZoomingToolkit.SetZoomingToolkitVisibility(olapChart, Visibility.Visibl
e);
```

VB.NET

```
ChartZoomingToolkit.SetZoomInButtonVisibility(olapChart, Visibility.Collapse
d)
ChartZoomingToolkit.SetZoomOutButtonVisibility(olapChart, Visibility.Hidden)
ChartZoomingToolkit.SetZoomResetButtonVisibility(olapChart, Visibility.Colla
psed)
ChartZoomingToolkit.SetZoomingToolkitVisibility(olapChart, Visibility.Visibl
e)
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version
Number>\WPF\OlapChart.WPF\Samples\Zooming and Scrolling\Zooming and Scrolling Demo

Palette in WPF Olap Chart

The chart palette is a pre-defined collection of a set of colors that can be applied to a chart series. The OLAP chart supports 23 chart palettes, which is used to provide rich look for your business applications.

The following are the available palettes in the OLAP chart control:

- Default
- DefaultAlpha
- EarthTone
- Analog
- Colorful
- Nature
- Pastel
- Triad
- WarmCold
- Grayscale
- Office2007Blue
- Office2007Black
- Office2007Silver
- Gradient
- Grayscale
- BlueScale
- MaroonRed
- GreenScale
- MixedViolet
- CoolBlueScale
- ChocolateOrange
- MixedFantasy
- Custom

The following code sample shows how to apply a palette to the OLAP chart.

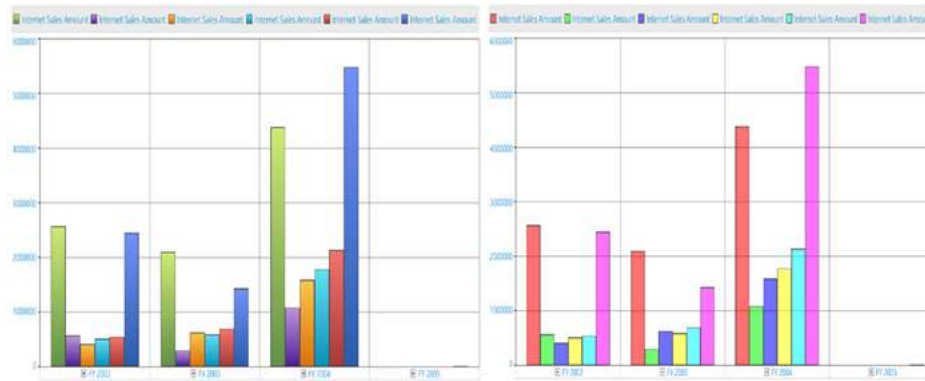
C#

```
this.olapChart.ColorModel.Palette =  
(Syncfusion.Windows.Chart.ChartColorPalette)Enum.Parse(typeof(Syncfusion.Windows.Chart.ChartColorPalette), "EarthTone");
```

VB.NET

```
Me.olapChart.ColorModel.Palette =  
CType(System.Enum.Parse(GetType(Syncfusion.Windows.Chart.ChartColorPalette),  
"EarthTone"), Syncfusion.Windows.Chart.ChartColorPalette)
```

The following image shows before and after applying the *EarthTone* palette.



Custom palette

Custom palettes can be applied to the OLAP chart by setting the “Interior” property with custom brush to each series in the OLAP chart.

A sample demo is available at the following location.

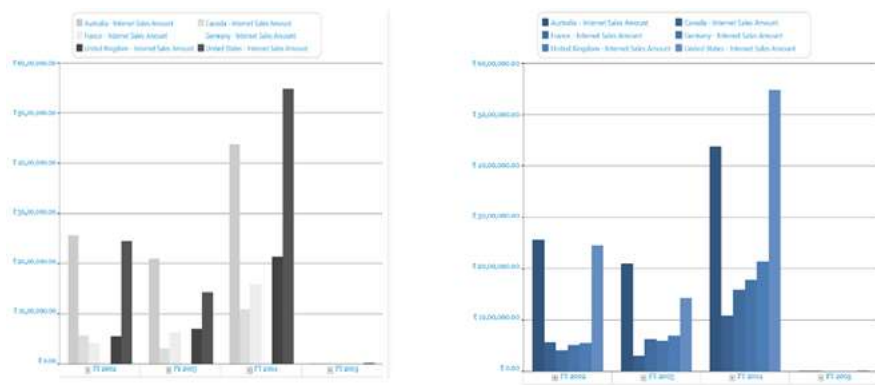
{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Customization\Series Customization Demo

Excel-like palette

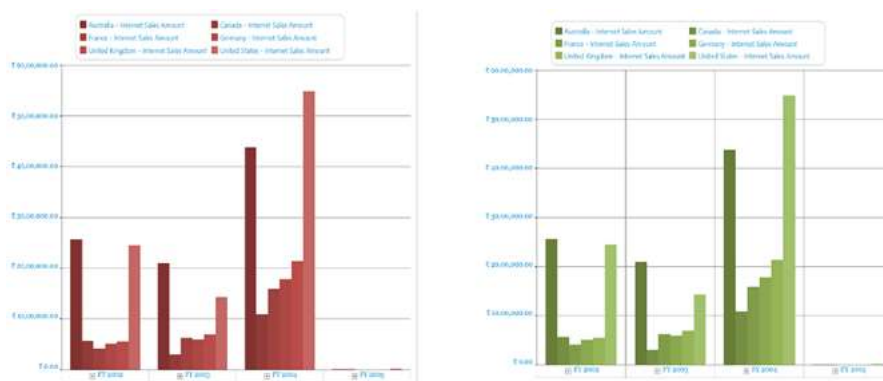
Excel-like palettes are used to display the OLAP chart in business applications.

The following are the available types of Excel-like palettes:

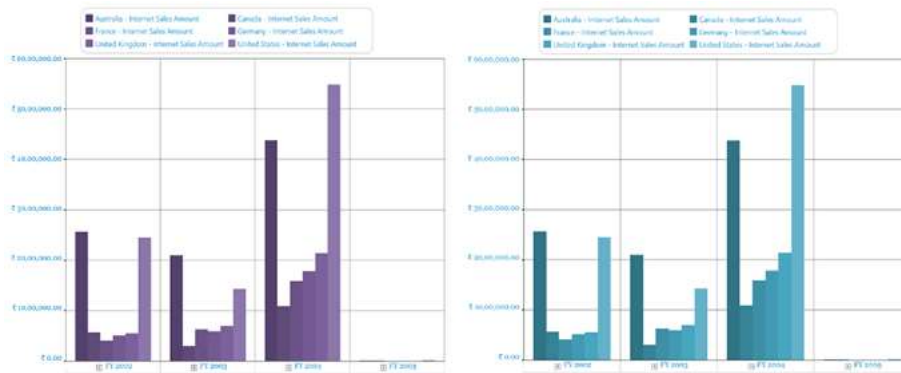
GrayScale, BlueScale



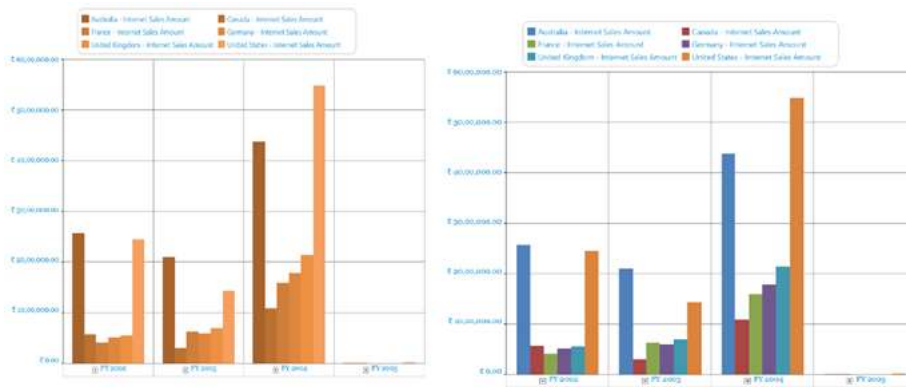
MaroonRed, GreenScale



MixedViolet, CoolBlueScale



ChocolateOrange, MixedFantasy



The following code sample shows how to apply Excel-like palettes.

Grayscale

C#

```
this.olapChart.ColorModel.Palette =
(Syncfusion.Windows.Chart.ChartColorPalette)Enum.Parse(typeof(Syncfusion.Windows.Chart.ChartColorPalette), "Grayscale");
```

VB.NET

```
Me.olapChart.ColorModel.Palette =
CType(System.Enum.Parse(GetType(Syncfusion.Windows.Chart.ChartColorPalette), "Grayscale"), Syncfusion.Windows.Chart.ChartColorPalette)
```

BlueScale

C#

```
this.olapChart.ColorModel.Palette =
(Syncfusion.Windows.Chart.ChartColorPalette)Enum.Parse(typeof(Syncfusion.Windows.Chart.ChartColorPalette), "BlueScale");
```

VB.NET

```
Me.olapChart.ColorModel.Palette =  
CType(System.Enum.Parse(GetType(Syncfusion.Windows.Chart.ChartColorPalette),  
"BlueScale"), Syncfusion.Windows.Chart.ChartColorPalette)
```

MaroonRed

C#

```
this.olapChart.ColorModel.Palette =  
(Syncfusion.Windows.Chart.ChartColorPalette)Enum.Parse(typeof(Syncfusion.Windows.Chart.ChartColorPalette), "MaroonRed");
```

VB.NET

```
Me.olapChart.ColorModel.Palette =  
CType(System.Enum.Parse(GetType(Syncfusion.Windows.Chart.ChartColorPalette),  
"MaroonRed"), Syncfusion.Windows.Chart.ChartColorPalette)
```

GreenScale

C#

```
this.olapChart.ColorModel.Palette =  
(Syncfusion.Windows.Chart.ChartColorPalette)Enum.Parse(typeof(Syncfusion.Windows.Chart.ChartColorPalette), "GreenScale");
```

VB.NET

```
Me.olapChart.ColorModel.Palette =  
CType(System.Enum.Parse(GetType(Syncfusion.Windows.Chart.ChartColorPalette),  
"GreenScale"), Syncfusion.Windows.Chart.ChartColorPalette)
```

MixedViolet

C#

```
this.olapChart.ColorModel.Palette =  
(Syncfusion.Windows.Chart.ChartColorPalette)Enum.Parse(typeof(Syncfusion.Windows.Chart.ChartColorPalette), "MixedViolet");
```

VB.NET

```
Me.olapChart.ColorModel.Palette =  
CType(System.Enum.Parse(GetType(Syncfusion.Windows.Chart.ChartColorPalette),  
"MixedViolet"), Syncfusion.Windows.Chart.ChartColorPalette)
```

CoolBlueScale

C#

```
this.olapChart.ColorModel.Palette =  
(Syncfusion.Windows.Chart.ChartColorPalette)Enum.Parse(typeof(Syncfusion.Windows.Chart.ChartColorPalette), "CoolBlueScale");
```

VB.NET

```
Me.olapChart.ColorModel.Palette =
CType(System.Enum.Parse(GetType(Syncfusion.Windows.Chart.ChartColorPalette),
"CoolBlueScale"), Syncfusion.Windows.Chart.ChartColorPalette)
```

ChocolateOrange

C#

```
this.olapChart.ColorModel.Palette =
(Syncfusion.Windows.Chart.ChartColorPalette)Enum.Parse(typeof(Syncfusion.Win
dows.Chart.ChartColorPalette), "ChocolateOrange");
```

VB.NET

```
Me.olapChart.ColorModel.Palette =
CType(System.Enum.Parse(GetType(Syncfusion.Windows.Chart.ChartColorPalette),
"ChocolateOrange"), Syncfusion.Windows.Chart.ChartColorPalette)
```

MixedFantasy

C#

```
this.olapChart.ColorModel.Palette =
(Syncfusion.Windows.Chart.ChartColorPalette)Enum.Parse(typeof(Syncfusion.Win
dows.Chart.ChartColorPalette), "MixedFantasy");
```

VB.NET

```
Me.olapChart.ColorModel.Palette =
CType(System.Enum.Parse(GetType(Syncfusion.Windows.Chart.ChartColorPalette),
"MixedFantasy"), Syncfusion.Windows.Chart.ChartColorPalette)
```

Appearance in WPF Olap Chart

The OLAP chart supports customizing the appearance of charts. You can customize the chart style, legend style, border and background style, point label style, and label style of the primary and secondary axes.

Chart style and legends

The OLAP chart provides options to set the chart type, chart color, chart legend position, chart legend, and legend check box visibility:

- **ChartType:** Sets the chart type for the OLAP chart control.
- **ColorModel.Palette:** Specifies the chart color for the OLAP chart control.
- **Legend.Visibility:** Specifies the visibility of the chart legend.
- **Legend.CheckBoxVisibility:** Specifies the visibility of the chart legend check box.
- **ChartDockPanel.SetDock:** Specifies the position of the chart legend.

The following code sample illustrates how to customize the chart style and legends.

C#

```
// Set the Chart Type.
this.olapChart.ChartType = ChartTypes.Column;
```

```
// Set the Chart Series Color.
this.olapChart.ColorModel.Palette = ChartColorPalette.Colorful;
// Set the Chart Legend and Legend Check Box Visibility.
this.olapChart.Legend.Visibility = Visibility.Visible;
this.olapChart.Legend.Visibility = Visibility.Collapsed;
this.olapChart.Legend.CheckBoxVisibility = Visibility.Visible;
this.olapChart.Legend.CheckBoxVisibility = Visibility.Collapsed;
// Set the Chart Legend Position.
ChartDockPanel.SetDock(this.olapChart.Legend, ChartDock.Right);
ChartDockPanel.SetDock(this.olapChart.Legend, ChartDock.Left);
ChartDockPanel.SetDock(this.olapChart.Legend, ChartDock.Top);
ChartDockPanel.SetDock(this.olapChart.Legend, ChartDock.Bottom);
ChartDockPanel.SetDock(this.olapChart.Legend, ChartDock.Floating);
```

VB.NET

```
' Set the Chart Type.
Me.olapChart.ChartType = ChartTypes.Column
' Set the Chart Series Color.
Me.olapChart.ColorModel.Palette = ChartColorPalette.Colorful
' Set the Chart Legend and Legend Check Box Visibility.
Me.olapChart.Legend.Visibility = Visibility.Visible
Me.olapChart.Legend.Visibility = Visibility.Collapsed
Me.olapChart.Legend.CheckBoxVisibility = Visibility.Visible
Me.olapChart.Legend.CheckBoxVisibility = Visibility.Collapsed
' Set the Chart Legend Position.
ChartDockPanel.SetDock(Me.olapChart.Legend, ChartDock.Right)
ChartDockPanel.SetDock(Me.olapChart.Legend, ChartDock.Left)
ChartDockPanel.SetDock(Me.olapChart.Legend, ChartDock.Top)
ChartDockPanel.SetDock(Me.olapChart.Legend, ChartDock.Bottom)
ChartDockPanel.SetDock(Me.olapChart.Legend, ChartDock.Floating)
```

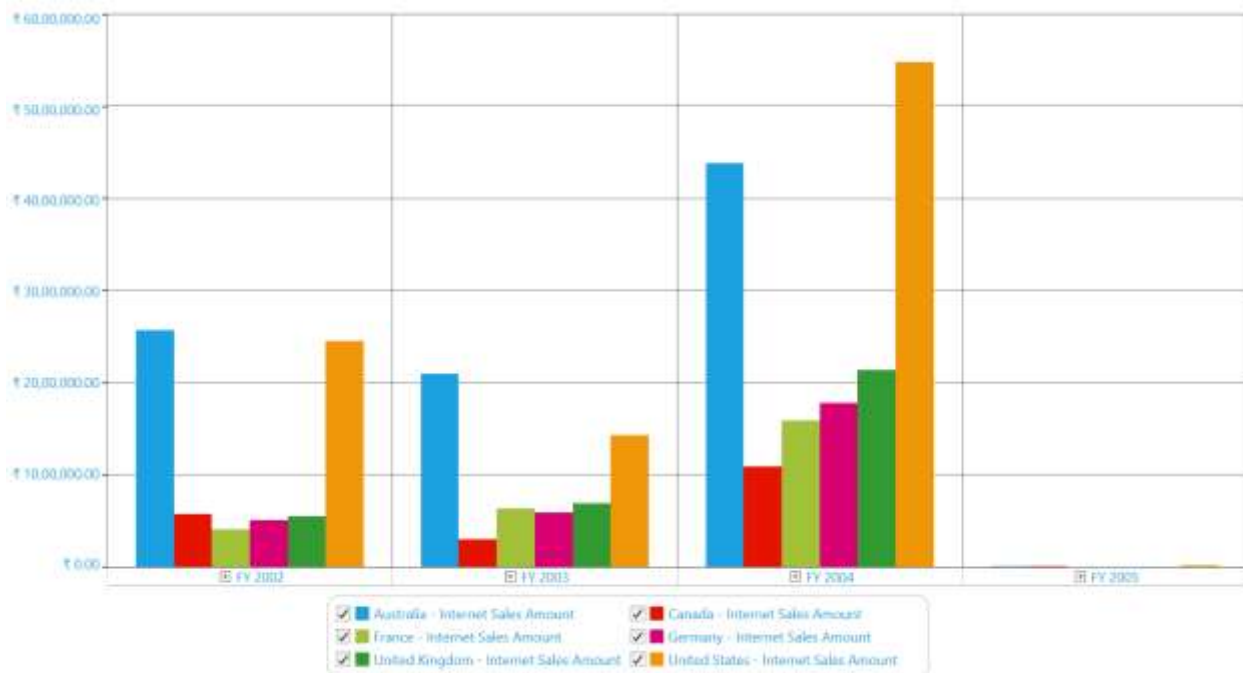


Chart border and background style

The OLAP chart provides options to set the chart border and background style:

- **BorderThickness:** Sets the border thickness for the OLAP chart control.
- **BorderBrush:** Specifies the border color for the OLAP chart control.
- **Background:** Specifies the background color for the OLAP chart control.
- **GridBackground:** Specifies the interior background color for the OLAP chart control.

The following code sample illustrates how to customize the chart border and the background style.

C#

```
// Set the Chart Border Style.
this.olapChart.BorderThickness = new Thickness(2);
this.olapChart.BorderBrush = Brushes.Blue;
// Set the Chart Background Style.
this.olapChart.Background = Brushes.LightBlue;
this.olapChart.GridBackground = Brushes.LightGray;
```

VB.NET

```
' Set the Chart Border Style.
Me.olapChart.BorderThickness = New Thickness(2)
Me.olapChart.BorderBrush = Brushes.Blue
' Set the Chart Background Style.
Me.olapChart.Background = Brushes.LightBlue
Me.olapChart.GridBackground = Brushes.LightGray
```

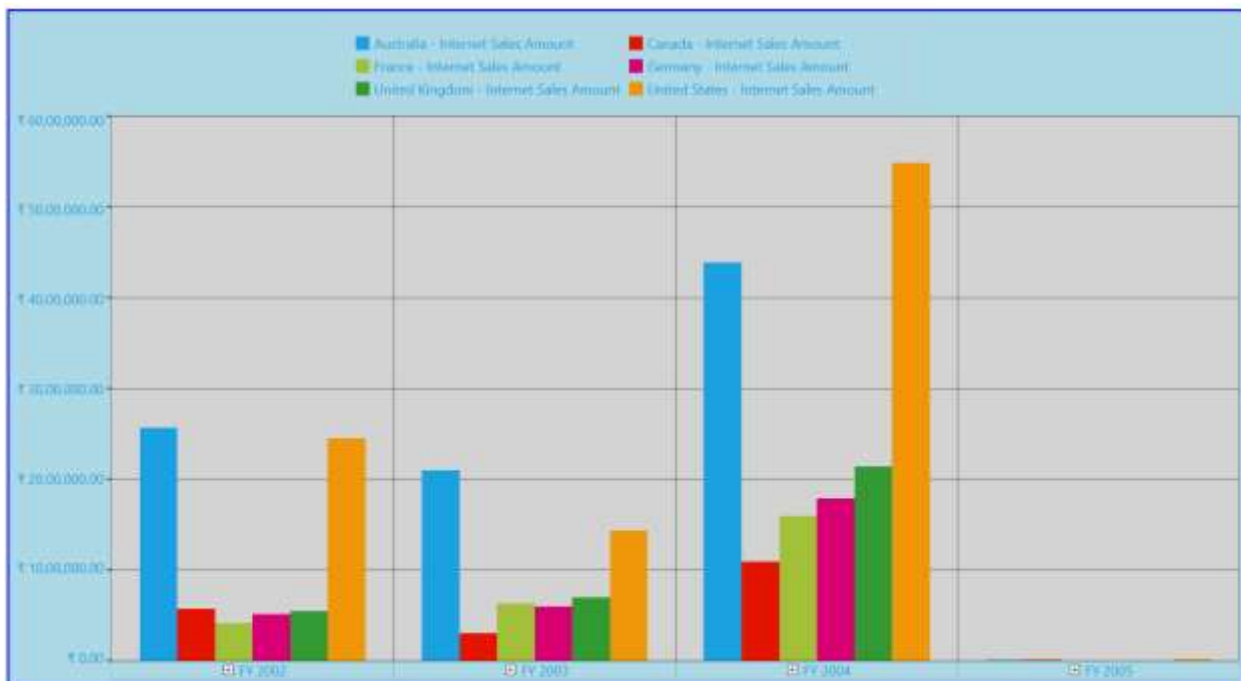


Chart points labels

The OLAP chart supports customizing the labels and symbols of chart points. This is illustrated in the following code sample.

C#

```
foreach (ChartSeries series in this.Series)
{
    series.AdornmentsInfo.Visible = true;
    ChartAdornmentInfo cai = series.AdornmentsInfo;
    // To display the x-axis label value.
    series.AdornmentsInfo.LabelContentPath = "DataPoint.X";
    // To display the y-axis label value.
    series.AdornmentsInfo.LabelContentPath = "DataPoint.Y";
    // To display the Series label value.
    series.AdornmentsInfo.LabelContentPath = "Series.Label";
}
```

VB.NET

```
For Each series As ChartSeries In Me.Series
    series.AdornmentsInfo.Visible = True
    Dim cai As ChartAdornmentInfo = series.AdornmentsInfo
    ' To display the x-axis label value.
    series.AdornmentsInfo.LabelContentPath = "DataPoint.X"
    ' To display the y-axis label value.
    series.AdornmentsInfo.LabelContentPath = "DataPoint.Y"
    ' To display the Series label value.
    series.AdornmentsInfo.LabelContentPath = "Series.Label"
Next series
```

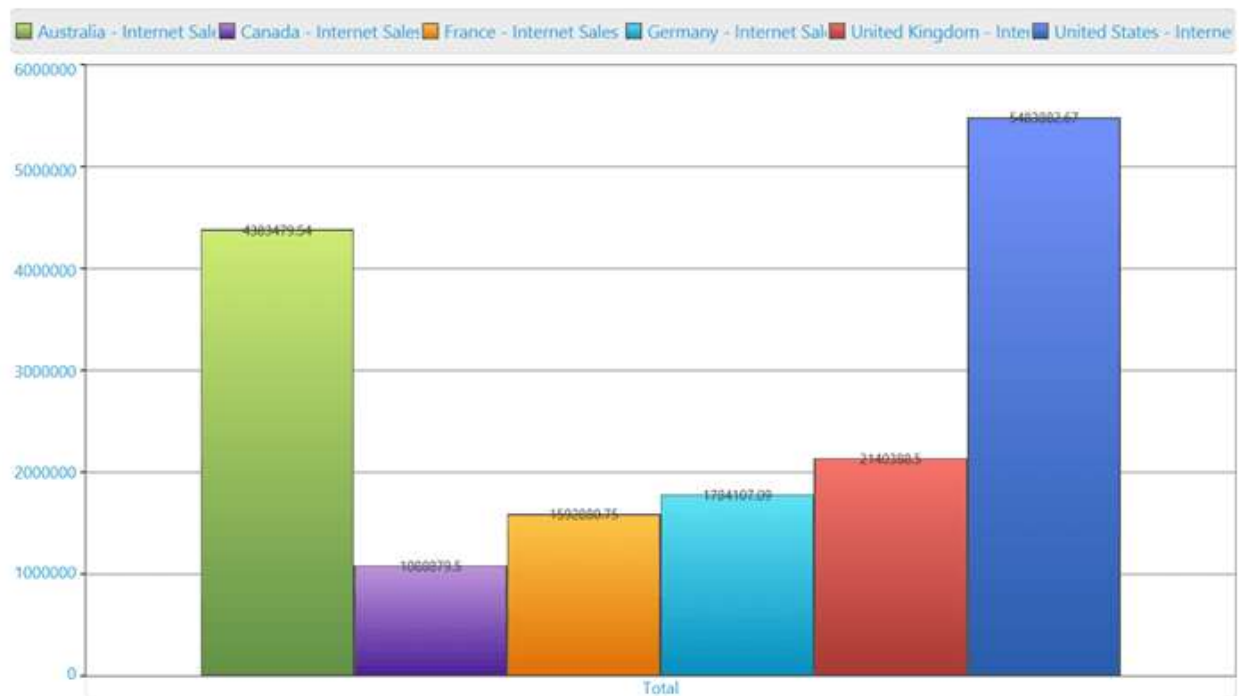


Chart axis labels

The OLAP chart supports customizing the labels of primary and secondary axes.

Customizing the font style of the primary axis

Dynamically change the font family, font color, and font weight for labels of the primary axis:

- **PrimaryAxis.LabelFontFamily:** Specifies the font family for the label of the primary axis.
- **PrimaryAxis.LabelForeground:** Specifies the font color for the label of the primary axis.
- **PrimaryAxis.LabelFontWeight:** Specifies the font weight for the label of the primary axis.

The following code sample illustrates how to customize the font style of the primary axis.

C#

```
// Set the Font Family.  
this.olapChart.PrimaryAxis.LabelFontFamily = new FontFamily("Arial");  
// Set the Font Color.  
this.olapChart.PrimaryAxis.LabelForeground = Brushes.LightBlue;  
// Set the Font Weight.  
this.olapChart.PrimaryAxis.LabelFontWeight = FontWeights.Bold;
```

VB.NET

```
' Set the Font Family.  
Me.olapChart.PrimaryAxis.LabelFontFamily = New FontFamily("Arial")  
' Set the Font Color.  
Me.olapChart.PrimaryAxis.LabelForeground = Brushes.LightBlue  
' Set the Font Weight.  
Me.olapChart.PrimaryAxis.LabelFontWeight = FontWeights.Bold
```

Customizing the font style of the secondary axis

Dynamically change the font family, font color, and font weight for labels of the secondary axis.

- **SecondaryAxis.LabelFontFamily:** Specifies the font family for the label of the secondary axis.
- **SecondaryAxis.LabelForeground:** Specifies the font color for the label of the secondary axis.
- **SecondaryAxis.LabelFontWeight:** Specifies the font weight for the label of the secondary axis.

The following code sample illustrates how to customize the font style of the secondary axis.

C#

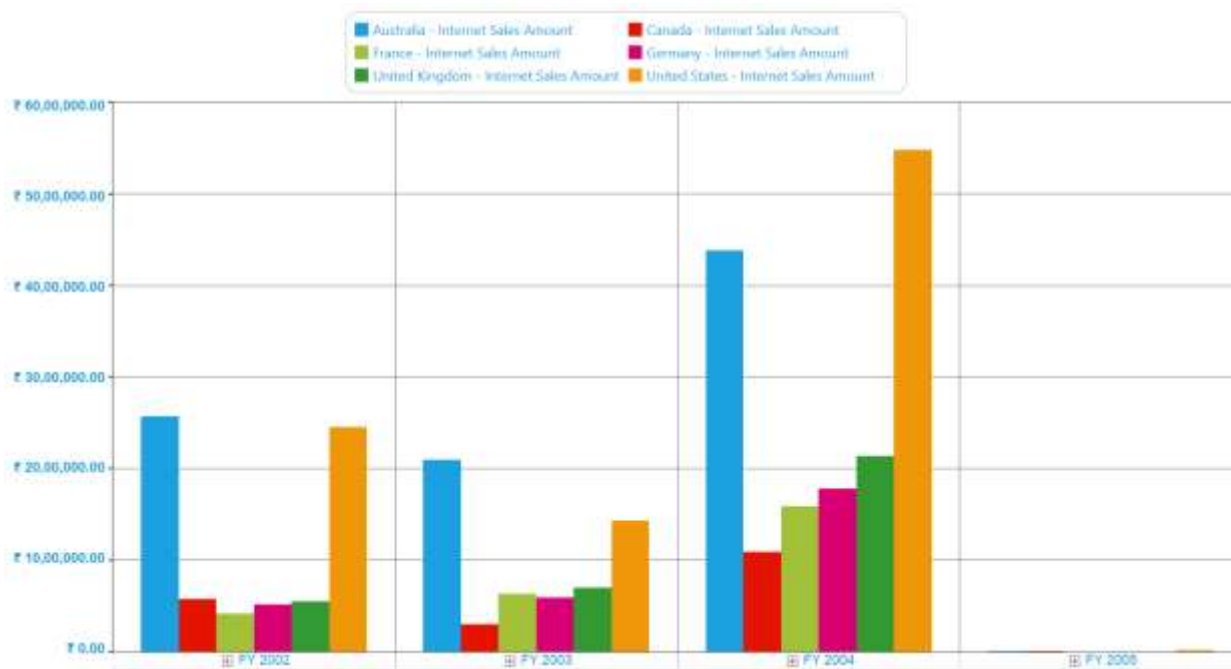
```
// Set the Font Family.  
this.olapChart.SecondaryAxis.LabelFontFamily = new FontFamily("Arial");  
// Set the Foreground Color.  
this.olapChart.SecondaryAxis.LabelForeground = Brushes.LightBlue;  
// Set the Font Weight.  
this.olapChart.SecondaryAxis.LabelFontWeight = FontWeights.Bold;
```

VB.NET

```
' Set the Font Family.  
Me.olapChart.SecondaryAxis.LabelFontFamily = New FontFamily("Arial")  
' Set the Foreground Color.
```



```
Me.olapChart.SecondaryAxis.LabelForeground = Brushes.LightBlue
' Set the Font Weight.
Me.olapChart.SecondaryAxis.LabelFontWeight = FontWeights.Bold
```



A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Appearance\Appearance

Tooltip in WPF Olap Chart

The OLAP chart provides the series information such as measure, primary x-axis and y-axis values, and series name through the series tooltip, when the mouse pointer is moved over chart points.

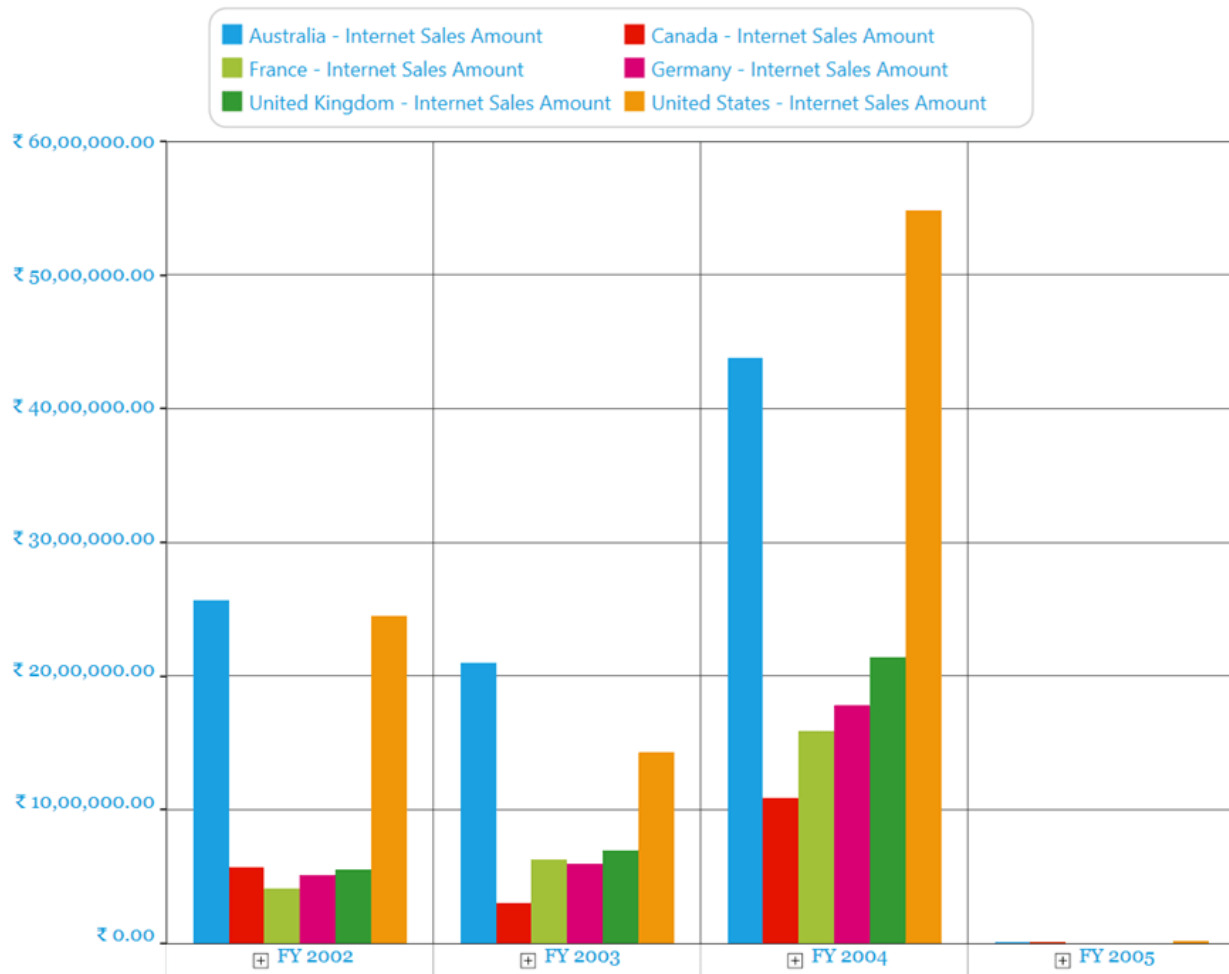
The following code sample shows how to disable the series tooltip using the `ShowToolTip` property.

C#

```
this.olapChart.Series[0].ShowToolTip = false;
```

VB.NET

```
Me.olapChart.Series(0).ShowToolTip = False
```



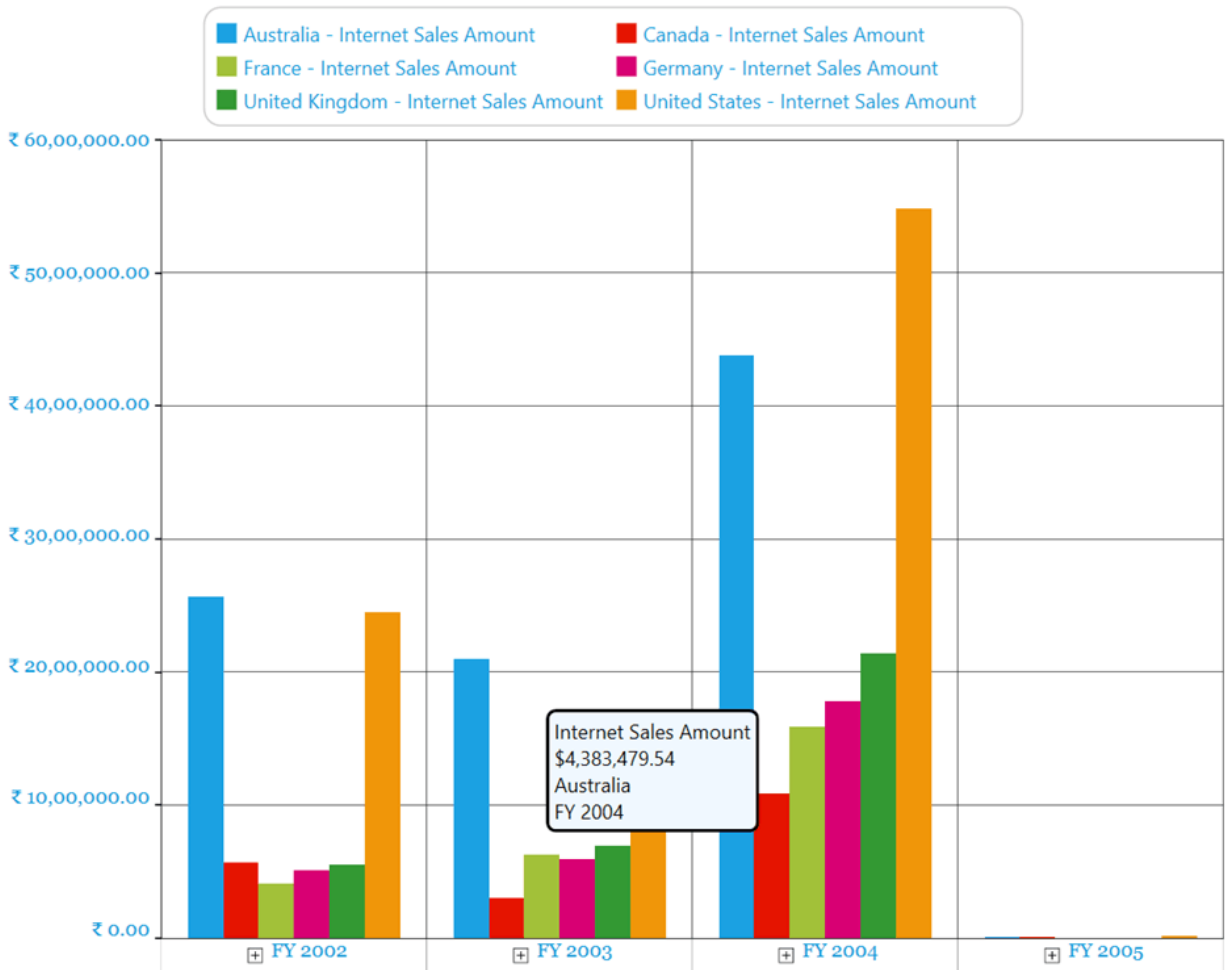
The following code sample shows how to enable the series tooltip using the `ShowToolTip` property.

C#

```
this.olapChart.Series[0].ShowToolTip = true;
```

VB.NET

```
Me.olapChart.Series(0).ShowToolTip = True
```

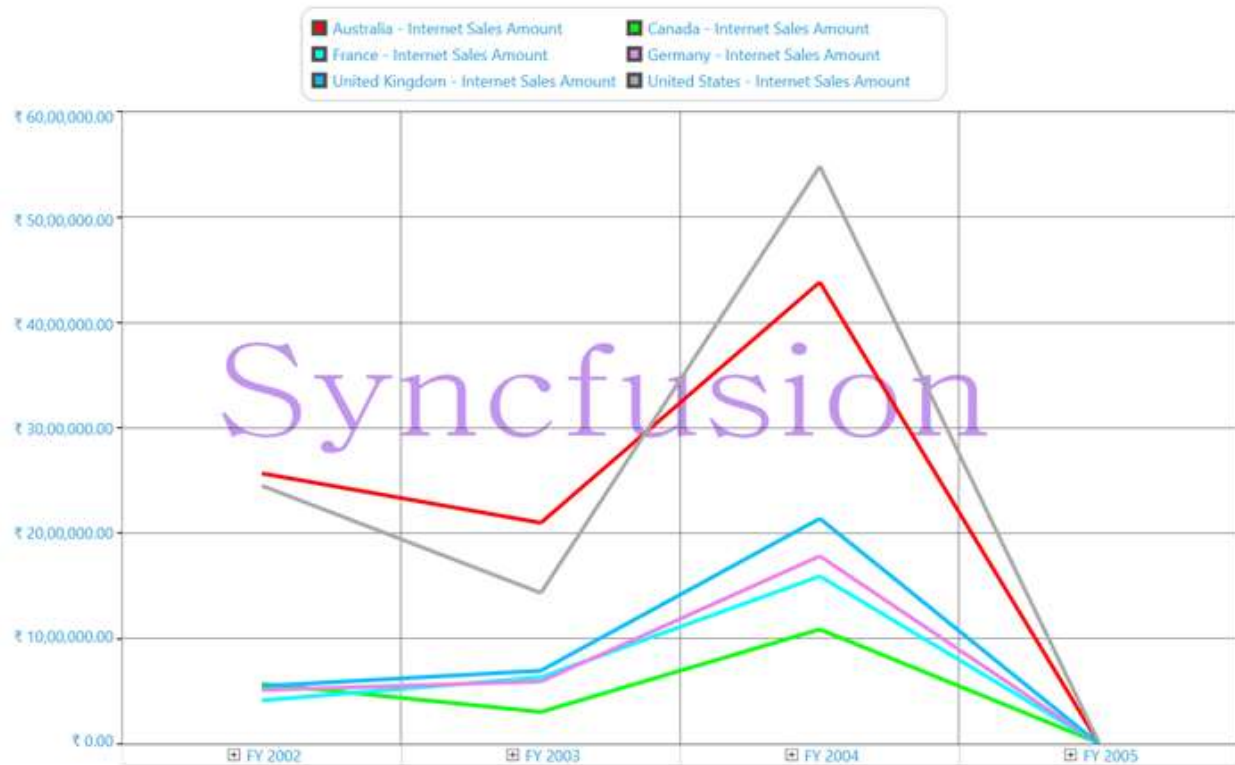


Watermark in WPF Olap Chart

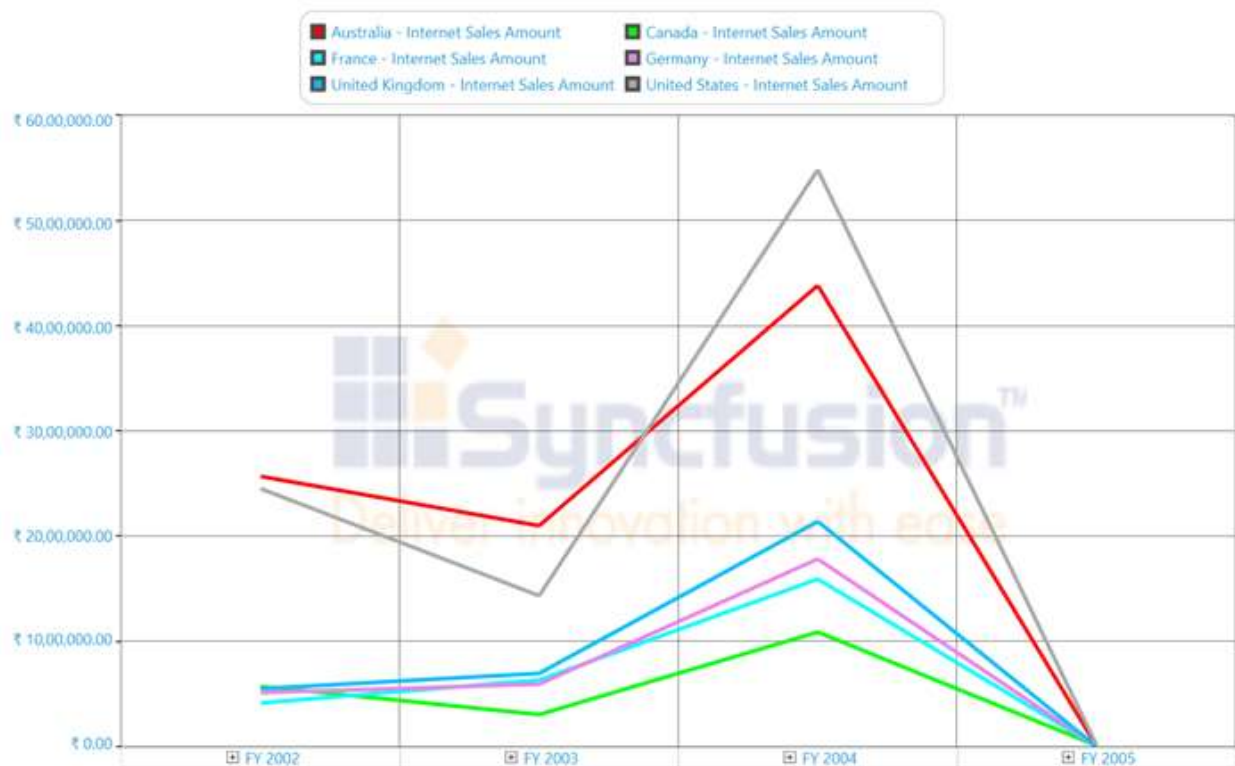
The OLAP chart for WPF supports watermark feature that is used to show text, image, or both as a watermark inside the chart area.

There are many customization options available for the watermarked content. The content can be aligned both horizontally and vertically, and the font style of the content can be changed. The interior of the content can be customized and the opacity can also be varied.

The following screenshot shows the text as a watermark.



The following screenshot shows the image as a watermark.



A sample demo is available at the following location.

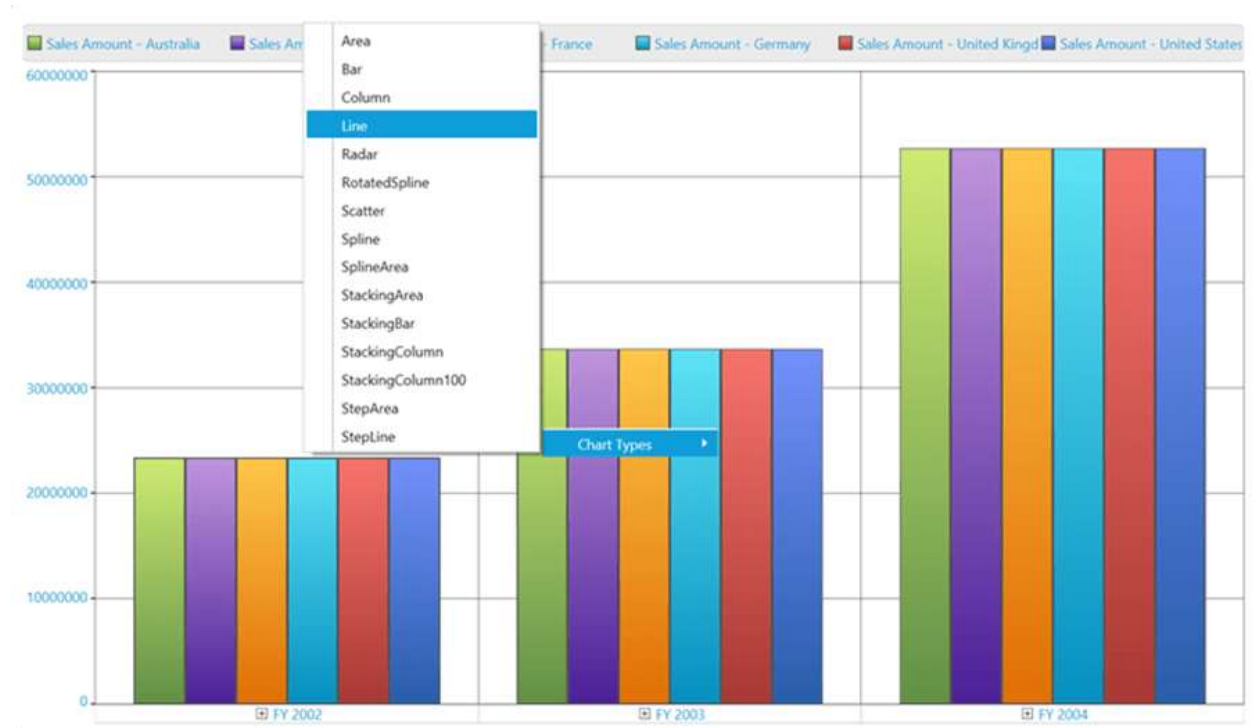
{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Chart Appearance\Watermark Demo

Chart Type for Specific Series in WPF Olap Chart

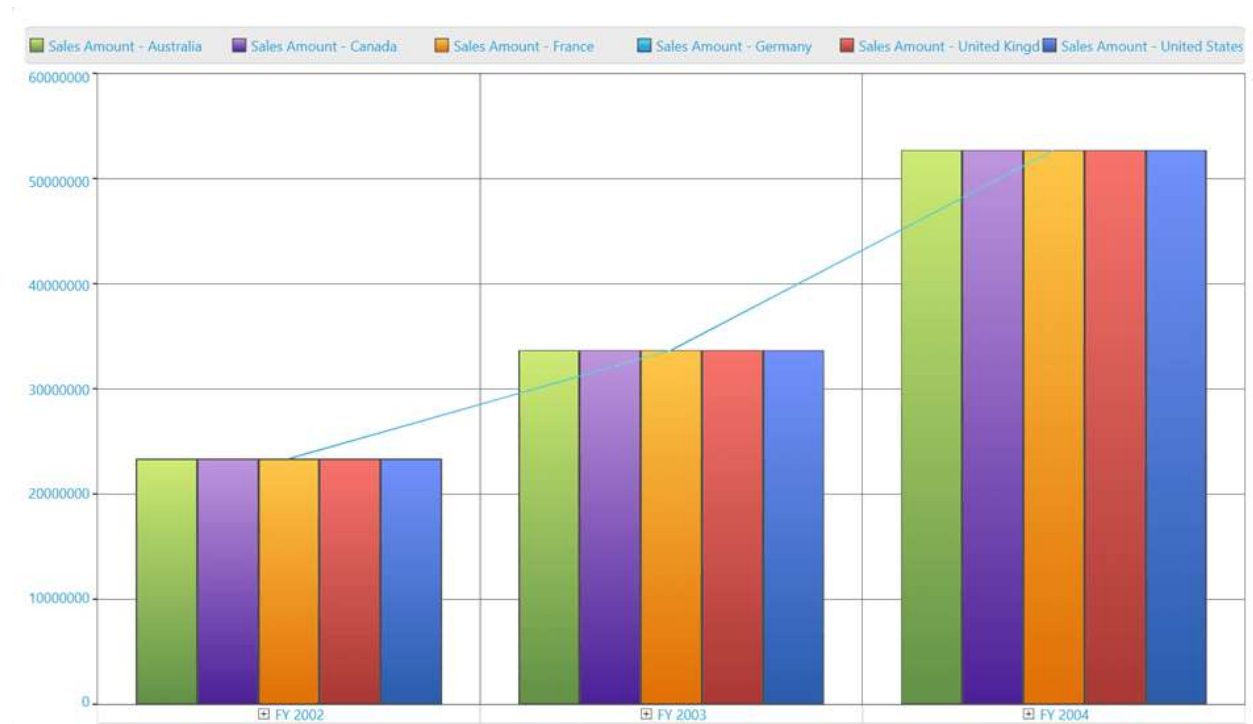
The OLAP chart in WPF supports changing the chart type of each series using the context menu.

To change the chart type of a specific series, right-click the particular series. A context menu with the list of available chart type appears. From the list of chart types in the context menu, you can select the required chart type.

The following screenshot shows how a specific column chart series is changed to line chart series.



The following screenshot shows a column chart series changed to line chart.



Exporting in WPF Olap Chart

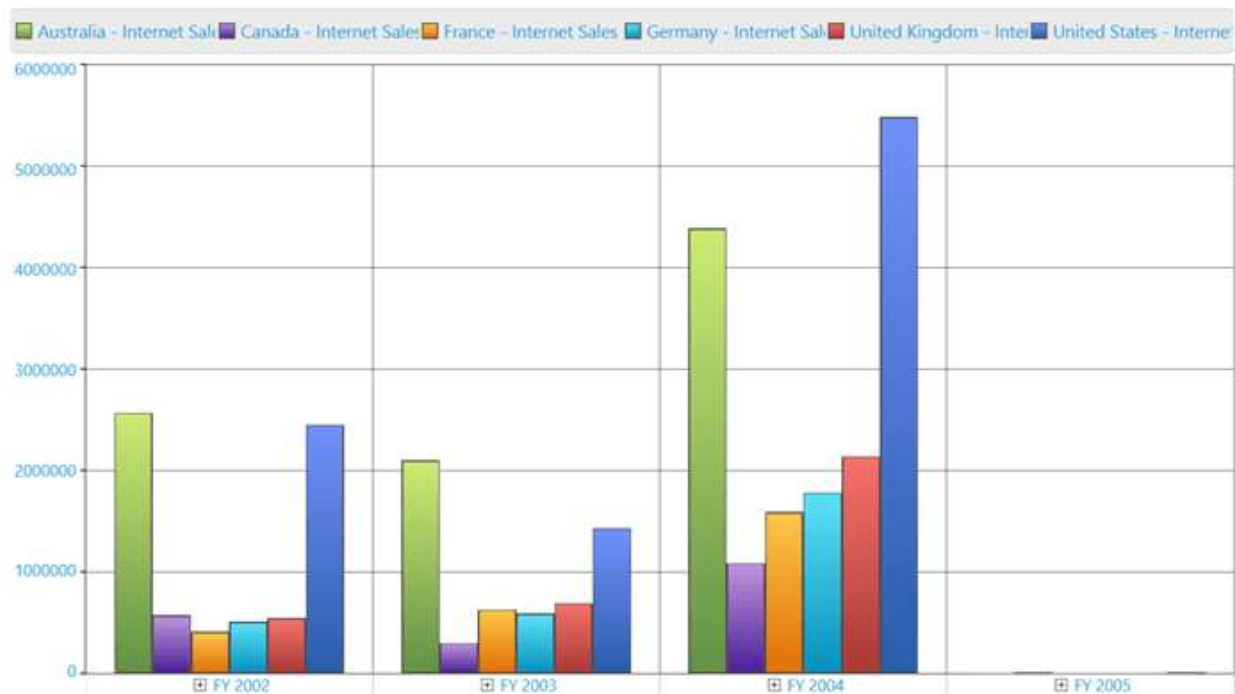
An OLAP chart can be exported to various image and document formats. The following topics illustrate this in detail:

- Exporting as an image.
- Exporting to Word document.
- Exporting to PDF document.

Exporting as an image

The OLAP chart can be copied to the clipboard or exported as an image. It can be exported in any one of the following image formats:

- Bitmap
- JPG
- PNG
- XPS
- GIF
- TIFF
- WDP



Exporting to Word document

The OLAP chart for WPF supports exporting an OLAP chart to an Microsoft Word document. It can also be exported into a template Word document file at a position specified by a marker.

The following methods are used to export an OLAP chart to a Microsoft Word document:

Methods

- **ExportIntoNewDoc:** Exports a chart to a new Word document file with specified file name.
- **ExportIntoTemplateDoc:** Exports a chart to an existing Word document file in the default marker string location.
- **ExportIntoTemplateDoc:** Exports a chart to an existing Word document file in the given marker string location.
- **ExportIntoTemplateDoc:** Exports a chart to an existing instance of a Word document in the default marker string location.
- **ExportIntoTemplateDoc:** Exports a chart to an existing instance of a Word document in the marker string location.

The following code sample illustrates how to export an OLAP chart to a Microsoft Word document.

C#

```
// Export the OlapChart into a new Word Document.
OlapChartWordExport olapChartWordExport = new
OlapChartWordExport(this.olapChart);
olapChartWordExport.ExportintoNewDoc(@"..\..\OutputDocument\Document.doc");
// Export the OlapChart into a new Word Document file in the default marker
string location.
OlapChartWordExport olapChartWordExport = new
OlapChartWordExport(this.olapChart);
```

```

olapChartWordExport.ExportIntoTemplateDoc(@"..\..\OutputDocument\Document.doc");
// Export the OlapChart into an existing Word Document file in the given marker string location.
OlapChartWordExport olapChartWordExport = new
OlapChartWordExport(this.olapChart);
olapChartWordExport.ExportIntoTemplateDoc(@"..\..\OutputDocument\Document.doc", "MarkerString1");

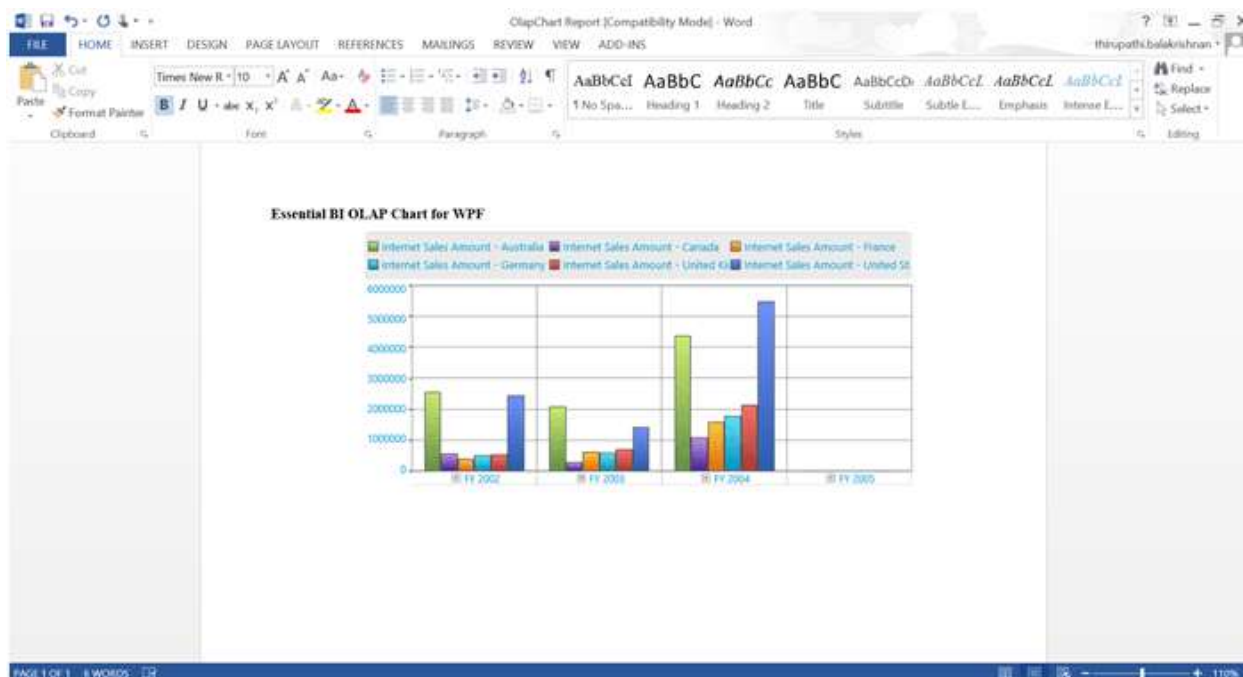
```

VB.NET

```

' Export the OlapChart into a new Word Document.
Dim olapChartWordExport As New OlapChartWordExport(Me.olapChart)
olapChartWordExport.ExportIntoNewDoc(@"..\..\OutputDocument\Document.doc")
' Export the OlapChart into a new Word Document file in the default marker string location.
Dim olapChartWordExport As New OlapChartWordExport(Me.olapChart)
olapChartWordExport.ExportIntoTemplateDoc(@"..\..\OutputDocument\Document.doc")
' Export the OlapChart into an existing Word Document file in the given marker string location.
Dim olapChartWordExport As New OlapChartWordExport(Me.olapChart)
olapChartWordExport.ExportIntoTemplateDoc(@"..\..\OutputDocument\Document.doc", "MarkerString1")

```



Exporting to a PDF document

The OLAP chart that is exported to a Word document is inserted into a PDF document. The **ExportIntoNewPDF** method is used for this purpose.

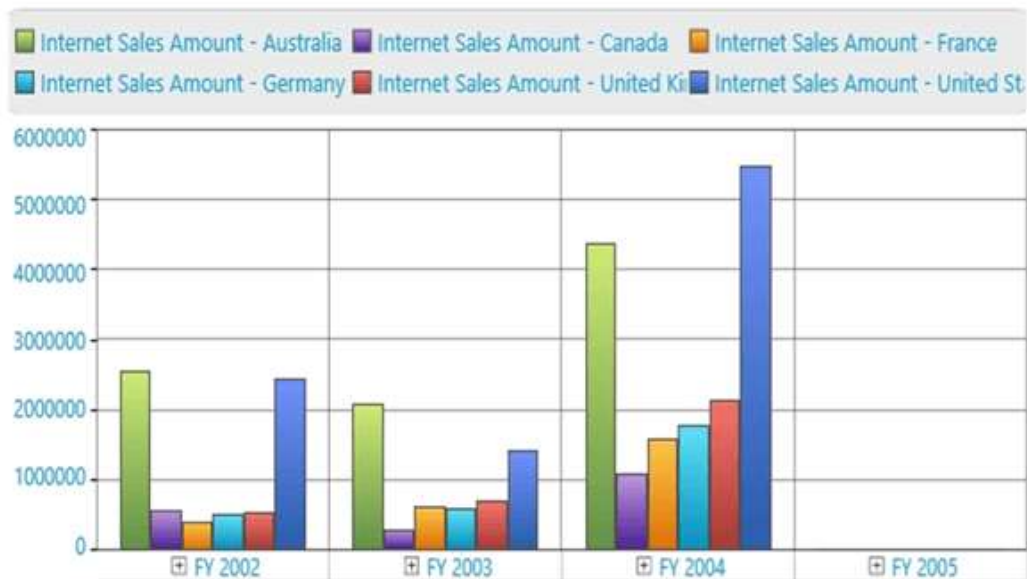
The following code sample illustrates how to set this method.

C#


```
OlapChartPdfExport chartPdfExport = new OlapChartPdfExport(this.olapChart);
chartPdfExport.ExportIntoNewPdf(@"..\..\TemplateDocument\PdfDocument.pdf");
```

VB.NET

```
Dim chartPdfExport As New OlapChartPdfExport(Me.olapChart)
chartPdfExport.ExportIntoNewPdf(@"..\..\TemplateDocument\PdfDocument.pdf")
```



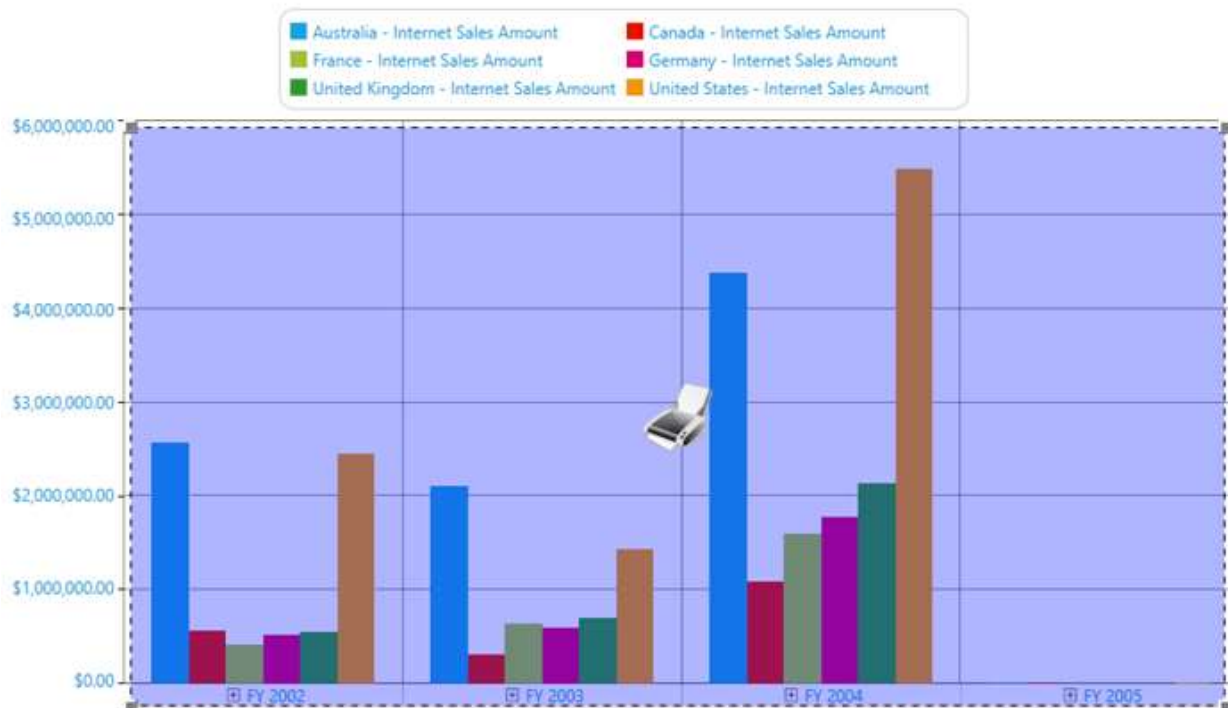
A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Exporting\Exporting Chart Demo

Printing in WPF Olap Chart

The OLAP chart can be printed in black, color, or white modes. It supports the cropping feature used to print a particular part of the chart.

The following image illustrates printing a cropped image.



A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Printing\Printing Chart Demo

Theming in WPF Olap Chart

Theming is the process of applying particular settings to visual elements of a product. This feature provides the following theming options:

- Office 2010 Blue
- Office 2010 Black
- Office 2010 Silver
- Transparent
- Office 2007 Blue
- Office 2007 Black
- Office 2007 Silver
- Blend
- Metro
- Office 2003
- Default

The `VisualStyle` property allows users to set the visual style of the OLAP chart control. The following code sample demonstrates how theming is added to the OLAP chart control.

XML

```
<syncfusion:OlapChart x:Name="olapChart" VisualStyle="Transparent"/>
```

C#

```
this.olapChart.VisualStyle = OlapChartVisualStyle.Transparent;
```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Appearance\Visual Styles demo

Localization in WPF Olap Chart

Localization is the key feature for providing software solutions targeted at global users. The OLAP chart allows users to localize the control to a specific locale and supports "resx" based localization.

The following steps should be performed to localize the control.

- Translation.
- Resource file and file name conventions.
- Tag inclusion into the project file.
- Specifying the CurrentUICulture.

Translation

The first step in localization is translating the strings that can be localized to the destination locale.

Note: Localization key field should be same for all locales. Do not translate it.

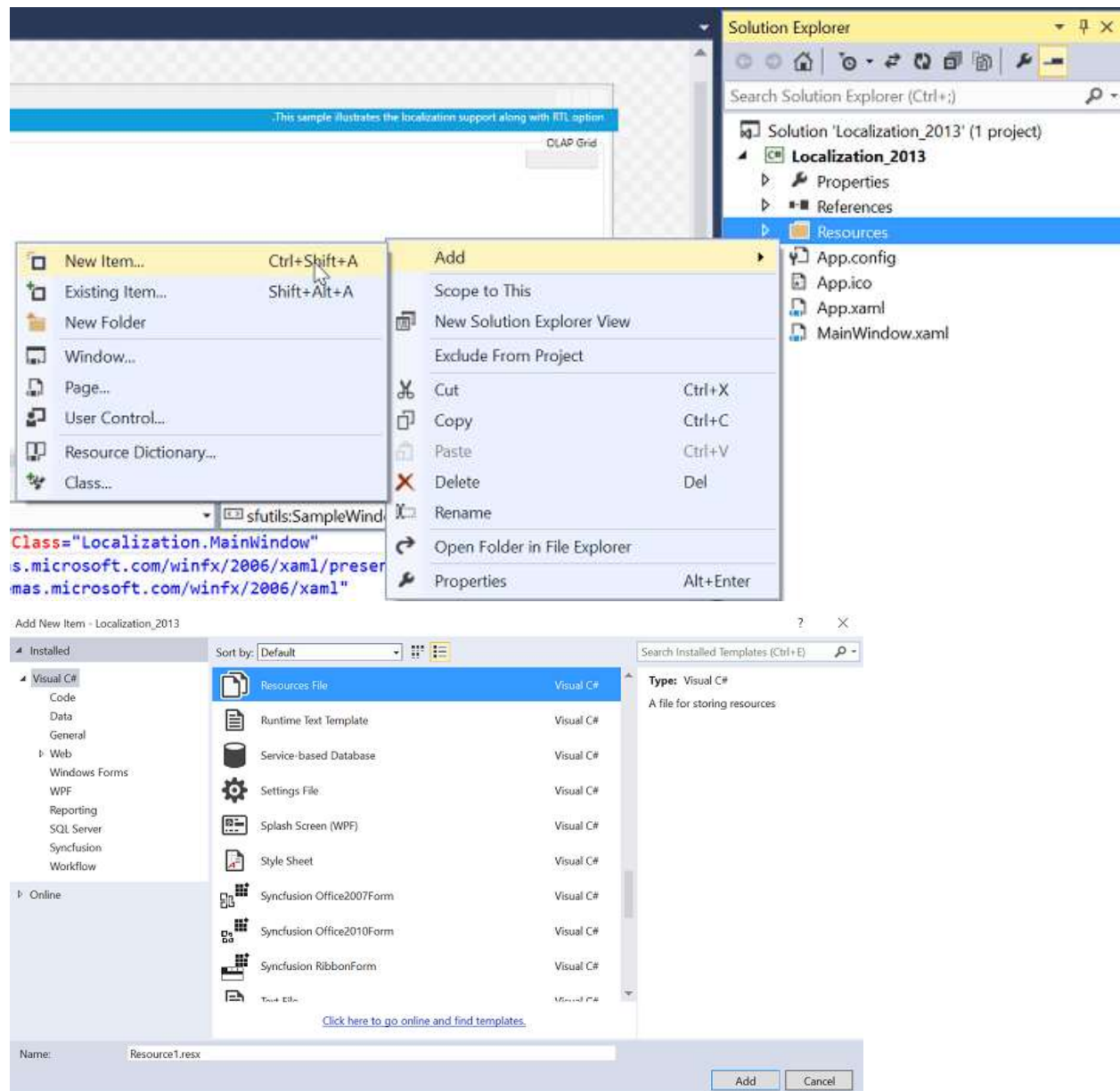
Resource file and file name conventions

After translating the strings that can be localized, perform the following in the application:

Right-click the project file to create a new folder in the project. Select Add > New Folder and rename the folder as "Resources".

Note: The folder name should strictly be "Resources".

Now, right-click the resources folder to create a new resource file in the Visual Studio project. Navigate to Add > New Item.



Select "Resources File" from the list. Then, name the resource file as Syncfusion.OlapChart.WPF.ar-AE.resx and click Add.

Note: The resource file name should strictly be in the format "Syncfusion.OlapChart.WPF.<Culture Code>.resx".

Copy and paste the translated locale to the resource file created in the earlier step.

Specifying the CurrentUICulture

Now, you need to specify the CurrentUICulture of the application. You can specify the CurrentUICulture either from Application_Startup in App.xaml.cs or from the constructor in the main page. (If you are specifying the current culture on the main page, then make sure that it is assigned before the InitializeComponent method).

C#

```
public MainWindow()
{
    //Set the current thread culture to load the localization resource file.
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("ar-AE");
    InitializeComponent();
    if (System.Globalization.CultureInfo.CurrentUICulture.ToString() == "ar-AE")
    this.FlowDirection = System.Windows.FlowDirection.RightToLeft;
}
```

RTL support

RTL support for OLAP chart is used to display the content from right to left by setting the **FlowDirection** property to "RightToLeft". The following code sample explains how to set this property.

XML

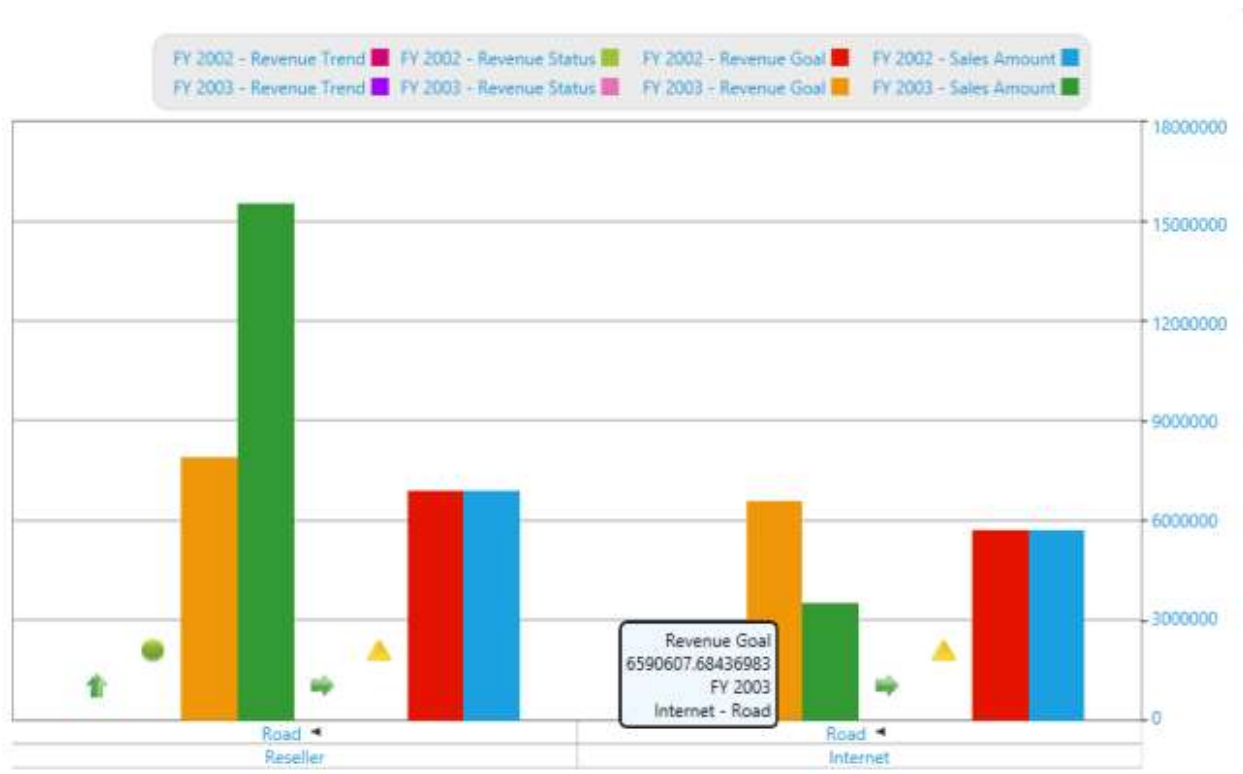
```
<syncfusion:OlapChart x:Name="olapChart" FlowDirection="RightToLeft"
HorizontalAlignment="Stretch"
olapshared:DataSource.DataManagerName="localManager"
olapshared:DataSource.ConnectionString="{Binding OlapConnectionString}" />
```

C#

```
this.olapChart.FlowDirection = System.Windows.FlowDirection.RightToLeft;
```

VB.NET

```
Me.olapChart.FlowDirection = System.Windows.FlowDirection.RightToLeft
```



A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapChart.WPF\Samples\Localization

OlapClient

WPF Olap Client Overview

The OLAP client control for WPF supports browsing multidimensional data that is organized in the form of dimensions, measures, named sets, and KPI (key performance indicators) in a cube format. You can visualize the results in graphical and tabular formats (chart and grid). The OLAP client control is also used to create and edit a report on-the-fly that is stored for later use.

Key features

- **Slice and dice:** You can slice and dice the cube database.
- **Saving the current session:** You can save your current session in an XML file or as a stream for future use and reload it whenever needed.
- **Filtering and sorting:** You can specify filters and sorting criteria by using the provided UI options.
- **Chart:** A chart is a graphical representation of data, in which data is represented by symbols, such as bars in a bar chart, lines in a line chart, or slices in a pie chart.
- **Grid:** A grid is a tabular representation of data, arranged in the form of rows and columns and categorized accordingly.
- **Cube selector:** Comprises multiple cubes obtained from the data source.
- **Cube dimension browser:** Tree view structure that comprises measures, dimensions, hierarchies, named sets, KPI, and so on, belonging to the current cube into independent logical groups.

- **AxisElementBuilder:** Allows building an OLAP report by placing elements in different axes of the OLAP client. There are three axes that are supported namely column, row, and slicer.
- **Member editor:** A tree view control that displays the member elements of the selected dimension.
- **Measure editor:** Comprises a collection of measures.
- **Toolbar:** Toolbar provides options for filter, sorting, exporting, and manipulating the OLAP report.
- **Export:** You can also export the OLAP grid and OLAP chart to PDF, Microsoft Excel, Microsoft Word, CSV, and other supported image formats.

Getting Started with WPF Olap Client

Important

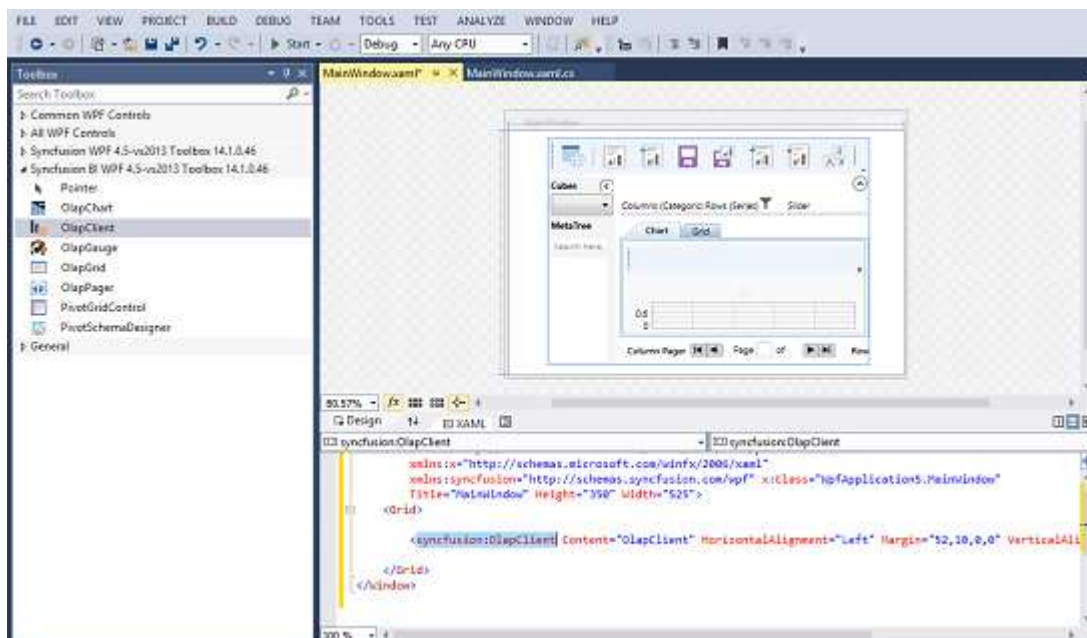
Starting with v16.2.0.x, if you refer to Syncfusion assemblies from trial setup or from the NuGet feed, include a license key in your projects. Refer to this [link](#) to learn about registering Syncfusion license key in your WPF application to use the components.

This section covers the information required to create a simple OLAP client bound to the OLAP data source.

Through Visual Studio

Open the Visual Studio IDE and navigate to File > New > Project > WPF application (inside Visual C# templates) to create a new WPF application.

After creating the WPF application, go to View menu and select Toolbox option. Now, the toolbox will appear inside the Visual Studio IDE. From the Visual Studio Toolbox, drag the OLAP client under the **Syncfusion BI WPF** tag. It will automatically add the required assemblies to the application.



Add a **Name** to the OLAP client component for accessing it through code-behind as shown in the following code sample.

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="WpfApplication5.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid>
<syncfusion:OlapClient Name="olapClient1" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Height="600" Width="700"/>
</Grid>
</Window>
```

Include the following namespace in the code-behind for using **OlapDataManager** in the application. The **OlapDataManager** class contains the connection details, current report, cube name, cube schema, and pivot engine for rendering the OLAP client control.

- Syncfusion.Olap.Manager

C#

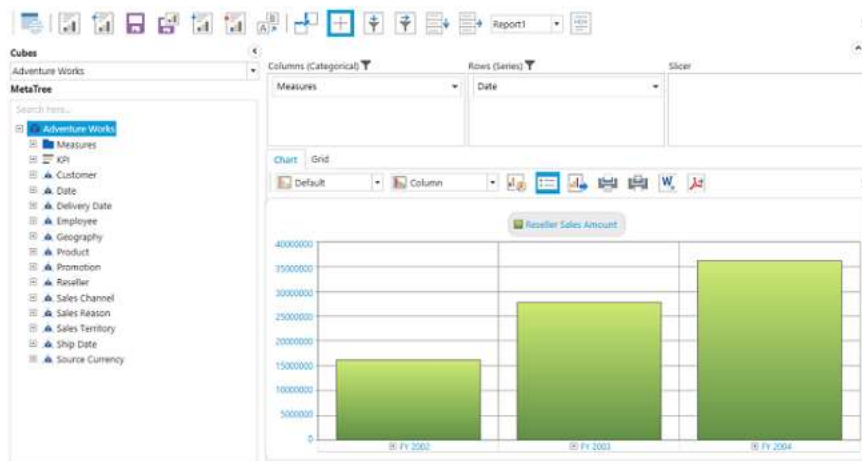
```
using Syncfusion.Olap.Manager;
namespace WpfApplication
{
public partial class MainWindow : SampleWindow
{
public MainWindow()
{
InitializeComponent();
string connectionString = "Enter a valid connection string";
//Connection string is passed to the OlapDataManager as an argument.
OlapDataManager olapDataManager = new OlapDataManager(connectionString);
// OlapClient gets information from the OlapDataManager.
this.olapClient1.OlapDataManager = olapDataManager;
this.olapClient1.DataBind();
}
}
}
```

VB.NET

```
Imports Syncfusion.Olap.Manager
Namespace WpfApplication
Partial Public Class MainWindow
Inherits SampleWindow
Public Sub New()
InitializeComponent()
Dim connectionString As String = "Enter a valid connection string"
'Connection string is passed to the OlapDataManager as an argument.
Dim olapDataManager As OlapDataManager = New
OlapDataManager(connectionString)
'OlapClient gets information from the OlapDataManager.
Me.olapClient1.OlapDataManager = olapDataManager
Me.olapClient1.DataBind()
End Sub
```


End Class
End Namespace

Run the application. The following output will be generated.



Through expression blend

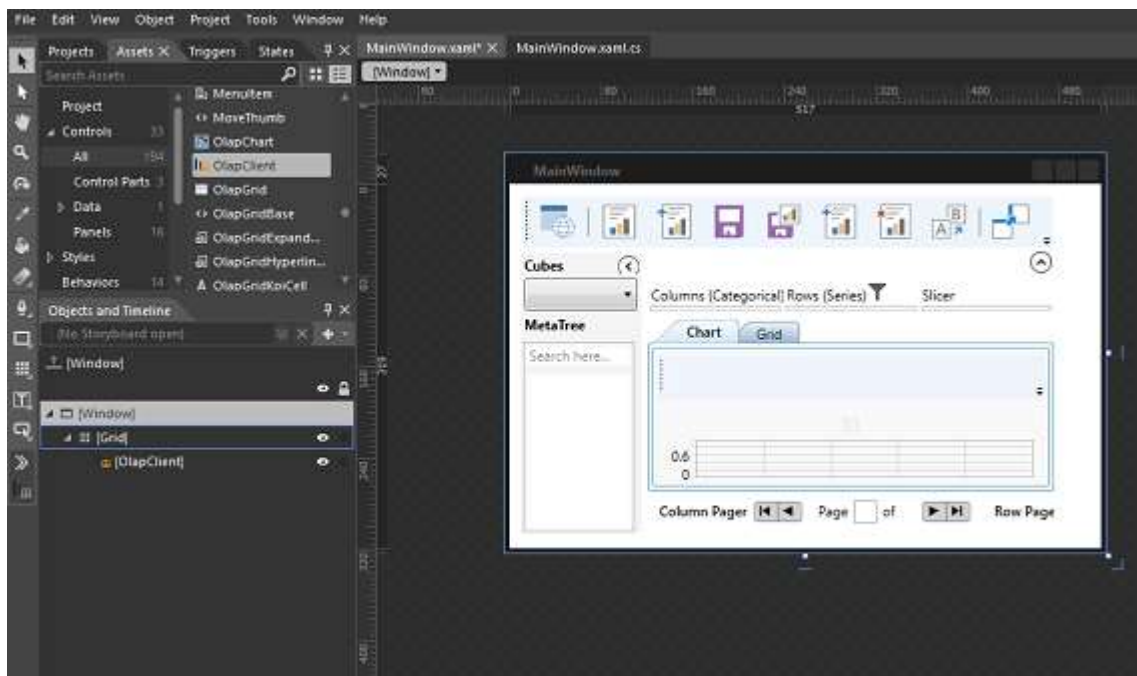
Open Blend for Visual Studio and navigate to File > New project > WPF > WPF application to create a new WPF application.

Select the **Project** tab available in the left-corner of the Blend IDE, right-click the **References** and select **Add Reference**. Now, browse and add the following Syncfusion assemblies to the project.

- Syncfusion.Grid.Wpf
- Syncfusion.Olap.Base
- Syncfusion.Chart.Wpf
- Syncfusion.OlapChart.Wpf
- Syncfusion.OlapGrid.Wpf
- Syncfusion.OlapGridCommon.Wpf
- Syncfusion.OlapClient.Wpf
- Syncfusion.OlapShared.Wpf
- Syncfusion.OlapTools.Wpf

Note: You can also get the assemblies by browsing to the default assembly location {System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<framework version>\

On adding the above assemblies, the OLAP client control will be added under the **Assets** tab automatically. Now, choose the **Assets** tab and drag the OLAP client to the designer.



Add a **Name** to the OLAP client component for accessing it through code-behind as shown in the following code sample.

XML

```
<Window
x:Class="WpfApplication.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="clr-namespace:Syncfusion.Windows.Client.Olap;assembly=Syncfusion.OlapClient.WPF"
Width="900" Height="630" >
<Grid>
<syncfusion:OlapClient Name="olapClient1" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"/>
</Grid>
</Window>
```

Include the following namespace in the code-behind for using OlapDataManager in the application.

- Syncfusion.Olap.Manager

C#

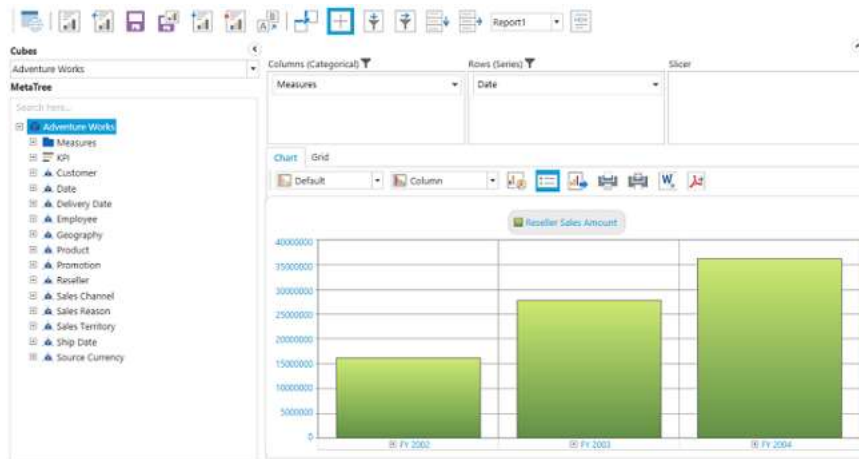
```
using Syncfusion.Olap.Manager;
namespace WpfApplication
{
public partial class MainWindow : SampleWindow
{
public MainWindow()
{
InitializeComponent();
String connectionString = "Enter a valid connection string";
```

```
//Connection string is passed to the OlapDataManager as an argument.
OlapDataManager olapDataManager = new OlapDataManager(connectionString);
// The OlapClient gets the information from the OlapDataManager.
this.olapClient1.OlapDataManager = olapDataManager;
this.olapClient1.DataBind();
}
}
}
```

VB.NET

```
Imports Syncfusion.Olap.Manager
Namespace WpfApplication
Partial Public Class MainWindow
Inherits SampleWindow
Public Sub New()
InitializeComponent()
Dim connectionString As String = "Enter a valid connection string"
'Connection string is passed to the OlapDataManager as an argument.
Dim olapDataManager As OlapDataManager = New
OlapDataManager(connectionString)
'The OlapClient gets the information from the OlapDataManager.
Me.olapClient1.OlapDataManager = olapDataManager
Me.olapClient1.DataBind()
End Sub
End Class
End Namespace
```

Run the application. The following output will be generated.



Through code-behind

Open the Visual Studio IDE and go to File > New > Project > WPF application (inside Visual C# Templates) to create a new WPF application.

To add the dependency assemblies within the application, right-click the **References** and select **Add Reference**. Then, add the following Syncfusion assemblies manually to the project from the installed location.

- Syncfusion.Chart.WPF

- Syncfusion.Shared.WPF
- Syncfusion.Olap.Base
- Syncfusion.OlapChart.WPF
- Syncfusion.OlapChartConverter.WPF
- Syncfusion.OlapClient.WPF
- Syncfusion.OlapGrid.WPF
- Syncfusion.OlapGridCommon.WPF
- Syncfusion.OlapGridConverter.WPF
- Syncfusion.OlapSampleUtils
- Syncfusion.OlapShared.WPF
- Syncfusion.OlapTools.WPF
- Syncfusion.Tools.WPF

Note: You can also get the assemblies by browsing to the default assembly location {System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<{framework version}>\

Include the following namespaces in code-behind for using OLAP client and OlapDataManager in the application.

- Syncfusion.Olap.Manager
- Syncfusion.Windows.Client.Olap

C#

```
using Syncfusion.Windows.Client.Olap;
using Syncfusion.Olap.Manager;
namespace WpfApplication
{
    public partial class MainWindow : SampleWindow
    {
        public MainWindow()
        {
            //OlapClient instantiation.
            OlapClient olapClient1 = new OlapClient();
            String connectionString = "Enter a valid connection string";
            //Connection string is passed to the OlapDataManager as an argument.
            OlapDataManager olapDataManager = new OlapDataManager(connectionString);
            // OlapClient gets information from the OlapDataManager.
            olapClient1.OlapDataManager = olapDataManager;
            olapClient1.DataBind();
            //OlapClient added to the Main Window Grid region.
            grid.Children.Add(olapClient1);
        }
    }
}
```

VB.NET

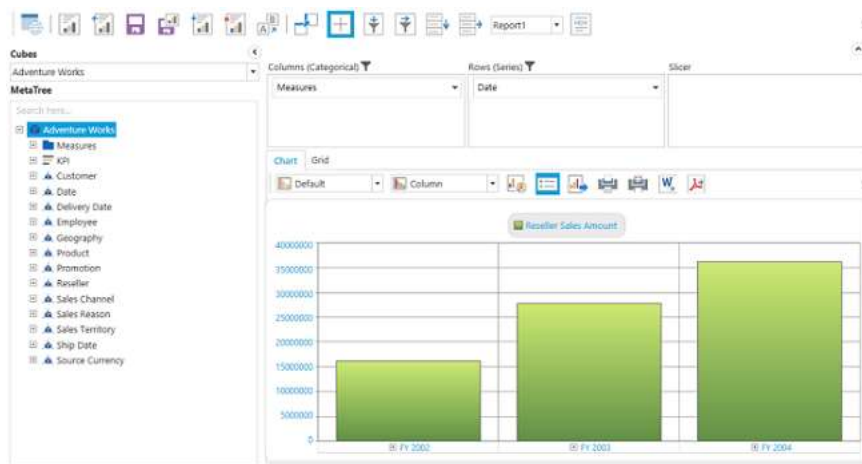
```
Imports Syncfusion.Olap.Manager
Imports Syncfusion.Windows.Client.Olap;
Namespace WpfApplication
    Partial Public Class MainWindow
```

```

Inherits SampleWindow
Public Sub New()
InitializeComponent()
'OlapClient instantiation.
Dim olapClient1 As OlapClient = New OlapClient()
Dim connectionString as String = "Enter a valid connection string"
'Connection string is passed to the OlapDataManager as an argument.
Dim olapDataManager As OlapDataManager = New OlapDataManager
(connectionString)
'OlapClient gets information from the OlapDataManager.
olapClient1.OlapDataManager = olapDataManager
olapClient1.DataBind()
'OlapClient added to the Main Window Grid region.
grid.Children.Add(olapClient1)
End Sub
End Class
End Namespace

```

Run the application. The following output will be generated.



Data Binding in WPF Olap Client

Binding OLAP client to offline cube

To connect to an OLAP cube available in the local machine, set the physical path of the cube set in the connection string. The following code example illustrates the same.

C#

```

string connectionString = @"Datasource =
systemdrive:\OfflineCube\Adventure_Works_Ext.cub; Provider= msolap;";
OlapDataManager olapDataManager = new OlapDataManager(connectionString);
this.olapClient1.OlapDataManager = olapDataManager;
this.olapClient1.DataBind();

```

VB.NET

```

Dim connectionString As String = @"Datasource =
systemdrive:\OfflineCube\Adventure_Works_Ext.cub; Provider= msolap;"
Dim olapDataManager As OlapDataManager = New
OlapDataManager(connectionString)

```

```
Me.olapClient1.OlapDataManager = olapDataManager  
Me.olapClient1.DataBind()
```

Binding OLAP client to cube in local SQL Server

To connect to the OLAP cube available in SQL Server Analysis Service in the local machine, set the server name and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code example illustrates the same.

C#

```
string connectionString = "Data source=localhost; Initial Catalog=Adventure  
Works DW";  
OlapDataManager olapDataManager = new OlapDataManager(connectionString);  
this.olapClient1.OlapDataManager = olapDataManager;  
this.olapClient1.DataBind();
```

VB.NET

```
Dim connectionString As String = "Datasource = localhost; Initial  
Catalog=Adventure Works DW"  
Dim olapDataManager As OlapDataManager = New  
OlapDataManager(connectionString)  
Me.olapClient1.OlapDataManager = olapDataManager  
Me.olapClient1.DataBind()
```

Binding OLAP client to cube in online SQL Server

To connect to the OLAP cube available in SQL Server Analysis Service in the online server through XML/A, set the host server link and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code example illustrates the same.

C#

```
string connectionString = "Data  
Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure  
Works DW 2008 SE;";  
OlapDataManager olapDataManager = new OlapDataManager(connectionString);  
this.olapClient1.OlapDataManager = olapDataManager;  
this.olapClient1.DataBind();
```

VB.NET

```
Dim connectionString As String = "Data  
Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure  
Works DW 2008 SE;"  
Dim olapDataManager As OlapDataManager = New  
OlapDataManager(connectionString)  
Me.olapClient1.OlapDataManager = olapDataManager  
Me.olapClient1.DataBind()
```

Binding OLAP client to cube in online Mondrian Server

To connect to the OLAP cube available in Mondrian Server through XML/A, set the host server link and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code example illustrates the same.

C#

```
string connectionString = @"Data Source =  
http://localhost:8080/mondrian/xmla; Initial Catalog =FoodMart;";  
OlapDataManager olapDataManager = new OlapDataManager(connectionString);  
olapDataManager.DataProvider.ProviderName =  
Syncfusion.Olap.DataProvider.Providers.Mondrian;  
this.olapClient1.OlapDataManager = olapDataManager;  
this.olapClient1.DataBind();
```

VB.NET

```
Dim connectionString As String = @"Data Source =  
http://localhost:8080/mondrian/xmla; Initial Catalog =FoodMart;";  
Dim olapDataManager As OlapDataManager = New  
OlapDataManager(connectionString)  
olapDataManager.DataProvider.ProviderName =  
Syncfusion.Olap.DataProvider.Providers.Mondrian;  
Me.olapClient1.OlapDataManager = olapDataManager  
Me.olapClient1.DataBind()
```

Binding OLAP client to cube in online ActivePivot Server

To connect to the OLAP cube available in ActivePivot Server through XML/A, set the host server link and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code example illustrates the same.

C#

```
string connectionString = @"Data Source = http://localhost:8080/cva_s/xmla;  
Initial Catalog = CVAS;";  
OlapDataManager olapDataManager = new OlapDataManager(connectionString);  
olapDataManager.DataProvider.ProviderName =  
Syncfusion.Olap.DataProvider.Providers.ActivePivot;  
this.olapClient1.OlapDataManager = olapDataManager;  
this.olapClient1.DataBind();
```

VB.NET

```
Dim connectionString As String = @"Data Source =  
http://localhost:8080/cva_s/xmla; Initial Catalog = CVAS;";  
Dim olapDataManager As OlapDataManager = New  
OlapDataManager(connectionString)  
olapDataManager.DataProvider.ProviderName =  
Syncfusion.Olap.DataProvider.Providers.ActivePivot;  
Me.olapClient1.OlapDataManager = olapDataManager  
Me.olapClient1.DataBind()
```

OlapClient: Elements in WPF Olap Client

Cube Selector

The cube selector allows you to select any one of the cubes available in the connected database. This can be achieved with a drop-down list displaying the list of cube names. On selecting a cube from the drop-down list, the corresponding cube element gets loaded.



Cube dimension browser

The cube dimension browser is a tree view-like structure that organizes the cube elements such as dimensions, hierarchies, and measures from the selected cube into independent logical groups.

Types of nodes in the cube dimension browser

- **Display folder:** A folder that contains a set of similar elements.
- **Measure:** Quantity available for analysis.
- **Dimension:** A name given to parts of the cube that categorize data.
- **Attribute hierarchy:** Level of attributes down the hierarchy.
- **User-defined hierarchy:** Members of a dimension in a hierarchical structure.
- **Level:** Denotes a specific level in the category.
- **Named Set:** A collection of tuples and members that can be defined and saved as a part of a cube definition.

Attribute hierarchy

Attribute hierarchy contains the following levels:

- A leaf level contains distinct attribute members and each member of the leaf level is known as a leaf member.
- Intermediate levels exist if the attribute hierarchy is a parent-child hierarchy.


User-defined hierarchy







A user-defined hierarchy organizes the members of a dimension into a hierarchical structure and provides the navigation paths in a cube. For example, take a dimension table that supports three attributes such as year, quarter, and month. These attributes are used to construct a user-defined hierarchy, named calendar, in the time dimension that relates to all levels.

Differentiating user-defined hierarchies and attribute hierarchies

- A user-defined hierarchy contains more than one level, whereas an attribute hierarchy contains only one level.
- A user-defined hierarchy provides a navigation path between the levels taken from the attribute hierarchies of the same dimension.

Symbolic representation of the nodes in cube dimension browser

Icon	Name	Node type	Is Draggable
	Display Folder	Display Folder	False

	Measure	Measure	True
	Dimension	Dimension	True
	User Defined Hierarchy	Hierarchy	True
	Attribute Hierarchy	Hierarchy	True
![[Levels icon]](OlapClient-Elementsimages/OlapClient-Elementsimg7.png) ![[Levels icon]](OlapClient-Elementsimages/OlapClient-Elementsimg8.png) 	Levels (in order)	Level Element	True
	Named Set	Named Set	True

Axis element builder

The axis element builder allows you to build elements in the axes of OLAP client. This supports three axes: categorical, series, and slicer. Based on the construction of the axes, the OLAP grid and OLAP chart will display the resultant data.

Categorical (column)

The categorical axis defines one or more elements that are displayed along the chart's y-axis as labels in the columns of the grid. If more than one dimension is present on the categorical axis, then the chart/grid will stack each dimension. The stacking order of dimensions is based on the order that they appear on the categorical axis.

Columns (Categorical)

Measures

Series (row)

The series axis defines one or more dimensions that are displayed along the chart's x-axis as labels and in the rows of the grid. If more than one dimension is present on the series axis, then the chart or grid will stack each dimension. The stacking order of dimensions is based on the order that they appear on the series axis.

Rows (Series)

Date

Slicer

The slicer axis is used as a filter to narrow the focus of the multidimensional data displayed in the chart or grid. It allows you to analyze a member of the dimension in depth. To display the member's data in the slicer, the corresponding member must not be present on both the categorical axis and series axis.

Slicer

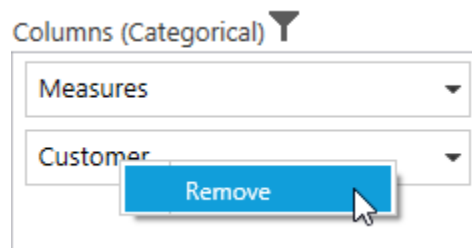
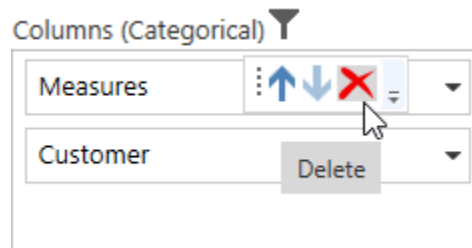


Adding elements to an axis

The measure, dimension, hierarchy, level, and named set elements can be dragged from the cube dimension browser and dropped into the axis element builder at the desired position by using the drag-and-drop operation.

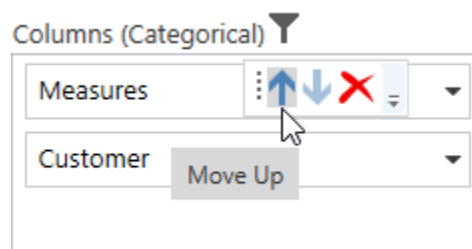
Removing elements from an axis

To remove measures, dimensions, hierarchy, levels, and named set elements from the axis element builder, hover over the element and click the delete icon. You can also use the context menu to remove an element by right-clicking on it.



Rearrange elements in an axis

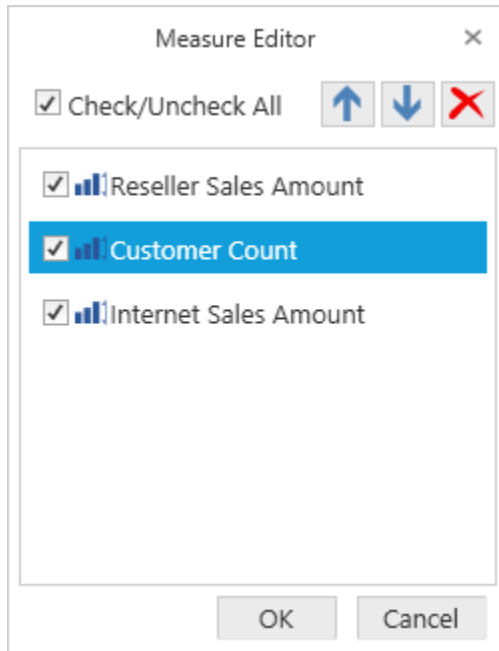
Rearranging can be done using Move Up/Move Down options visible when hovering over an element.



Elements editor

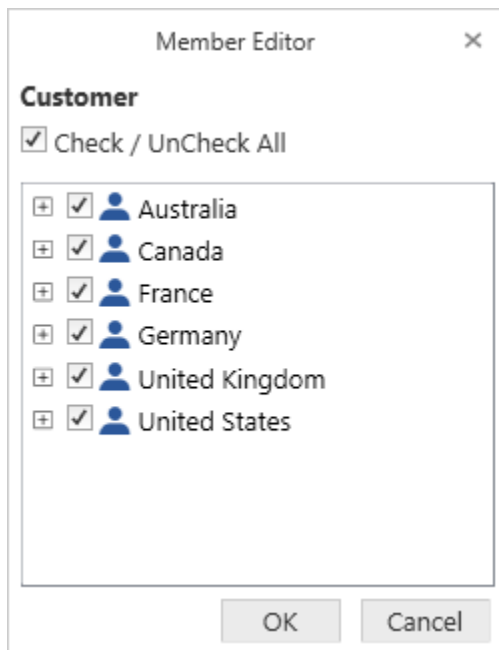
Measure editor

Measure editor is a dialog that displays the collection of measures in the current report. It can be opened by clicking the split button at the right corner of the measure node in the axis element builder.



Member editor

Member editor is a dialog that displays the members of the current hierarchy in a tree view structure. It can be opened by clicking the split button at the right corner of the member node in the axis element builder.



The Check All and Uncheck All options in the dialog allows you to select and deselect the entire nodes for filtering.

Toolbar



The options available in the toolbar are:

- **Connect to server:** Connects the data source with OLAP client through the offline cube, online server, or connection string.
- **New report:** Creates a new report list and clears the existing report collection to provide a new platform for new deployment based on the existing cube elements.
- **Load report:** Picks a saved report collection from the database and loads it by clearing the existing collection of reports.
- **Save report:** Stores the report collection at that instant in the local system.
- **SaveAs report:** Store a copy of the report collection with a new name in the local system.
- **Add report:** Adds a new report to the existing list of reports.
- **Remove report:** Removes the current report from the report list. You cannot remove the report, if the report list contains only one report in it.
- **Rename report:** Changes the name of the current report.
- **Toggle axis:** Interchanges the items between categorical and series axes.
- **Show expander:** Displays an expander option for the grid and chart to perform drill-down operations.
- **Filter/sort column:** Filters or sorts the data in the OLAP report with respect to the column.
- **Filter/sort row:** Filters or sorts the data in the OLAP report with respect to the row.
- **Report list:** Holds all reports of the current session of the OLAP client control and displays them in a drop-down list. You can select a report from the drop-down list.



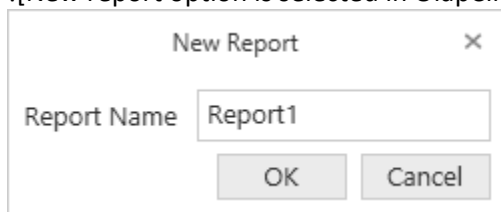
- **MDX query:** Displays the executed MDX query to retrieve the data at that instant.

Report manipulation

New report

This option helps users create a new report collection with a single report by clearing the existing report collection. By clicking the new report icon on the toolbar, the new report dialog opens, prompting you for a name for the report.

![[New report option is selected in OlapClient toolbar](OlapClient-Elements



images/OlapClient-Elementsimg20.png)

![New report dialog](OlapClient-Elements



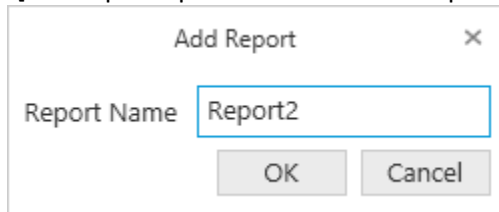
images/OlapClient-Elementsimg21.png)

After giving the required name, click OK. The report collection gets loaded with only one empty report having the entered name. By clicking Cancel, the new report creation gets canceled.

Add report

This option allows users to add a report to the existing report collection. By clicking the add report icon on the toolbar, the add report dialog opens prompting you for a name for the report.

![Add report option is selected in OlapClient toolbar](OlapClient-Elements



images/OlapClient-Elementsimg22.png)

![Add report dialog](OlapClient-Elements



images/OlapClient-Elementsimg23.png)

After giving an appropriate name in the respective column, click OK. A report with the entered name is added to the collection. By clicking Cancel, the report creation gets canceled.

Remove report

This option removes the current or active report from the report collection. This option works only if the report collection has more than one report.

![Remove report option is selected in OlapClient toolbar](OlapClient-Elements

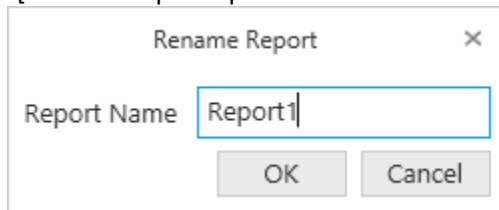


images/OlapClient-Elementsimg24.png)

Rename report

This option allows users to rename the current or active report. By clicking the rename icon on the toolbar, the rename report dialog opens, prompting users for a new name.

![Rename report option is selected in OlapClient toolbar](OlapClient-Elements



images/OlapClient-Elementsimg25.png)

![Rename report dialog](OlapClient-Elements



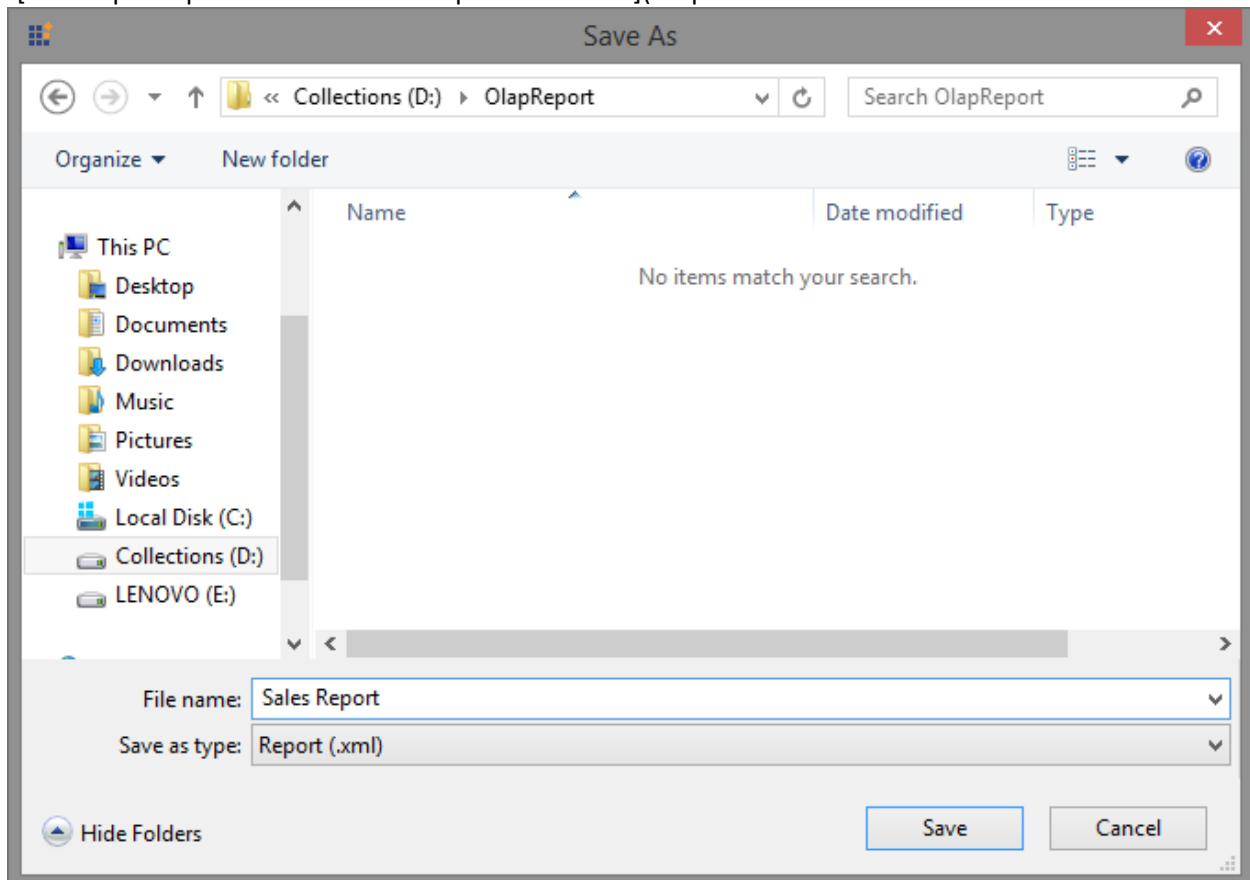
images/OlapClient-Elementsimg26.png)

After providing an appropriate name, click OK. The active report is refreshed with the new name. By clicking Cancel, the operation gets canceled.

[Save report](#)

This option saves the report in the local system. By clicking the save report icon, the SaveAs report dialog opens prompting you for a name with which the report needs to be stored.

![Save report option is selected in OlapClient toolbar](OlapClient-Elements



images/OlapClient-Elementsimg27.png)

![To save the report using file dialog](OlapClient-Elementsimages/OlapClient-Elementsimg28.png)

After providing an appropriate name, click

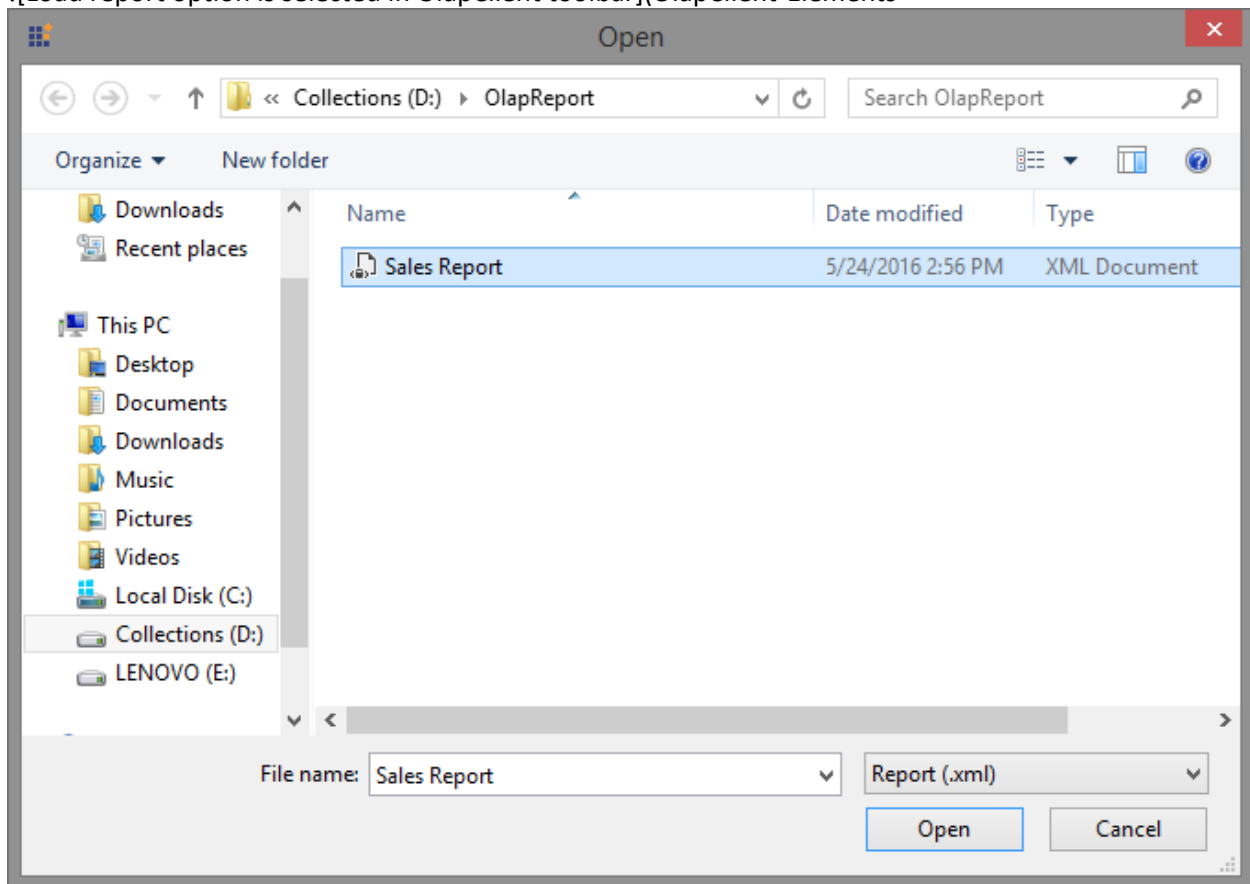


Save. The report will be saved in the selected system location.

[Load report](#)

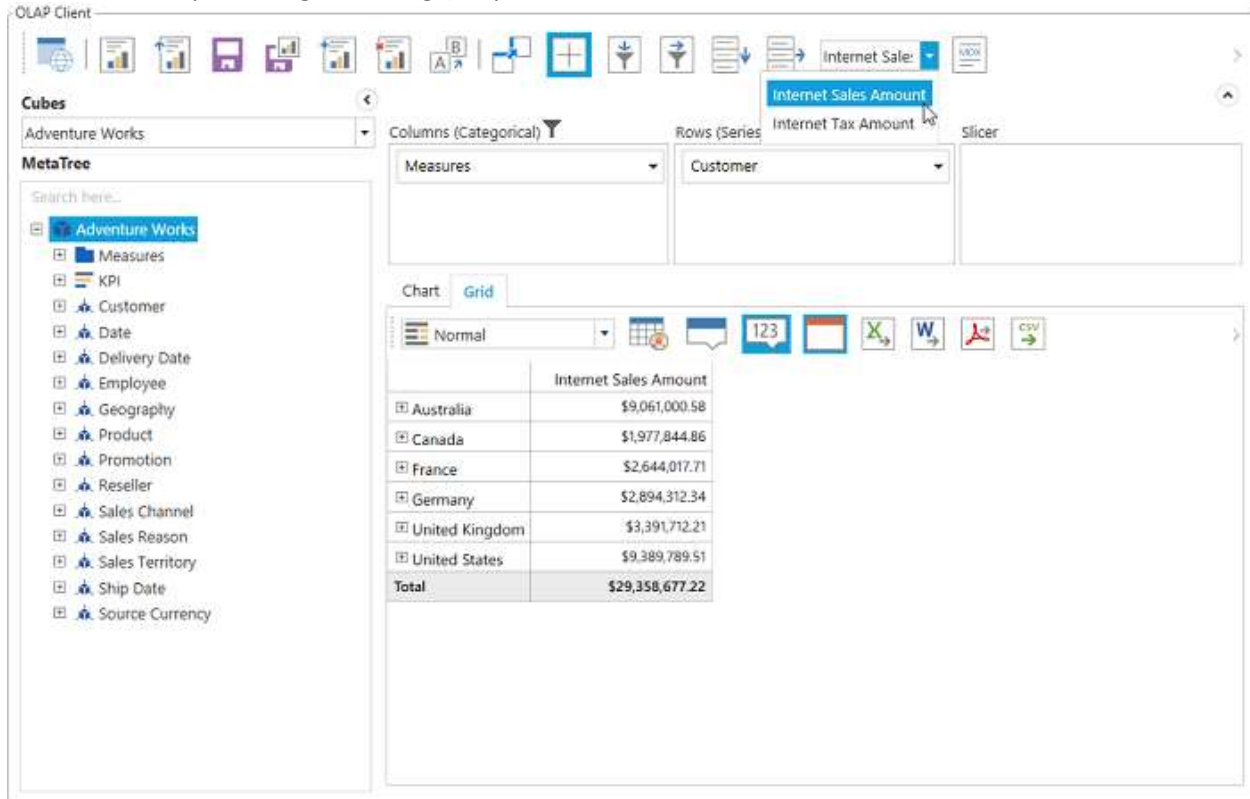
This option loads a report from the local system. Click the load report icon on the toolbar to load a report into the OLAP client.

! [Load report option is selected in OlapClient toolbar] (OlapClient-Elements



images/OlapClient-Elementsimg29.png)

! [To load the report using file dialog] (OlapClient-Elements



images/OlapClient-Elementsimg30.png)

Report list

The report list drop-down contains the names of all the reports in the report collection.

! [To change the reports at runtime using Report list] (OlapClient-Elementsimages/OlapClient-Elementsimg31.png)

Select the required report from the report list. The selected report will be set as an active report and loaded.

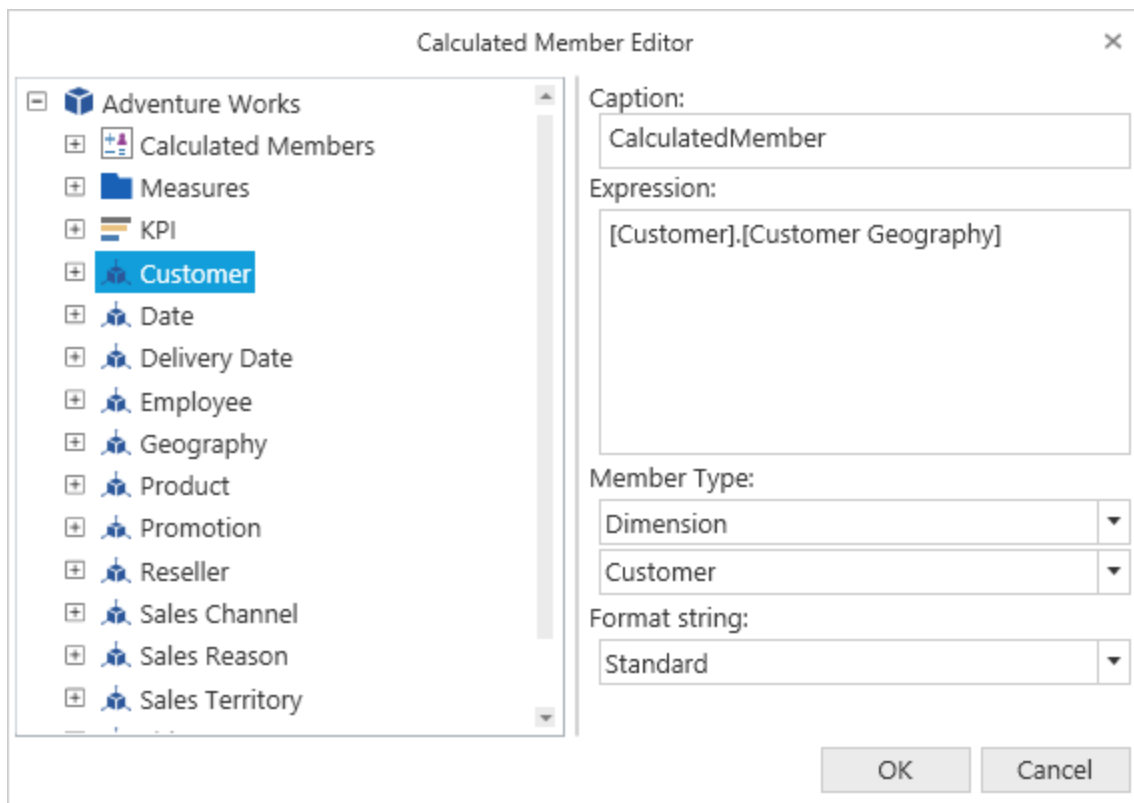
OLAP grid and OLAP chart

The [OLAP grid](#) and [OLAP chart](#) controls will be rendered with respect to the operations done in the axis element builder.

Calculated Members in WPF Olap Client

This feature allows users to define measures and members using the calculated member editor. The calculated member editor can be opened just by clicking the respective icon available in the OLAP client toolbar. The icon will be visible only by setting the `IsCalculatedMembersEnabled` property to true.



**XML**

```
<CheckBox Name="chk_CalcMember"
ToolTip="Enable/Disable Calculated Members" Content="Enable Calculated Members"
IsChecked="{Binding ElementName=olapClient1, Path=IsCalculatedMembersEnabled}" />
```

C#

```
this.olapClient1.IsCalculatedMembersEnabled = true;
```

VB.NET

```
Me.olapClient1.IsCalculatedMembersEnabled = True
```

A sample demo is available at the following location.

```
{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version
Number>\WPF\OlapClient.WPF\Samples\Product Showcase\CalculatedMembers
```

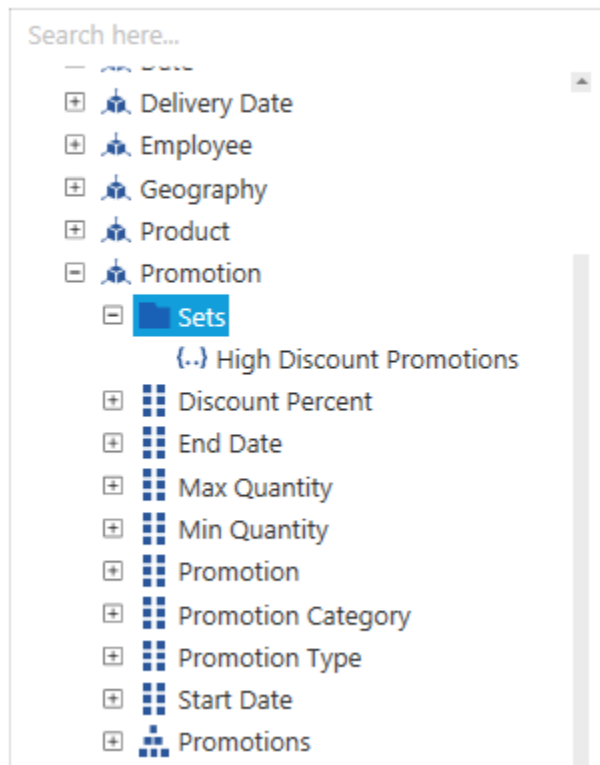
Named Set in WPF Olap Client

The OLAP client supports the binding of OLAP data with named set records pre-defined in the cube. A named set is a collection of tuples and members, which can be defined and saved as a part of the cube definition. Named set records reside inside the sets folder, which is under a dimension element. These elements can be dragged to categories/series/slicer axis of the axis element builder. To easily work with

a lengthy, complex, or commonly used expression, use Multidimensional Expressions (MDX) that allows you to define a named set.

The cube dimension browser displays the dimensions, measures, and KPIs along with named sets from the selected cube inside a tree on the left. To visualize these members, you can drag the members to the axis element builder.

MetaTree



Drill Through in WPF Olap Client

Drill-through retrieves raw data that are used to create a specified cell in a cube. You can enable or disable the drill-through action using the `EnableDrillThrough` property.

The following code sample explains how to enable the drill-through option in your application.

C#

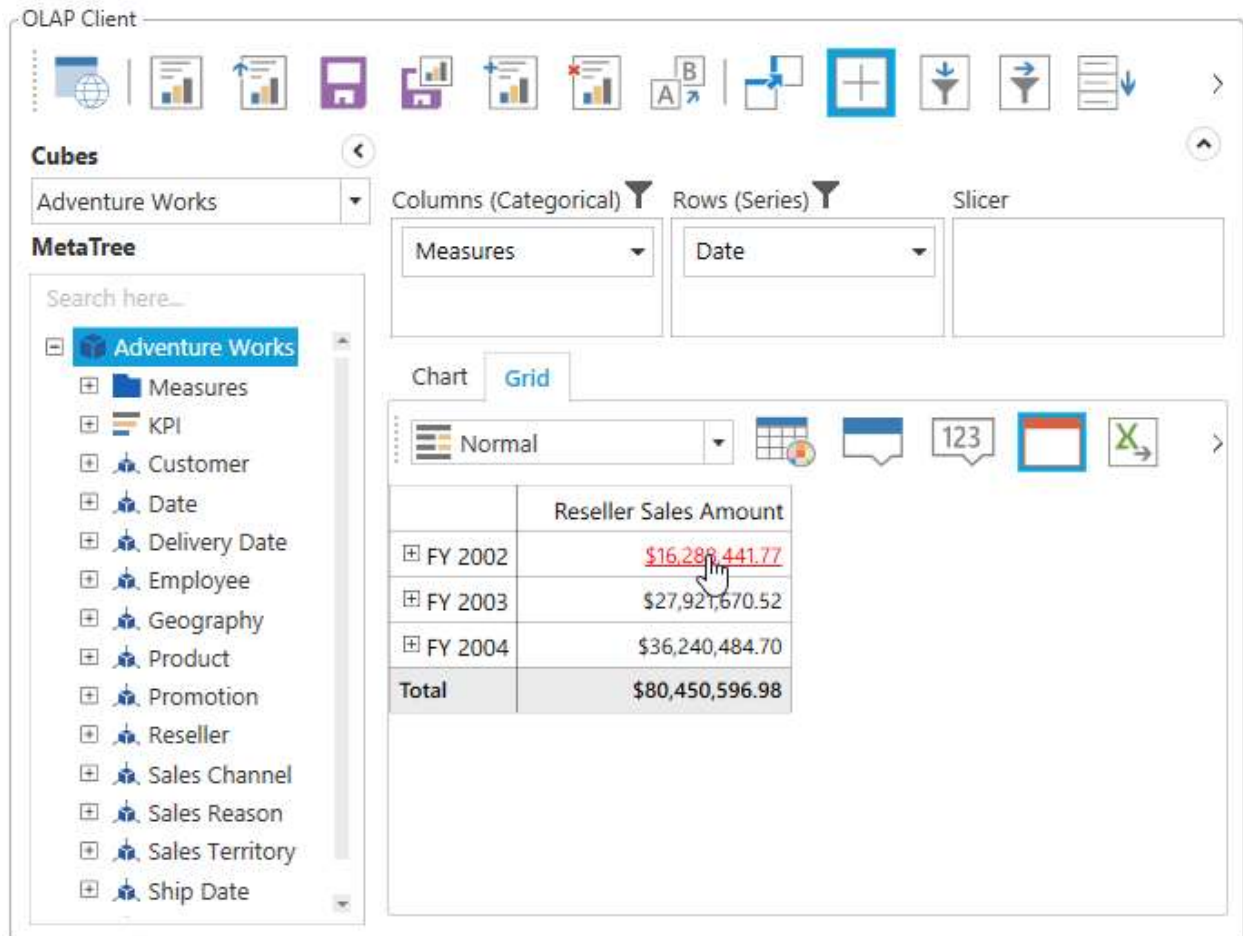
```
this.olapClient1.DisplayMode =
Syncfusion.Windows.Client.Olap.DisplayModes.GridOnly;
this.olapClient1.OlapGrid.ValueCellStyle.IsHyperlinkCell = true;
this.olapClient1.OlapGrid.EnableDrillThrough = true;
this.olapClient1.OlapGrid.LinkClick += new
Syncfusion.Windows.Grid.Olap.LinkLabelClickEventHandler(OlapGrid_LinkClick);
void OlapGrid_LinkClick(object sender,
Syncfusion.Windows.Grid.Olap.LinkLabelEventArgs e)
{
    DataTable DrillThroughData = e.DrillThroughData;
}
```

VB.NET

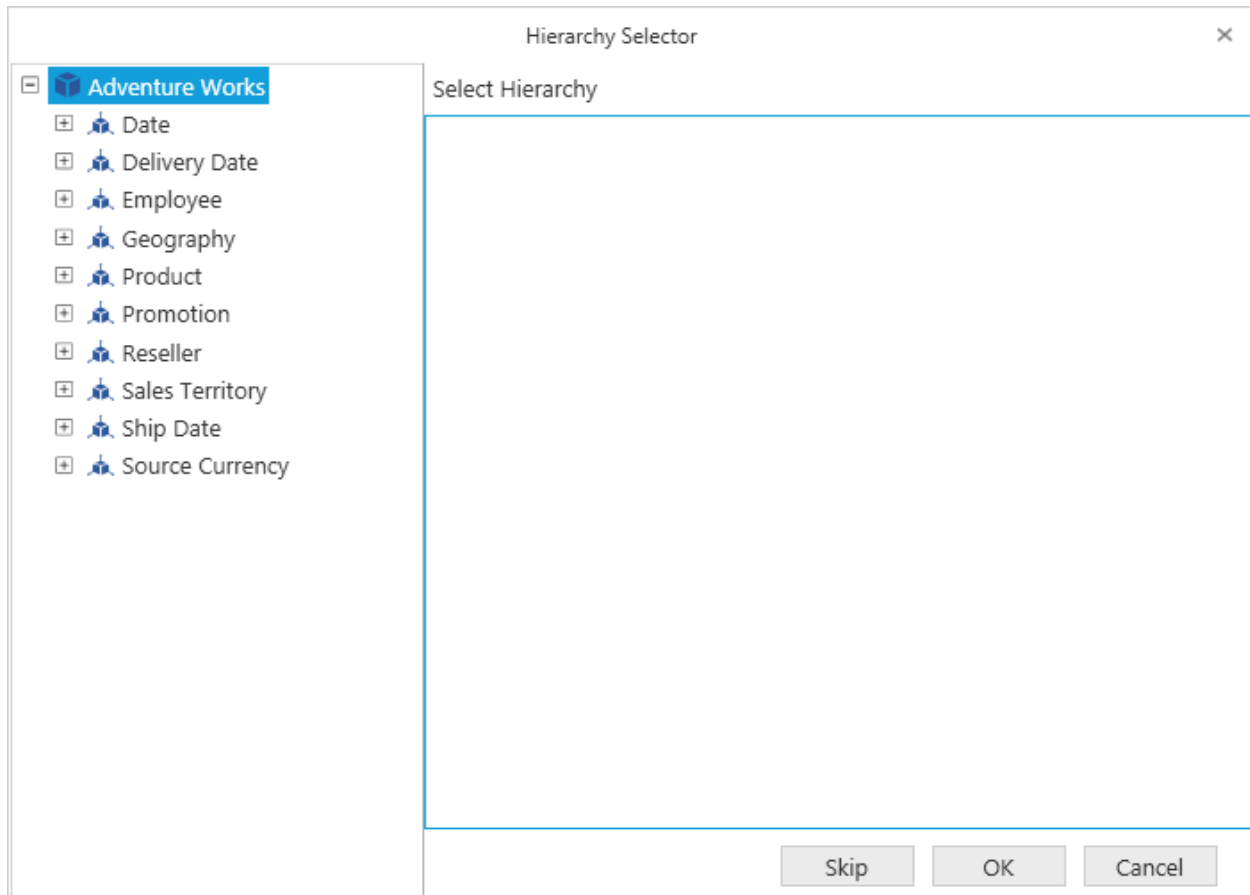
```

Me.olapClient1.DisplayMode =
Syncfusion.Windows.Client.Olap.DisplayModes.GridOnly
Me.olapClient1.OlapGrid.ValueCellStyle.IsHyperlinkCell = True
Me.olapClient1.OlapGrid.EnableDrillThrough = True
Me.olapClient1.OlapGrid.LinkClick += New
Syncfusion.Windows.Grid.Olap.LinkLabelClickEventHandler(OlapGrid_LinkClick)
Private Sub OlapGrid_LinkClick(sender As Object, e As
Syncfusion.Windows.Grid.Olap.LinkLabelEventArgs)
Dim DrillThroughData As DataTable = e.DrillThroughData
End Sub

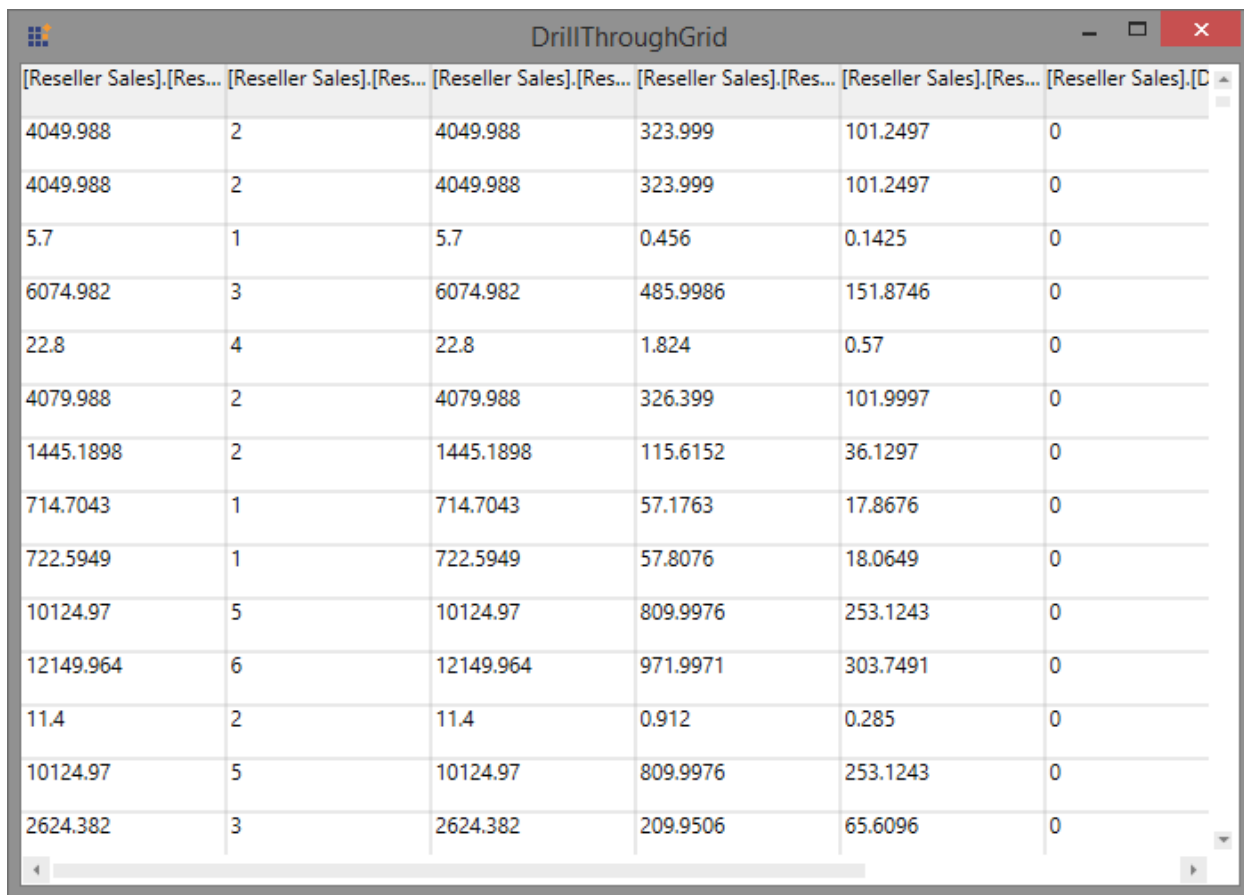
```



Hyperlink Cell Click



Attribute Hierarchy Selector



[Reseller Sales].[Res...]	[Reseller Sales].[Res...]	[Reseller Sales].[Res...]	[Reseller Sales].[Res...]	[Reseller Sales].[Res...]	[Reseller Sales].[Res...]
4049.988	2	4049.988	323.999	101.2497	0
4049.988	2	4049.988	323.999	101.2497	0
5.7	1	5.7	0.456	0.1425	0
6074.982	3	6074.982	485.9986	151.8746	0
22.8	4	22.8	1.824	0.57	0
4079.988	2	4079.988	326.399	101.9997	0
1445.1898	2	1445.1898	115.6152	36.1297	0
714.7043	1	714.7043	57.1763	17.8676	0
722.5949	1	722.5949	57.8076	18.0649	0
10124.97	5	10124.97	809.9976	253.1243	0
12149.964	6	12149.964	971.9971	303.7491	0
11.4	2	11.4	0.912	0.285	0
10124.97	5	10124.97	809.9976	253.1243	0
2624.382	3	2624.382	209.9506	65.6096	0

Grid with drill-through information

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapClient.WPF\Samples\Product Showcase\Drill Through

Serialization in WPF Olap Client

This feature allows users to store the current session as a stream anywhere and load the saved stream back to populate the OLAP client control.

The following code sample describes getting the report as stream.

C#

```
Stream stream = this.olapClient1.GetReportStream();
```

VB.NET

```
Dim stream As Stream = Me.olapClient1.GetReportStream()
```

The following code sample illustrates loading the report which is in stream type to OLAP client.

C#

```
this.olapClient1.LoadReportStream(reportStream);
```

VB.NET

```
Me.olapClient1.LoadReportStream(reportStream)
```

The following code sample describes

- storing of the report as stream in database and
- retrieving the report as stream from the database and loading it back to OLAP client.

Storing report as stream in database**C#**

```
Stream stream = this.olapClient1.GetReportStream();
if (stream != null)
{
    MemoryStream reportStream = stream as MemoryStream;
    ReportNameWindow saveReport = new ReportNameWindow(this.ReportNames);
    saveReport.WindowStartupLocation = WindowStartupLocation.CenterOwner;
    saveReport.Owner = App.Current.Windows[0];
    if (saveReport.ShowDialog() == true)
    {
        string repotName = saveReport.reportName;
        con.Open();
        SqlCeCommand sqlCmd = new SqlCeCommand("insert into ReportsTable
        Values (@ReportName,@Report)", con);
        sqlCmd.Parameters.Add("@ReportName", repotName);
        sqlCmd.Parameters.Add("@Report", reportStream.ToArray());
        sqlCmd.ExecuteNonQuery();
        con.Close();
        this.ReportNames.Add(saveReport.reportName);
    }
}
```

VB.NET

```
Dim stream As Stream = Me.olapClient1.GetReportStream()
If stream IsNot Nothing Then
    reportStream = TryCast(stream, MemoryStream)
    Me.saveReport = New ReportNameWindow(Me.reportNames)
    If saveReport.ShowDialog() = True Then
        Dim repotName As String = "SalesReport"
        con.Open()
        Dim cmd1 As SqlCeCommand = New SqlCeCommand("insert into ReportsTable
        Values (@ReportName,@Report)", con)
        cmd1.Parameters.Add("@ReportName", repotName)
        cmd1.Parameters.Add("@Report", reportStream.ToArray())
        cmd1.ExecuteNonQuery()
        con.Close()
    End If
End If
```

Loading report as stream from database**C#**

```

string reportname = "RevenueReport";
Stream reportStream = null;
con.Open();
SqlCeDataAdapter da = new SqlCeDataAdapter("Select * from ReportsTable",
con);
DataSet dSet = new DataSet();
da.Fill(dSet);
DataTable table = dSet.Tables[0];
foreach (DataRow row in table.Rows)
{
    if ((row.ItemArray[0] as string).Equals(reportname))
    {
        reportStream = new MemoryStream(row.ItemArray[1] as byte[]);
        break;
    }
}
this.olapClient1.LoadReportStream(reportStream);
con.Close();

```

VB.NET

```

Dim reportname As String = "RevenueReport"
Dim reportStream As Stream = Nothing
con.Open()
Dim da As SqlCeDataAdapter = New SqlCeDataAdapter("Select * from
ReportsTable", con)
Dim dSet As DataSet = New DataSet()
da.Fill(dSet)
Dim table As DataTable = dSet.Tables(0)
For Each row As DataRow In table.Rows
    If (TryCast(row.ItemArray(0), String).Equals(reportname) Then
        reportStream = New MemoryStream(TryCast(row.ItemArray(1), Byte()))
    Exit For
    End If
Next row
Me.olapClient1.LoadReportStream(reportStream)
con.Close()

```

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapClient.WPF\Samples\Serialization\Report Serialization

Paging in WPF Olap Client

Paging in the OLAP client supports you to load and render the large amount of data without any performance constraint.

The OLAP pager (user control) is included and bound with the OlapDataManager object of the respective OLAP client. To enable paging, set the **EnablePaging** property to true.

When you process the large CellSet, it is split into several number of segments and each segment is assigned and rendered in a separate page. You can navigate back and forth in all possible ways by using the UI options in the OLAP pager. You can also change the page size and other pager settings at runtime by using the **PageSetting** window.

Include the following Syncfusion assembly from the installed location to add the OLAP pager (user control) with OLAP client.

- Syncfusion.OlapShared.Wpf

Note: You can also get the assemblies by browsing to the default assembly location: {System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<framework version>\

Enable paging through XAML

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="SampleApplication.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<GroupBox Header="OlapClient" Grid.Row="0">
<syncfusion:OlapClient Name="olapClient" EnablePaging="True"
Background="Transparent" SeriesStrokeThickness="0">
</syncfusion:OlapClient>
</GroupBox>
</Grid>
</Window>
```

Enable paging through report:

C#

```
using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
namespace SampleApplication
{
    public partial class MainWindow : SampleWindow
    {
        private string _connectionString;
        private OlapDataManager _olapDataManager;
        public MainWindow()
        {
            InitializeComponent();
            _connectionString = "Enter a valid connection string";
            //Created connection string is passed to OlapDataManager as argument
            _olapDataManager = new OlapDataManager(_connectionString);
            //Created OlapReport is set as a current report to OlapDataManager
            _olapDataManager.SetCurrentReport(SimpleDimensions());
            //Finally OlapClient control gets the data from the created OlapDataManager
            this.olapClient.OlapDataManager = _olapDataManager;
            this.olapClient.DataBind();
        }
    }
}
```



```

private OlapReport SimpleDimensions()
{
    OlapReport olapReport = new OlapReport();
    olapReport.CurrentCubeName = "Adventure Works";
    olapReport.EnablePaging = true;
    olapReport.PagerOptions.CategoricalPageSize = 10;
    olapReport.PagerOptions.SeriesPageSize = 10;
    DimensionElement dimensionElement = new DimensionElement() { Name =
    "Customer", HierarchyName = "Customer" };
    dimensionElement.AddLevel("Customer Geography", "Country");
    olapReport.CategoricalElements.Add(dimensionElement);
    MeasureElements measureElements = new MeasureElements();
    measureElements.Add(new MeasureElement { Name = "Internet Sales Amount" });
    olapReport.SeriesElements.Add(measureElements);
    dimensionElement = new DimensionElement() { Name = "Geography",
    HierarchyName = "Geography" };
    dimensionElement.AddLevel("Geography", "Country");
    olapReport.CategoricalElements.Add(dimensionElement);
    dimensionElement = new DimensionElement() { Name = "Date" };
    dimensionElement.AddLevel("Fiscal", "Fiscal Year");
    olapReport.SeriesElements.Add(dimensionElement);
    return olapReport;
}
}
}

```

VB.NET

```

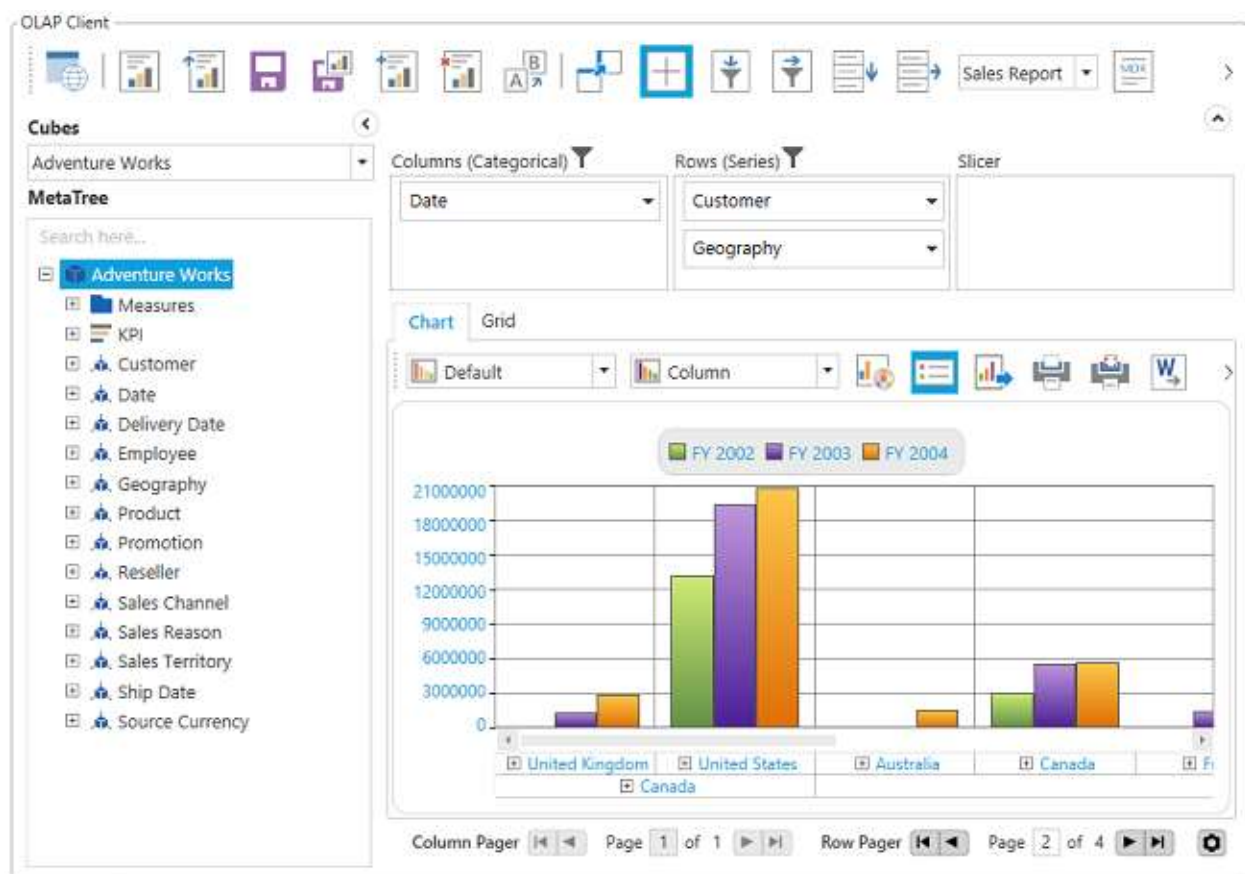
Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Namespace SampleApplication
Partial Public Class MainWindow
Inherits SampleWindow
Private _connectionString As String
Private _olapDataManager As OlapDataManager
Public Sub New()
InitializeComponent()
_connectionString = "Enter a valid connection string"
'Created connection string is passed to OlapDataManager as argument
_olapDataManager = New OlapDataManager(_connectionString)
'Created OlapReport is set as a current report to OlapDataManager
_olapDataManager.SetCurrentReport(SimpleDimensions())
'Finally OlapClient control gets the data from the created OlapDataManager
Me.olapClient.OlapDataManager = _olapDataManager
Me.olapClient.DataBind()
End Sub
Private Function SimpleDimensions() As OlapReport
Dim olapReport As New OlapReport()
olapReport.CurrentCubeName = "Adventure Works"
olapReport.EnablePaging = True
olapReport.PagerOptions.CategoricalPageSize = 10
olapReport.PagerOptions.SeriesPageSize = 10
Dim dimensionElement As New DimensionElement() With {.Name = "Customer",
.HierarchyName = "Customer"}
dimensionElement.AddLevel("Customer Geography", "Country")
olapReport.CategoricalElements.Add(dimensionElement)

```

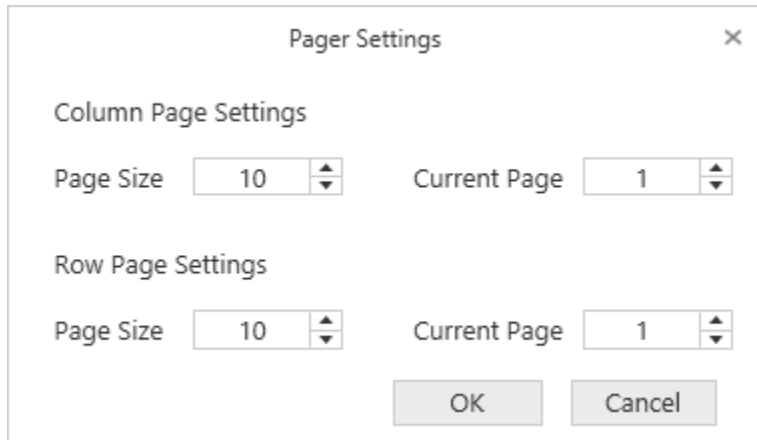
```

Dim measureElements As New MeasureElements()
measureElements.Add(New MeasureElement With {.Name = "Internet Sales Amount"})
olapReport.SeriesElements.Add(measureElements)
dimensionElement = New DimensionElement() With {.Name = "Geography", .HierarchyName = "Geography"}
dimensionElement.AddLevel("Geography", "Country")
olapReport.CategoricalElements.Add(dimensionElement)
dimensionElement = New DimensionElement() With {.Name = "Date"}
dimensionElement.AddLevel("Fiscal", "Fiscal Year")
olapReport.SeriesElements.Add(dimensionElement)
Return olapReport
End Function
End Class
End Namespace

```



OlapPager in OlapClient control



Page Setting Window

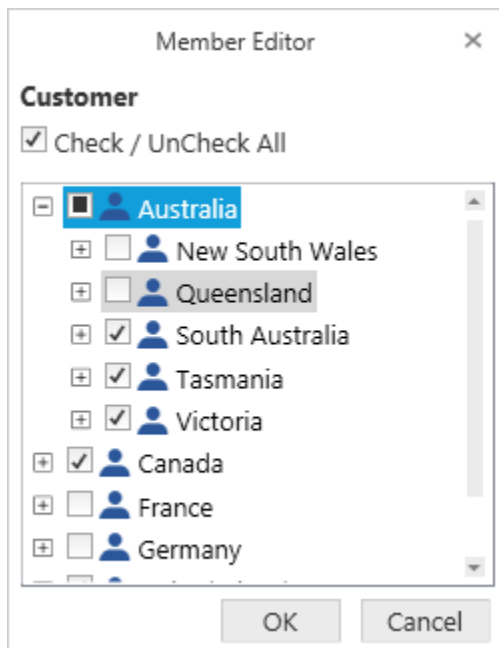
A sample demo is available at the following location.

{system drive}\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapClient.WPF\Samples\Paging\Paging

Filtering in WPF Olap Client

Filtering by member

After clicking the split button of a dimension, the member editor dialog opens through which members are filtered by checking and unchecking the check boxes corresponding to members. By clicking OK, the OLAP report gets updated and refreshes the OLAP grid and OLAP chart controls based on the selected members in the member editor dialog. The Cancel button is used for canceling the selection.



The above filter illustrates that the members "France" and "Germany" along with "New South Wales" and "Queensland" are filtered from the grid and chart views.

Filtering by value

The filter tab in the filtering and sorting dialog box provides the options to specify custom filters on the multidimensional data. It enables users to filter the rows and columns of the selected measure.

- **Column filter:** Column filter checks each and every row of a column against the filter condition. The column will be included in the result set only if all the rows of that column satisfy the condition; otherwise, the column will be filtered.
- **Row filter:** Row filter checks each and every column of a row against the filter condition. The row will be included in the result set only if all the columns of that row satisfy the condition; otherwise, the row will be filtered.

Filtering and Sorting dialog for rows/columns can be opened by clicking the corresponding icon in the toolbar.



Filtering by row



Filtering by column

The following screenshot displays the filter tab in Filtering and Sorting dialog.

The screenshot shows a dialog box titled "On Column" with a close button (X) in the top right corner. Inside the dialog, there is a section titled "Filtering and Sorting" with two tabs: "Filter" (selected) and "Sorting". Below the tabs, there is a section titled "Empty Results" with a checkbox labeled "Filter Empty Columns" which is currently unchecked. Below this, there are two filter sections. The first section, "Filter 1", is checked and contains a "Condition" dropdown set to "GreaterThan", a "Filter On" dropdown set to "Customer Count", and a "Value" input field containing "3000". The second section, "Filter 2", is unchecked and contains empty dropdowns for "Condition" and "Filter On", and an empty "Value" input field. At the bottom of the dialog, there are two buttons: "Update" and "Cancel".

The options in the filtering tab are as follows:

Filter empty rows/columns: Filters the empty rows or columns appeared in the result set.

Filter 1 and Filter 2: You can apply two filter expressions to a report at the same time. The options in the filter group box are as follows:

- **Condition:** You can choose any one condition required to appear in the filter expression.
- **Filter on:** You can choose any one measure element from the list, on which you want to apply the filter.
- **Value:** Enter the conditional value for the expression.

You can toggle the visibility of the filter and sort buttons in the OLAP client toolbar by using the `ShowFilterSortButtons` property.

C#

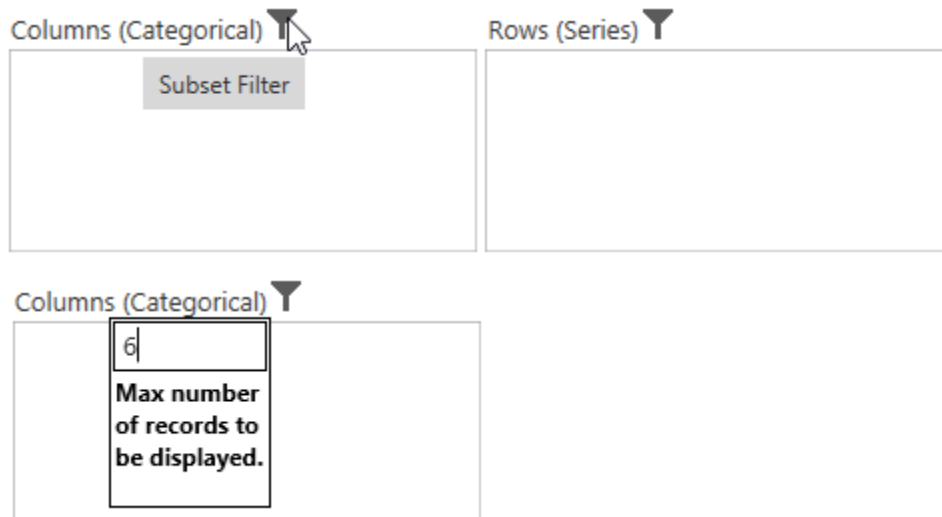
```
this.olapClient1.ShowFilterSortButtons = false;
```

VB.NET

```
Me.olapClient1.ShowFilterSortButtons = False
```

Subset filter

Subset filter is used to filter the number of records in the result set. The subset filter gets a numeric number as input and restricts the number of records within that count. You can specify the subset filter for both the row and column.



Users can toggle the visibility of subset filter by using the `ShowSubsetFilters` property.

C#

```
this.olapClient1.ShowSubsetFilters = false;
```

VB.NET

```
Me.olapClient1.ShowSubsetFilters = False
```

Sorting in WPF Olap Client

The sorting tab in the Filtering and Sorting dialog box provides an option to sort the results by rows or columns in ascending or descending order.

- **Column sorting:** Sorts the columns in the result set based on the column total of each column.
- **Row sorting:** Sorts the rows in the result set based on the row total of each row.

Filtering and Sorting dialog box for rows or columns can be opened by clicking the corresponding icon in the toolbar.

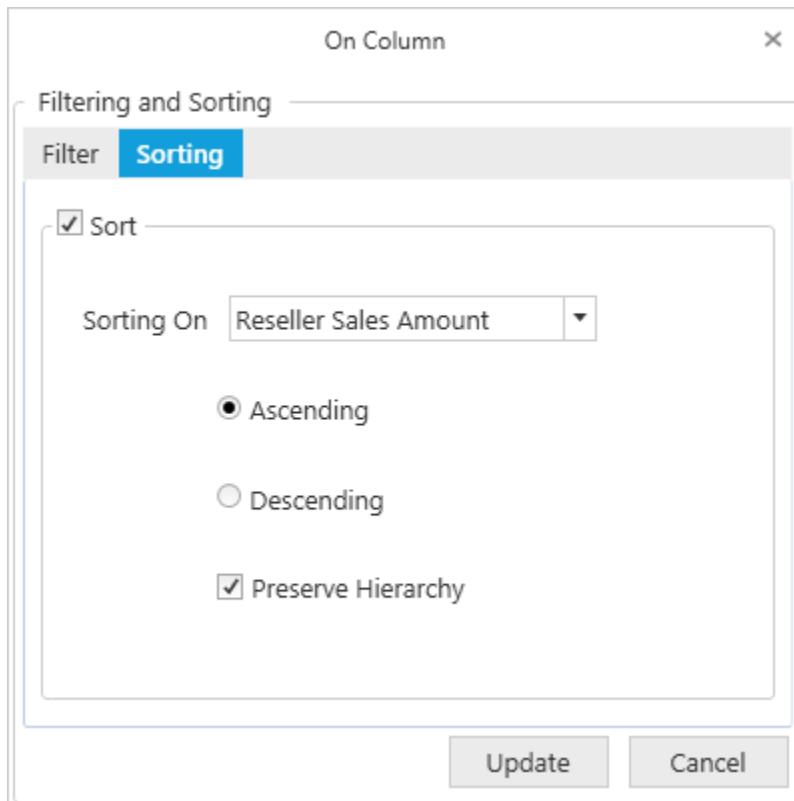


Sorting by row



Sorting by column

The following screenshot displays the sorting tab in the Filtering and Sorting dialog box.



The options in the sorting tab are as follows:

- **Sorting on:** Displays the list of measure elements to choose one to set as the key field.
- **Ascending or descending:** Specifies the sorting order.
- **Preserve hierarchy:** Sorts the records without changing the hierarchy order.

To toggle the visibility of the filter and sort buttons in the OLAP client toolbar, use the `ShowFilterSortButtons` property.

C#

```
this.olapClient1.ShowFilterSortButtons = false;
```

VB.NET

```
Me.olapClient1.ShowFilterSortButtons = False
```

Exporting in WPF Olap Client

When creating an OLAP report in the OLAP client, the report will be visualized in the OLAP chart and OLAP grid. The OLAP client has an option to export the current view of the OLAP chart and OLAP grid to various forms.

Exporting OLAP chart

The current view of the OLAP chart can be exported to Microsoft Word, PDF, and image and printed as well.





You can perform these exports in two ways:

- Through OLAP chart toolbar.
- Through API's.

Through OLAP chart toolbar menu

By clicking the respective icons in the OLAP chart toolbar, You can export the OLAP chart to the corresponding mode.



Icon	Name	Description
	Export to Image	Export the current view of the OLAP chart as image
	Export to Word	Export the current view of the OLAP chart to a Word document.
	Export to PDF	Export the current view of the OLAP chart to PDF.
	Print	Print the current view of the OLAP chart.

Through API

You can achieve the export feature of OLAP chart by using the following API's. The following code sample illustrates exporting the OLAP chart into corresponding format.

Word export

C#

```

SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.FileName = "OlapChart Report";
saveFileDialog.AddExtension = true;
saveFileDialog.DefaultExt = ".doc";
saveFileDialog.Filter = "Word (.doc)|*.doc";
if (saveFileDialog.ShowDialog() == true)
{
    string fileName = saveFileDialog.FileName;
    OlapChartWordExport olapChartWordExport = new
    OlapChartWordExport(this.olapChart);
    olapChartWordExport.ExportintoNewDoc(fileName);
}

```

VB.NET

```

Dim saveFileDialog As SaveFileDialog()
saveFileDialog.FileName = "OlapChart Report"
saveFileDialog.AddExtension = True
saveFileDialog.DefaultExt = ".doc"
saveFileDialog.Filter = "Word (.doc)|*.doc"
If saveFileDialog.ShowDialog() = True Then
Dim fileName As String = saveFileDialog.FileName
Dim olapChartWordExport As New OlapChartWordExport(Me.olapChart)
olapChartWordExport.ExportintoNewDoc(fileName)
End If

```

PDF export

C#

```

SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.FileName = "OlapChart Report";
saveFileDialog.AddExtension = true;
saveFileDialog.DefaultExt = ".pdf";
saveFileDialog.Filter = "PDF (.pdf)|*.pdf";
if (saveFileDialog.ShowDialog() == true)
{
    string fileName = saveFileDialog.FileName;
    OlapChartPdfExport chartPdfExport = new OlapChartPdfExport(this.olapChart);
    chartPdfExport.ExportIntoNewPdf(fileName);
}

```

VB.NET

```

Dim saveFileDialog As SaveFileDialog()
saveFileDialog.FileName = "OlapChart Report"
saveFileDialog.AddExtension = True
saveFileDialog.DefaultExt = ".pdf"
saveFileDialog.Filter = "PDF (.pdf)|*.pdf"
If saveFileDialog.ShowDialog() = True Then
Dim fileName As String = saveFileDialog.FileName
Dim olapChartPdfExport As New OlapChartPdfExport(Me.olapChart)
olapChartPdfExport.ExportIntoNewPdf(fileName)
End If

```


Image export

C#

```

private const string c_imageFilesFilter =
    "Bitmap(*.bmp)|*.bmp|JPEG(*.jpg,*.jpeg)|*.jpg;*.jpeg|GIF(*.gif)|*.gif|TIFF(*.tiff)|*.tiff|PNG(*.png)|*.png|WDP(*.wdp)|*.wdp|All files (*.*)|*.*";
SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.FileName = "OlapChart Report";
saveFileDialog.Filter = c_imageFilesFilter;
if (saveFileDialog.ShowDialog() == true)
{
    string fileName = saveFileDialog.FileName;
    string extension = new
    FileInfo(fileName).Extension.ToLower(CultureInfo.InvariantCulture);
    if (c_imageFilesFilter.Contains(extension))
    {
        using (Stream stream = File.Create(saveFileDialog.FileName))
        {
            BitmapEncoder encoder = Chart.CreateBitmapEncoderByExtension(extension);
            RenderTargetBitmap bmpSource = new
            RenderTargetBitmap((int)chart.ActualWidth, (int)chart.ActualHeight, 96, 96,
            PixelFormats.Default);
            Rectangle backgroundRect = new Rectangle();
            backgroundRect.Fill = Brushes.White;
            backgroundRect.Arrange(new Rect(chart.RenderSize));
            bmpSource.Render(backgroundRect);
            bmpSource.Render(visual);
            encoder.Frames.Add(BitmapFrame.Create(bmpSource));
            encoder.Save(stream);
        }
    }
}

```

VB.NET

```

Dim c_imageFilesFilter As String =
    "Bitmap(*.bmp)|*.bmp|JPEG(*.jpg,*.jpeg)|*.jpg;*.jpeg|GIF(*.gif)|*.gif|TIFF(*.tiff)|*.tiff|PNG(*.png)|*.png|WDP(*.wdp)|*.wdp|All files (*.*)|*.*"
Dim saveFileDialog As SaveFileDialog()
saveFileDialog.FileName = "OlapChart Report"
saveFileDialog.Filter = c_imageFilesFilter;
If saveFileDialog.ShowDialog() = True Then
    Dim extension As String = New
    FileInfo(fileName).Extension.ToLower(CultureInfo.InvariantCulture);
    Dim fileName As String = saveFileDialog.FileName
    If c_imageFilesFilter.Contains(extension) Then
        Using stream As Stream = File.Create(saveFileDialog.FileName)
            Dim encoder As BitmapEncoder =
            Chart.CreateBitmapEncoderByExtension(extension)
            Dim bmpSource As New RenderTargetBitmap(CInt(chart.ActualWidth),
            CInt(chart.ActualHeight), 96, 96, PixelFormats.[Default])
            Dim backgroundRect As New Rectangle()
            backgroundRect.Fill = Brushes.White
            backgroundRect.Arrange(New Rect(chart.RenderSize))
            bmpSource.Render(backgroundRect);
            bmpSource.Render(visual);
        End Using
    End If
}

```

```

encoder.Frames.Add(BitmapFrame.Create(bmpSource));
encoder.Save(stream);
End Using
End If
End If

```

Exporting OLAP grid

The current view of the OLAP grid can be exported to the following forms:

- Microsoft Word
- PDF
- Microsoft Excel
- CSV





The user can perform these exports in two ways:

1. Through OLAP grid toolbar.
2. Through API's.

Through OLAP grid toolbar menu

By clicking the respective export buttons in the OLAP grid toolbar, the user can export OLAP grid to the corresponding format.



Icon	Name	Description
	Export to Excel	Export the OLAP grid to an Excel document.
	Export to Word	Export the OLAP grid to a Word document.
	Export to PDF	Export the OLAP grid to a PDF document.
	Export to CSV	Export the OLAP grid to a CSV document.

Through API

You can achieve the export feature of OLAP grid by using the following API's. The following code sample illustrates exporting the OLAP grid into corresponding format.

Excel export

C#

```

SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.FileName = "OlapGrid Report";
saveFileDialog.AddExtension = true;
saveFileDialog.DefaultExt = ".xlsx";
saveFileDialog.Filter = "Excel (.xlsx) | *.xlsx";
if (saveFileDialog.ShowDialog() == true)

```

```
{
    string fileName = saveFileDialog.FileName;
    GridExcelExport excelExport = new
    GridExcelExport(this.olapGrid.InternalGrid.Engine,
        this.olapGrid.GridStyleInfo, this.olapGrid.InternalGrid.Layout,
        saveFileDialog.DefaultExt, this.olapGrid.OlapDataManager.ItemSource == null
        ? false : true);
    excelExport.Export(fileName);
}
```

VB.NET

```
Dim saveFileDialog As SaveFileDialog()
saveFileDialog.FileName = "OlapGrid Report"
saveFileDialog.AddExtension = True
saveFileDialog.DefaultExt = ".xlsx"
saveFileDialog.Filter = "Excel (.xlsx)|*.xlsx"
If saveFileDialog.ShowDialog() = True Then
    Dim fileName As String = saveFileDialog.FileName
    Dim excelExport As New GridExcelExport(Me.olapGrid.InternalGrid.Engine,
        Me.olapGrid.GridStyleInfo, Me.olapGrid.InternalGrid.Layout,
        saveFileDialog.DefaultExt, Me.olapGrid.OlapDataManager.ItemSource == null ?
        False : True)
    excelExport.Export(fileName)
End If
```

Word export

C#

```
SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.FileName = "OlapGrid Report";
saveFileDialog.AddExtension = true;
saveFileDialog.DefaultExt = ".doc";
saveFileDialog.Filter = "Word (.doc)|*.doc";
if (saveFileDialog.ShowDialog() == true)
{
    string fileName = saveFileDialog.FileName;
    GridWordExport gridWordExport = new
    GridWordExport(this.olapGrid.InternalGrid.Engine,
        this.olapGrid.InternalGrid.Layout);
    gridWordExport.Export(fileName, this.olapGrid.GridStyleInfo);
}
```

VB.NET

```
Dim saveFileDialog As SaveFileDialog()
saveFileDialog.FileName = "OlapGrid Report"
saveFileDialog.AddExtension = True
saveFileDialog.DefaultExt = ".doc"
saveFileDialog.Filter = "Word (.doc)|*.doc"
If saveFileDialog.ShowDialog() = True Then
    Dim fileName As String = saveFileDialog.FileName
    Dim gridWordExport As New GridWordExport(Me.olapGrid.InternalGrid.Engine,
        Me.olapGrid.InternalGrid.Layout)
    gridWordExport.Export(fileName, Me.olapGrid.GridStyleInfo)
```

End If

PDF export

C#

```
SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.FileName = "OlapGrid Report";
saveFileDialog.AddExtension = true;
saveFileDialog.DefaultExt = "pdf";
saveFileDialog.Filter = "PDF (.pdf) | *.pdf";
if (saveFileDialog.ShowDialog() == true)
{
    string fileName = saveFileDialog.FileName;
    GridPdfExport gridPdfExport = new
    GridPdfExport(this.olapGrid.InternalGrid.Engine,
    this.olapGrid.GridStyleInfo);
    gridPdfExport.Export(fileName);
}
```

VB.NET

```
Dim saveFileDialog As SaveFileDialog()
saveFileDialog.FileName = "OlapGrid Report"
saveFileDialog.AddExtension = True
saveFileDialog.DefaultExt = "pdf"
saveFileDialog.Filter = "PDF (.pdf) | *.pdf"
If saveFileDialog.ShowDialog() = True Then
    Dim fileName As String = saveFileDialog.FileName
    Dim gridPdfExport As New GridPdfExport(Me.olapGrid.InternalGrid.Engine,
    Me.olapGrid.GridStyleInfo)
    gridPdfExport.Export(fileName)
End If
```

CSV export

C#

```
SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.FileName = "OlapGrid Report";
saveFileDialog.AddExtension = true;
saveFileDialog.DefaultExt = "csv";
saveFileDialog.Filter = "CSV (.csv) | *.csv";
if (saveFileDialog.ShowDialog() == true)
{
    string fileName = saveFileDialog.FileName;
    GridCsvExport gridCsvExport = new
    GridCsvExport(this.olapGrid.InternalGrid.Engine);
    gridCsvExport.Export(fileName);
}
```

VB.NET

```
Dim saveFileDialog As SaveFileDialog()
saveFileDialog.FileName = "OlapGrid Report"
saveFileDialog.AddExtension = True
```

```

saveFileDialog.DefaultExt = "csv"
saveFileDialog.Filter = "CSV (.csv) | *.csv"
If saveFileDialog.ShowDialog() = True Then
Dim fileName As String = saveFileDialog.FileName
Dim gridCsvExport As New GridCsvExport(Me.olapGrid.InternalGrid.Engine)
gridCsvExport.Export(fileName)
End If

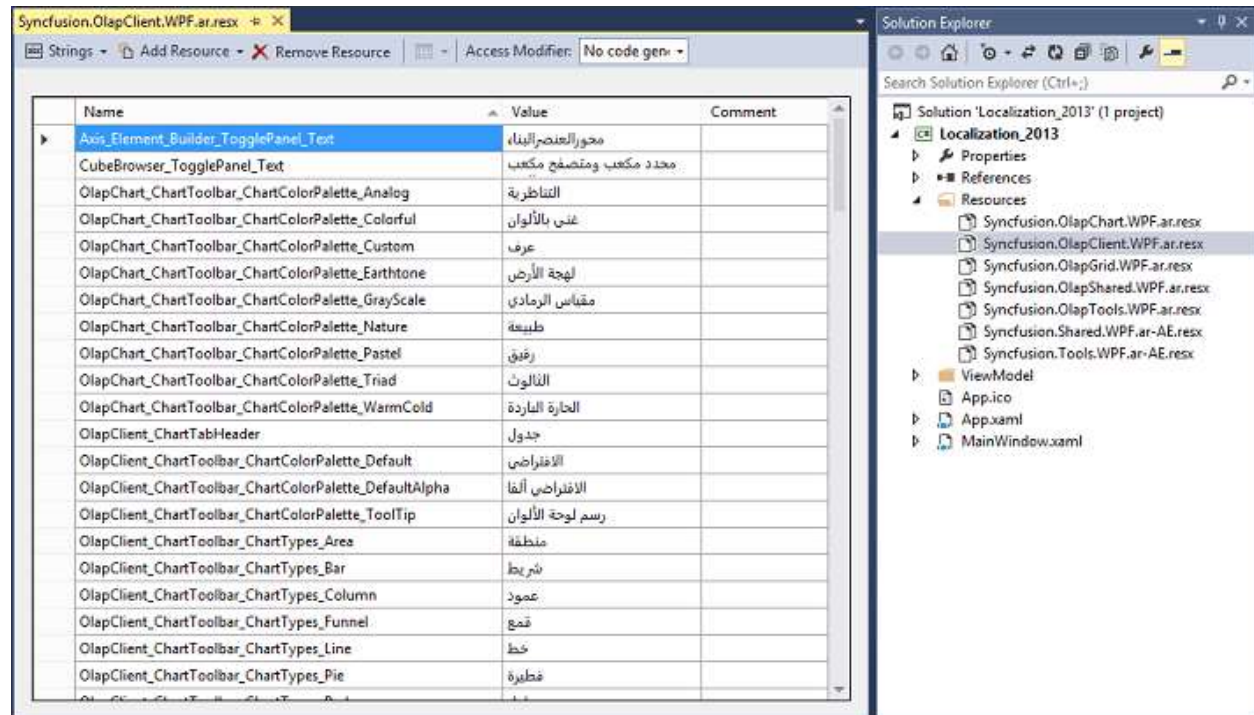
```

Theming in WPF Olap Client

Theming is the process of applying particular settings to visual elements of a product. This feature provides the following theming options:

- Office 2010 Blue
- Office 2010 Black
- Office 2010 Silver
- Transparent
- Office 2007 Blue
- Office 2007 Black
- Office 2007 Silver
- Blend
- Metro
- Office 2003
- Default

The



VisualStyle property allows users to set the Visual Style of the OLAP client control. The following code sample demonstrates how theming is added to the OLAP client control.

XML

```
<syncfusion:OlapClient x:Name="olapClient" VisualStyle="Transparent"/>
```

C#

```
this.olapClient.VisualStyle = OlapClientVisualStyle.Transparent;
```

Localization in WPF Olap Client

Localization deals with customizing data and resources for specific culture or language. The built-in localization and globalization mechanism in WPF allows you to localize any string resource used by the OLAP client control.

You can create and place the resource files in a separate location in the user application. Then, you can access the culture specific resources from the current application assembly.

For this, first create a resource file for our OLAP client control and translate the strings to your culture. Basically, the OLAP client contains control assemblies such as OlapChart.WPF and OlapGrid.WPF and Tools assemblies such as OlapShared.WPF and OlapTools.WPF within it. So, it is mandatory to localize the necessary strings available in those assemblies. Once it is translated, you might use the resources in your projects by setting the corresponding culture in your application.

![Adding the localized string in resource file](Localizationimages/Localizationimg1.png)

Refer to the following code sample.

C#

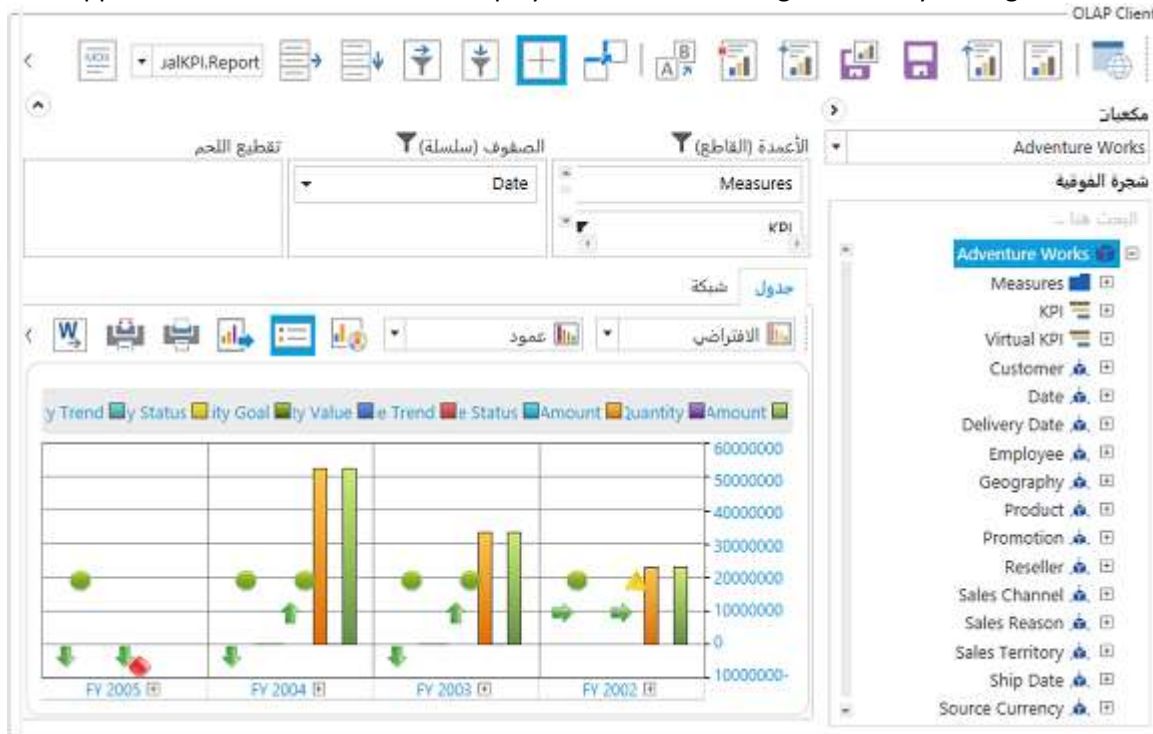
```
public MainWindow()
{
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("ar-AE");
    InitializeComponent();
}
```

VB.NET

```
Public Sub New()
    System.Threading.Thread.CurrentThread.CurrentUICulture = New
    System.Globalization.CultureInfo("ar-AE")
    InitializeComponent()
End Sub
```

RTL support

RTL support for OLAP client is used to display the content from right to left by setting the



FlowDirection property to "RightToLeft". The following code sample explains how to set this property.

XML

```
<syncfusion:OlapClient x:Name="olapClient1" FlowDirection="RightToLeft"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
OlapDataManager="{Binding ClientDataManager}" />
```

C#

```
this.olapClient1.FlowDirection = System.Windows.FlowDirection.RightToLeft;
```

VB.NET

```
Me.olapClient1.FlowDirection = System.Windows.FlowDirection.RightToLeft;
```

! [Localized Olapclient and also display the data from right to left] (Localizationimages/Localizationimg2.png)

A sample demo is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapClient.WPF\Samples\Localization\Localization

OlapGrid

WPF Olap Grid Overview

The OLAP grid is an easily configurable, presentation-quality business control that reads OLAP data source to create multi-dimensional views for analyzing and satisfying business user needs.

Key features

- Drill up/down: Provides multi-level drill up/down capability for both row and column headers.
- Layouts: Supports different grid layouts such as normal layout, Excel-like layout, Excel-like layout With member properties, normal top summary, and no summaries layout.
- Conditional formatting: You can define certain conditions, format cell's font, color, and border settings.
- Exporting: You can export the OLAP grid to Microsoft Excel, Microsoft Word, PDF, and CSV formats.

Getting Started with WPF Olap Grid

Important

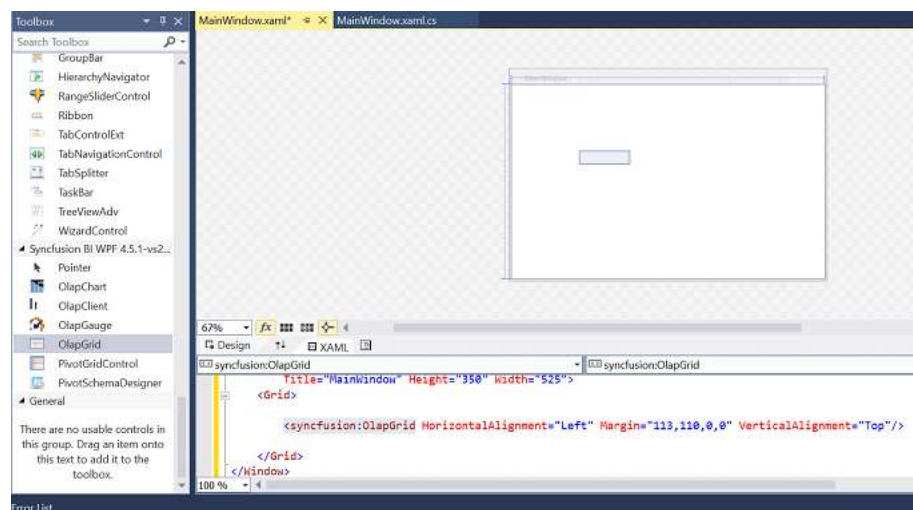
Starting with v16.2.0.x, if you refer to Syncfusion assemblies from trial setup or from the NuGet feed, include a license key in your projects. Refer to this [link](#) to learn about registering Syncfusion license key in your WPF application to use the components.

This section covers the information required to create a simple OLAP grid bound to the OLAP data source.

Through Visual Studio

Open Visual Studio IDE and go to File > New > Project > WPF Application (inside Visual C# Templates) to create a new WPF application.

After creating the WPF application, go to View menu and select Toolbox option. Now, the Toolbox will appear inside Visual Studio IDE. From the Visual Studio Toolbox, drag and drop the OLAP grid under the **Syncfusion BI WPF** tag. It will automatically add the required assemblies into the application.



Add a **Name** to the OLAP grid component for accessing it through code-behind as shown in the following code sample.

XML

```
<Window x:Class="WpfApplication.MainWindow1"
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">
<Grid>
<syncfusion:OlapGrid Name="olapGrid1" HorizontalAlignment="Left"
Margin="248,137,0,0" VerticalAlignment="Top"/>
</Grid>
</Window>
```

Include the following namespaces in the code-behind for using OlapReport and OlapDataManager in the application.

- Syncfusion.Olap.Manager
- Syncfusion.Olap.Reports

C#

```
using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
public partial class MainWindow : SampleWindow
{
private OlapDataManager olapDataManager = null;
private string _connectionString = "Enter a valid connection string";
public MainWindow
{
//Connection string is passed to OlapDataManager as an argument
olapDataManager = new OlapDataManager(_connectionString);
InitializeComponent();
//A default OlapReport is set to OlapDataManager
olapDataManager.SetCurrentReport(CreateOlapReport());
// Finally OlapGrid gets information from the OlapDataManager
this.olapGrid1.OlapDataManager = olapDataManager;
this.olapGrid1.DataBind();
}
/// <summary>
/// Defining OlapReport with Dimension and Measure
/// </summary>
private OlapReport CreateOlapReport()
{
OlapReport olapReport = new OlapReport();
//Setting the Cube name
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the name of the Dimension
dimensionElementColumn.Name = "Customer";
//Specifying the Hierarchy and Level name
dimensionElementColumn.AddLevel("Customer Geography", "Country");
MeasureElements measureElementColumn = new MeasureElements();
//Specifying the Measure name
measureElementColumn.Elements.Add(new MeasureElement { Name = "Reseller
Sales Amount" });
DimensionElement dimensionElementRow = new DimensionElement();
```

```
// Specifying the name of the Dimension
dimensionElementRow.Name = "Date";
// Specifying the Hierarchy and Level name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
///Adding Dimension in column axis
olapReport.CategoricalElements.Add(dimensionElementColumn);
///Adding Measure in column axis
olapReport.CategoricalElements.Add(measureElementColumn);
///Adding Dimension in row axis
olapReport.SeriesElements.Add(dimensionElementRow);
return olapReport;
}
}
```

VB.NET

```
Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Partial Public Class MainWindow
Inherits SampleWindow
Private olapDataManager As OlapDataManager = Nothing
Private _connectionSting As String = " Enter valid connection string;"
Private Sub MainWindow() As Public
'Connection string is passed to OlapDataManager as an argument
olapDataManager = New OlapDataManager(_connectionSting)
InitializeComponent()
'A default OlapReport is set to OlapDataManager
'Finally OlapGrid gets information from the OlapDataManager
Me.olapGrid1.OlapDataManager = olapDataManager
Me.olapGrid1.DataBind()
End Sub
''' <summary>
''' Defining OlapReport with Dimension and Measure
''' </summary>
Private Function CreateOlapReport() As OlapReport
Dim olapReport As OlapReport = New OlapReport()
'Setting the Cube name
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the name of the Dimension
dimensionElementColumn.Name = "Customer"
'Specifying the Hierarchy and Level name
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
' Specifying the Measure name
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Reseller
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementRow.Name = "Date"
' Specifying the Hierarchy and Level name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
''' Adding Dimension in column axis
olapReport.CategoricalElements.Add(dimensionElementColumn)
''' Adding Measure in column axis
olapReport.CategoricalElements.Add(measureElementColumn)
```

```
''' Adding Dimension in row axis
olapReport.SeriesElements.Add(dimensionElementRow)
Return olapReport
End Function
End Class
```

Run the application. The following output will be generated.

	Reseller Sales Amount						Total
	Australia	Canada	France	Germany	United Kingdom	United States	
FY 2002	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77
FY 2003	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52
FY 2004	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70
Total	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98

Through Expression Blend

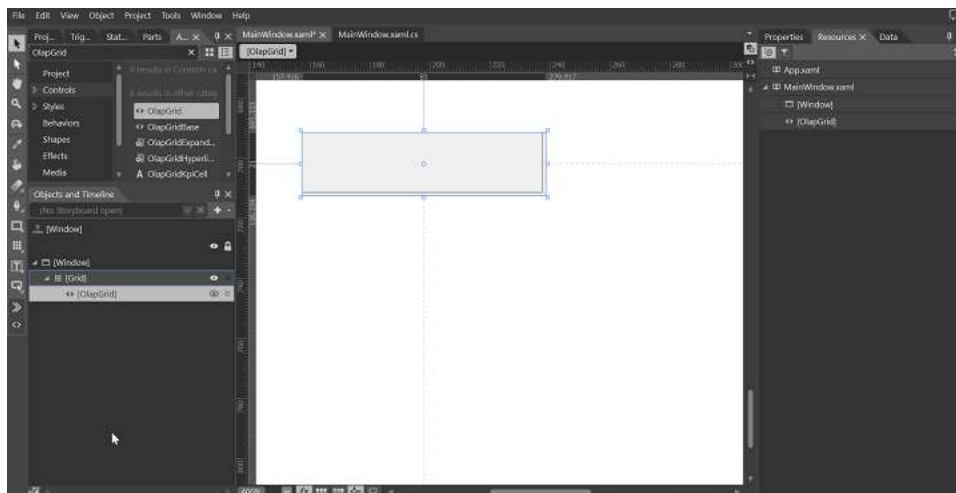
Open Blend for Visual Studio and go to File > New project > WPF > WPF Application to create a new WPF application.

Select the **Project** tab available in the left corner of the Blend IDE. Right-click the **References** and select **Add Reference**. Now, browse and add the following assemblies to the project.

- Syncfusion.Grid.Wpf
- Syncfusion.Olap.Base
- Syncfusion.OlapGrid.Wpf
- Syncfusion.OlapGridCommon.Wpf
- Syncfusion.OlapShared.Wpf

Note: You can also get the assemblies by browsing to the default assembly location: {System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<framework version>\

On adding the above assemblies, the OLAP grid control will be added under the **Assets** tab automatically. Now, choose the **Assets** tab and drag the OLAP grid to the designer.



Add a **Name** to the OLAP grid component for accessing it through code-behind as shown in the following code sample.

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="WpfApplication.MainWindow"
xmlns:local="c"
Title="MainWindow" Height="350" Width="525"
Loaded="Window_Loaded_1" >
<Grid>
<syncfusion:olapGrid Name="olapGrid1"/>
</Grid>
</Window>

```

Include the following namespaces in the code-behind for using OlapReport and OlapDataManager in the application.

- Syncfusion.Olap.Manager
- Syncfusion.Olap.Reports

C#

```

using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
public partial class MainWindow : SampleWindow
{
    private OlapDataManager olapDataManager = null;
    private string _connectionString = "Enter a valid connection string";
    public MainWindow()
    {
        InitializeComponent();
        //Connection string is passed to OlapDataManager as an argument
        olapDataManager = new OlapDataManager(_connectionString);
        //A default OlapReport is set to OlapDataManager
        olapDataManager.SetCurrentReport(CreateOlapReport());
        // Finally OLAP Grid gets the information from the OlapDataManager
        this.olapGrid1.OlapDataManager = olapDataManager;
        this.olapGrid1.DataBind();
    }
    /// <summary>
    /// Defining OlapReport with Dimension and Measure
    /// </summary>
    private OlapReport CreateOlapReport()
    {
        OlapReport olapReport = new OlapReport();
        // Setting the Cube name
        olapReport.CurrentCubeName = "Adventure Works";
        DimensionElement dimensionElementColumn = new DimensionElement();
        // Specifying the name of the Dimension
        dimensionElementColumn.Name = "Customer";
        // Specifying the Hierarchy and Level name
        dimensionElementColumn.AddLevel("Customer Geography", "Country");
        MeasureElements measureElementColumn = new MeasureElements();
        //Specifying the Measure name
        measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet
Sales Amount" });
    }
}

```

```

DimensionElement dimensionElementRow = new DimensionElement();
// Specifying the name of the Dimension
dimensionElementRow.Name = "Date";
// Specifying the Hierarchy and Level name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
///Adding Dimension in column axis
olapReport.CategoricalElements.Add(dimensionElementColumn);
///Adding Measure in column axis
olapReport.CategoricalElements.Add(measureElementColumn);
///Adding Dimension in row axis
olapReport.SeriesElements.Add(dimensionElementRow);
return olapReport;
}
}

```

VB.NET

```

Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Partial Public Class MainWindow Inherits SampleWindow
Private olapDataManager As OlapDataManager = Nothing
Private _connectionString As String = "Enter a valid connection string"
Private Sub MainWindow() As Public
Connection string is passed to OlapDataManager as an argument
olapDataManager = New OlapDataManager(_connectionString)
InitializeComponent()
'A default OlapReport is set to OlapDataManager
olapdata.SetCurrentReport(CreateOlapReport())
'Finally OLAP Grid gets the information from the OlapDataManager
Me.olapGrid1.OlapDataManager = olapDataManager
Me.olapGrid1.DataBind()
End Sub
''' <summary>
''' Defining OlapReport with Dimension and Measure
''' </summary>
Private Function CreateOlapReport() As OlapReport
Dim olapReport As OlapReport = New OlapReport()
' Setting the Cube name
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementColumn.Name = "Customer"
' Specifying the Hierarchy and Level name
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
' Specifying the Measure name
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementRow.Name = "Date"
' Specifying the Hierarchy and Level name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
''' Adding Dimension in column axis
olapReport.CategoricalElements.Add(dimensionElementColumn)
''' Adding Measure in column axis

```

```

olapReport.CategoricalElements.Add(measureElementColumn)
''' Adding Dimension in row axis
olapReport.SeriesElements.Add(dimensionElementRow)
Return olapReport
End Function
End Class

```

Run the application. The following output will be generated.

	Reseller Sales Amount						Total
	Australia	Canada	France	Germany	United Kingdom	United States	
FY 2002	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77
FY 2003	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52
FY 2004	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70
Total	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98

Through code-behind

Open Visual Studio IDE and go to File > New > Project > WPF Application (inside Visual C# Templates) to create a new WPF application.

To add the dependency assemblies within the application, right-click the **References** and select **Add Reference**. Then, add the following Syncfusion assemblies manually to the project from the installed location.

- Syncfusion.Core
- Syncfusion.Grid.Wpf
- Syncfusion.Olap.Base
- Syncfusion.OlapGrid.Wpf
- Syncfusion.OlapGridCommon.Wpf
- Syncfusion.OlapShared.Wpf

Note: You can also get the assemblies by browsing to the default assembly location: {System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<framework version>\

Include the following namespace in code-behind for OlapGrid, OlapDataManager, and OlapReport.

- Syncfusion.Windows.Grid.Olap
- Syncfusion.Olap.Manager
- Syncfusion.Olap.Reports

C#

```

using Syncfusion.Windows.Grid.Olap;
using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
namespace WpfApplication1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            //OlapGrid Instantiation
            OlapGrid olapGrid1 = new OlapGrid();

```

```

InitializeComponent();
//Connection string is passed to OlapDataManager as an argument
OlapDataManager olapDataManager = new OlapDataManager("Enter a valid
connection string");
//A default OlapReport is set to OlapDataManager
olapDataManager.SetCurrentReport(CreateOlapReport());
//Finally OlapGrid gets information from the OlapDataManager
olapGrid1.OlapDataManager = olapDataManager;
olapGrid1.DataBind();
// OlapGrid added to the Main Window grid region
grid.Children.Add(olapGrid1);
}
/// <summary>
/// Defining OlapReport with Dimension and Measure
/// </summary>
private OlapReport CreateOlapReport()
{
    OlapReport olapReport = new OlapReport();
    // Setting the Cube name
    olapReport.CurrentCubeName = "Adventure Works";
    DimensionElement dimensionElementColumn = new DimensionElement();
    DimensionElement dimensionElementRow = new DimensionElement();
    // Specifying the name of the Dimension
    dimensionElementColumn.Name = "Customer";
    // Specifying the Hierarchy and Level name
    dimensionElementColumn.AddLevel("Customer Geography", "Country");
    MeasureElements measureElementColumn = new MeasureElements();
    //Specifying the Measure name
    measureElementColumn.Elements.Add(new MeasureElement { Name = "Reseller
Sales Amount" });
    // Specifying the name of the Dimension
    dimensionElementRow.Name = "Date";
    // Specifying the Hierarchy and Level name
    dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
    ///Adding Dimension in column axis
    olapReport.CategoricalElements.Add(dimensionElementColumn);
    ///Adding Measure in column axis
    olapReport.CategoricalElements.Add(measureElementColumn);
    ///Adding Dimension in row axis
    olapReport.SeriesElements.Add(dimensionElementRow);
    return olapReport;
}
}
}

```

VB.NET

```

Imports Syncfusion.Windows.Grid.Olap
Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Partial Public Class MainWindow Inherits Window
Public Sub New()
    'OlapGrid Instantiation
    Dim olapGrid1 As OlapGrid = New OlapGrid()
    InitializeComponent()
    'Connection string is passed to OlapDataManager as an argument

```

```

Dim olapDataManager As OlapDataManager = New OlapDataManager("Enter a valid
connection string")
'A default OlapReport is set to OlapDataManager
olapDataManager.SetCurrentReport(CreateOlapReport())
'Finally OlapGrid gets the information from the OlapDataManager
olapGrid1.OlapDataManager = olapDataManager
olapGrid1.DataBind()
'OlapGrid added to the Main Window grid region
grid.Children.Add(olapGrid1)
End Sub
''' <summary>
''' Defining OlapReport with Dimension and Measure
''' </summary>
Private Function CreateOlapReport() As OlapReport
Dim olapReport As OlapReport = New OlapReport()
' Setting the Cube name
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementColumn.Name = "Customer"
' Specifying the Hierarchy and Level name
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
' Specifying the Measure name
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Reseller
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
' Specifying the name of the Dimension
dimensionElementRow.Name = "Date"
' Specifying the Hierarchy and Level name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
''' Adding Dimension in column axis
olapReport.CategoricalElements.Add(dimensionElementColumn)
''' Adding Measure in column axis
olapReport.CategoricalElements.Add(measureElementColumn)
''' Adding Dimension in row axis
olapReport.SeriesElements.Add(dimensionElementRow)
Return olapReport
End Function
End Class

```

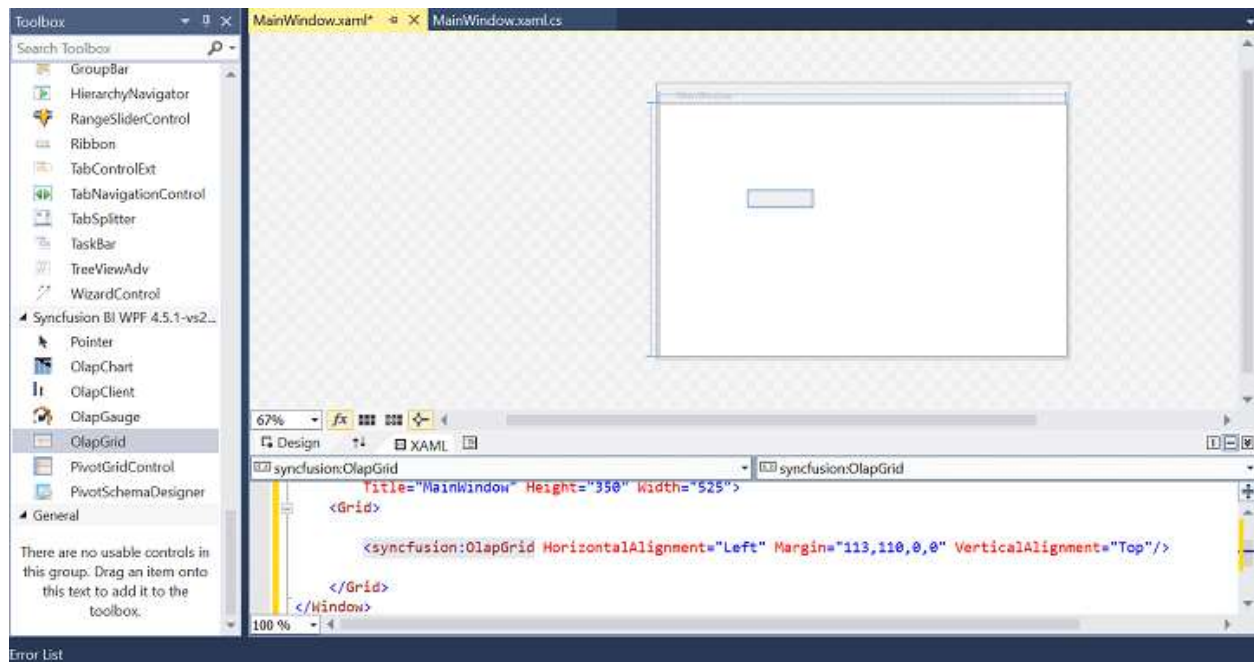
Run the application. The following output will be generated.

	Reseller Sales Amount						Total
	Australia	Canada	France	Germany	United Kingdom	United States	
FY 2002	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77
FY 2003	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52
FY 2004	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70
Total	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98

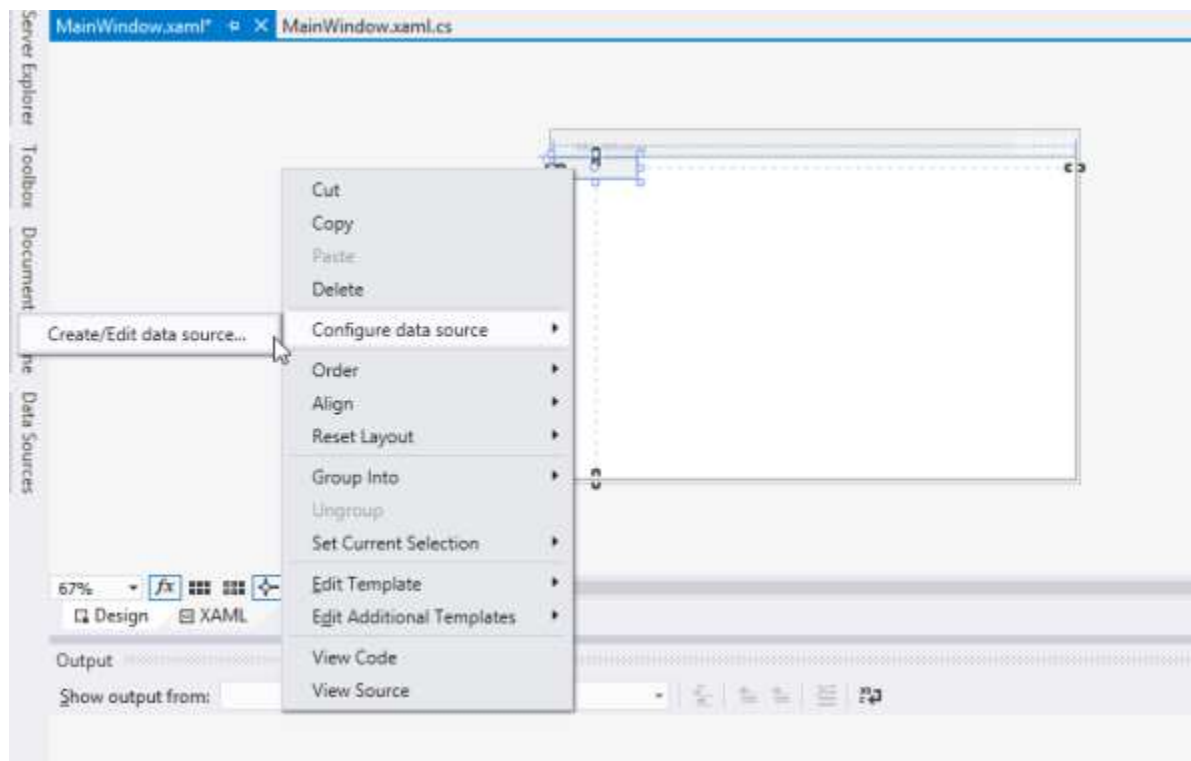
Design-time binding

Design-time binding support for OLAP grid allows you to reduce the time spent on creating and customizing the report. Normally, it takes 5 to 10 minutes for creating a report and in the case of unfamiliar cubes it may extend further, but by using design-time support you can create a report in a couple of minutes. The following section explain how to create a report during design-time.

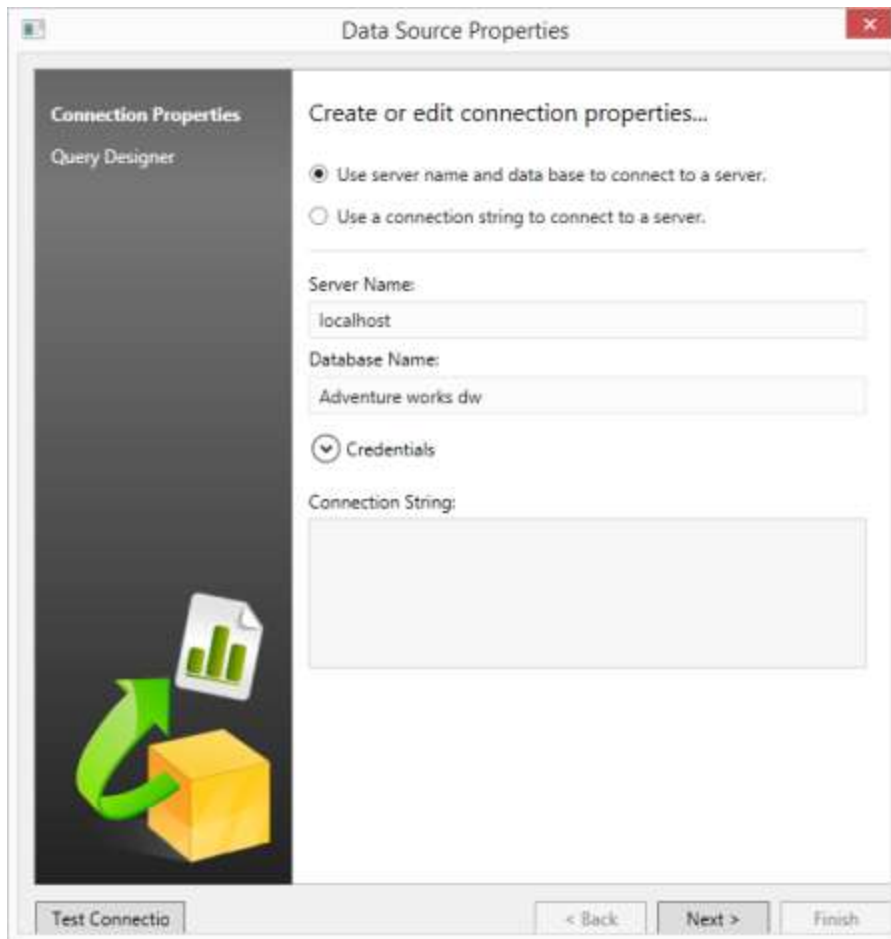
First drag the OLAP grid control from the toolbox to the Visual Studio designer surface.



Right-click the OLAP grid available in the designer and go to **Configure data source > Create/Edit data source...** option in the context menu. Now, the **Data Source Properties** wizard opens.



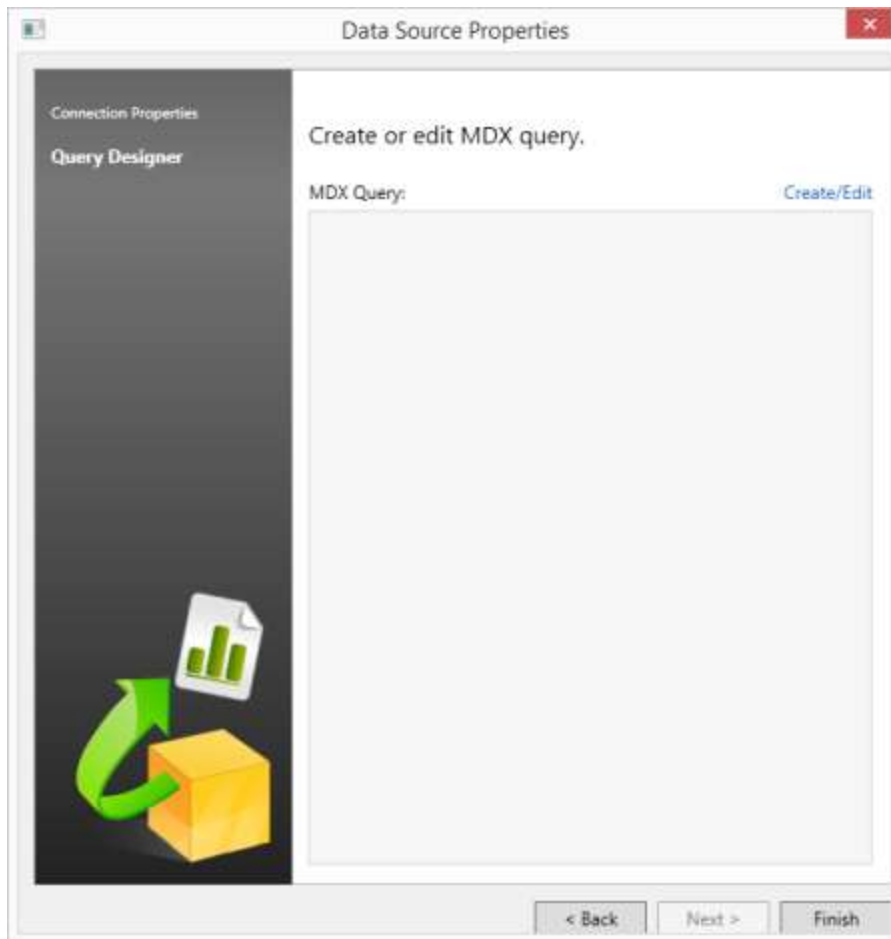
From the **Data Source Properties** wizard, select the connection type. If you want to connect to SSAS, select **Use server name and database to connect to a server** and specify the necessary information to connect to the server. If you want to connect to an offline cube, select **Use a connection string to connect to a server** and enter your connection string path.



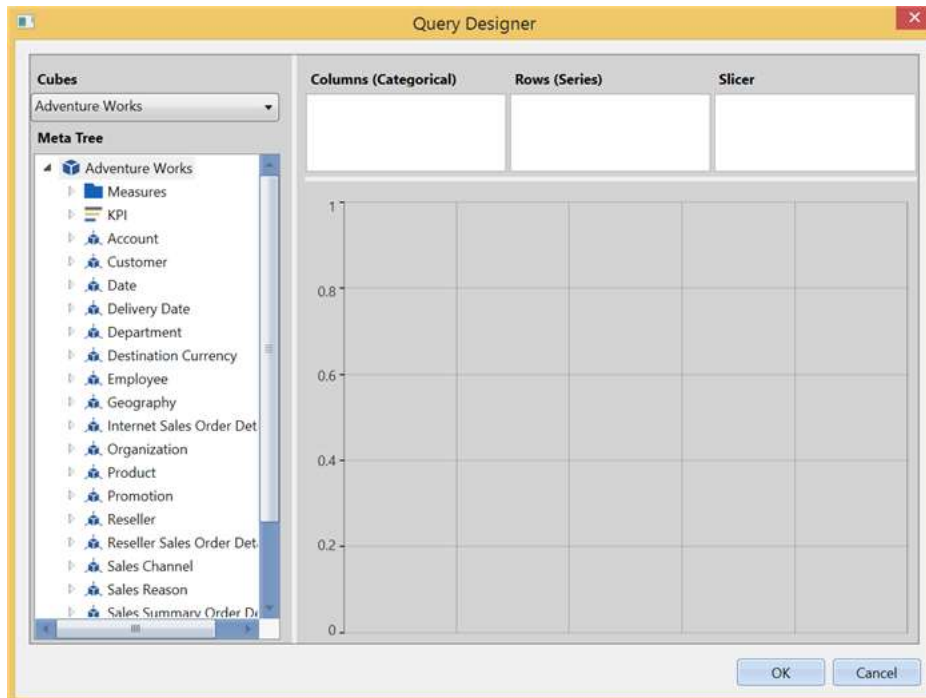
If you want to test the connection, click the **Test Connection** button that is displayed on the bottom-left corner of the window. Click the **Next** button, to proceed.

Note: The next button is enabled only when any one of the connection options is filled properly.

When the connection is valid, it displays the summary page of the **Data Source Properties** wizard. When you create a query for the first time, the MDX query textbox in the summary page is empty. When you edit an existing query, it displays the current query in the text box.



For creating or editing a query, click the **Create/Edit** link that is displayed on the top-right side of the MDX query textbox. This opens a **Query Designer** dialog.

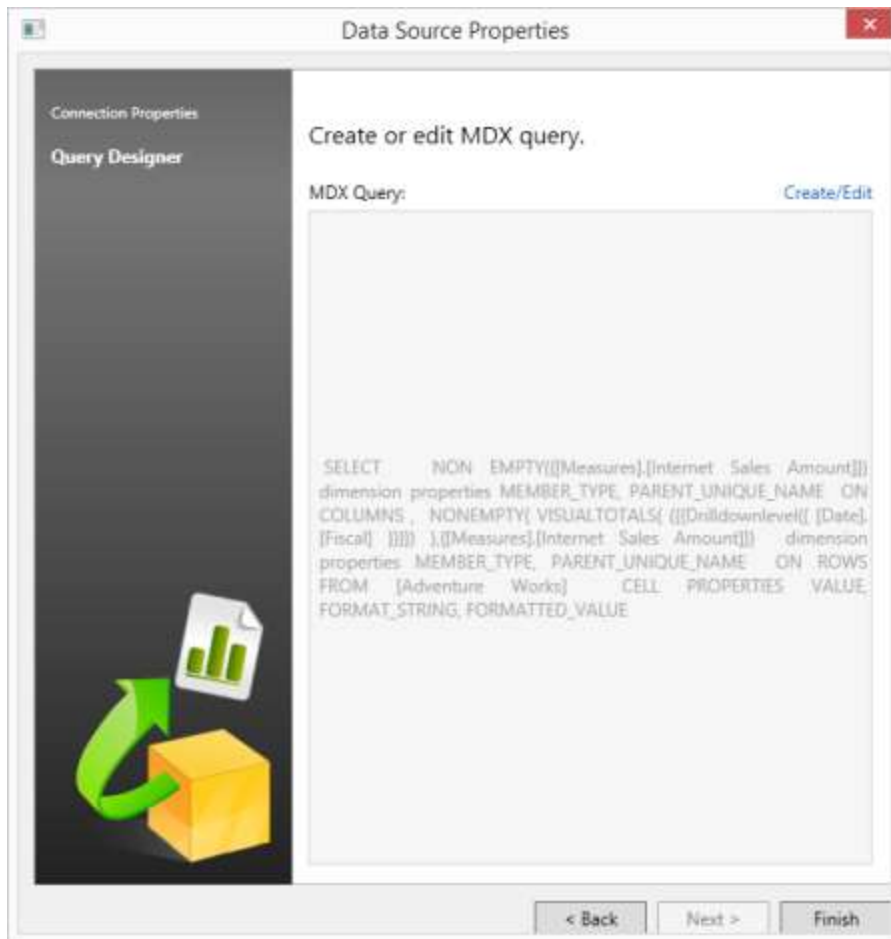


When you edit an existing query, it displays the required dimensions in the specific axis of the query designer and the preview of that query is displayed in a grid control.

Note: It does not display any style/formatting applied to the grid. It only displays the result of the query.

Drag and drop dimensions to frame a new query. Then, click OK, to save the query or click Cancel, to revert the changes made during this session.

The summary page of the **Data Source Properties** wizard displays the resultant MDX query.



Click **Finish** and run the application.

	Internet Sales Amount
⊞ FY 2002	\$7,072,084.24
⊞ FY 2003	\$5,762,134.30
⊞ FY 2004	\$16,473,618.05
⊞ FY 2005	\$50,840.63
Total	\$29,358,677.22

Data Source in WPF Olap Grid

Binding OLAP grid to offline cube

To connect to an OLAP cube available in the local machine, set the physical path of the cube set in the connection string. The following code sample illustrates the same.

C#

```
string connectionString = @"DataSource = system
drive:\OfflineCube\Adventure_Works_Ext.cub; Provider = MSOLAP;";
OlapDataManager DataManager = new OlapDataManager(connectionString);
```

Binding OLAP grid to cube in local SQL Server

To connect to the OLAP cube available in SQL Server Analysis Service in the local machine, set the server name and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code sample illustrates the same.

C#

```
string connectionString = "Data source=localhost; Initial Catalog=Adventure Works DW;";  
OlapDataManager DataManager = new OlapDataManager(connectionString);
```

Binding OLAP grid to cube in online SQL Server

To connect to the OLAP cube available in SQL Server Analysis Service in online server through **XML/A**, set the host server link and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code sample illustrates the same.

C#

```
string connectionString = "Data Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure Works DW 2008 SE;";  
OlapDataManager DataManager = new OlapDataManager(connectionString);
```

Binding OLAP grid to cube in online Mondrian Server

To connect to the OLAP cube available in Mondrian Server through **XML/A**, set the host server link and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code sample illustrates the same.

C#

```
string connectionString = @"Data Source =  
http://localhost:8080/mondrian/xmla; Initial Catalog =FoodMart;";  
OlapDataManager DataManager = new OlapDataManager(connectionString);  
DataManager.DataProvider.ProviderName =  
Syncfusion.Olap.DataProvider.Providers.Mondrian;
```

Binding OLAP grid to cube in online ActivePivot Server

To connect to the OLAP cube available in ActivePivot Server through **XML/A**, set the host server link and database name in the connection string. If you have any credentials to connect your cube, then set the user ID and password attributes accordingly. The following code sample illustrates the same.

C#

```
string connectionString = @"Data Source = http://localhost:8080/cva_s/xmla;  
Initial Catalog = CVAS;";  
OlapDataManager DataManager = new OlapDataManager(connectionString);  
DataManager.DataProvider.ProviderName=Syncfusion.Olap.DataProvider.Providers  
.ActivePivot;
```

XAML Configuration in WPF Olap Grid

XAML configuration is one of the most important features of the OLAP grid, as it helps users to configure the control entirely through XAML by eliminating the required code in code behind.

Properties



- **DataSource.ConnectionString**: Specifies the connection string of the data manager.
- **DataSource.ConnectionName**: Specifies the connection name, which is available in the App.Config file of the application.
- **DataSource.DataManagerName**: Specifies the data manager name.
- **SharedDataManagerName**: Specifies the data manager name, which is available in the shared data manager collection.
- **ReportName**: Specifies the OLAP report name.
- **CurrentCubeName**: Specifies the current cube name of an OLAP report.
- **CategoricalAxis**: Specifies the categorical axis of the OLAP report.
- **SeriesAxis**: Specifies the series axis of the OLAP report.
- **SlicerAxis**: Specifies the slicer axis of the OLAP report.
- **CalculatedMembers**: Specifies the calculated members of the OLAP report.

Adding an OLAP report to the OLAP grid in design time is described in the following code sample.

XML

```
<syncfusion:OlapGrid x:Name="olapGrid"
HorizontalAlignment="Stretch"
ReportName="SalesReport"
CurrentCubeName="Adventure Works"
SharedDataManagerName="localManager"
olapshared:DataSource.DataManagerName="localManager"
olapshared:DataSource.ConnectionString="datasource=localhost;
initial catalog=adventure works dw">
<!-- Adding Elements to Categorical Axis -->
<syncfusion:OlapGrid.CategoricalAxis>
<syncfusion:Dimension Name="Date" HierarchyName="Fiscal" LevelName="Fiscal Y
ear" IncludeMembers="FY 2002, FY 2003" /><!-- Multiple members where
specified by comma separate -->
<syncfusion:Kpi Name="Revenue" ShowGoal="True" ShowStatus="True" ShowValue="
True" ShowTrend="True" />
</syncfusion:OlapGrid.CategoricalAxis>
<!-- Adding Elements to Series Axis -->
<syncfusion:OlapGrid.SeriesAxis>
<syncfusion:Dimension Name="Sales Channel" HierarchyName="Sales Channel" Lev
elName="Sales Channel" />
<syncfusion:Dimension Name="Product" HierarchyName="Product Model Lines" Lev
elName="Product Line" IncludeMembers="Road" />
</syncfusion:OlapGrid.SeriesAxis>
</syncfusion:OlapGrid>
```

Run the application. The following output will be generated.

		FY 2002				
		Sales Amount		Revenue Goal	Revenue Status	Revenue Trend
Internet	Road	\$5,730,963.20	\$5,730,963.20	5730963.20379985		
	Total	\$5,730,963.20	\$5,730,963.20	5730963.20379985		
Reseller	Road	\$6,896,260.51	\$6,896,260.51	6896260.51409998		
	Total	\$6,896,260.51	\$6,896,260.51	6896260.51409998		
Total	Road	\$12,627,223.72	\$12,627,223.72	12627223.7178998		
	Total	\$12,627,223.72	\$12,627,223.72	12627223.7178998		

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Defining Reports\XAML Configuration

KPI in WPF Olap Grid

KPI is a collection of calculations that are associated with a measure group in a cube that are used to evaluate business success. Typically, these calculations are a combination of multi-dimensional expressions (MDX) or calculated members. KPIs also have additional metadata that provides information about how grid applications should display the results of KPI calculations.

The following are the different types of indicators:

- KPI goal
- KPI status
- KPI trend
- KPI value

C#

```

/// <summary>
/// OlapReport with KPI Elements
/// </summary>
/// <returns></returns>
private OlapReport LoadBasicKPI()
{
    OlapReport olapReport = new OlapReport();
    // Selecting the Cube
    olapReport.CurrentCubeName = "Adventure Works";
    KpiElements kpiElement = new KpiElements();
    // Specifying the KPI Element name and configuring its Indicators
    kpiElement.Elements.Add(new KpiElement
    {
        Name = "Internet Revenue",
        ShowKPIGoal = true,
        ShowKPIStatus = true,
        ShowKPIValue = true,
        ShowKPI Trend = true
    });
    DimensionElement dimensionElementRow = new DimensionElement();
    // Specifying the Name for Row Dimension Element
    dimensionElementRow.Name = "Date";
    // Specifying the Level element
    dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
    // Adding Row Elements

```



```

olapReport.SeriesElements.Add(dimensionElementRow);
// Adding Column Elements
olapReport.CategoricalElements.Add(kpiElement);
return olapReport;
}

```

VB.NET

```

''' <summary>
''' OlapReport with KPI Elements
''' </summary>
''' <returns></returns>
Private Function LoadBasicKPI() As OlapReport
Dim olapReport As OlapReport = New OlapReport()
olapReport.CurrentCubeName = "Adventure Works"
Dim kpiElement As KpiElements = New KpiElements()
kpiElement.Elements.Add(New KpiElement With {.Name = "Internet Revenue",
.ShowKPIGoal = True, .ShowKPIStatus = True, .ShowKPIValue = True,
.ShowKPIStatus = True})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
dimensionElementRow.Name = "Date"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
olapReport.SeriesElements.Add(dimensionElementRow)
olapReport.CategoricalElements.Add(kpiElement)
Return olapReport
End Function

```

	Bikes			
	Internet Sales Amount	Internet Revenue Goal	Internet Revenue Status	Internet Revenue Trend
FY 2002	\$7,072,084.24	7072084.2438		
FY 2003	\$5,762,134.30	7779292.66818		
FY 2004	\$15,483,926.11	6338347.72659		
FY 2005		17032318.721		
Total	\$28,318,144.65	28318144.6507		

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Product Showcase\KPI

Drill Down/Up in WPF Olap Grid

This is the basic feature of OLAP grid through which the amount of information can be limited. It allows you to drill down to access the detailed level of data, or drill up to see the summarized data using the expanders present in the grid. The expander here refers to the plus or minus or arrow sign present in the grid prior to a member. The expanders can be made hidden by setting the `ShowExpander` property of the OLAP report to `"false"`.

C#

```

// Hiding Expanders
this.OlapGrid1.OlapDataManager.CurrentReport.ShowExpanders = false;

```

VB.NET

```
' Hiding Expanders  
Me.OlapGrid1.OlapDataManager.CurrentReport.ShowExpanders = False
```

Drill Position in WPF Olap Grid

Drill position allows users to drill only the current position of the selected member and exclude the drilled data of selected member in other positions by using the MDX query. This can be enabled by setting the “**DrillType**” enumeration to “**DrillPosition**” in the OLAP report.

C#

```
olapDataManager.CurrentReport.DrillType = DrillType.DrillPosition;
```

VB.NET

```
olapDataManager.CurrentReport.DrillType = DrillType.DrillPosition
```

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Data Relation\Drill Types

Grid Layout in WPF Olap Grid

The position of summary cells in the OLAP grid can be customized with the help of grid layouts. It can be positioned at the top or bottom of each parent member.

The following are the five different types of layouts supported by the OLAP grid:

- Normal layout.
- Excel-like layout.
- Excel-like layout with member properties.
- Normal top summary.
- No summaries layout.

Normal layout

Normal layout is the default layout of OLAP grid. In normal layout, the summary cells are positioned at the bottom of each parent member and the child members are positioned adjacent to it.

C#

```
/// Grid Layout will be Normal  
this.OlapGrid1.Layout = GridLayout.Normal;
```

VB.NET

```
' Grid Layout will be Normal  
Me.OlapGrid1.Layout = GridLayout.Normal
```

		Australia	Canada	France	Germany	United Kingdom	United States	Total
		Internet Sales Amount						
FY 2002	Bikes	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
	Total	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
FY 2003	Bikes	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
	Total	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
FY 2004	Accessories	\$132,860.61	\$96,106.31	\$60,878.53	\$59,503.04	\$73,771.80	\$243,895.03	\$667,015.32
	Bikes	\$4,183,763.19	\$943,190.73	\$1,505,930.70	\$1,701,913.93	\$2,035,740.36	\$5,113,387.20	\$15,483,926.11
	Clothing	\$66,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$30,876.34	\$126,600.44	\$322,676.62
	Total	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
FY 2005	Accessories	\$5,830.02	\$7,271.54	\$2,528.25	\$2,729.55	\$2,858.24	\$12,527.04	\$33,744.64
	Clothing	\$3,404.21	\$3,582.16	\$963.70	\$875.28	\$1,363.17	\$6,907.47	\$17,095.99
	Total	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	Accessories	\$138,690.63	\$103,377.85	\$63,406.78	\$62,232.59	\$76,630.04	\$256,422.07	\$700,759.96
	Bikes	\$8,852,050.00	\$1,821,302.39	\$2,553,575.71	\$2,808,514.35	\$3,282,842.66	\$8,999,859.53	\$28,318,144.65
	Clothing	\$70,259.95	\$53,164.62	\$27,035.22	\$23,565.40	\$32,239.51	\$133,507.91	\$339,772.61
	Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

Excel-like layout

In the Excel-like layout, the summary cells are positioned at the bottom alone and the child members are positioned below the parent member with some indent space.

C#

```
/// Excel-like Grid Layout
this.OlapGrid1.Layout = GridLayout.ExcelLikeLayout;
```

VB.NET

```
' Excel-like Grid Layout
Me.OlapGrid1.Layout = GridLayout.ExcelLikeLayout
```

		Australia	Canada	France	Germany	United Kingdom	United States	Total
		Internet Sales Amount						
FY 2002		\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
Bikes		\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
FY 2003		\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
Bikes		\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
FY 2004		\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
Accessories		\$132,860.61	\$96,106.31	\$60,878.53	\$59,503.04	\$73,771.80	\$243,895.03	\$667,015.32
Bikes		\$4,183,763.19	\$943,190.73	\$1,505,930.70	\$1,701,913.93	\$2,035,740.36	\$5,113,387.20	\$15,483,926.11
Clothing		\$66,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$30,876.34	\$126,600.44	\$322,676.62
FY 2005		\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Accessories		\$5,830.02	\$7,271.54	\$2,528.25	\$2,729.55	\$2,858.24	\$12,527.04	\$33,744.64
Clothing		\$3,404.21	\$3,582.16	\$963.70	\$875.28	\$1,363.17	\$6,907.47	\$17,095.99
Grand Total		\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

Excel-like layout with member properties

This layout is used to display member properties along with dimension members. These properties appear adjacent to each member.

Note: This layout is applicable only for members having properties defined in the OLAP cube and those members are bound to the OLAP grid through the OLAP report.

C#

```
/// Excel-like Grid Layout with Member Properties
this.OlapGrid1.Layout = GridLayout.ExcelLikeLayoutWithMemberProperties;
```

VB.NET

```
' Excel-like Grid Layout with Member Properties
Me.OlapGrid1.Layout = GridLayout.ExcelLikeLayoutWithMemberProperties
```

	Title	Phone	Email Address	Sales Amount Quota
<input type="checkbox"/> Ken J. Sánchez	Chief Executive Officer	697-555-0142	ken0@adventure-works.com	\$113,129,150.00
<input type="checkbox"/> Brian S. Welcker	Vice President of Sales	716-555-0127	brian3@adventure-works.com	\$113,129,150.00
<input type="checkbox"/> Amy E. Alberts	European Sales Manager	775-555-0164	amy0@adventure-works.com	\$23,077,600.00
<input type="checkbox"/> Jae B. Pak	Sales Representative	1 (11) 500 555-0145	jae0@adventure-works.com	\$12,547,100.00
<input type="checkbox"/> Rachel B. Valdez	Sales Representative	1 (11) 500 555-0140	rachel0@adventure-works.com	\$3,269,800.00
<input type="checkbox"/> Ranjit R. Varkey Chudukatil	Sales Representative	1 (11) 500 555-0117	ranjit0@adventure-works.com	\$7,260,700.00
<input type="checkbox"/> Stephen Y. Jiang	North American Sales Manager	238-555-0197	stephen0@adventure-works.com	\$87,336,050.00
<input type="checkbox"/> Syed E. Abbas	Pacific Sales Manager	926-555-0182	syed0@adventure-works.com	\$2,715,500.00
Grand Total				\$113,129,150.00

Normal top summary layout

In normal top summary layout, the summary cells are positioned at the top of each parent member and the child members are positioned adjacent to it.

C#

```
/// Grid Layout will be Normal with summary positioned at the top
this.OlapGrid1.Layout = GridLayout.NormalTopSummary;
```

VB.NET

```
' Grid Layout will be Normal with summary positioned at the top
Me.OlapGrid1.Layout = GridLayout.NormalTopSummary
```

		Total	Australia	Canada	France	Germany	United Kingdom	United States
		Internet Sales Amount						
Total	Total	\$29,358,677.22	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51
	Accessories	\$700,759.96	\$138,690.63	\$103,377.85	\$63,406.78	\$62,232.59	\$76,630.04	\$256,422.07
	Bikes	\$28,318,144.65	\$8,852,050.00	\$1,821,302.39	\$2,553,575.71	\$2,808,514.35	\$3,282,842.66	\$8,999,859.53
	Clothing	\$339,772.61	\$70,259.95	\$53,164.62	\$27,035.22	\$23,565.40	\$32,239.51	\$133,507.91
FY 2002	Total	\$7,072,084.24	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07
	Bikes	\$7,072,084.24	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07
FY 2003	Total	\$5,762,134.30	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26
	Bikes	\$5,762,134.30	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26
FY 2004	Total	\$16,473,618.05	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67
	Accessories	\$667,015.32	\$132,860.61	\$96,106.31	\$60,878.53	\$59,503.04	\$73,771.80	\$243,895.03
	Bikes	\$15,483,926.11	\$4,183,763.19	\$943,190.73	\$1,505,930.70	\$1,701,913.93	\$2,035,740.36	\$5,113,387.20
	Clothing	\$322,676.62	\$66,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$30,876.34	\$126,600.44
FY 2005	Total	\$50,840.63	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51
	Accessories	\$33,744.64	\$5,830.02	\$7,271.54	\$2,528.25	\$2,729.55	\$2,858.24	\$12,527.04
	Clothing	\$17,095.99	\$3,404.21	\$3,582.16	\$963.70	\$875.28	\$1,363.17	\$6,907.47

No summaries layout

In no summaries layout, the summary cells are hidden and the child members appear adjacent to the parent member.

C#

```
/// Grid Layout without Summaries
this.OlapGrid1.Layout = GridLayout.NoSummaries;
```

VB.NET

```
' Grid Layout without Summaries
Me.OlapGrid1.Layout = GridLayout.NoSummaries
```

		Australia	Canada	France	Germany	United Kingdom	United States
		Internet Sales Amount					
FY 2002	Bikes	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07
FY 2003	Bikes	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26
FY 2004	Accessories	\$132,860.61	\$96,106.31	\$60,878.53	\$59,503.04	\$73,771.80	\$243,895.03
	Bikes	\$4,183,763.19	\$943,190.73	\$1,505,930.70	\$1,701,913.93	\$2,035,740.36	\$5,113,387.20
	Clothing	\$66,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$30,876.34	\$126,600.44
FY 2005	Accessories	\$5,830.02	\$7,271.54	\$2,528.25	\$2,729.55	\$2,858.24	\$12,527.04
	Clothing	\$3,404.21	\$3,582.16	\$963.70	\$875.28	\$1,363.17	\$6,907.47

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Appearance\Grid Layout

Paging in WPF Olap Grid

Paging in the OLAP grid supports loading and rendering large amounts of data without any performance constraints.

The OLAP pager (user control) is included and bound with the OlapDataManager object of the respective OLAP grid. To enable paging, set the `EnablePaging` property to true.

When you process a large cell set, it is split into several segments and each segment is assigned and rendered in a separate page. You can navigate back and forth in all possible ways by using the UI options in the OLAP pager. You can also change the page size and other pager settings at runtime by using the **PageSetting** window.

Include the following Syncfusion assembly from the installed location to add the OLAP pager with OLAP grid:

- Syncfusion.OlapShared.Wpf

Note: You can also get the assemblies by browsing to the default assembly location: {System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\{version number}\precompiledassemblies\{version number}\{framework version}\

Enable paging through XAML

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:pager="clr-
namespace:Syncfusion.Windows.Shared.Olap;assembly=Syncfusion.OlapShared.WPF"
x:Class="SampleApplication.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<GroupBox Header="OlapGrid" Grid.Row="0">
<syncfusion:OlapGrid Name="olapGrid" Background="Transparent"
SeriesStrokeThickness="0"></syncfusion:OlapGrid>
</GroupBox>
<GroupBox Grid.Row="1" Header="OlapPager" Margin="5" >
<pager:OlapPager x:Name="olapPager" ></pager:OlapPager>
</GroupBox>
</Grid>
</Window>
```

Enable paging through report:

C#

```
using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
namespace SampleApplication
{
public partial class MainWindow : SampleWindow
{
private string _connectionString;
private OlapDataManager _olapDataManager;
public MainWindow()
{
InitializeComponent();
```



```

_connectionString = "Enter a valid connection string";
//Created connection string is passed to OlapDataManager as argument
_olapDataManager = new OlapDataManager(_connectionString);
//Created OlapReport is set as a current report to OlapDataManager
_olapDataManager.SetCurrentReport(SimpleDimensions());
//Finally OlapGrid control gets the data from the created OlapDataManager
this.olapGrid.OlapDataManager = _olapDataManager;
this.olapPager.OlapDataManager= _olapDataManager;
this.olapGrid.DataBind();
}
private OlapReport SimpleDimensions()
{
OlapReport olapReport = new OlapReport();
olapReport.CurrentCubeName = "Adventure Works";
olapReport.EnablePaging = true;
olapReport.PagerOptions.CategoricalPageSize = 10;
olapReport.PagerOptions.SeriesPageSize = 10;
DimensionElement dimensionElement = new DimensionElement() { Name =
"Customer", HierarchyName = "Customer" };
dimensionElement.AddLevel("Customer Geography", "Country");
olapReport.CategoricalElements.Add(dimensionElement);
MeasureElements measureElements = new MeasureElements();
measureElements.Add(new MeasureElement { Name = "Internet Sales Amount" });
olapReport.SeriesElements.Add(measureElements);
dimensionElement = new DimensionElement() { Name = "Geography",
HierarchyName = "Geography" };
dimensionElement.AddLevel("Geography", "Country");
olapReport.CategoricalElements.Add(dimensionElement);
dimensionElement = new DimensionElement() { Name = "Date" };
dimensionElement.AddLevel("Fiscal", "Fiscal Year");
olapReport.SeriesElements.Add(dimensionElement);
return olapReport;
}
}
}

```

VB.NET

```

Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Namespace SampleApplication
Partial Public Class MainWindow
Inherits SampleWindow
Private _connectionString As String
Private _olapDataManager As OlapDataManager
Public Sub New()
InitializeComponent()
_connectionString = "Enter a valid connection string"
'Created connection string is passed to OlapDataManager as argument
_olapDataManager = New OlapDataManager(_connectionString)
'Created OlapReport is set as a current report to OlapDataManager
_olapDataManager.SetCurrentReport(SimpleDimensions())
'Finally OlapGrid control gets the data from the created OlapDataManager
Me.olapGrid.OlapDataManager = _olapDataManager
Me.olapPager.OlapDataManager= _olapDataManager
Me.olapGrid.DataBind()

```

```
End Sub
Private Function SimpleDimensions() As OlapReport
Dim olapReport As New OlapReport()
olapReport.CurrentCubeName = "Adventure Works"
olapReport.EnablePaging = True
olapReport.PagerOptions.CategoricalPageSize = 10
olapReport.PagerOptions.SeriesPageSize = 10
Dim dimensionElement As New DimensionElement() With {.Name = "Customer",
.HierarchyName = "Customer"}
dimensionElement.AddLevel("Customer Geography", "Country")
olapReport.CategoricalElements.Add(dimensionElement)
Dim measureElements As New MeasureElements()
measureElements.Add(New MeasureElement With {.Name = "Internet Sales
Amount"})
olapReport.SeriesElements.Add(measureElements)
dimensionElement = New DimensionElement() With {.Name = "Geography",
.HierarchyName = "Geography"}
dimensionElement.AddLevel("Geography", "Country")
olapReport.CategoricalElements.Add(dimensionElement)
dimensionElement = New DimensionElement() With {.Name = "Date"}
dimensionElement.AddLevel("Fiscal", "Fiscal Year")
olapReport.SeriesElements.Add(dimensionElement)
Return olapReport
End Function
End Class
End Namespace
```

		United States					
		Australia	Canada	France	Germany	United Kingdom	United States
Internet Sales Amount	FY 2002	\$2,452,176.07	\$2,452,176.07	\$2,452,176.07	\$2,452,176.07	\$2,452,176.07	\$2,452,176.07
	FY 2003	\$1,434,296.26	\$1,434,296.26	\$1,434,296.26	\$1,434,296.26	\$1,434,296.26	\$1,434,296.26
	FY 2004	\$5,483,882.67	\$5,483,882.67	\$5,483,882.67	\$5,483,882.67	\$5,483,882.67	\$5,483,882.67
	FY 2005	\$19,434.51	\$19,434.51	\$19,434.51	\$19,434.51	\$19,434.51	\$19,434.51
Column Pager		Row Pager					

Pager Settings

Column Page Settings

Page Size

Current Page

Row Page Settings

Page Size

Current Page

OK Cancel

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Paging

Conditional Formatting in WPF Olap Grid

Conditional formatting allows you to format the grid cells based on a certain condition.

Conditional formatting can be specified through the `OlapGrid.ConditionalFormats` property. This is an observable collection, into which you can add required number of formatters of type `"OlapGridDataConditionalFormat"`. Using the `"OlapGridDataConditionalFormat"` class, you can specify the filter criteria for cells and the style to be applied for filtered cells. After these specifications are defined, the given styles are applied to only those cells that satisfy the specified condition. The filter criteria are specified by the `OlapGridDataConditionalFormat.Conditions` property, which is a collection of `OlapGridDataCondition` objects. The style for each `ConditionalFormat` can be specified using the `OlapGridDataConditionalFormat.CellStyle` property which should be of `OlapGridCellStyle` type.

The following code sample shows how to add conditional formats.

XML

```
<syncfusion:OlapGrid>
<syncfusion:OlapGrid.ConditionalFormats>
<!-- Adding Conditions -->
<syncfusion:OlapGridDataConditionalFormat Name="C1">
<!-- Specifying Cell Style -->
< syncfusion:OlapGridDataConditionalFormat.CellStyle>
<syncfusion:OlapGridCellStyle Background="Yellow" FontFamily="Calibri"
FontSize="12"/>
</syncfusion:OlapGridDataConditionalFormat.CellStyle>
<!-- Specifying Conditions -->
<syncfusion:OlapGridDataConditionalFormat.Conditions>
<syncfusion:OlapGridDataCondition ConditionType="GreaterThan"
Value="2000000" MeasureElement="Internet Sales Amount" PredicateType="Or"/>
<syncfusion:OlapGridDataCondition ConditionType="LessThan" Value="5000000"
MeasureElement="Internet Sales Amount" PredicateType="And"/>
</syncfusion:OlapGridDataConditionalFormat.Conditions>
</syncfusion:OlapGridDataConditionalFormat>
</syncfusion:OlapGrid.ConditionalFormats>
</syncfusion:OlapGrid>
```

C#

```
OlapGrid olapGrid1 = new OlapGrid();
// Instantiating OlapDataManager with Connection string.
OlapDataManager olapDataManager = new OlapDataManager(connectionString);
// Set current report for OlapDataManager.
olapDataManager.SetCurrentReport(CurrentReport());
// Specifying OlapDataManager to Grid.
olapGrid1.OlapDataManager = olapDataManager;
// Specifying OlapGridData Conditional Format.
OlapGridDataConditionalFormat conditionalFormat = new
OlapGridDataConditionalFormat();
// Adding Conditions to OlapGridData Conditional Format.
conditionalFormat.Conditions.Add(new OlapGridDataCondition() {
ConditionType= OlapGridDataConditionType.GreaterThan ,
MeasureElement="Internet Sales Amount",
Value="2000000",
PredicateType = PredicateType.Or });
conditionalFormat.Conditions.Add(new OlapGridDataCondition() {
```

```

ConditionType= OlapGridDataConditionType.LessThan ,
MeasureElement="Internet Sales Amount",
Value="5000000",
PredicateType = PredicateType.And });
// Specifying Cell Style to Conditional Format.
conditionalFormat.CellStyle = new OlapGridCellStyle() { Background=
Brushes.Yellow, FontFamily = new FontFamily("Calibri"), FontSize=12 };
// Adding Conditions to Grid.
this.olapgrid1.ConditionalFormats.Add(conditionalFormat);
// Data Binding.
this.olapgrid1.DataBind();

```

VB.NET

```

Dim olapGrid1 As OlapGrid = New OlapGrid()
' Instantiating OlapDataManager with Connection string.
Dim olapDataManager As OlapDataManager = New
OlapDataManager(connectionString)
' Set current report for OlapDataManager.
olapDataManager.SetCurrentReport(CurrentReport())
' Specifying OlapDataManager to Grid.
olapGrid1.OlapDataManager = OlapDataManager
' Specifying OlapGridData Conditional Format.
Dim conditionalFormat As OlapGridDataConditionalFormat = New
OlapGridDataConditionalFormat()
' Adding Conditions to OlapGridData Conditional Format.
conditionalFormat.Conditions.Add(New OlapGridDataCondition() With
{.ConditionType= OlapGridDataConditionType.GreaterThan,
.MeasureElement="Internet Sales Amount", .Value="2000000", .PredicateType =
PredicateType.Or})
conditionalFormat.Conditions.Add(New OlapGridDataCondition() With
{.ConditionType= OlapGridDataConditionType.LessThan,
.MeasureElement="Internet Sales Amount", .Value="5000000", .PredicateType =
PredicateType.And})
' Specifying Cell Style to Conditional Format.
conditionalFormat.CellStyle = New OlapGridCellStyle()
Dim TempFontFamily As FontFamily = New FontFamily("Calibri"), FontSize=12
Brushes.Yellow, FontFamily = New FontFamily("Calibri"), FontSize
Background= Brushes.Yellow, FontFamily
' Adding Conditions to Grid.
Me.olapgrid1.ConditionalFormats.Add(conditionalFormat)
' Data Binding.
Me.olapgrid1.DataBind()

```

		Australia	Canada	France	Germany	United Kingdom	United States	Total
		Internet Sales Amount						
Accessories	FY 2004	\$132,860.61	\$96,106.31	\$60,878.53	\$59,503.04	\$73,771.80	\$243,895.03	\$667,015.32
	FY 2005	\$5,830.02	\$7,271.54	\$2,528.25	\$2,729.55	\$2,858.24	\$12,527.04	\$33,744.64
	Total	\$138,690.63	\$103,377.85	\$63,406.78	\$62,232.59	\$76,630.04	\$256,422.07	\$700,759.96
Bikes	FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
	FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
	FY 2004	\$4,183,763.19	\$943,190.73	\$1,505,930.70	\$1,701,913.93	\$2,035,740.36	\$5,113,387.20	\$15,483,926.11
	Total	\$8,852,050.00	\$1,821,302.39	\$2,553,575.71	\$2,808,514.35	\$3,282,842.66	\$8,999,859.53	\$28,318,144.65
Clothing	FY 2004	\$66,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$30,876.34	\$126,600.44	\$322,676.62
	FY 2005	\$3,404.21	\$3,582.16	\$963.70	\$873.18	\$1,383.17	\$6,907.47	\$17,095.99
	Total	\$70,259.95	\$53,164.62	\$27,035.22	\$23,565.40	\$32,239.51	\$133,507.91	\$339,772.61
Total	FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
	FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
	FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
	FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
	Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Appearance\Conditional Formatting

Cell Selection in WPF Olap Grid

The OLAP grid supports Excel-like cell selection, where you can select grid value cells as in the Microsoft Excel. This can be achieved by setting the `AllowSelection` property of the OLAP grid to true.

On cell selection, an event named **"SelectionChanged"** will be triggered, and the **"OlapGridSelectionChangedEventArgs"** returns an `IEnumerator` collection of column, row, and value of the corresponding selected cell. The event argument also returns the cell range and selection reason such as mouse down, mouse move, and mouse up.

The following code sample shows how to enable cell selection.

XML

```
<!--Adding OlapGrid and enabling cell selection-->
<syncfusion:OlapGrid AllowSelection="True"
SelectionChanged="olapGrid1_SelectionChanged">
</syncfusion:OlapGrid>
```

C#

```
public MainWindow()
{
InitializeComponent();
//OlapGrid Instantiation
OlapGrid olapGrid1 = new OlapGrid();
//OlapGrid added to an parent Grid present in the Main Window
this.gridContainer.Children.Add(olapGrid1);
//Connection string is passed to OlapDataManager as an argument
OlapDataManager olapDataManager = new OlapDataManager("Enter a valid
connection string");
//Set current report for OlapDataManager.
```

```

olapDataManager.SetCurrentReport(CreateOlapReport());
//Specifying OlapDataManager to Grid.
olapGrid1.OlapDataManager = olapDataManager;
//Enable Cell Selection
olapGrid1.AllowSelection = true;
olapGrid1.SelectionChanged += olapGrid1_SelectionChanged;
olapGrid1.DataBind();
}
private void olapGrid1_SelectionChanged(object sender,
OlapGridSelectionChangedEventArgs e)
{
//Enter your code
}

```

VB.NET

```

Public Sub New()
InitializeComponent()
Dim OlapGrid1 As OlapGrid = New OlapGrid()
' OlapGrid added to an parent Grid present in the Main Window
Me.gridContainer.Children.Add(olapGrid1);
' Instantiating OlapDataManager.
Dim olapDataManager As OlapDataManager = New OlapDataManager("Enter a valid
connection string")
' Set current report for OlapDataManager.
olapDataManager.SetCurrentReport(olapReport())
' Specifying OlapDataManager to Grid.
Me.OlapGrid1.OlapDataManager = OlapDataManager
' Enable Cell Selection.
Me.OlapGrid1.AllowSelection = True
Me.OlapGrid1.SelectionChanged += OlapGrid1_SelectionChanged;
Me.OlapGrid1.DataBind()
End Sub
Private Sub OlapGrid1_SelectionChanged(ByVal sender As Object, ByVal e As
Syncfusion.Windows.Grid.Olap.OlapGridSelectionChangedEventArgs)
'Enter your code here
End Sub

```

	Australia	Canada	France	Germany	United Kingdom	United States	Total
	Reseller Sales Amount						
Accessories	\$571,297.93	\$571,297.93	\$571,297.93	\$571,297.93	\$571,297.93	\$571,297.93	\$571,297.93
Bikes	\$66,302,381.56	\$66,302,381.56	\$66,302,381.56	\$66,302,381.56	\$66,302,381.56	\$66,302,381.56	\$66,302,381.56
Clothing	\$1,777,840.84	\$1,777,840.84	\$1,777,840.84	\$1,777,840.84	\$1,777,840.84	\$1,777,840.84	\$1,777,840.84
Components	\$11,799,076.66	\$11,799,076.66	\$11,799,076.66	\$11,799,076.66	\$11,799,076.66	\$11,799,076.66	\$11,799,076.66
Total	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Selection\Cell Selection

Freeze Headers in WPF Olap Grid

The OLAP grid provides built-in support to freeze the column and row headers. This can be achieved by setting the `FreezeHeaders` property of OLAP grid to `“true”`.

XML

```
<syncfusion:OlapGrid FreezeHeaders="True">
</syncfusion:OlapGrid>
```

C#

```
// To freeze OlapGrid Headers
this.OlapGrid1.FreezeHeaders = true;
```

VB.NET

```
' To freeze OlapGrid Headers
Me.OlapGrid1.FreezeHeaders = True
```

		Canada	France	Germany
	Amount			
Document Control	\$1,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34
Engineering	\$1,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34
Executive	\$1,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34
Facilities and Maintenance	\$1,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34
Finance	\$1,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34
Human Resources	\$1,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34
Information Services	\$1,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34

A sample demo is available in the following location.

{system drive:}\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Appearance\Frozen Header

Hyperlink Cells in WPF Olap Grid

The OLAP grid allows hyperlinking of cells to retrieve a detailed information about a particular cell. The OLAP grid generates a separate event called **“LinkClick”** for the clicked hyperlink cell and the **“LinkLabelClickEventHandler”** will return the clicked OlapGridCellInfo.

The hyperlink cell’s property can be applied to the following regions:

- Column header
- Row header
- Value cell
- Summary cell

It is possible to enable or disable the hyperlink options separately for all row header, column header, summary header, and value cells through the **IsHyperlinkCell** property. Refer to the following code sample.

C#

```
public MainWindow ()
{
```

```

InitializeComponent();
//OlapGrid Instantiation
OlapGrid olapGrid1 = new OlapGrid();
//OlapGrid added to an parent Grid present in the Main Window
this.gridContainer.Children.Add(olapGrid1);
//Connection string is passed to OlapDataManager as an argument
OlapDataManager olapDataManager = new OlapDataManager("Enter a valid
connection string");
//Set current report for OlapDataManager.
olapDataManager.SetCurrentReport(CreateOlapReport());
//Specifying OlapDataManager to Grid.
olapGrid1.OlapDataManager = olapDataManager;
// To enable hyperlink for Column Header
this.OlapGrid1.ColumnHeaderStyle.IsHyperlinkCell = true;
// To enable hyperlink for Row Header
this.OlapGrid1.RowHeaderStyle.IsHyperlinkCell = true;
// To enable hyperlink for Value Cell
this.OlapGrid1.ValueCellStyle.IsHyperlinkCell = true;
// To enable hyperlink for Summary Column
this.OlapGrid1.SummaryColumnStyle.IsHyperlinkCell = true;
// To enable hyperlink for Summary Row
this.OlapGrid1.SummaryRowStyle.IsHyperlinkCell = true;
// Tag hyperlink cell click event
this.OlapGrid1.LinkClick += new
Syncfusion.Windows.Grid.Olap.LinkLabelClickEventHandler(OlapGrid1_LinkClick)
;
olapGrid1.DataBind();
}
private void OlapGrid1_LinkClick(object sender,
Syncfusion.Windows.Grid.Olap.LinkLabelEventArgs e)
{
    string uniqueName = e.PivotCellDescriptor.UniqueName;
}

```

VB.NET

```

Public Sub New()
InitializeComponent()
Dim OlapGrid1 As OlapGrid = New OlapGrid()
' OlapGrid added to an parent Grid present in the Main Window
Me.gridContainer.Children.Add(olapGrid1);
' Instantiating OlapDataManager.
Dim olapDataManager As OlapDataManager = New OlapDataManager("Enter a valid
connection string")
' Set current report for OlapDataManager.
olapDataManager.SetCurrentReport(olapReport())
' Specifying OlapDataManager to Grid.
' To enable hyperlink for Column Header
Me.OlapGrid1.ColumnHeaderStyle.IsHyperlinkCell = True
' To enable hyperlink for Row Header
Me.OlapGrid1.RowHeaderStyle.IsHyperlinkCell = True
' To enable hyperlink for Value Cell
Me.OlapGrid1.ValueCellsStyle.IsHyperlinkCell = True
' To enable hyperlink for Summary Column
Me.OlapGrid1.SummaryColumnStyle.IsHyperlinkCell = True
' To enable hyperlink for Summary Row

```



```

Me.OlapGrid1.SummaryRowStyle.IsHyperlinkCell = True
' Tag hyperlink cell click event
Me.OlapGrid1.LinkClick += New
Syncfusion.Windows.Grid.Olap.LinkLabelClickEventHandler(OlapGrid1_LinkClick)
;
Me.OlapGrid1.DataBind()
End Sub
Private Sub OlapGrid1_LinkClick(ByVal sender As Object, ByVal e As
Syncfusion.Windows.Grid.Olap.LinkLabelEventArgs)
Dim uniqueName As String = e.PivotCellDescriptor.UniqueName
End Sub

```

	Australia	Canada	France	Germany	United Kingdom	United States	Total
Internet Sales Amount							
FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

	Australia	Canada	France	Germany	United Kingdom	United States	Total
Internet Sales Amount							
FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

	Australia	Canada	France	Germany	United Kingdom	United States	Total
Internet Sales Amount							
FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Appearance\Hyperlink Cell

Tooltip in WPF Olap Grid

The OLAP grid can additionally display the member information within a tooltip when the mouse pointer is moved over the header cells or value cells of the OLAP grid control.

Header tooltip

The OLAP grid provides drill-down information through the header cell tooltip for hierarchical dimensions by enabling efficient preview of data before drilling down. It can be enabled using the following property of the OLAP grid.

C#

```

// Enabling Header Cell Tooltip
this.OlapGrid1.ShowHeaderCellsToolTip = true;

```

VB.NET

```

' Enabling Header Cell Tooltip

```

```
Me.OlapGrid1.ShowHeaderCellsToolTip = True
```

Reseller Sales Amount							
	Australia	Canada	France	Germany	United Kingdom	United States	Total
FY 2002	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77
FY 2003	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52
FY 2004							
Total	Reseller Sales Amount						
	Australia	Canada	France	Germany	United Kingdom	United States	
FY 2002	H1 FY 2002	\$8,065,435.31	\$8,065,435.31	\$8,065,435.31	\$8,065,435.31	\$8,065,435.31	\$8,065,435.31
	H2 FY 2002	\$8,223,006.46	\$8,223,006.46	\$8,223,006.46	\$8,223,006.46	\$8,223,006.46	\$8,223,006.46

Value cell tooltip

The OLAP grid provides cell information (measure, column header, row header, and value cell) when the mouse pointer is hovered over the value cells. This can be enabled using the following property of the OLAP grid.

C#

```
// Enabling Value Cell Tooltip
this.OlapGrid1.ShowValueCellToolTip = true;
```

VB.NET

```
' Enabling Value Cell Tooltip
Me.OlapGrid1.ShowValueCellToolTip = True
```

Reseller Sales Amount							
	Australia	Canada	France	Germany	United Kingdom	United States	Total
FY 2002	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77	\$16,288,441.77
FY 2003	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52	\$27,921,670.52
FY 2004	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70	\$36,240,484.70
Total	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98	\$80,450,596.98

Grid Style Dialog in WPF Olap Grid

The OLAP grid can be formatted the following ways:

- Style dialog.
- Configuring the properties of cell style.

Style dialog

The OLAP grid style dialog is used to format the cells of the control. Styling can be applied to column headers, row headers, summary cells, and value cells. The following properties of headers and summary cells can be formatted:

- Background color
- Foreground color
- Font name
- Font size

The following are the properties of value cells that can be formatted:

- Font name
- Font style
- Font color
- Font size



The following code sample will launch the OLAP grid style dialog.

C#

```
// To display style dialog
this.OlapGrid1.ShowStyleDialog();
```

VB.NET

```
' To display style dialog
Me.OlapGrid1.ShowStyleDialog()
```

Configuring the properties of cell style

The following properties allow the OLAP grid cell to be customized, so that it appears in a custom style rather than the default one.

- **Background:** Gets or sets the background color of the cell.
- **FontFamily:** Gets or sets the font family of the cell.
- **FontSize:** Gets or sets the font size of the cell.
- **FontWeight:** Gets or sets the font weight of the cell.
- **Foreground:** Gets or sets the foreground color of the cell.

The column, row, summary, and value cells of OLAP grid can be formatted independently using the following properties.

- ColumnHeaderStyle
- RowHeaderStyle
- SummaryColumnStyle
- SummaryRowStyle
- ValueCellsStyle

C#

```
// Specifying the background color for column header  
this.OlapGrid1.ColumnHeaderStyle.Background = new  
SolidColorBrush(Color.FromRgb(175, 209, 255));  
// Specifying the background color for row header  
this.OlapGrid1.RowHeaderCellStyle.Background = new  
SolidColorBrush(Color.FromRgb(175, 209, 255));  
// Specifying the background color for summary cell  
this.OlapGrid1.SummaryColumnStyle.Background = new  
SolidColorBrush(Color.FromRgb(206, 225, 248));
```

VB.NET

```
' Specifying the background color for column header  
Me.OlapGrid1.ColumnHeaderStyle.Background = New  
SolidColorBrush(Color.FromRgb(175, 209, 255))  
' Specifying the background color for row header  
Me.OlapGrid1.RowHeaderCellStyle.Background = New  
SolidColorBrush(Color.FromRgb(175, 209, 255))  
' Specifying the background color for summary cell  
Me.OlapGrid1.SummaryColumnStyle.Background = New  
SolidColorBrush(Color.FromRgb(206, 225, 248))
```

The value cell text alignment can be changed using the following property of the OLAP grid.

C#

```
// Specifying the value cell text alignment  
this.OlapGrid1.ValueCellTextAlignment = HorizontalAlignment.Center;
```

VB.NET

```
' Specifying the value cell text alignment  
Me.OlapGrid1.ValueCellTextAlignment = HorizontalAlignment.Center
```

		Australia	Canada	France	Germany	United Kingdom
		Internet Sales Amount				
Accessories	FY 2004	\$132,860.61	\$96,106.31	\$60,878.53	\$59,503.04	\$73,771.80
	FY 2005	\$5,830.02	\$7,271.54	\$2,528.25	\$2,729.55	\$2,858.24
	Total	\$138,690.63	\$103,377.85	\$63,406.78	\$62,232.59	\$76,630.04
Bikes	FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33
	FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97
	FY 2004	\$4,183,763.19	\$943,190.73	\$1,505,930.70	\$1,701,913.93	\$2,035,740.36
	Total	\$8,852,050.00	\$1,821,302.39	\$2,553,575.71	\$2,808,514.35	\$3,282,842.66
Clothing	FY 2004	\$66,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$30,876.34
	FY 2005	\$3,404.21	\$3,582.16	\$963.70	\$875.28	\$1,363.17
	Total	\$70,259.95	\$53,164.62	\$27,035.22	\$23,565.40	\$32,239.51
Total	FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33
	FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97
	FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50
	FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41
	Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Exporting\Exporting Grid

Member Properties in WPF Olap Grid

The OLAP grid allows binding of members along with their properties. Member properties cover the basic information about each member in each tuple. This basic information includes the member name, parent level, number of children, and so on. Member properties are available for all members at a given level. To display the member properties along with the dimension member, OLAP report requires member properties to be defined in the concerned dimension element. Also, the OLAP grid layout should be set to **“ExcelLikeLayoutWithMemberProperties”**.

C#

```
private OlapReport ReportWithMemberProperties()
{
    OlapReport olapReport = new OlapReport();
    // Specifying the current cube name
    olapReport.CurrentCubeName = "Adventure Works";
    MeasureElements measureElementColumn = new MeasureElements();
    // Specifying the Name for the Measure Element
    measureElementColumn.Elements.Add(new MeasureElement { Name = "Sales Amount Quota" });
    DimensionElement dimensionElementRow = new DimensionElement();
    // Specifying the Dimension Name
    dimensionElementRow.Name = "Employee";
    // Specifying the Hierarchy and level name for the Dimension Element
    dimensionElementRow.AddLevel("Employees", "Employee Level 02");
    dimensionElementRow.Hierarchy.LevelElements["Employee Level 02"].IncludeAvailableMembers = true;
    // Adding the Member properties to the Dimension Element
    dimensionElementRow.MemberProperties.Add(new MemberProperty("Title", "[Employee].[Employees].[Title]"));
}
```

```

dimensionElementRow.MemberProperties.Add(new MemberProperty("Phone",
"[Employee].[Employees].[Phone]"));
dimensionElementRow.MemberProperties.Add(new MemberProperty("Email Address",
"[Employee].[Employees].[Email Address]"));
// Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow);
///Adding Column Members
olapReport.CategoricalElements.Add(measureElementColumn);
return olapReport;
}

```

VB.NET

```

''' <summary>
''' OlapReport with KPI Elements
''' </summary>
''' <returns></returns>
Private Function ReportWithMemberProperties() As OlapReport
Dim olapReport As New OlapReport()
' Selecting the Cube
olapReport.CurrentCubeName = "Adventure Works"
Dim measureElements As New MeasureElements()
measureElements.Add(New MeasureElement With {.Name = " Sales Amount Quota"})
olapReport.SeriesElements.Add(measureElements)
Dim dimensionElementRow As New DimensionElement()
' Specifying the Name for Row Dimension Element
dimensionElementRow.Name = "Employee"
' Specifying the Level element
dimensionElementRow.AddLevel("Employees", "Employee Level 02")
dimensionElementRow.Hierarchy.LevelElements["Employee Level
02"].IncludeAvailableMembers = true;
' Adding the Member properties to the Dimension Element
dimensionElementRow.MemberProperties.Add(new MemberProperty("Title",
"[Employee].[Employees].[Title]"));
dimensionElementRow.MemberProperties.Add(new MemberProperty("Phone",
"[Employee].[Employees].[Phone]"));
dimensionElementRow.MemberProperties.Add(new MemberProperty("Email Address",
"[Employee].[Employees].[Email Address]"));
' Adding Row Elements
olapReport.SeriesElements.Add(dimensionElementRow)
' Adding Column Elements
olapReport.CategoricalElements.Add(measureElements)
Return olapReport
End Function

```

	Title	Phone	Email Address	Sales Amount Quota
☐ Ken J. Sánchez	Chief Executive Officer	697-555-0142	ken0@adventure-works.com	\$114,253,550.00
☐ Brian S. Welcker	Vice President of Sales	716-555-0127	brian3@adventure-works.com	\$114,253,550.00
☐ Amy E. Alberts	European Sales Manager	775-555-0164	amy0@adventure-works.com	\$24,202,000.00
☐ Stephen Y. Jiang	North American Sales Manager	238-555-0197	stephen0@adventure-works.com	\$87,336,050.00
☐ Syed E. Abbas	Pacific Sales Manager	926-555-0182	syed0@adventure-works.com	\$2,715,500.00
Grand Total				\$114,253,550.00

To display member properties through the header tooltip, the following property of OLAP grid should be set to true.

C#

```
// To Display Member Properties in ToolTip
this.OlapGrid1.ShowMemberPropertiesToolTip = true;
```

VB.NET

```
' To Display Member Properties in ToolTip
Me.OlapGrid1.ShowMemberPropertiesToolTip = True
```

	Title	Phone	Email Address	Sales Amount Quota
<input type="checkbox"/> Ken J. Sánchez	Chief Executive Officer	697-555-0142	ken0@adventure-works.com	\$114,253,550.00
<input type="checkbox"/> Brian S. Welcker	Vice President of Sales	716-555-0127	brian3@adventure-works.com	\$114,253,550.00
<input type="checkbox"/> Amy E. Alberts	European Sales Manager	775-555-0164	amy0@adventure-works.com	\$24,202,000.00
<input type="checkbox"/> Stephen Y. Jiang	North American Sales Manager	238-555-0197	stephen0@adventure-works.com	\$87,336,050.00
<input type="checkbox"/> Syed E. Abbas	Pacific Sales Manager	926-555-0182	syed0@adventure-works.com	\$2,715,500.00
Grand Total				\$114,253,550.00

Title	North American Sales Manager
Phone	238-555-0197
Email Address	stephen0@adventure-works.com

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Product Showcase\Member Properties

"All" - Level Member

This feature enables you to display the "All" level type member across the rows and columns in the OLAP grid. This member behaves as parent to other members in its hierarchy by controlling their visibility through an expander.

To display the "All" level type member, set the *ShowLevelTypeAll* property to *true*. By default this is set to *false*. Refer to the following code sample.

C#

```
OlapDataManager DataManager = new OlapDataManager() { ShowLevelTypeAll = true};
```

VB.NET

```
OlapDataManager DataManager = New OlapDataManager() { ShowLevelTypeAll = True }
```

			All Customers						
			Australia	Canada	France	Germany	United Kingdom	United States	Total
			Internet Sales Amount						
All Periods	FY 2002	All	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.22
	FY 2003	All	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
	FY 2004	All	\$4,383,479.50	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.00
	FY 2005	All	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
	Total	All	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Defining Reports\Reports-in-code

Exporting in WPF Olap Grid

The OLAP grid can be exported to Microsoft Excel, Microsoft Word, PDF, and CSV file formats. To perform exporting operation, add the following assembly along with default assemblies in the reference folder.

- Syncfusion.OlapGridConverter.Wpf

The OLAP grid can be exported using the following methods.

Excel export

C#

```
SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.AddExtension = true;
saveFileDialog.FileName = "OlapGrid Report";
saveFileDialog.DefaultExt = ".xls";
saveFileDialog.Filter = "Excel file (.xls)|*.xls";
if (saveFileDialog.ShowDialog() == true)
{
    if (this.olapGrid.InternalGrid != null && this.olapGrid.InternalGrid.Engine
        != null)
    {
        GridExcelExport gridExcelExport = new
        GridExcelExport(this.olapGrid.InternalGrid.Engine);
        gridExcelExport.Export(saveFileDialog.FileName);
        MessageBox.Show("Excel document exported successfully!");
    }
}
```

VB.NET

```
Dim saveFileDialog As New SaveFileDialog()
saveFileDialog.AddExtension = True
saveFileDialog.FileName = "OlapGrid Report"
saveFileDialog.DefaultExt = ".xls"
saveFileDialog.Filter = "Excel file (.xls)|*.xls"
If saveFileDialog.ShowDialog() = True Then
    If Me.olapGrid.InternalGrid IsNot Nothing AndAlso
        Me.olapGrid.InternalGrid.Engine IsNot Nothing Then
        Dim gridExcelExport As New GridExcelExport(Me.olapGrid.InternalGrid.Engine)
        gridExcelExport.Export(saveFileDialog.FileName)
        MessageBox.Show("Excel document exported successfully!")
    End If
End If
```

		Australia	Canada	France	Germany	United Kingdom	United States
		Internet Sales Amount					
FY 2002	Bikes	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$660,507.33	\$2,452.17
	Total	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$660,507.33	\$2,452.17
FY 2003	Bikes	\$2,099,586.43	\$305,010.89	\$633,398.70	\$593,247.24	\$696,594.97	\$1,434.29
	Total	\$2,099,586.43	\$305,010.89	\$633,398.70	\$593,247.24	\$696,594.97	\$1,434.29
FY 2004	Accessories	\$132,860.61	\$96,106.31	\$60,878.53	\$58,503.04	\$73,771.80	\$243.89
	Bikes	\$4,183,763.19	\$943,190.73	\$1,505,930.70	\$1,701,913.93	\$2,035,740.36	\$5,113.38
	Clothing	\$60,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$30,876.34	\$126.00
	Total	\$4,383,479.54	\$1,088,879.90	\$1,592,980.75	\$1,784,107.09	\$2,140,388.50	\$5,483.88
FY 2005	Accessories	\$5,839.02	\$7,271.54	\$2,526.25	\$2,729.55	\$2,858.24	\$12.52
	Clothing	\$3,404.21	\$3,582.16	\$983.70	\$875.28	\$1,383.17	\$8.90
	Total	\$9,243.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19.43
Total	Accessories	\$138,690.63	\$103,377.85	\$63,406.78	\$62,232.59	\$76,630.04	\$256.42
	Bikes	\$8,652,050.00	\$1,821,302.39	\$2,553,875.71	\$2,808,514.35	\$3,282,642.68	\$8,999.85
	Clothing	\$93,259.95	\$83,164.62	\$27,035.22	\$23,565.40	\$32,239.51	\$133.50
	Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,381,712.21	\$9,389.78

Word export

C#

```
SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.AddExtension = true;
saveFileDialog.FileName = "OlapGrid Report";
saveFileDialog.DefaultExt = ".Doc";
saveFileDialog.Filter = "Word file (.Doc)|*.Doc";
if (saveFileDialog.ShowDialog() == true)
{
    if (this.olapGrid.InternalGrid != null && this.olapGrid.InternalGrid.Engine
        != null)
    {
        GridWordExport gridWordExport = new
        GridWordExport(this.olapGrid.InternalGrid.Engine);
        gridWordExport.Export(saveFileDialog.FileName);
        MessageBox.Show("Word document exported successfully!");
    }
}
```

VB.NET

```
Dim saveFileDialog As New SaveFileDialog()
saveFileDialog.AddExtension = True
saveFileDialog.FileName = "OlapGrid Report"
saveFileDialog.DefaultExt = ".Doc"
saveFileDialog.Filter = "Word file (.Doc)|*.Doc"
If saveFileDialog.ShowDialog() = True Then
    If Me.olapGrid.InternalGrid IsNot Nothing AndAlso
        Me.olapGrid.InternalGrid.Engine IsNot Nothing Then
        Dim gridWordExport As New GridWordExport(Me.olapGrid.InternalGrid.Engine)
        gridWordExport.Export(saveFileDialog.FileName)
        MessageBox.Show("Word document exported successfully!")
    End If
End If
```


		Australia	Canada	France	Germany	United Kingdom
		Internet Sales Amount	Internet Sales Amount	Internet Sales Amount	Internet Sales Amount	Internet Sales Amount
FY 2002	Bikes	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$55
	Total	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$55
FY 2003	Bikes	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$66
	Total	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$66
FY 2004	Accessories	\$132,860.61	\$96,106.31	\$60,878.53	\$59,503.04	\$7
	Bikes	\$4,183,763.19	\$943,199.73	\$1,505,930.70	\$1,701,913.93	\$2.03
	Clothing	\$66,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$3
	Total	\$4,383,479.54	\$1,088,878.50	\$1,592,880.75	\$1,784,107.09	\$2.14
FY 2005	Accessories	\$5,630.02	\$7,271.54	\$2,528.25	\$2,729.55	\$
	Clothing	\$3,404.21	\$3,582.16	\$963.70	\$875.28	\$
	Total	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$
Total	Accessories	\$138,690.63	\$103,377.85	\$63,406.78	\$62,232.59	\$7
	Bikes	\$8,852,050.00	\$1,821,302.39	\$2,553,876.71	\$2,808,514.35	\$3.28
	Clothing	\$70,259.95	\$53,164.62	\$27,035.22	\$23,565.40	\$3
	Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3.35

PDF export

C#

```
SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.AddExtension = true;
saveFileDialog.FileName = "OlapGrid Report";
saveFileDialog.DefaultExt = "Pdf";
saveFileDialog.Filter = "Pdf file (.pdf)|*.pdf";
if (saveFileDialog.ShowDialog() == true)
{
    if (this.olapGrid.InternalGrid != null && this.olapGrid.InternalGrid.Engine
        != null)
    {
        GridPdfExport gridPdfExport = new
        GridPdfExport(this.olapGrid.InternalGrid.Engine);
        gridPdfExport.Export(saveFileDialog.FileName);
        MessageBox.Show("Pdf document exported successfully!");
    }
}
```

VB.NET

```
Dim saveFileDialog As New SaveFileDialog()
saveFileDialog.AddExtension = True
saveFileDialog.FileName = "OlapGrid Report"
saveFileDialog.DefaultExt = "Pdf"
saveFileDialog.Filter = "Pdf file (.pdf)|*.pdf"
If saveFileDialog.ShowDialog() = True Then
    If Me.olapGrid.InternalGrid IsNot Nothing AndAlso
        Me.olapGrid.InternalGrid.Engine IsNot Nothing Then
        Dim gridPdfExport As New GridPdfExport(Me.olapGrid.InternalGrid.Engine)
        gridPdfExport.Export(saveFileDialog.FileName)
        MessageBox.Show("Pdf document exported successfully!")
    End If
End If
```




The screenshot shows a PDF document titled 'OlapReportData.pdf' in Adobe Reader. The document contains a table with the following data:

		Australia	Canada	France	Germany	United Kingdom
	Internet Sales Amount					
FY 2002	Bikes	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33
	Total	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33
FY 2003	Bikes	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97
	Total	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97
FY 2004	Accessories	\$132,860.61	\$96,106.31	\$60,878.53	\$59,503.04	\$73,771.80
	Bikes	\$4,183,763.19	\$943,190.73	\$1,505,930.70	\$1,701,913.93	\$2,035,740.36
	Clothing	\$66,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$30,876.34
	Total	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50
FY 2005	Accessories	\$5,830.02	\$7,271.54	\$2,528.25	\$2,729.55	\$2,858.24
	Clothing	\$3,404.21	\$3,582.16	\$963.70	\$875.28	\$1,363.17
	Total	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41
Total	Accessories	\$138,690.63	\$103,377.85	\$63,406.78	\$62,232.59	\$76,630.04
	Bikes	\$8,852,050.00	\$1,821,302.39	\$2,553,575.71	\$2,808,514.35	\$3,282,842.66
	Clothing	\$70,259.95	\$53,164.62	\$27,035.22	\$23,565.40	\$32,239.51
	Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21

CSV export

C#

```

SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.AddExtension = true;
saveFileDialog.FileName = "OlapGrid Report";
saveFileDialog.DefaultExt = "CSV";
saveFileDialog.Filter = "Csv file (.csv)|*.csv";
if (saveFileDialog.ShowDialog() == true)
{
    if (this.olapGrid.InternalGrid != null && this.olapGrid.InternalGrid.Engine
        != null)
    {
        GridCsvExport gridCsvExport = new
        GridCsvExport(this.olapGrid.InternalGrid.Engine);
        gridCsvExport.Export(saveFileDialog.FileName);
        MessageBox.Show("CSV document exported successfully!");
    }
}

```

VB.NET

```

Dim saveFileDialog As New SaveFileDialog()
saveFileDialog.AddExtension = True
saveFileDialog.FileName = "OlapGrid Report"
saveFileDialog.DefaultExt = "CSV"
saveFileDialog.Filter = "Csv file (.csv)|*.csv"
If saveFileDialog.ShowDialog() = True Then
    If Me.olapGrid.InternalGrid IsNot Nothing AndAlso
        Me.olapGrid.InternalGrid.Engine IsNot Nothing Then
        Dim gridCsvExport As New GridCsvExport(Me.olapGrid.InternalGrid.Engine)
        gridCsvExport.Export(saveFileDialog.FileName)
        MessageBox.Show("CSV document exported successfully!")
    End If
End If

```

```
End If
End If
```

	A	B	C	D	E	F	G	H	I
1			Australia	Canada	France	Germany	United Kingdom	United States	Total
2			Internet Sales Amount						
3	FY 2002	Bikes	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
4		Total	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
5	FY 2003	Bikes	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
6		Total	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
7	FY 2004	Accessories	\$132,860.61	\$96,106.31	\$60,878.53	\$59,503.04	\$73,771.80	\$243,895.03	\$667,015.32
8		Bikes	\$4,183,763.19	\$943,190.73	\$1,505,930.70	\$1,701,913.93	\$2,035,740.36	\$5,113,387.20	\$15,483,926.11
9		Clothing	\$66,855.74	\$49,582.46	\$26,071.52	\$22,690.12	\$30,876.34	\$126,600.44	\$322,676.62
10		Total	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
11	FY 2005	Accessories	\$5,830.02	\$7,271.54	\$2,528.25	\$2,729.55	\$2,858.24	\$12,527.04	\$33,744.64
12		Clothing	\$3,404.21	\$3,582.16	\$963.70	\$875.28	\$1,363.17	\$6,907.47	\$17,095.99
13		Total	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
14	Total	Accessories	\$138,690.63	\$103,377.85	\$63,406.78	\$62,232.59	\$76,630.04	\$256,422.07	\$700,759.96
15		Bikes	\$8,852,050.00	\$1,821,302.39	\$2,553,575.71	\$2,808,514.35	\$3,282,842.66	\$8,999,859.53	\$28,318,144.65
16		Clothing	\$70,259.95	\$53,164.62	\$27,035.22	\$23,565.40	\$32,239.51	\$133,507.91	\$339,772.61
17		Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

A sample demo is available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Exporting\Exporting Grid

Theming in WPF Olap Grid

Theming is the process of applying particular settings to the visual elements of a control. This feature provides the following themes options:

Default Blend Metro Office2010Blue Office2010Black Office2010Silver Office2013LightGray Office2013DarkGray Office2013White VS

The `VisualStyle` property allows you to set a theme for the OLAP grid control. The following code sample demonstrates how to add theming to the OLAP grid control.

XML

```
<syncfusion:OlapGrid x:Name="olapGrid" VisualStyle="Transparent"/>
```

C#

```
this.olapGrid.VisualStyle = OlapGridVisualStyle.Transparent;
```

A sample demo available in the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGrid.WPF\Samples\Appearance\Visual Styles

Localization in WPF Olap Grid

Localization is the key feature for providing software solutions targeted at global users. The OLAP grid allows users to localize the control to a specific locale and supports "resx" based localization.

The following steps should be performed to localize the control:

- Translation.
- Resource file and file name conventions.
- Tag inclusion into the project file.
- Specification of the CurrentUICulture.

Translation

The first step in localization is translating the strings that can be localized to the destination locale.

Note: Localization key field should be same for all locales. Do not translate it.

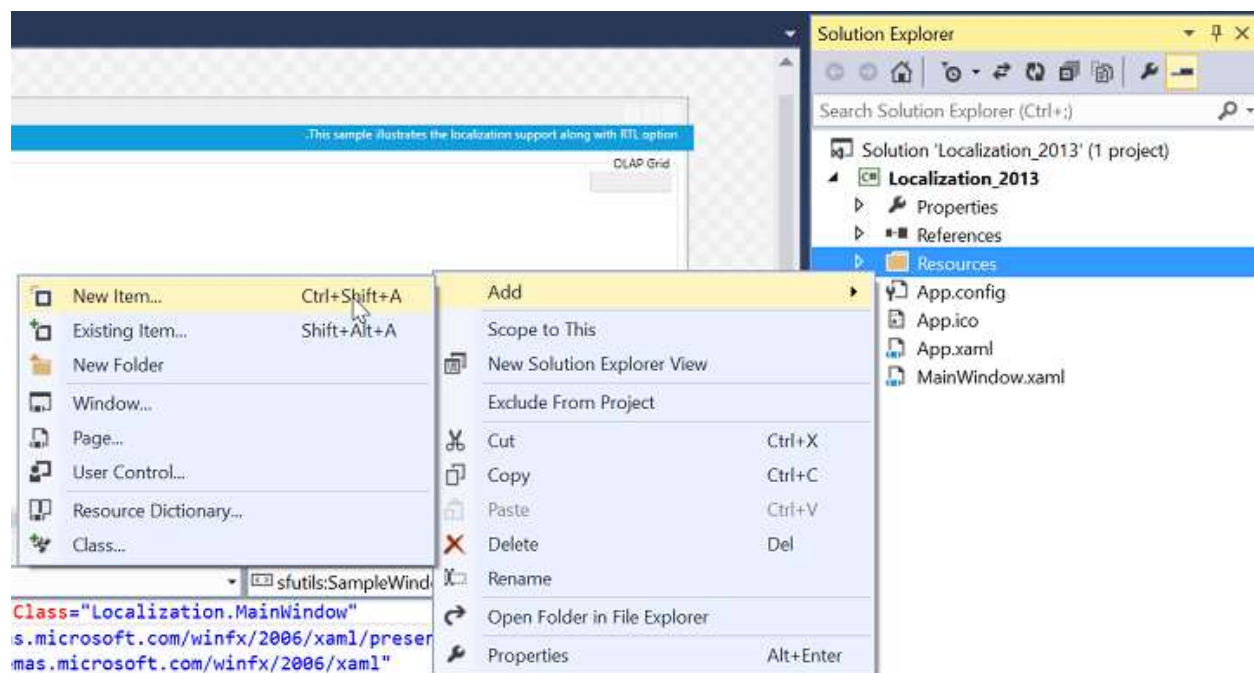
Resource file and file name conventions

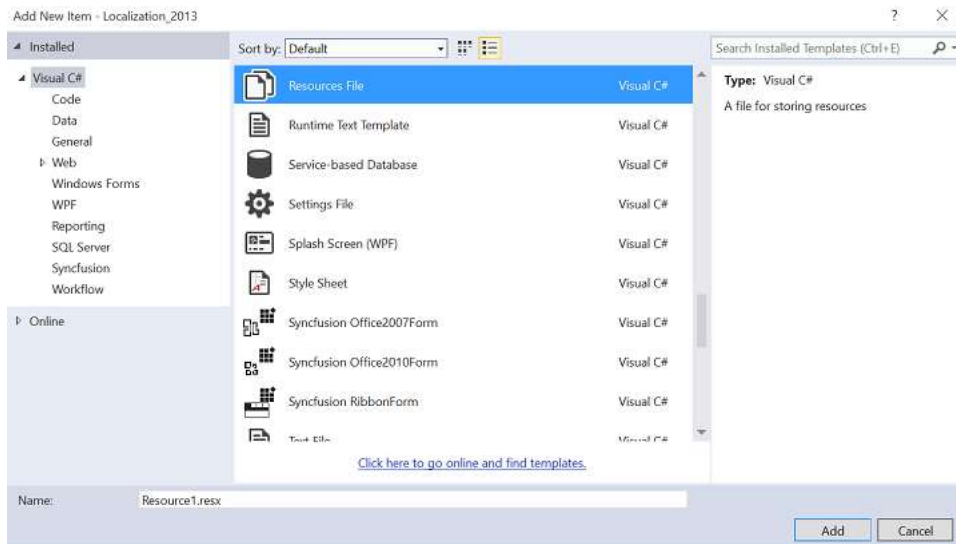
Steps to be followed in the application, after translating the strings that can be localized:

1. Right-click the **Project File** to create a new folder in the project. Select **Add > New Folder** and rename the folder as "Resources".

Note: The folder name should strictly be "Resources".

2. Right-click the Resources folder to create a new resource file in the Visual Studio project. Go to **Add > New Item**.





3. Select "Resources File" from the list. Then, name the resource file: Syncfusion.OlapGrid.WPF.ar-AE.resx and click **Add**.

Note: The resource file name should strictly be in the format "Syncfusion.OlapGrid.WPF.<Culture Code>.resx".

4. Copy and paste the translated locale to the resource file created in the earlier step.

Specifying the CurrentUICulture

Now, you need to specify the CurrentUICulture of the application. You can specify the CurrentUICulture from the Application_Startup in App.xaml.cs or the constructor in the main page (if you are specifying the current culture on the main page, make sure that it is assigned before the InitializeComponent method).

C#

```
public MainWindow()
{
    //Set the current thread culture to load the localization resource file.
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("ar-AE");
    InitializeComponent();
    if (System.Globalization.CultureInfo.CurrentUICulture.ToString() == "ar-AE")
    this.FlowDirection = System.Windows.FlowDirection.RightToLeft;
}
```

RTL support

The RTL support for OLAP grid is used to display the content from right-to-left by setting the **FlowDirection** property to "RightToLeft". The following code sample explains how to set this property.

XML

```
<syncfusion:OlapGrid x:Name="olapGrid" FlowDirection="RightToLeft"
ReportName="SalesReport" SharedDataManagerName="localManager"
olapshared:DataSource.DataManagerName="localManager"/>
```

C#

```
this.olapGrid.FlowDirection = System.Windows.FlowDirection.RightToLeft;
```

VB.NET

```
Me.olapGrid.FlowDirection = System.Windows.FlowDirection.RightToLeft;
```

FY 2003				FY 2002			
Revenue Goal	Sales Amount	Revenue Trend	Revenue Status	Revenue Goal	Sales Amount		
42337.07336	124,433.35 د.ا	←	▲	36814.8464	36,814.85 د.ا	United States	Accessories
42337.07336	124,433.35 د.ا	←	▲	36814.8464	36,814.85 د.ا	Total	
25404211.062249	28,179,553.99 د.ا	←	▲	22090618.3149991	22,090,618.31 د.ا	United States	Bikes
25404211.062249	28,179,553.99 د.ا	←	▲	22090618.3149991	22,090,618.31 د.ا	Total	
76276.66479	750,716.33 د.ا	←	▲	66327.5346	66,327.53 د.ا	United States	Clothing
76276.66479	750,716.33 د.ا	←	▲	66327.5346	66,327.53 د.ا	Total	
1341780.112595	4,629,101.14 د.ا	←	▲	1166765.3153	1,166,765.32 د.ا	United States	Components
1341780.112595	4,629,101.14 د.ا	←	▲	1166765.3153	1,166,765.32 د.ا	Total	
...26864604.9129	...33,683,804. د.ا	←	▲	23360526.0112991	23,360,526.01 د.ا	United States	Total
...26864604.9129	...33,683,804. د.ا	←	▲	23360526.0112991	23,360,526.01 د.ا	Total	

A sample is locally available in the following location.

```
{system drive}\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version  
Number>\WPF\OlapGrid.WPF\Samples\Localization\Localization
```

OlapGauge

WPF Olap Gauge Overview

The OLAP gauge control for WPF is ideal for highlighting business critical Key Performance Indicator (KPI) information in executive dashboards and report cards. It allows you to present values against goals in a very intuitive manner. Built-in data binding support allows you to easily bind an OLAP gauge to a KPI result from your OLAP database. The OLAP gauge control with sophisticated customization support provides endless possibility for control customization.

Key features

The key features of the OLAP gauge control are:

- **Data source:** Supports OLAP data binding with XML/A data sources.
- **Multiple gauges and layouts:** Supports to customize the layout while rendering multiple controls.
- **Tooltip:** Displays the value and goal information in the tooltip.
- **Indicators:** Displays the active/inactive state of the OLAP gauge.
- **Ranges:** Highlights the range of values in scale.
- **Pointers:** Points the actual value and goal information.
- **Frame types:** Various built-in frame types provide a rich appearance of the OLAP gauge control.
- **Themes:** Supports different themes, such as Office 2007 Blue, Office 2007 Black, Office 2007 Silver, and so on to customize the appearance of the OLAP gauge.

- **Visibility:** Shows or hides the headers, factors, and labels.

Getting Started with WPF Olap Gauge

Important

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

This section covers the information required to create a simple OLAP gauge control bound to the OLAP data source.

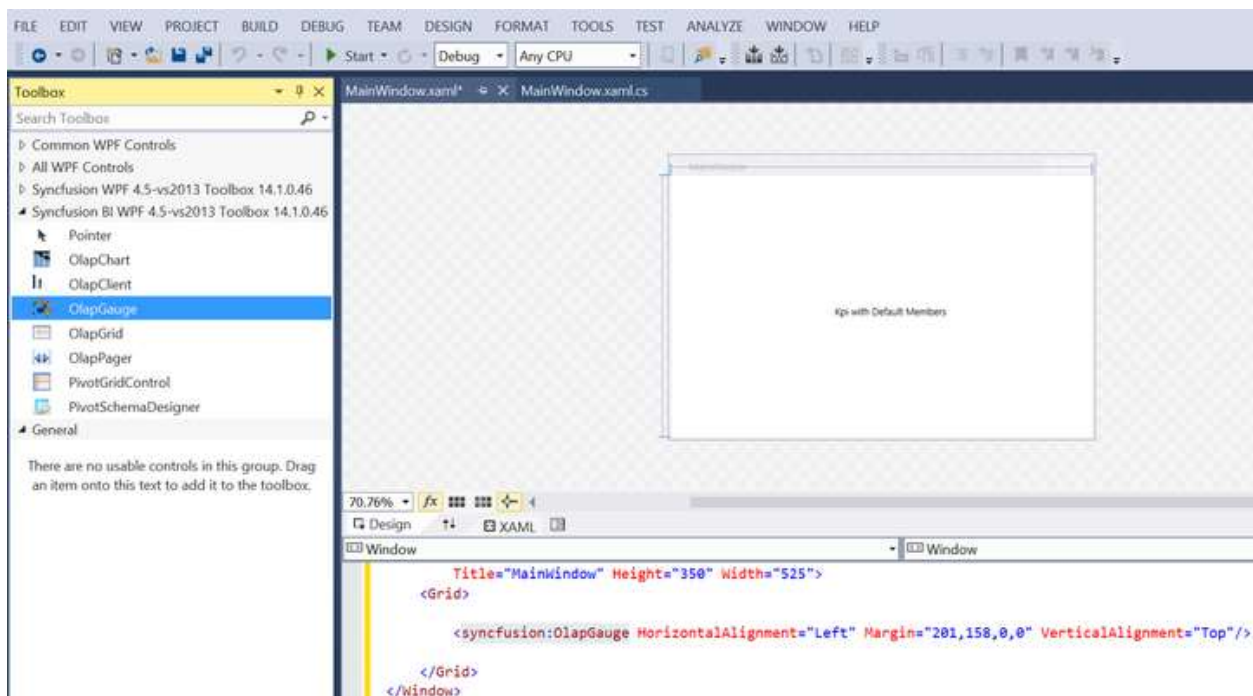
Control initialization

The OLAP gauge control can be initialized and added to an application through the following ways:

- Through Visual Studio
- Through Expression Blend
- Through code-behind

Adding control through Visual Studio

1. Open Visual Studio IDE and go to File > New > Project > WPF Application inside the Visual C# Templates to create a new WPF application.
2. Select the toolbox option from the view menu. It will appear inside the Visual Studio IDE.
3. From the toolbox, select the OLAP gauge control under “Syncfusion BI WPF” group, and then drag it to the designer section of the MainPage.xaml file.



- Then, name the added OLAP gauge control as “OlapGauge1” in MainPage.xaml to refer it in code-behind as follows:

XML

```
<syncfusion:OlapGauge x:Name="OlapGauge1"/>
```

OLAP report and OLAP data manager declaration

Include the following namespaces in the code-behind for using the OlapReport and OlapDataManager in the application.

- Syncfusion.Olap.Reports
- Syncfusion.Olap.Manager

C#

```
using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
using System.Windows;
namespace OlapGaugeApp
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            var olapDataManager = new OlapDataManager("Data
Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure
Works DW 2008 SE;");
            olapDataManager.SetCurrentReport(CreateOlapReport());
            this.OlapGauge1.OlapDataManager = olapDataManager;
        }
        /// <summary>
        /// Defining OlapReport with Dimension and Measure
        /// </summary>
        private OlapReport CreateOlapReport()
        {
            OlapReport report = new OlapReport();
            report.CurrentCubeName = "Adventure Works";
            KpiElements kpiElement = new KpiElements();
            kpiElement.Elements.Add(new KpiElement { Name = "Revenue", ShowKPIGoal =
true, ShowKPIStatus = true, ShowKPIValue = true, ShowKPITrend = true });
            DimensionElement dimensionElement1 = new DimensionElement();
            DimensionElement dimensionElement2 = new DimensionElement();
            DimensionElement dimensionElement3 = new DimensionElement();
            dimensionElement1.Name = "Date";
            dimensionElement1.AddLevel("Fiscal Year", "Fiscal Year");
            dimensionElement2.Name = "Sales Channel";
            dimensionElement2.AddLevel("Sales Channel", "Sales Channel");
            dimensionElement2.Hierarchy.LevelElements["Sales Channel"].Add("Reseller");
```

```

dimensionElement2.Hierarchy.LevelElements["Sales
Channel"].IncludeAvailableMembers = true;
dimensionElement3.Name = "Product";
dimensionElement3.AddLevel("Product Model Lines", "Product Line");
dimensionElement3.Hierarchy.LevelElements["Product Line"].Add("Road");
dimensionElement3.Hierarchy.LevelElements["Product
Line"].IncludeAvailableMembers = true;
report.CategoricalElements.Add(new Item { ElementValue = dimensionElement2
});
report.CategoricalElements.Add(new Item { ElementValue = dimensionElement1
});
report.CategoricalElements.Add(new Item { ElementValue = kpiElement });
report.SeriesElements.Add(new Item { ElementValue = dimensionElement3 });
return report;
}
}
}

```

VB.NET

```

Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Imports System.Windows
Namespace OlapGaugeApp
    ''' <summary>
    ''' Interaction logic for MainWindow.xaml
    ''' </summary>
    Partial Public Class MainWindow
        Inherits Window
        Public Sub New()
            InitializeComponent()
            Dim olapDataManager = New OlapDataManager("Data
Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure
Works DW 2008 SE;")
            olapDataManager.SetCurrentReport(CreateOlapReport())
            Me.OlapGauge1.OlapDataManager = olapDataManager
        End Sub
        ''' <summary>
        ''' Defining OlapReport with Dimension and Measure
        ''' </summary>
        Private Function CreateOlapReport() As OlapReport
            Dim report As New OlapReport()
            report.CurrentCubeName = "Adventure Works"
            Dim kpiElement As New KpiElements()
            kpiElement.Elements.Add(New KpiElement With {.Name = "Revenue", .ShowKPIGoal
= True, .ShowKPIStatus = True, .ShowKPIValue = True, .ShowKPITrend = True})
            Dim dimensionElement1 As New DimensionElement()
            Dim dimensionElement2 As New DimensionElement()
            Dim dimensionElement3 As New DimensionElement()
            dimensionElement1.Name = "Date"
            dimensionElement1.AddLevel("Fiscal Year", "Fiscal Year")
            dimensionElement2.Name = "Sales Channel"
            dimensionElement2.AddLevel("Sales Channel", "Sales Channel")
            dimensionElement2.Hierarchy.LevelElements("Sales Channel").Add("Reseller")
            dimensionElement2.Hierarchy.LevelElements("Sales
Channel").IncludeAvailableMembers = True

```



```
dimensionElement3.Name = "Product"
dimensionElement3.AddLevel("Product Model Lines", "Product Line")
dimensionElement3.Hierarchy.LevelElements("Product Line").Add("Road")
dimensionElement3.Hierarchy.LevelElements("Product
Line").IncludeAvailableMembers = True
report.CategoricalElements.Add(New Item With {.ElementValue =
dimensionElement2})
report.CategoricalElements.Add(New Item With {.ElementValue =
dimensionElement1})
report.CategoricalElements.Add(New Item With {.ElementValue = kpiElement})
report.SeriesElements.Add(New Item With {.ElementValue = dimensionElement3})
Return report
End Function
End Class
End Namespace
```

Adding control through Expression Blend

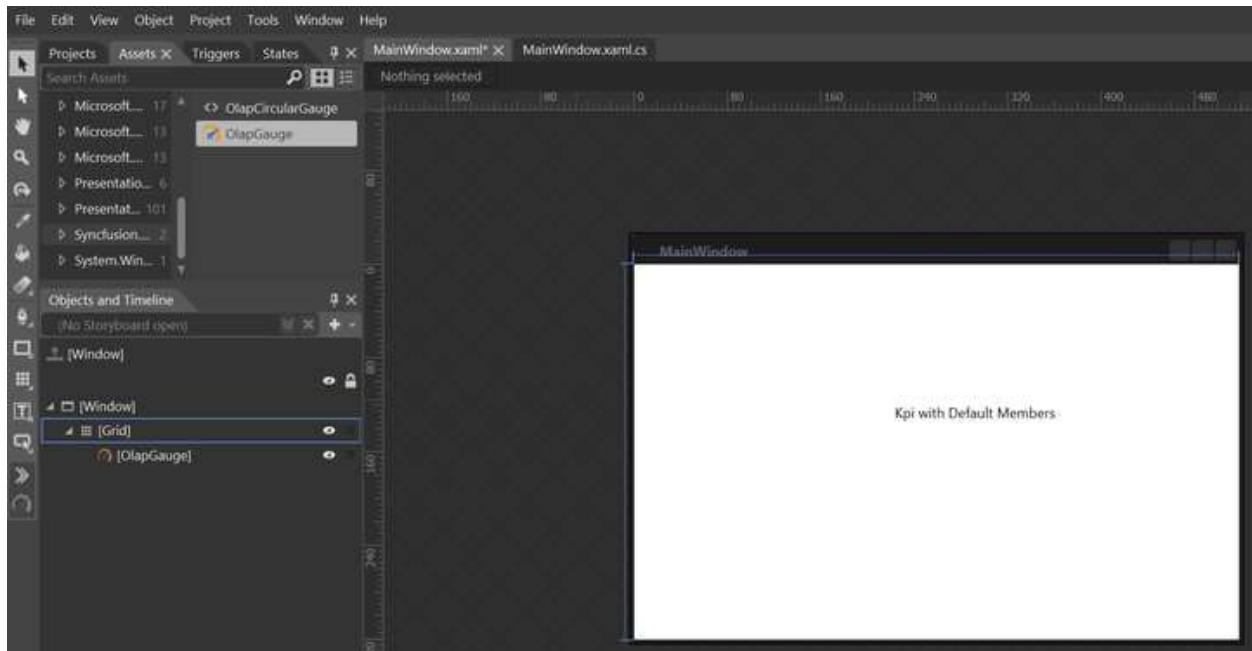
1. Open Blend for Visual Studio IDE and navigate to File > New project > WPF > WPF Application to create a new WPF application.
2. Select the **Project** tab available in the left corner of the Blend IDE and right-click **References** to select **Add Reference**.
3. Then, browse the following assemblies and add it to the project.
 - Syncfusion.Gauge.WPF
 - Syncfusion.Core
 - Syncfusion.Olap.Base
 - Syncfusion.OlapGauge.WPF
 - Syncfusion.OlapShared.WPF
 - Syncfusion.Shared.WPF

Note: You can find these libraries under the following location:

{Installed Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<framework version>\

On adding the above assemblies, the OLAP gauge control will be added under the **Assets** tab automatically.

4. Now, choose the **Assets** tab and drag the OLAP gauge control to the designer.



- Then, name the added OLAP gauge control as "OlapGauge1" in MainPage.xaml to refer it in code-behind as follows:

XML

```
<syncfusion:OlapGauge x:Name="OlapGauge1"/>
```

To add the OlapReport and OlapDataManager in the application, refer to the [OlapReport and OlapDataManager declaration](#) section.

Adding control through code-behind

- Open Visual Studio IDE and navigate to File > New > Project > WPF Application inside the Visual C# Templates to create a new WPF application.
- To add the dependency assemblies within the application, right-click the **References** and select **Add Reference**.
- Add the following Syncfusion assemblies manually to the project from the installed location.
 - Syncfusion.Gauge.WPF
 - Syncfusion.Core
 - Syncfusion.Olap.Base
 - Syncfusion.OlapGauge.WPF
 - Syncfusion.OlapShared.WPF
 - Syncfusion.Shared.WPF

Note: You can find these libraries under the following location:

```
{Installed Drive}:\Program Files (x86)\Syncfusion\Essential Studio\<version number>\precompiledassemblies\<version number>\<framework version>
```

- Then, name the grid in MainWindow.xaml as "RootGrid" as specified below.

XML

```
<Grid x:Name="RootGrid"/>
```

- Include the following namespaces in code-behind for using OlapGauge, OlapReport, and OlapDataManager in the application.
 - Syncfusion.Olap.Reports
 - Syncfusion.Olap.Manager
 - Syncfusion.Windows.Gauge.Olap

C#

```
using System.Windows;
using Syncfusion.Olap.Manager;
using Syncfusion.Olap.Reports;
using Syncfusion.Windows.Gauge.Olap;
namespace OlapGaugeApp
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            var olapGauge = new OlapGauge();
            var olapDataManager = new OlapDataManager("Data
Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure
Works DW 2008 SE;");
            olapDataManager.SetCurrentReport(CreateOlapReport());
            olapGauge.OlapDataManager = olapDataManager;
            this.RootGrid.Children.Add(olapGauge);
        }
        /// <summary>
        /// Defining OlapReport with Dimension and Measure
        /// </summary>
        private OlapReport CreateOlapReport()
        {
            OlapReport report = new OlapReport();
            report.CurrentCubeName = "Adventure Works";
            KpiElements kpiElement = new KpiElements();
            kpiElement.Elements.Add(new KpiElement { Name = "Revenue", ShowKPIGoal =
true, ShowKPIStatus = true, ShowKPIValue = true, ShowKPI Trend = true });
            DimensionElement dimensionElement1 = new DimensionElement();
            DimensionElement dimensionElement2 = new DimensionElement();
            DimensionElement dimensionElement3 = new DimensionElement();
            dimensionElement1.Name = "Date";
            dimensionElement1.AddLevel("Fiscal Year", "Fiscal Year");
            dimensionElement2.Name = "Sales Channel";
            dimensionElement2.AddLevel("Sales Channel", "Sales Channel");
            dimensionElement2.Hierarchy.LevelElements["Sales Channel"].Add("Reseller");
```

```

dimensionElement2.Hierarchy.LevelElements["Sales
Channel"].IncludeAvailableMembers = true;
dimensionElement3.Name = "Product";
dimensionElement3.AddLevel("Product Model Lines", "Product Line");
dimensionElement3.Hierarchy.LevelElements["Product Line"].Add("Road");
dimensionElement3.Hierarchy.LevelElements["Product
Line"].IncludeAvailableMembers = true;
report.CategoricalElements.Add(new Item { ElementValue = dimensionElement2
});
report.CategoricalElements.Add(new Item { ElementValue = dimensionElement1
});
report.CategoricalElements.Add(new Item { ElementValue = kpiElement });
report.SeriesElements.Add(new Item { ElementValue = dimensionElement3 });
return report;
}
}
}

```

VB.NET

```

Imports System.Windows
Imports Syncfusion.Olap.Manager
Imports Syncfusion.Olap.Reports
Imports Syncfusion.Windows.Gauge.Olap
Namespace OlapGaugeApp
    ''' <summary>
    ''' Interaction logic for MainWindow.xaml
    ''' </summary>
    Partial Public Class MainWindow
        Inherits Window
        Public Sub New()
            InitializeComponent()
            Dim olapGauge = New OlapGauge()
            Dim olapDataManager = New OlapDataManager("Data
Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure
Works DW 2008 SE;")
            olapDataManager.SetCurrentReport(CreateOlapReport())
            olapGauge.OlapDataManager = olapDataManager
            Me.RootGrid.Children.Add(olapGauge)
        End Sub
        ''' <summary>
        ''' Defining OlapReport with Dimension and Measure
        ''' </summary>
        Private Function CreateOlapReport() As OlapReport
            Dim report As New OlapReport()
            report.CurrentCubeName = "Adventure Works"
            Dim kpiElement As New KpiElements()
            kpiElement.Elements.Add(New KpiElement With { .Name = "Revenue", .ShowKPIGoal
= True, .ShowKPIStatus = True, .ShowKPIValue = True, .ShowKPI Trend = True })
            Dim dimensionElement1 As New DimensionElement()
            Dim dimensionElement2 As New DimensionElement()
            Dim dimensionElement3 As New DimensionElement()
            dimensionElement1.Name = "Date"
            dimensionElement1.AddLevel("Fiscal Year", "Fiscal Year")
            dimensionElement2.Name = "Sales Channel"
            dimensionElement2.AddLevel("Sales Channel", "Sales Channel")

```

```

dimensionElement2.Hierarchy.LevelElements("Sales Channel").Add("Reseller")
dimensionElement2.Hierarchy.LevelElements("Sales Channel").IncludeAvailableMembers = True
dimensionElement3.Name = "Product"
dimensionElement3.AddLevel("Product Model Lines", "Product Line")
dimensionElement3.Hierarchy.LevelElements("Product Line").Add("Road")
dimensionElement3.Hierarchy.LevelElements("Product Line").IncludeAvailableMembers = True
report.CategoricalElements.Add(New Item With {.ElementValue = dimensionElement2})
report.CategoricalElements.Add(New Item With {.ElementValue = dimensionElement1})
report.CategoricalElements.Add(New Item With {.ElementValue = kpiElement})
report.SeriesElements.Add(New Item With {.ElementValue = dimensionElement3})
Return report
End Function
End Class
End Namespace

```

Finally, run the application. The OLAP gauge control is rendered as follows.



Data Binding in WPF Olap Gauge

Binding an OLAP gauge to offline cube

To connect to an OLAP cube available in the local machine, the physical path of the cube should be specified in the connection string. The following code snippet illustrates the same.

C#

```

string connectionString = @"DataSource = system
drive:\Cube\Adventure_Works_Ext.cub; Provider = MSOLAP;";
OlapDataManager DataManager = new OlapDataManager(connectionString);

```

Binding OLAP gauge to cube in local SQL Server

To connect to the OLAP cube available in SQL Server Analysis Service in the local machine, the server name and database name should be specified in the connection string. The following code example illustrates the same.

Note: If any credentials are maintained to connect the cube, then the user ID and password attributes should be mentioned accordingly.

C#

```
string connectionString = "Data source=localhost; Initial Catalog=Adventure Works DW;";  
OlapDataManager DataManager = new OlapDataManager(connectionString);
```

VB.NET

```
Dim connectionString As String = "Data source=localhost; Initial Catalog=Adventure Works DW;"  
Dim DataManager As New OlapDataManager(connectionString)
```

Binding OLAP gauge to cube in online SQL Server

To connect to the OLAP cube available in SQL server Analysis Service in the online server through **XML/A**, the host server link and database name should be specified in the connection string. The following code example illustrates the same.

Note: If any credentials are maintained to connect the cube, then the user ID and password attributes should be mentioned accordingly.

C#

```
string connectionString = "Data Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure Works DW 2008 SE;";  
OlapDataManager DataManager = new OlapDataManager(connectionString);
```

VB.NET

```
Dim connectionString As String = "Data Source=http://bi.syncfusion.com/olap/msmdpump.dll; Initial Catalog=Adventure Works DW 2008 SE;"  
Dim DataManager As New OlapDataManager(connectionString)
```

Binding OLAP gauge to cube in online Mondrian server

To connect to the OLAP cube available in Mondrian server through **XML/A**, the host server link and database name should be specified in the connection string. The following code example illustrates the same.

Note: If any credentials are maintained to connect the cube, then the user ID and password attributes should be mentioned accordingly.

C#

```
string connectionString = @"Data Source =  
http://localhost:8080/mondrian/xmla; Initial Catalog =FoodMart;";  
OlapDataManager DataManager = new OlapDataManager(connectionString);  
DataManager.DataProvider.ProviderName =  
Syncfusion.Olap.DataProvider.Providers.Mondrian;
```

VB.NET

```
Dim connectionString As String = "Data Source =  
http://localhost:8080/mondrian/xmla; Initial Catalog =FoodMart;"  
Dim DataManager As New OlapDataManager(connectionString)
```

```
DataManager.DataProvider.ProviderName =
Syncfusion.Olap.DataProvider.Providers.Mondrian
```

Binding OLAP gauge to cube in online ActivePivot Server

To connect to the OLAP cube available in ActivePivot server through **XML/A**, the host server link and database name should be specified in the connection string. The following code example illustrates the same.

Note: If any credentials are maintained to connect the cube, then the user ID and password attributes should be mentioned accordingly.

C#

```
string connectionString = @"Data Source = http://localhost:8080/cva_s/xmla;
Initial Catalog = CVAS;";
OlapDataManager DataManager = new OlapDataManager(connectionString);
DataManager.DataProvider.ProviderName=Syncfusion.Olap.DataProvider.Providers
.ActivePivot;
```

VB.NET

```
Dim connectionString As String = "Data Source =
http://localhost:8080/cva_s/xmla; Initial Catalog = CVAS;"
Dim DataManager As New OlapDataManager(connectionString)
DataManager.DataProvider.ProviderName=Syncfusion.Olap.DataProvider.Providers
.ActivePivot
```

XAML Configuration in WPF Olap Gauge

XAML configuration is one of the important features of the OLAP gauge, as it helps users to configure the control entirely through XAML by eliminating the required code in code behind.

Properties

- **DataSource.ConnectionString:** Specifies the connection string of the data manager.
- **DataSource.ConnectionName:** Specifies the connection name, which is available in the App.Config file of the application.
- **DataSource.DataManagerName:** Specifies the data manager name.
- **SharedDataManagerName:** Specifies the data manager name, which is available in the shared data manager collection.
- **ReportName:** Species the name of the OLAP report.
- **CurrentCubeName:** Specifies the current cube name of the OLAP report.
- **CategoricalAxis:** Specifies the categorical axis of the OLAP report.
- **SeriesAxis:** Specifies the series axis of the OLAP report.
- **SlicerAxis:** Specifies the slicer axis of the OLAP report.
- **CalculatedMembers:** Specifies the calculated members of the OLAP report.

The following code snippet illustrates about adding an OLAP report to OLAP gauge in design time.

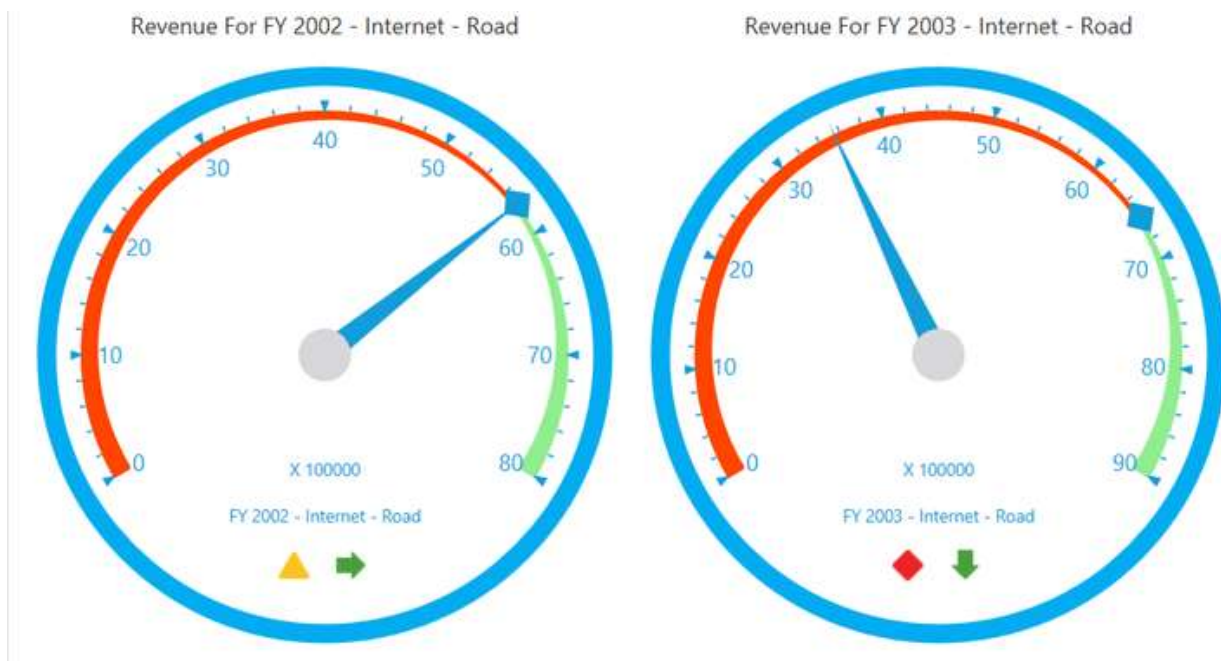
XML

```
<syncfusion:OlapGauge x:Name="olapGauge" CurrentCubeName="Adventure Works"
ReportName="SalesReport"
```

```

SharedDataManagerName="localManager"
olapshared:DataSource.DataManagerName="localManager"
olapshared:DataSource.ConnectionString="datasource=localhost;
initial catalog=adventure works dw">
<!-- Adding Elements to Categorical Axis -->
<syncfusion:OlapGauge.CategoricalAxis>
<syncfusion:Dimension Name="Date" HierarchyName="Fiscal" LevelName="Fiscal Y
ear" IncludeMembers="FY 2002, FY 2003" />      <!-- Multiple Members where
specified by comma separate -->
<syncfusion:Kpi Name="Revenue" ShowGoal="True"
ShowStatus="True" ShowValue="True" ShowTrend="True" />
</syncfusion:OlapGauge.CategoricalAxis>
<!-- Adding Elements to Series Axis -->
<syncfusion:OlapGauge.SeriesAxis>
<syncfusion:Dimension Name="Sales Channel" HierarchyName="Sales Channel" Lev
elName="Sales Channel" />
<syncfusion:Dimension Name="Product" HierarchyName="Product Model Lines" Lev
elName="Product Line" IncludeMembers="Road" />
</syncfusion:OlapGauge.SeriesAxis>
</syncfusion:OlapGauge>

```

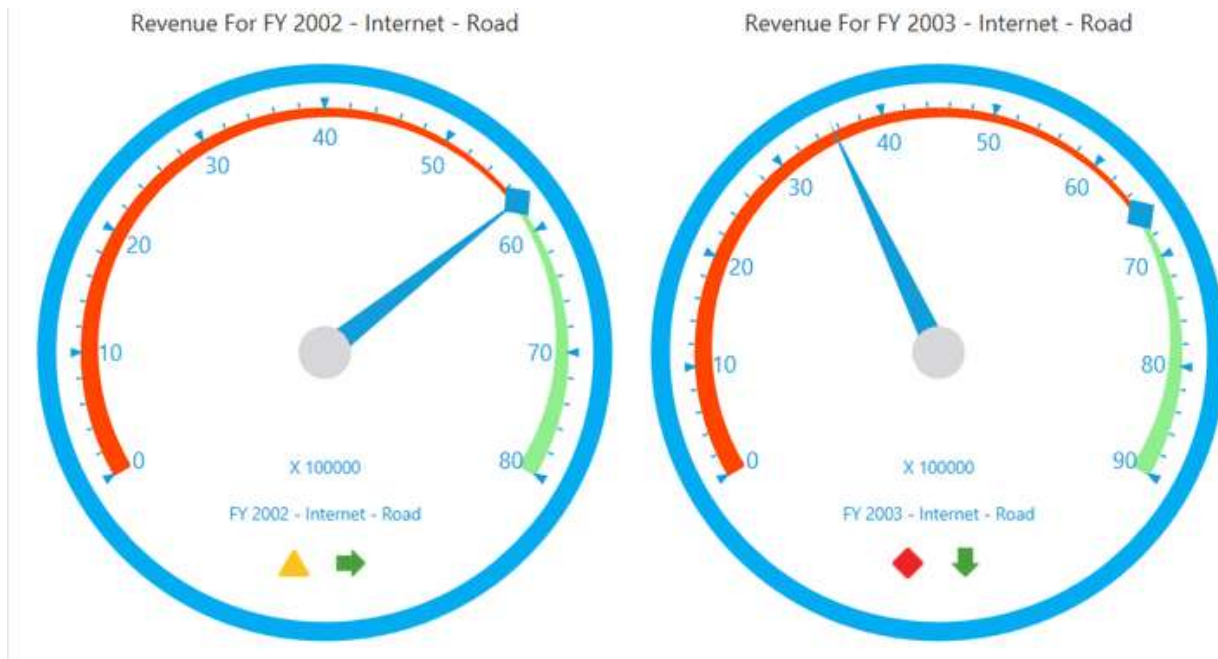


A demo sample is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGauge.WPF\Samples\Defining Reports\XAML Configuration\

KPI in WPF Olap Gauge

The OLAP gauge supports displaying Key Performance Indicators (KPI) from the OLAP cube. KPI value is represented with the help of *Pointers* and KPI goal is represented with the help of *Markers*. The KPI status and KPI trend values are represented through user friendly images such as traffic light, road signs, and standard arrow in the OLAP gauge of WPF. Each gauge represents a member against one KPI combination.



A sample demo is available at the following location.

{system drive}:\Users\\{User Name}\AppData\Local\Syncfusion\EssentialStudio\\{Version Number}\WPF\OlapGauge.WPF\Samples\Product Showcase\KPI\

Appearance in WPF Olap Gauge

Gauge radius

The OLAP gauge supports adjusting its radius and this can be achieved by assigning a proper value to the **Radius** property of OLAP gauge. The following code snippet illustrates about modifying the radius of OLAP gauge.

XML

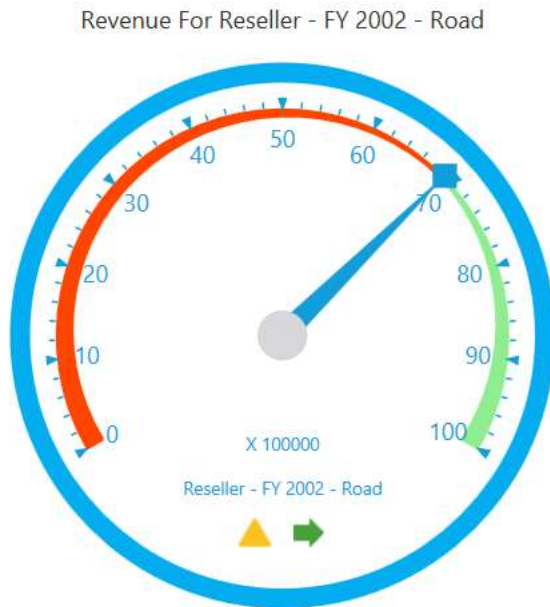
```
<syncfusion:OlapGauge Name="olapGauge1" Radius="120"/>
```

C#

```
this.OlapGauge1.Radius = 100;
```

VB.NET

```
Me.OlapGauge1.Radius = 100
```



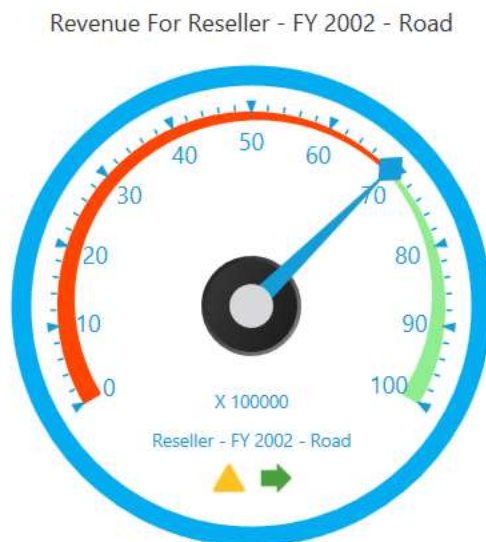
A demo sample is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGauge.WPF\Samples\Appearance\Customization\

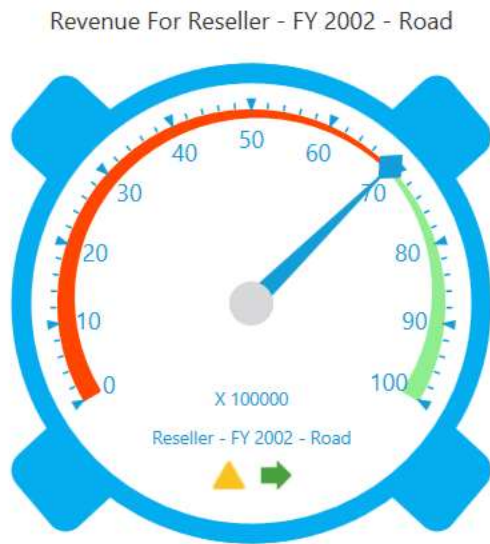
Built-in frame types

The OLAP gauge supports four types of built-in frames to provide effective rim styles. The **FrameType** property is used to set the frame type for the OLAP gauge. The following are the frame types supported by the OLAP gauge:

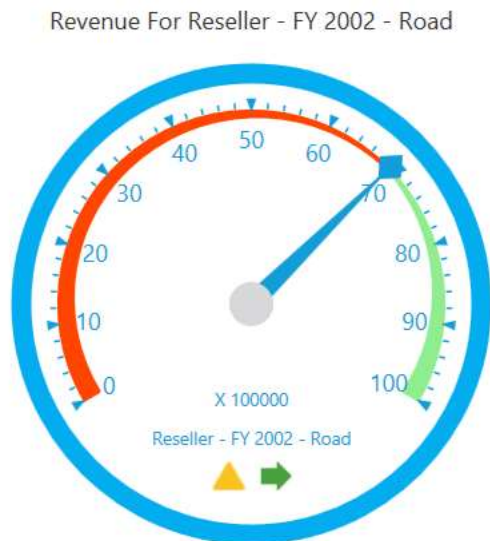
- CircularCenterGradient



- CircularWithDarkOuterFrames



- FullCircle



- HalfCircle

Revenue For Reseller - FY 2003 - Road



The following code snippet illustrates about how to set frame type for the OLAP gauge.

C#

```
this.OlapGauge1.FrameType = GaugeFrameType.CircularWithInnerLeftGradient;
```

VB.NET

```
Me.OlapGauge1.FrameType = GaugeFrameType.CircularWithInnerLeftGradient
```

A demo sample is available at the following location.

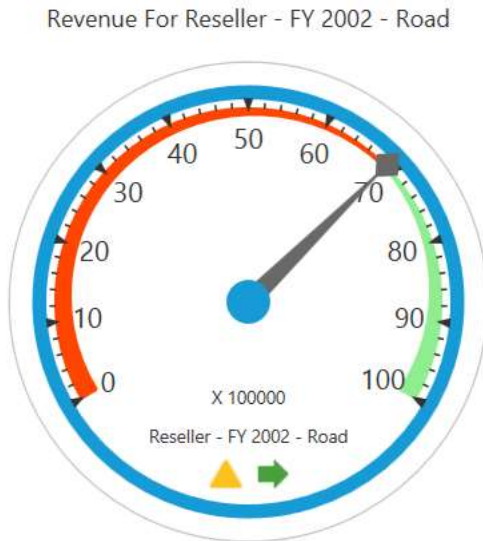
{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGauge.WPF\Samples\Appearance\Customization\

Skins

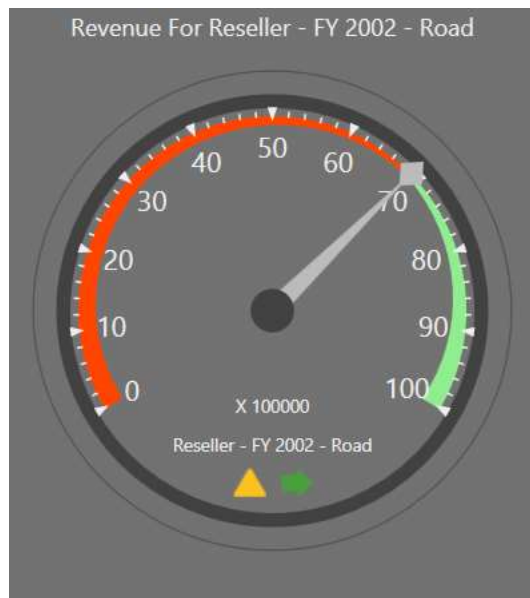
Built-in-skins allow you to customize the theme and style to improve the look and feel of the OLAP gauge control in various rich color schemes. You can use the skin manager framework to apply a wide range of skins to the OLAP gauge control. These skins have been designed to suit the needs of wide range of audience.

Types of skins:

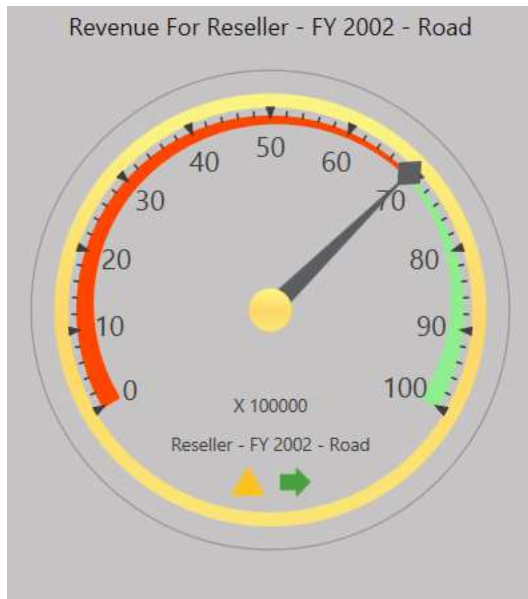
- Metro: This skin is similar to the Windows 8 Metro style.



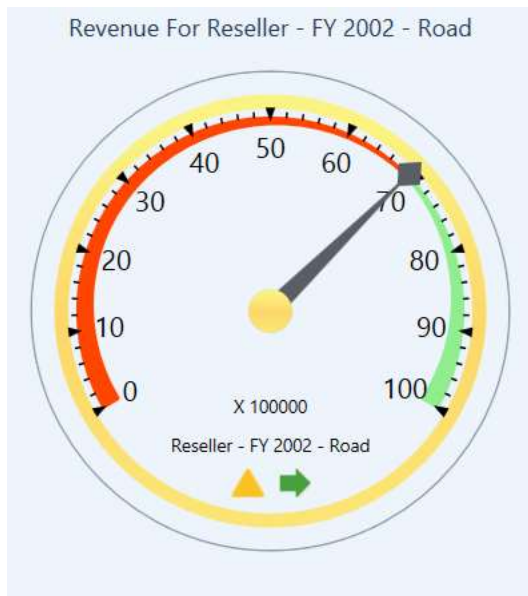
- Blend: This skin is similar to the Microsoft Blend skin.



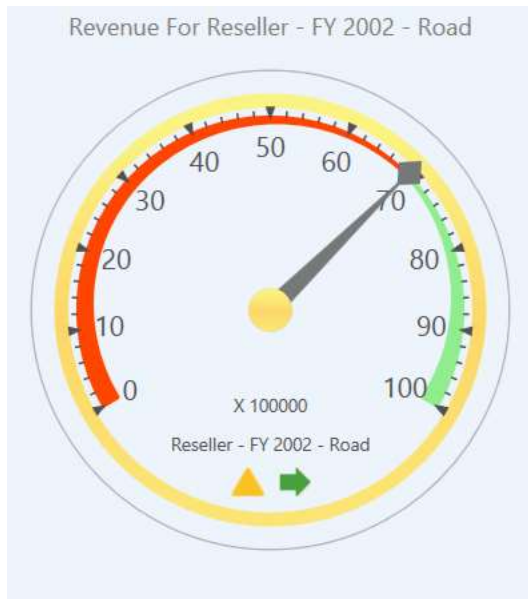
- Office2010Black: This skin is similar to the Microsoft Office2010Black skin.



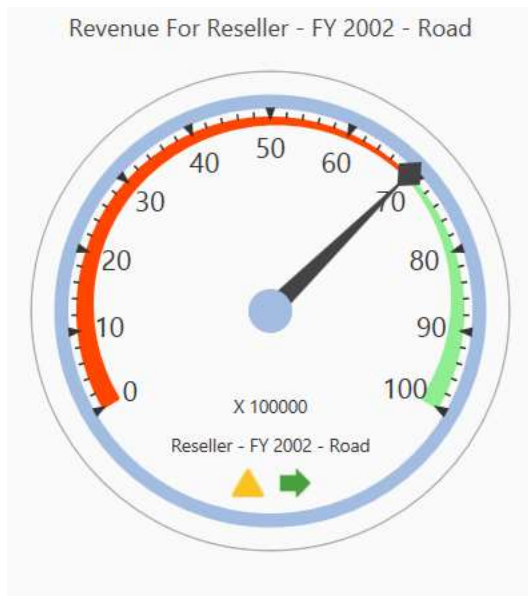
- Office2010Blue: This skin is similar to the Microsoft Office2010Blue skin.



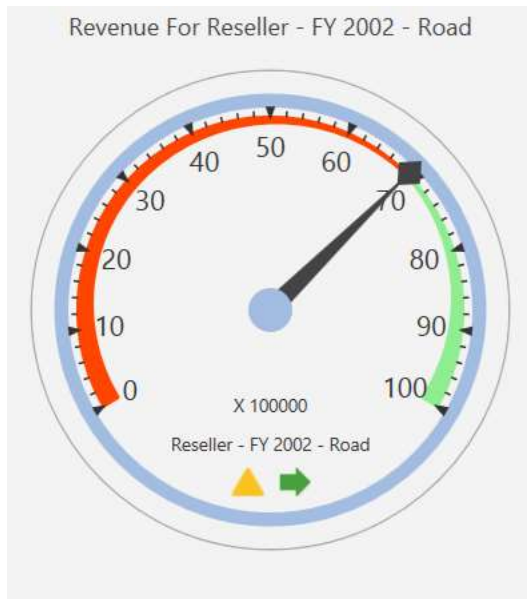
- Office2010Silver: This skin is similar to the Microsoft Office2010Silver skin.



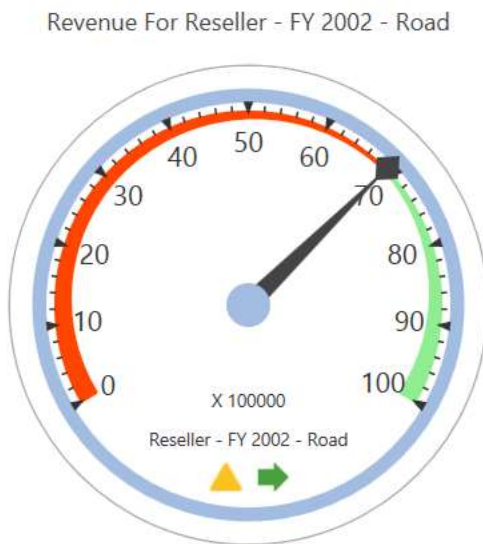
- Office2013LightGray: This skin is similar to the Microsoft Office2013LightGray skin.



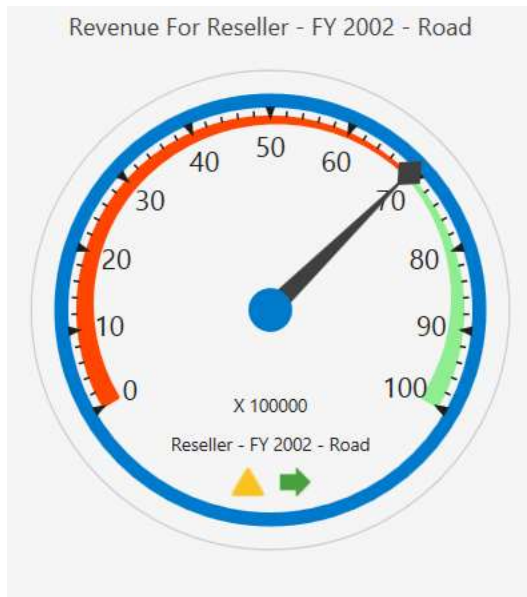
- Office2013DarkGray: This skin is similar to the Microsoft Office2013DarkGray skin.



- Office2013White: This skin is similar to the Microsoft Office2013White skin.



- VisualStudio2013: This skin is similar to the VisualStudio2013 skin.



To apply a skin to the OLAP gauge, use the `SkinStorageVisualStyle` property. The following code snippet shows how to set the visual style for the control.

XML

```
<syncfusion:OlapGauge Name="olapGauge1"
sfshared:SkinStorage.VisualStyle="Metro"/>
```

C#

```
SfSkinManager.SetVisualStyle(olapGauge1,
Syncfusion.SfSkinManager.VisualStyle.Metro);
```

VB.NET

```
SfSkinManager.SetVisualStyle(olapGauge1,
Syncfusion.SfSkinManager.VisualStyle.Metro);
```

A demo sample is available at the following location.

```
{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version
Number>\WPF\OlapGauge.WPF\Samples\Appearance\Visual Styles\
```

Gauge Customization in WPF Olap Gauge

Layout customization

The OLAP gauge displays multiple gauges in a structured layout. You can customize the layout by using the `ColumnsCount` and `RowCount` properties. These properties are used to specify the number of columns and rows for displaying the control.

XML

```
<syncfusion:OlapGauge x:Name="OlapGauge1" RowCount="2" ColumnsCount="2"/>
```

C#

```
this.OlapGauge1.ColumnsCount = 2;  
this.OlapGauge1.RowsCount = 2;
```

VB.NET

```
Me.OlapGauge1.ColumnsCount = 2  
Me.OlapGauge1.RowsCount = 2
```

Revenue For Reseller - FY 2002 - Road



Revenue For Reseller - FY 2003 - Road



Revenue For Reseller - FY 2004 - Road



Revenue For Reseller - FY 2005 - Road



Gauge header

The gauge header is the combination of details about the measure and KPI. The header components of the OLAP gauge can be hidden by using the `ShowGaugeHeaders` property as specified in the following code snippet.

XML

```
<syncfusion:OlapGauge x:Name="OlapGauge1" ShowGaugeHeaders="False"/>
```

C#

```
OlapGauge1.ShowGaugeHeaders = false;
```

VB.NET

```
OlapGauge1.ShowGaugeHeaders = False
```



Gauge label

The visibility of gauge labels that are displayed inside the gauge can be toggled with the help of `ShowGaugeLabels` property. The following code snippet shows how to hide labels of the OLAP gauge.

XML

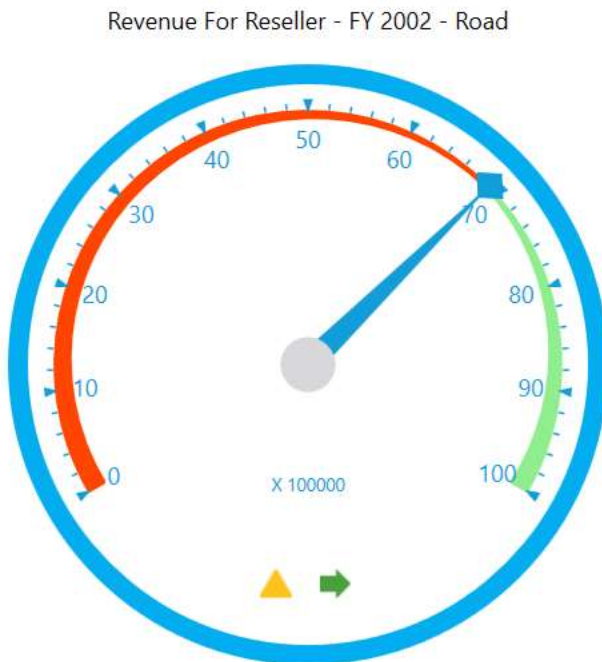
```
<syncfusion:OlapGauge x:Name="OlapGauge1" ShowGaugeLabels="False"/>
```

C#

```
OlapGauge1.ShowGaugeLabels = false;
```

VB.NET

```
OlapGauge1.ShowGaugeLabels = False
```



Gauge factor

The gauge factor component can be hidden by using the `ShowGaugeFactors` property as specified in the following code snippet.

XML

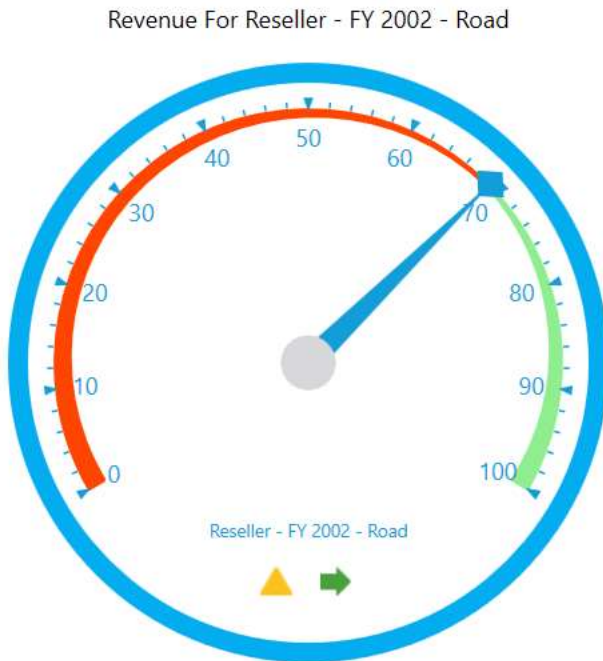
```
<syncfusion:OlapGauge x:Name="OlapGauge1" ShowGaugeFactors="False"/>
```

C#

```
OlapGauge1.ShowGaugeFactors = false;
```

VB.NET

```
OlapGauge1.ShowGaugeFactors = False
```



Tooltip in WPF Olap Gauge

The OLAP gauge provides the information about values when the mouse pointer is moved over the gauge.

Pointer tooltip

The OLAP gauge provides value information when the mouse pointer is moved over the pointer. This can be achieved by enabling the `ShowPointersTooltip` property.

The following code snippet illustrates how to show a tooltip for pointers.

XML

```
<syncfusion:OlapGauge x:Name="OlapGauge1" ShowPointersTooltip="True"/>
```

C#

```
this.OlapGauge1.ShowPointersTooltip = true;
```

VB.NET

```
Me.OlapGauge1.ShowPointersTooltip = True
```

Revenue For Reseller - FY 2004 - Accessory



Marker tooltip

The OLAP gauge provides goal information when the mouse pointer is moved over the marker. This can be achieved by enabling the `ShowMarkersTooltip` property.

The following code snippet illustrates how to show a tooltip for markers.

XML

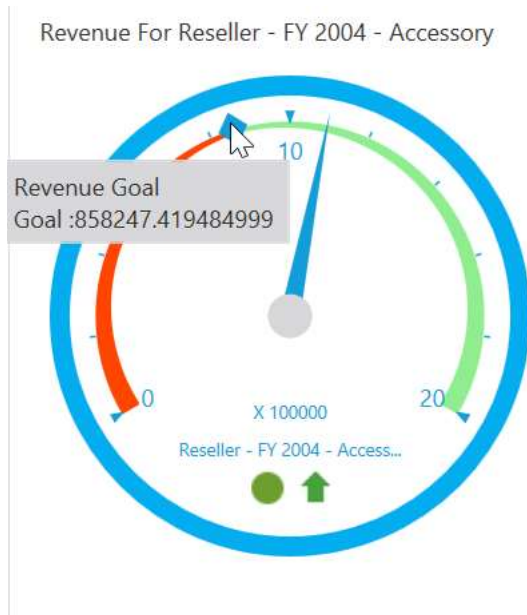
```
<syncfusion:OlapGauge x:Name="OlapGauge1" ShowMarkersTooltip="True"/>
```

C#

```
this.OlapGauge1.ShowMarkersTooltip = true;
```

VB.NET

```
Me.OlapGauge1.ShowMarkersTooltip = True
```



A demo sample is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGauge.WPF\Samples\Product ShowCase\KPI\

Localization in WPF Olap Gauge

Localization is the key feature to provide software solutions that are targeted at global users. The OLAP gauge allows users to localize the control to a specific locale and supports “resx” based localization.

You should perform the following steps to localize the control:

- Translation.
- Resource file and file name conventions.
- Culture specification.

Translation

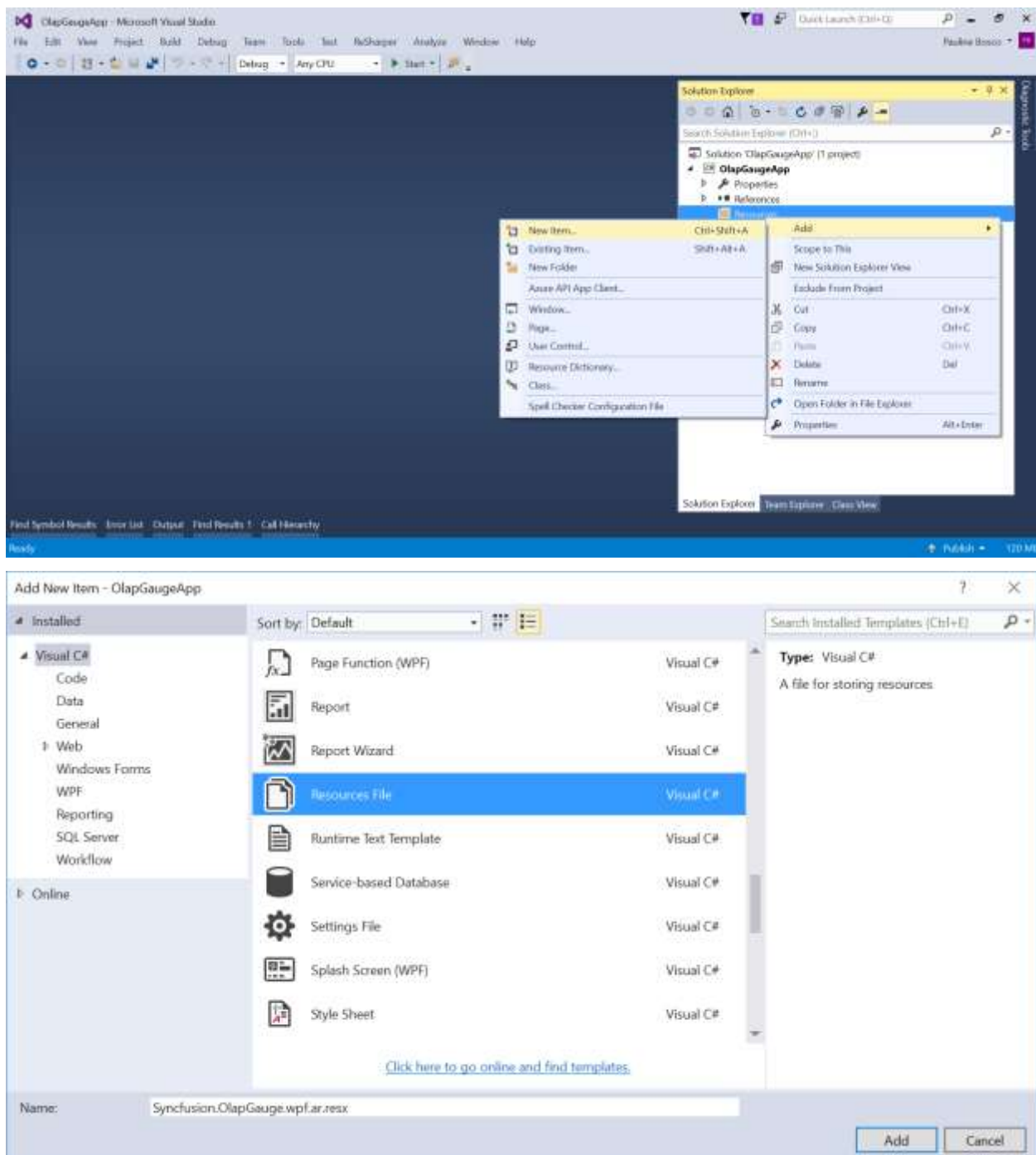
The first step in localization is translating the strings that can be localized to the destination locale.

Note: Localization key field should be same for all the locales. Do not translate the key fields.

Resource file and file name conventions

After translating the strings, perform the following steps in the application:

1. Right-click the project file to create a new folder in the project by selecting Add > New Folder and rename the folder as “Resources”.
2. Then, right-click the **Resources** folder to create a new resource file by selecting Add > New Item.



Note: The resource file name should be in the format “Syncfusion.OlapGauge.wpf.<Culture Code>.resx”.

3. Copy and paste the translated locale to the resource file which is created in the previous step.

Culture specification

You should specify the `CurrentUICulture` in the `Application_Startup` method of `App.xaml.cs` file or in the constructor of `MainPage.xaml.cs` file.

Note: If you are specifying the current culture in the constructor of MainPage, then ensure that the culture is specified before calling the InitializeComponent() method.

C#

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

VB.NET

```
Public Sub New()  
    System.Threading.Thread.CurrentThread.CurrentUICulture = New  
    System.Globalization.CultureInfo("ar")  
    InitializeComponent()  
End Sub
```

RTL

The OLAP gauge provides RTL support to display the content from right-to-left direction by setting the `FlowDirection` property to **RightToLeft**.

XML

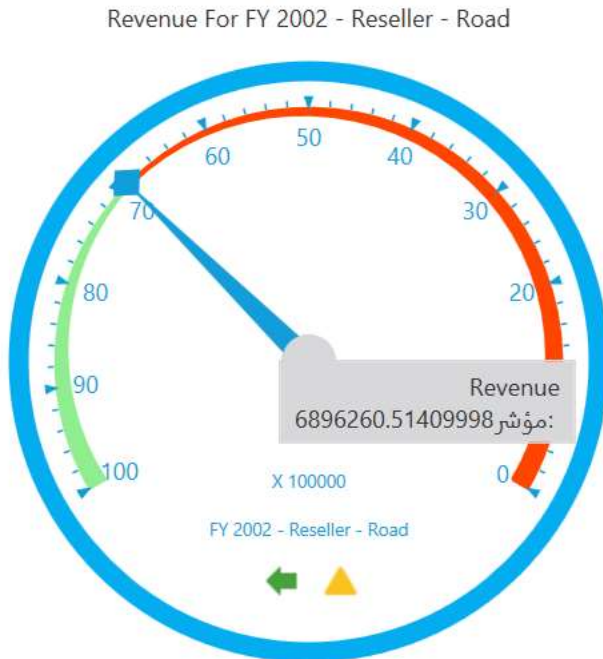
```
<syncfusion:OlapGauge x:Name="OlapGauge1" FlowDirection="RightToLeft"/>
```

C#

```
OlapGauge1.FlowDirection = FlowDirection.RightToLeft;
```

VB.NET

```
OlapGauge1.FlowDirection = FlowDirection.RightToLeft
```



A demo sample is available at the following location.

{system drive}:\Users\<User Name>\AppData\Local\Syncfusion\EssentialStudio\<Version Number>\WPF\OlapGauge.WPF\Samples\Localization\Localization Demo\

OLAP Common

Business Intelligence BI in WPF OLAP Common

What is BI?

Business Intelligence (BI) simplifies information to enable all decision makers of an organization to access information easily. This helps the decision makers at all level to understand, analyze, collaborate, and act on information anytime, anywhere.

Here is how Wikipedia defines [BI](#).

Why to use BI?

It is hard to over emphasize the importance of Business Intelligence (BI) in today's world. It is impossible to make strategic business decisions without analyzing the past business performance. Businesses are increasingly investing heavily in tools and services that help decision makers to visually analyze the data in myriad ways.

Several products and solutions have emerged to cater to the Business Intelligence market; Lesson have been learn, and currently Online Analytical Processing (OLAP) is the de facto standard for persisting and accessing BI data. Applications built using OLAP include sales reports, executive reports, forecasting, and so on.

What's new in BI?

While BI has been an expensive affair in the past, requiring several thousands of dollars in investment for integrating a solution into an enterprise, recent technological developments have considerably reduced the cost to implement and own an OLAP-based BI solution.

Microsoft's SQL Server Analysis Services (SSAS) is one such solution, which is a set of OLAP services provided as part of Microsoft SQL Server. The SSAS is available for almost a decade. It is a mature and very cost-effective solution for maintaining multidimensional (also called cube) data. With an efficient OLAP storage mechanism, you only need another efficient OLAP visualization tool set to complete your BI needs. This is where the Syncfusion OLAP controls come into place.

Multi-dimensional data

Multi-dimensional structure is defined as "a variation of the relational model that uses multidimensional structures to organize data and express the relationships between data". The structure is broken into cubes which are able to store and access data within the confines of each cube. Each cell within a multidimensional structure contains aggregated data related to elements along each of its dimensions. Even when data is manipulated, it is still easy to access as well as be a compact type of database. The data still remains interrelated. Multidimensional structure is quite popular for analytical databases that use OLAP applications. Analytical databases use these databases because of their ability to deliver answers quickly to complex business queries. Data can be seen through different ways, which gives a broader picture of a problem unlike other models.

Multi-dimensional database products were commercially popularized as Online Analytical Processing (OLAP) systems to help analysts do decision support on large historical data. They expose a multidimensional view of the data with categorical attributes like products and stores forming the dimensions and numeric attributes like sales and revenue forming the measures or cells of the multidimensional cube. Dimensions are usually associated with the hierarchies that specify aggregation levels. For instance, store-name ->city ->state is a hierarchy on the store dimension. The measure attributes are aggregated to various levels of detail about the combination of dimension attributes using functions like sum, average, count, and variance. OLAP products provide convenient tools for exploring the data cubes through navigational operators like select, drill-down, roll-up, and pivot conforming to the multidimensional view of data. An analyst can interactively invoke sequences of these simple operations to visualize the measures along various combinations of dimensions and at various levels of aggregation. Our OLAP Architecture provides you the easiest way to the extreme analysis of data.

Online Analytical Processing (OLAP) in WPF OLAP Common

Online Analytical Processing (OLAP) is a service that performs a real time, multi-dimensional analysis of complex data stored in a database. The OLAP typically consists of a server component that uses specialized algorithms, indexing tools to efficiently process data mining tasks and an OLAP client that efficiently allows you visualize the OLAP data pertaining to your business needs.

Here is how Wikipedia defines [OLAP](#).

ADOMD.NET

The Syncfusion OLAP controls use the [ADOMD.NET](#), which is a Microsoft .NET Framework provider for retrieving data from OLAP servers in .NET platform. While ADOMD.NET primarily retrieves OLAP data from the SQL Server Analysis Services (Microsoft's OLAP Server), it is adherence to industry standards like XML/A, which allows you access any OLAP server (SAP, SAS, Hyperion, etc.) using it. This provides you the ability to visualize the OLAP data from myriads of data sources including Microsoft's SSAS using the Syncfusion OLAP control.

ADOMD.NET assembly and setup files information

The following assembly is required to run the Syncfusion' OLAP samples.

- Microsoft.AnalysisServices.AdomdClient.dll

Install the following setup files for the above assembly.

- *SQLServer2005ADOMD.msi and SQLServer2005ASOLEDB9.msi*

Or

- *SQLSERVER2008ASADOMD10.msi and SQLSERVER2008ASOLEDB10.msi*

These setup files can be downloaded at [Microsoft download center](#).

Note: By default, the following setup files will be installed while installing the Syncfusion' Essential Studio setup for BI edition.

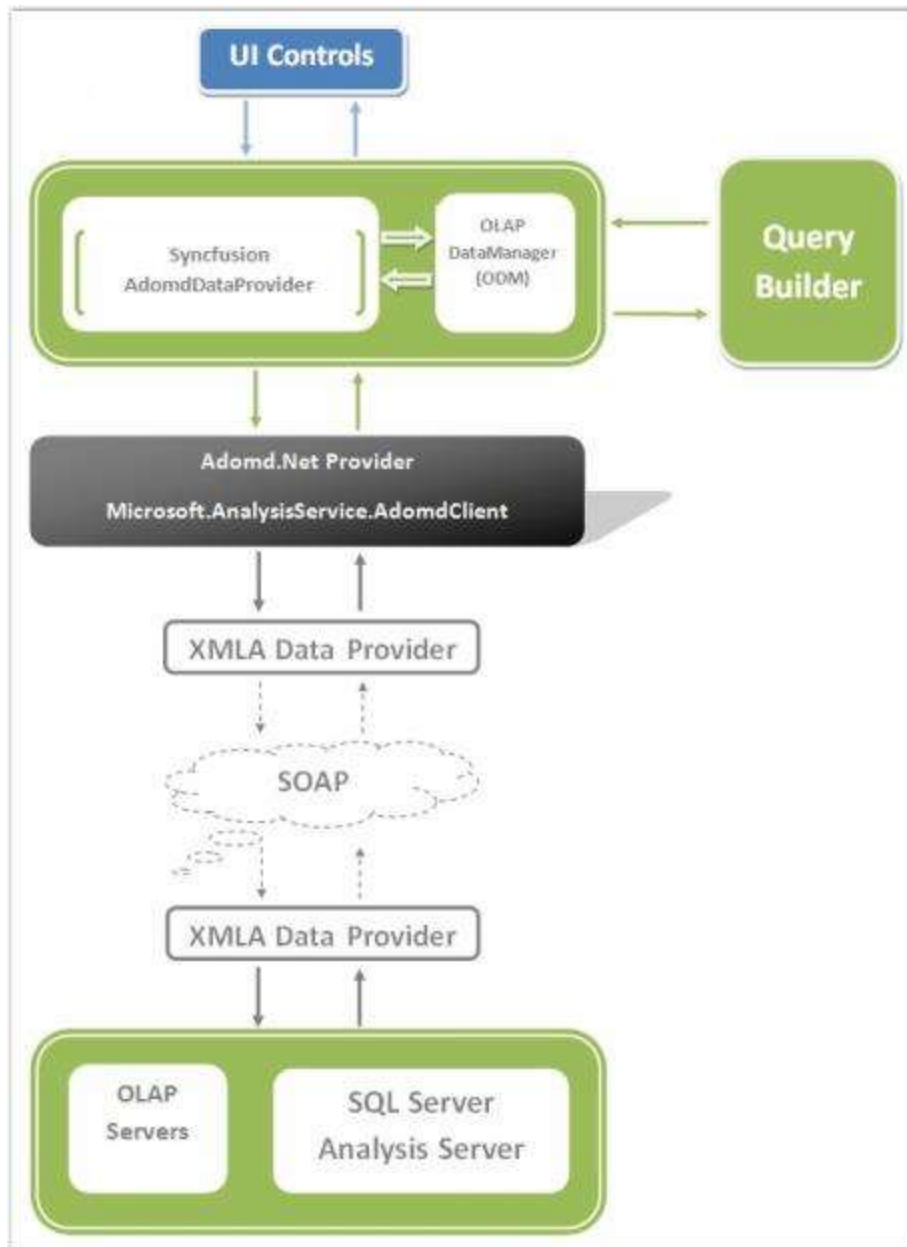
SQLSERVER2008ASADOMD10.msi and SQLSERVER2008ASOLEDB10.msi

Syncfusion OLAP Architecture in WPF OLAP Common

Syncfusion OLAP architecture allows you to build a full life cycle reporting solution for your enterprise. Here are the important pieces of the architecture:

- OLAP access layer: Built on the top of ADOMD.NET and provides a high level object model to define the reports easily.
- OLAP controls: Chart, grid, gauge, client for ASP.NET (excluding Gauge), WPF, silverlight, and ASP.NET MVC (grid only).
- OLAP report builder: Rapid Application Development (RAD) tool allows you to select the dimensions you are interested in visualizing and define the appearance for the chart and grid.

The following screenshot shows how the Syncfusion OLAP components allow you to build a full life cycle reporting solution for your enterprise.

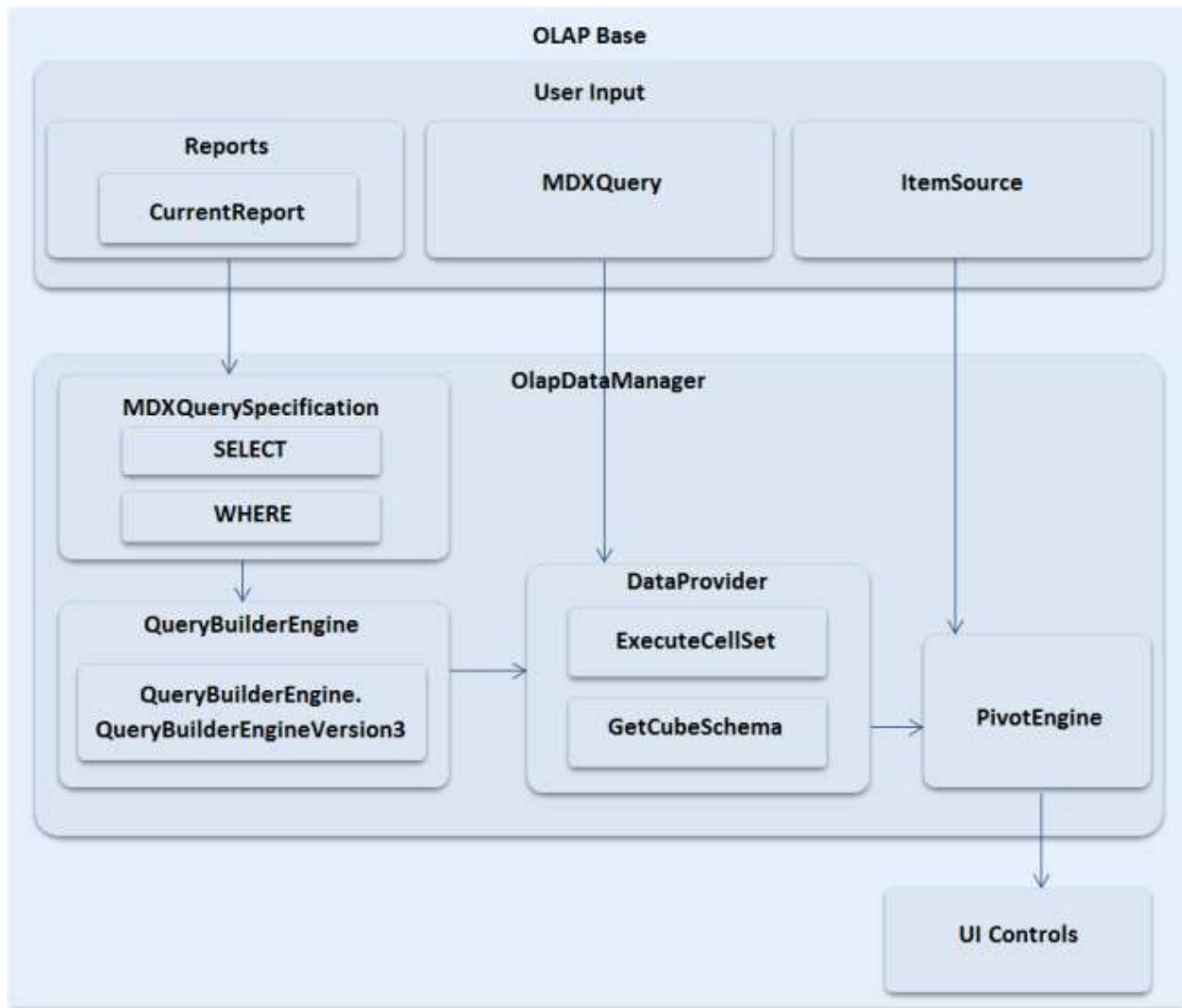


Syncfusion OLAP Architecture

OLAP base

The OLAP base is a class library that contains several namespaces and classes to perform data processing operations required by OLAP controls. OLAP base is the processing unit of all Syncfusion OLAP controls. For establishing connection and retrieving data, format the input and provide it to each control. This is controlled by the OLAP base.

Syncfusion OLAP controls communicate to the OLAP cube through the `OlapDataManager` class available in the `Syncfusion.Olap.Base` namespace.



OLAP Base Architecture

Note: This class library was organized under the Syncfusion.Olap.Base assembly.

OLAP Silverlight base

OLAP Silverlight base is a class library, which contains several namespaces and classes to perform data processing operation required by OLAP Silverlight controls. The *OlapDataManager* retrieves OLAP data and binds the result to an OLAP control.

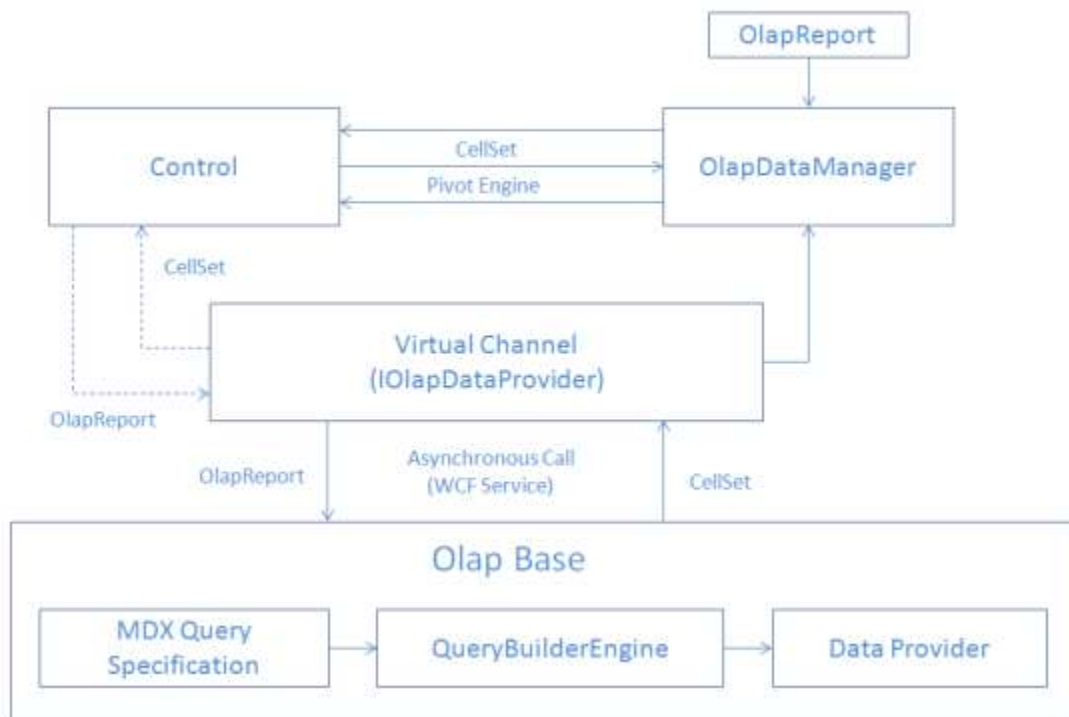
Note: This class library was organized under the Syncfusion.OlapSilverlight.Base assembly.

OLAP Silverlight base wrapper

The OLAP Silverlight base wrapper is a class library, which contains several namespaces and classes. This library helps to perform data conversion between OLAP Silverlight base and OLAP base. Data conversion process is used to achieve the following features:

1. Establishing the connection and retrieving data by converting the OLAP Silverlight base information to OLAP base information.

2. Send retrieved data to OLAP Silverlight base by converting the OLAP base data to OLAP Silverlight base data.



Data flow in Silverlight

Note: This class library was organized under the Syncfusion.OlapSilverlight.BaseWrapper assembly.

WCF service

To query the database, the OLAP Silverlight controls use the WCF service. The **DataProvider** property of **OlapDataManager** performs the service call operations.

OlapDataProvider in WPF OLAP Common

The database connectivity related works are all taken care of by this part of the base. Here, the **Microsoft.AnalysisService.AdomdClient** data provider is used. Establishing the connection, checking the state of the connection, and closing the connection are basic operations provided by the general data provider, but some more information is required to provide the input for controls.

This part of the base will get the connection information and establish a connection with the specified data source and retrieve the information from the database and store it in its classes. This part of the base has the required logic to retrieve the information from the database and store it in the object of class in data namespace.

All the information about the connected cube will intersect and store in the object of classes in the data namespace, which are equivalent to the classes in the **AdomdClient**. This information is required to provide the input for OLAP controls.

Important class in **DataProvider** namespace:

- **AdomdDataProvider**

AdomdDataProvider

The important properties and methods in AdomdDataProvider class are tabulated below:

Properties

- **CatalogName:** Gets the connected database name.
- **ConnectionString:** Sets or gets the connection string.
- **CurrentCellSet:** Gets the currently executed CellSet.
- **GetCubes:** Gets the information about the cubes in the connected data source.

Methods

Method Name	Description	Parameters	Return Type	Reference Link
ExecuteCellSet	Four arguments should be given to invoke this method. The arguments are MDX query as string, drill-down state of result set as bool, query append info as bool, and finally get the OlapReport. This method will generate the CellSet for the given query or OlapReport.	MDX Query as string, drill-down state as bool, property append status as bool and current OlapReport of OlapDataManager.	CellSet	-
GetCubeSchema	This method will get the cube name and intersect the cube to get all the information about the cube and return an object of type CubeSchema, which will contain all the information. CubeSchema is a class in data namespace.	Cube Name as string	CubeSchema	-
GetChildMembers	This method will get the member element and the expander state and return the child members of the given member as MemberCollection.	Parent member as Member and drill down state as bool	MemberCollection	-

GetLevelMembers	This method will get the level element as argument and return the member elements of that level as MemberCollection.	Level	MemberCollection	-
GetParentMember	This method is used to find the parent member of a member element. By passing a member element as an argument, this element will return its parent member.	Member	Member	-
ValidateConnectionString	This method will validate the specified connection string in the data manager or in the data provider and return the validated status.	-	bool	-

OlapDataManager in WPF OLAP Common

OlapDataManager is the most important class in the whole OLAP base. All the information are transferred from the control to OLAP base through this class and this will retain the current state of the base objects. The connection is established in the data provider of the OLAP base, but the information required in establishing the connection is given to the data provider through the OlapDataManager.

Constructors

Constructors	Description	Parameters	Return Type	Reference Link
OlapDataManager()	Default constructor	-	Void	-
OlapDataManager(string)	Accepts the connection string as an argument and passes it to the data provider to establish the connection with data source.	String	Void	-
OlapDataManager(AdomdDataProvider)	Accepts the data provider as an argument and	AdomdDataProvider	Void	-

	processes the cube that is connected with the given data provider.			
--	--	--	--	--

Establishing connection with the SSAS server

The following code snippet describes establishing connection with the server.

C#

```
OlapDataManager olapDataManager = new OlapDataManager("DataSource=localhost;
Initial Catalog=Adventure Works DW");
```

VB.NET

```
OlapDataManager olapDataManager = New OlapDataManager("DataSource=localhost;
Initial Catalog=Adventure Works DW")
```

Establishing connection with the SSAS server through data provider

The following code snippet describes establishing connection with the server.

C#

```
AdomdDataProvider dataProvider = new
AdomdDataProvider("DataSource=localhost;
Initial Catalog=Adventure Works DW");
OlapDataManager olapDataManager = new OlapDataManager(dataProvider);
```

VB.NET

```
Dim dataProvider As AdomdDataProvider = New
AdomdDataProvider("DataSource=localhost;
Initial Catalog=Adventure Works DW")
Dim olapDataManager As OlapDataManager = New OlapDataManager(dataProvider)
```

Establishing connection with the offline cube

The following code snippet describes establishing connection with the offline cube.

C#

```
OlapDataManager olapDataManager = new OlapDataManager(@"Data Source = C:\
Common\Data\OfflineCube\Adventure Works Ext.cub; Provider = MSOLAP;");
```

VB.NET

```
OlapDataManager olapDataManager = New OlapDataManager("Data Source = C:\\
Common\\Data\\OfflineCube\\Adventure Works Ext.cub; Provider = MSOLAP;")
```

Establishing connection with XMLA Server

XML for Analysis (XMLA) is a standard that allows the client applications to transfer multi-dimensional or OLAP data sources, which is available in the Mondrian Server. The back and forth communication is

done using the web standards – HTTP, SOAP, and XML. The query language used is MDX, which is most widely supported for reporting from multi-dimensional data stores.

Connecting to Mondrian Server

The following code illustrates how to connect to the Mondrian Server.

C#

```
// Connecting to Mondrian Server
OlapDataManager DataManager = new OlapDataManager("Data Source =
http://localhost:8080/mondrian/xmla; Initial Catalog = FoodMart;");
DataManager.DataProvider.ProviderName = Syncfusion.Olap.DataProvider.Provide
rs.Mondrian;
```

VB.NET

```
' Connecting to Mondrian Server
Dim DataManager As OlapDataManager = New OlapDataManager("Datasource =
http://bi.syncfusion.com:8080/mondrian/xmla; Initial Catalog=FoodMart;")
DataManager.DataProvider.ProviderName = Syncfusion.Olap.DataProvider.Provide
rs.Mondrian
```

[Click here](#) for more information about Mondrian XMLA configurations.

Connecting to Active Pivot Server

The following code illustrates how to connect to the Active Pivot Server.

C#

```
// Connecting to Active Pivot Server
OlapDataManager DataManager = new
OlapDataManager(@"Data Source=http://localhost:8081/var_s/xmla; Initial Cat
alog=VaRCubes; User ID=; Password=; Transport Compression=None;");
DataManager.DataProvider.ProviderName = Syncfusion.Olap.DataProvider.Provide
rs.ActivePivot;
```

VB.NET

```
' Connecting to Active Pivot Server
Dim DataManager As OlapDataManager = New OlapDataManager("Data
Source=http://localhost:8081/var_s/xmla; Initial Catalog=VaRCubes; User
ID=; Password=; Transport Compression=None;")
DataManager.DataProvider.ProviderName = Syncfusion.Olap.DataProvider.Provide
rs.ActivePivot
```

[Click here](#) for more information on Active Pivot server.

Connecting to SAP Business Warehouse Server

The following code illustrates how to connect to the SAP Business Warehouse server.

C#

```
// Connecting to SAP BW Server
```

```
OlapDataManager DataManager = new OlapDataManager(@"Data Source=
http://sapdomain:50000/sap/bw/xml/soap/xmla; Initial Catalog=$INFOCUBE; User
ID=username; Password=password;");
DataManager.DataProvider.ProviderName = Syncfusion.Olap.DataProvider.Provide
rs.SAPBW;
```

VB.NET

```
' Connecting to SAP BW Server
Dim DataManager As OlapDataManager = New OlapDataManager("Data Source=
http://sapdomain:50000/sap/bw/xml/soap/xmla; Initial Catalog=$INFOCUBE; User
ID=username; Password=password;")
DataManager.DataProvider.ProviderName = Syncfusion.Olap.DataProvider.Provide
rs.SAPBW
```

[Click here](#) for more information on SAP BW server.

Properties

- **ConnectionString:** Passes the connection string to establish the connection. Users can also get the connection string using this property.
- **CurrentCellSet:** Gets the CellSet of user's input from here.
- **CurrentCubeName:** Sets or gets the current cube name. When set, the whole data procession will change to the specified cube name.
- **CurrentCubeSchema:** Gets the CubeSchema of the currently connected cube.
- **CurrentReport:** Loads an OlapReport or get the currently loaded report.
- **DataProvider:** Sets a data provider and gets the existing data provider.
- **IsCurrentReportModified:** Gets or sets the modified status of the currently loaded report.
- **MdxQuery:** Passes the MDX query as input.
- **PivotEngine:** Gets the pivot engine for the given input.
- **QuerySpecification:** Gets the MDXQuerySpecification for the given OlapReport.
- **ReportPath:** Gets or sets the report path to store the report as an XML file.
- **Reports:** Contains the report collection of the OlapDataManager.

Methods

Method Name	Description	Parameters	Return Type	Reference Link
AddReport	Used to add a new OlapReport to the report collection of OlapDataManager.	OlapReport	Void	-
CloneOlapDataManager	Create a clone from the current state of OlapDataManager and return the new copy of OlapDataManager.	-	OlapDataManager	-

	The IDataProvider will be same for both the managers.			
ExecuteCellSet	This is an overloaded method that can be invoked by calling or passing the MDXQuerySpecification or by passing the MDX query as a string. This method invokes the data processing and returns the processed CellSet, which will be further formatted.	-	CellSet	ExecuteCellSet
ExecuteOlapTable	Used to create pivot engine from the CellSet.	-	PivotEngine	
GetMDXQuery	This method generates an MDX query from the MDXQuerySpecification and returns the query as a string.	-	string	-
GetReport	This method returns the OlapReport collection by giving the XML report file name as an input.	string	OlapReportCollection	-
GetReportAsStream	This method returns the current report collection along with current state of OlapDataManager as a stream.	-	Stream	GetReportAsStream

LoadOlapDataManager	Used to accept the OlapReport as an argument and loads the OlapDataManager with the given OlapReport.	OlapReport	Void	LoadOlapDataManager
LoadReport	Used to get the report name as an argument and pick the specified report from the report collection to load the OlapDataManager with that report.	string	Void	-
LoadReportDefinitionFile	Used to get the XML report file as an input and load the OlapDataManager with the report in that file.	string	Void	LoadReportDefinitionFile
LoadReportDefinitionFromStream	Used to get a stream as an input and read the report from that stream and load the OlapDataManager with that report.	Stream	Void	LoadReportDefinitionFromStream
NotifyElementModified	Used to trigger the ElementModified event.	-	Void	-
NotifyReportChanged	Used to trigger the ReportChanged event.	-	Void	-
RemoveReport	Used to get the report name as an input and remove that report from the report collection.	string	Void	RemoveReport

RenameReport	Used to rename the current report of OlapDataManager.	Index as int and new Report Name as string	void	RenameReport
SaveReport	Used to get the file name and store the current report collection along the current state of the OlapDataManager as an XML file.	string	Void	SaveReport
SetCurrentReport	Used to set an OlapReport as the current report of the OlapDataManager. This method is the initial method that will start data processing.	OlapReport	Void	SetCurrentReport

UseWhereClauseForSlicing

The UseWhereClauseForSlicing property facilitates the user to decide whether the MDX query parser engine should consider 'Where' or 'Select' clause for slicing data.

Properties

- **UseWhereClauseForSlicing:** Enables users to decide whether the MDX Query Parser Engine should consider the 'Where' or 'Select' clause for slicing operation.

Drill through

This feature enables the user to drill through any value and see the data which formed the value.

The following code snippet illustrates how to drill through using the MDX Query in OlapDataManager.

C#

```
string query = @"drillthrough Select { [Customer].[Customer Geography].[Country].&[Australia] } on 0, { [Date].[Fiscal].[Fiscal Year].&[2002] } on 1 from [Adventure Works] Where [Internet Sales Amount]";
// executing the drill-through operation.
olapDataManager.Execute(query);
```

VB.NET

```
Dim query As String = "drillthrough Select { [Customer].[Customer  
Geography].[Country].&[Australia] } on 0, { [Date].[Fiscal].[Fiscal  
Year].&[2002] } on 1 from [Adventure Works] Where [Internet Sales Amount]"  
'Executing the drill-through operation.'  
olapDataManager.Execute(query)
```

OlapReport in WPF OLAP Common

The OLAP report is an object that contains information about the cube element that has to be included for processing along its axis position and filter and sorting constraints. The OLAP report has categorized the elements based on their characteristics as follows:

- Dimension element
- Hierarchy element
- Level element
- Member elements
- Measure element
- KPI element
- NamedSet element
- Sort element
- Calculated member
- Subset element
- Summary element

These elements are used to get the cube element information from users. You can create an instance of this element and add the required information about the element to pick that specific element from the database for processing.

These elements are come under the following four elements group, which are based on the axis position and filtering constraints.

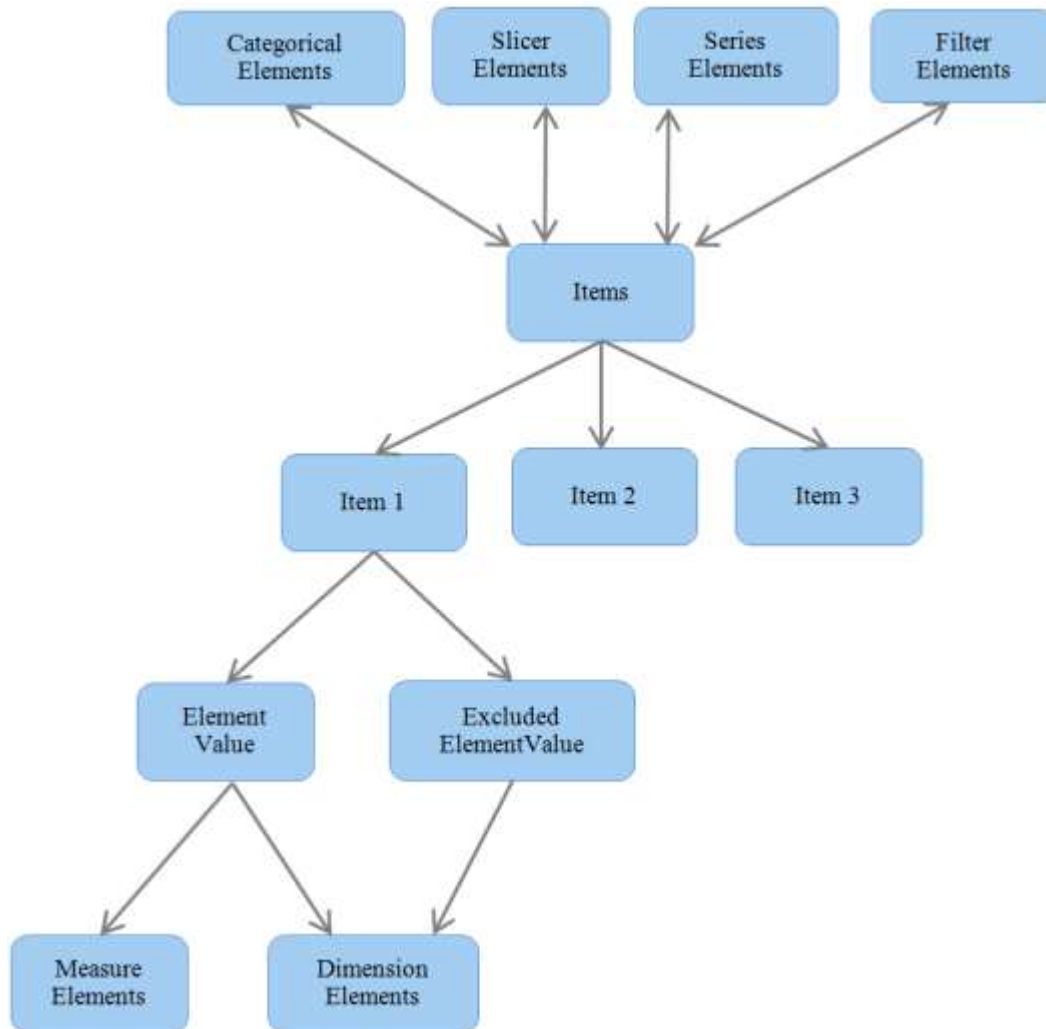
- Categorical elements: Contains elements that have to come under categorical axis.
- Series elements: Contains elements that have to come under series axis.
- Slicer elements: Contains elements that have to come under slicer elements.
- Filter elements: Contains elements that are subjected to the filter.

All the elements are internally maintained as items.

Properties

- **CalculatedMembers:** Sets and gets the calculated members of the OLAP report.
- **CategoricalElements:** Contains elements that are said to be in categorical axis. You can add an element and get the collection of elements that comes under the categorical axis.
- **CurrentCubeName:** Sets or gets the current cube name of the report.
- **FilterElements:** Contains elements that are subjected to filter constraints and a filter expression along the measure on which the filter expression is built.
- **Name:** Sets or gets the report name.
- **SeriesElements:** Contains elements that are said to be in series axis. You can add an element and get the collection of elements that comes under the categorical axis.
- **ShowEmptyColumnData:** Shows/hides the empty column in the result set.

- **ShowEmptyRowData:** Shows/hides the empty row in the result set.
- **ShowExpanders:** Shows/hides the expander buttons in the OLAP control.
- **SliceElements:** Contains the element that are said to be in slicer axis. You can add an element and get the collection of elements that comes under the categorical axis.
- **TogglePivot:** Swaps the elements in the column axis and row axis.
- **Tag:** Holds the backup information of the OLAP report.



Architecture of Items

Elements of OLAP report

Dimension element

A simple [dimension](#) object is composed of basic information such as name, hierarchy, level, and members. You can create a dimension element by specifying its name and providing the hierarchy and level name.

The dimension element contains the hierarchical details and information about each included level elements in that hierarchy. A hierarchy can have any number of level elements and the level elements can have any number of members and the member elements can have any number of child members.

Hierarchy element

Each element of a dimension can be summarized using a [hierarchy](#). The hierarchy is a series of parent-child relationship, where a parent member represents the consolidation of members which are its children. Parent members can be further aggregated as the children of another parent.

For example, May 2005 can be summarized into Second Quarter 2005 which in turn would be summarized in the year 2005.

Level element

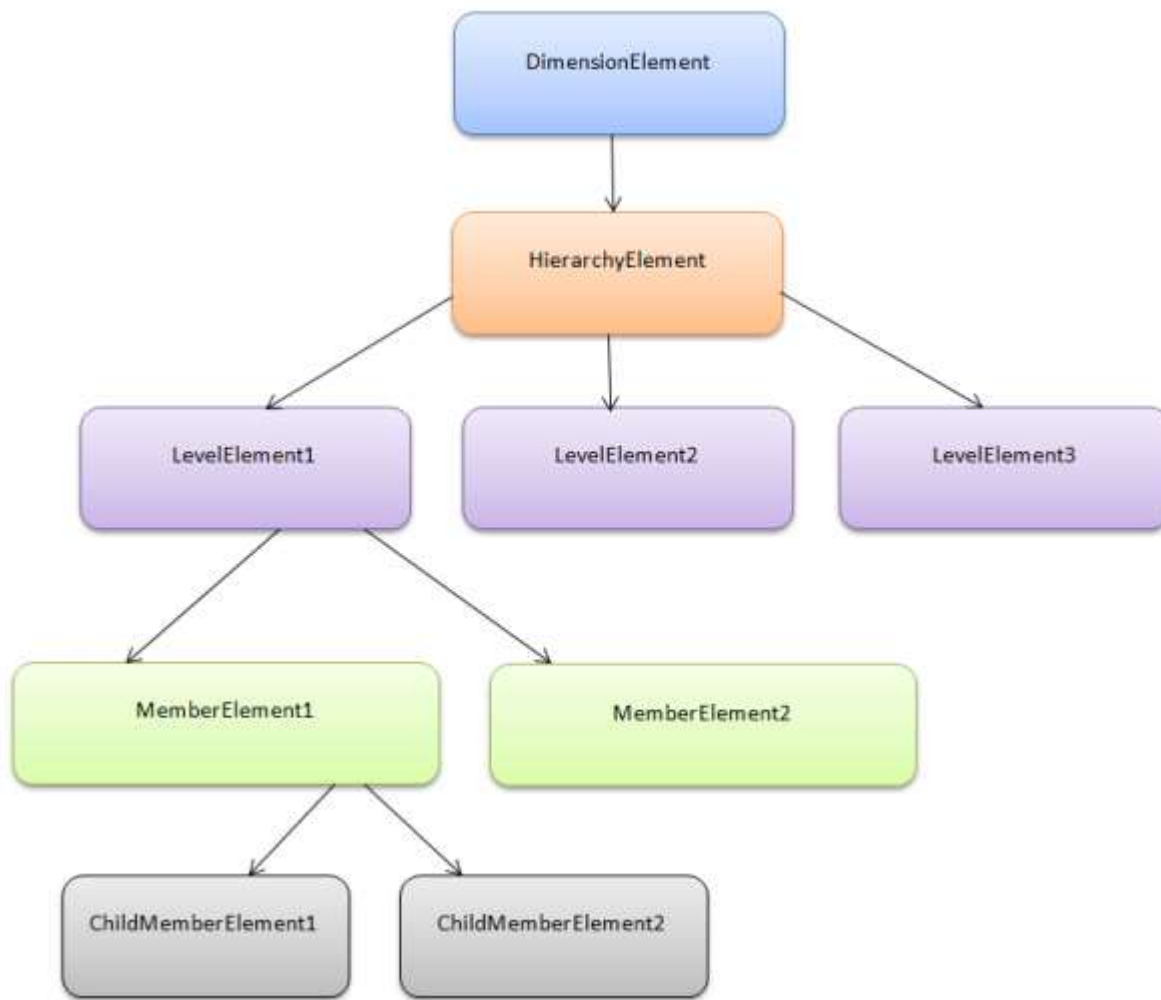
Level element is the child of hierarchy element which contains a set of members, each of which has the same rank within a hierarchy.

Member element

Member element represents a member of level in a cube, the children of a member of level.

Member properties

Member properties cover the basic information about each member in each tuple. This basic information includes the member name, parent level, the number of children, and so on. The member properties are available for all members at a given level.



The following code illustrates the creation of different types of dimensions.

Creating simple dimension element

C#

```
DimensionElement dimensionElementColumn = new DimensionElement();
// Specifying the Name for Column Dimension Element
dimensionElementColumn.Name = "Customer";
// Specifying the Hierarchy and Level Element Name
dimensionElementColumn.AddLevel("Customer Geography", "Country");
```

VB.NET

```
Dim dimensionElementRow As DimensionElement = New DimensionElement()
' Specifying the Name for Row Dimension Element
dimensionElementRow.Name = "Date"
' Specifying the Hierarchy and Level Element Name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
```

Creating sliced dimension

C#

```
DimensionElement dimensionElementSlicer = new DimensionElement();
// Specifying the dimension Name
dimensionElementSlicer.Name = "Product";
// Adding the Level Name along with the Hierarchy Name
dimensionElementSlicer.AddLevel("Product Categories", "Category");
// Adding the Member Element
dimensionElementSlicer.Hierarchy.LevelElements["Category"].Add("Bikes");
dimensionElementSlicer.Hierarchy.LevelElements["Category"].IncludeAvailableMembers = true;
```

VB.NET

```
Dim dimensionElementSlicer As DimensionElement = New DimensionElement()
' Specifying the dimension Name
dimensionElementSlicer.Name = "Product"
' Adding the Level Name along with the Hierarchy Name
dimensionElementSlicer.AddLevel("Product Categories", "Category")
' Adding the Member Element
dimensionElementSlicer.Hierarchy.LevelElements("Category").Add("Bikes")
dimensionElementSlicer.Hierarchy.LevelElements("Category").IncludeAvailableMembers = True
```

Measure element

In a cube, a measure is a set of values that are based on a column in the cube's [fact table](#) and are usually numeric. The measures are the central values of a cube that are analyzed. That is, measures are the numeric data of primary interest to users browsing a cube. You can select measures depend on the types of users request. Some common measures are sales, costs, expenditures, and production count.

You can create a measure element by specifying its name. The following code illustrates the creation of a measure element.

C#

```
MeasureElements measureElementColumn = new MeasureElements();
//Specifying the Measure Elements
measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet
Sales Amount" });
```

VB.NET

```
Dim measureElementColumn As MeasureElements = New MeasureElements()
' Specifying the Measure Elements
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
```

Key Performance Indicator (KPI) element

KPI is a collection of calculations that are associated with a measure group in a cube used to evaluate business success. Typically, these calculations are a combination of Multidimensional Expressions (MDX) or calculated members. The KPIs also have an additional metadata that provides information about how client applications should display the results of the KPI's calculations.

The different types of KPI indicators are:

- KPI goal
- KPI status
- KPI trend
- KPI value

You can create a KPI element by specifying its name and giving details of the indicator that are included in the element.

C#

```
KpiElements kpiElement = new KpiElements();
// Specifying the KPI Element name and configuring its Indicators
kpiElement.Elements.Add(new KpiElement { Name = "Internet Revenue",
ShowKPIGoal = true, ShowKPIStatus = true, ShowKPIValue = true,
ShowKPITrend = true });
```

VB.NET

```
Dim kpiElement As KpiElements = New KpiElements()
' Specifying the KPI Element name and configuring its Indicators
kpiElement.Elements.Add(New KpiElement With {.Name =
"Internet Revenue", .ShowKPIGoal = True, .ShowKPIStatus = True,
.ShowKPIValue = True, .ShowKPITrend = True})
```

NamedSet element

A named set is a collection of tuples and members, which can be defined and saved as a part of the cube definition. Named set records reside inside the sets folder, which is under a dimension element. These elements can be dragged to categories/series/slicer axis of the axes element builder. To work with a lengthy, complex, or commonly used expression easier, Multidimensional Expressions (MDX) allows you to define a named set.

The following code describes the creation of a named set element.

C#

```
NamedSetElement dimensionElementRow = new NamedSetElement();
// Specifying the dimension name
dimensionElementRow.Name = "Negative Margin Products";
```

VB.NET

```
Dim dimensionElementRow As NamedSetElement = New NamedSetElement()
' Specifying the dimension name
dimensionElementRow.Name = "Negative Margin Products"
```

Sort element

The result set can be sorted by using the SortElement. You can create sort elements and add it to the OLAP report. There are four types of sorting orders. They are:

- ASC: Sort the elements in ascending order.
- BASC: Sort the elements in ascending order by breaking the hierarchy.
- DESC: Sort the elements in descending order.
- BDESC: Sort the elements in descending order by breaking the hierarchy.

The following report illustrates the creation of sort element.

C#

```
SortElement sortElement = new SortElement(AxisPosition.Categorical,
SortOrder.BDESC, true);
sortElement.Element.UniqueName = "[Measures].[Internet Sales Amount]";
```

VB.NET

```
Dim sortElement As SortElement = New SortElement(AxisPosition.Categorical,
SortOrder.BDESC, True)
sortElement.Element.UniqueName = "[Measures].[Internet Sales Amount]"
```

Calculated member

Calculated members are the customized measures or dimension members created with the cube. The values are calculated at runtime. It is a user defined element. The two types of calculated members are as follows:

1. Calculated measure: Creates a calculated member from a measure element.
2. Calculated dimension: Creates a calculated member from a dimension.

Calculated member to be defined in OLAP report requires the following definitions

- Name: Name of the calculated member.
- Expression: Expression to form the calculated member.
- Measure/dimension element: You should add a measure or dimension element from which the calculated member should be created.

The following steps explain how to create and add a calculated member in an OLAP report.

1. Create a dimension measure or dimension element from which the calculated member has to be created.
2. Create a calculated member by giving the name and expression.
3. Add the created dimension element to the calculated member.
4. After defining the calculated member, add that member to the calculated member collection in the OLAP report.
5. Add the newly created calculated members in the categorical or series axis of the OLAP report.

The following code snippet describes the creation and addition of a calculated member in the OLAP report.

Calculated measure

C#

```
MeasureElement measureElement = new MeasureElement();
measureElement.Name = "Order Quantity";
CalculatedMember calculatedMeasure = new CalculatedMember();
calculatedMeasure.Name = "Order on Discount";
calculatedMeasure1.Expression = "[Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)";
calculatedMeasure1.AddElement(measureElement);
olapReport.CalculatedMembers.Add(calculatedMeasure);
```

VB.NET

```
Private measureElement As MeasureElement = New MeasureElement()
measureElement.Name = " Order Quantity "
Dim calculatedMeasure As CalculatedMember = New CalculatedMember()
calculatedMeasure.Name = " Order on Discount "
calculatedMeasure1.Expression = "[Measures].[Order Quantity] + ([Measures].[Order Quantity] *0.10) "
calculatedMeasure1.AddElement(measureElement)
olapReport.CalculatedMembers.Add(calculatedMeasure)
```

Calculated dimension

C#

```
DimensionElement internalDimension = new DimensionElement();
internalDimension.Name = "Product";
internalDimension.AddLevel("Product Categories", "Category");
// Calculated Dimension
CalculatedMember calculateDimension = new CalculatedMember();
calculateDimension.Name = "Bikes & Components";
calculateDimension.Expression = "([Product].[Product Categories].[Category].[Bikes] + [Product].[Product Categories].[Category].[Components] ) ";
calculateDimension.AddElement(internalDimension);
olapReport.CalculatedMembers.Add(calculateDimension)
```

VB.NET

```

Dim internalDimension As DimensionElement = New DimensionElement()
internalDimension.Name = "Product"
internalDimension.AddLevel("Product Categories", "Category")
' Calculated Dimension
Dim calculateDimension As CalculatedMember = New CalculatedMember()
calculateDimension.Expression = "Bikes & Components"
calculateDimension.Expression = "([Product].[Product Categories].
[Category].[Bikes] + [Product].[Product Categories].[Category].
[Components] )"
calculateDimension.AddElement(internalDimension)
olapReport.CalculatedMembers.Add(calculateDimension)

```

Subset element

Subset elements are used to filter the result set by their count. It filters the number of records and number of fields in the result set.

The following codes illustrate the creation of a subset element.

C#

```

SubsetElement subSetElementColumn = new SubsetElement(5);
subSetElementColumn.Name = "Top 5 Elements";
SubsetElement subSetElementRow = new SubsetElement(3);
subSetElementRow.Name = "Top 3 Elements";
olapReport.CategoricalElements.SubSetElement = subSetElementColumn;
olapReport.SeriesElements.SubSetElement = subSetElementRow;

```

VB.NET

```

Dim subSetElementColumn As SubsetElement = New SubsetElement(5)
subSetElementColumn.Name = "Top 5 Elements"
Dim subSetElementRow As SubsetElement = New SubsetElement(3)
subSetElementRow.Name = "Top 3 Elements"
olapReport.CategoricalElements.SubSetElement = subSetElementColumn
olapReport.SeriesElements.SubSetElement = subSetElementRow

```

Filtering slicer elements by range

This feature enables you to specify a range for filter elements in the slicer field. You have to specify the start and end values to set the range. Multiple ranges can be added to filter elements in the slicer field.

Class table

Name	Description
SlicerRangeFiltersInfo	Used to filter values from one range to another. Unique name of the member element for start and end values need to be specified. The name of the member element can also be specified for start and end values when customer builds the unique name*.

- Name of the member element can be specified only when name is formed with dimension name, hierarchy name, and level name.

Constructor

Syntax	Description	Parameter
<code>SlicerRangeFiltersInfo(string startValueUniqueName, string endValueUniqueName)</code>	Initializes <code>SlicerRangeFiltersInfo</code> with unique name as star and end values.	Unique name for start and end value.
<code>SlicerRangeFiltersInfo(string dimensionName, string hierarchyName, string levelName, string startValueName, string endValueName)</code>	Initializes <code>SlicerRangeFiltersInfo</code> with name of dimension, hierarchy, level, star value, and end value.	Name for dimension, hierarchy, level, start value, and end value.

Properties

Following table consists of `SlicerRangeFiltersInfo` class's property.

- **DimensionName:** Specifies the dimension name.
- **HierarchyName:** Specifies the hierarchy name.
- **LevelName:** Specifies the level name.
- **StartValue:** Specifies the unique name or name of the member element.
- **EndValue:** Specifies the unique name or name of the member element.
- Name of the member element can be specified only when the name is formed with dimension name, hierarchy name, and level name.

Adding a range for filter elements in the slicer field

There are two methods to add a range for filter elements in the slicer field.

In the first method, you can specify the unique name for start and end values. The following code illustrates this:

C#

```
olapReport.SlicerRangeFilters.Add(new SlicerRangeFiltersInfo("[TimeFlat].[201010100031]", "[TimeFlat].[201010100037]"));
```

VB.NET

```
olapReport.SlicerRangeFilters.Add(New SlicerRangeFiltersInfo("[TimeFlat].[201010100031]", "[TimeFlat].[201010100037]"))
```

In the second method, you can specify the member name along with dimension name, hierarchy name and level name. Entering the unique name for start and end value is not mandatory. The following code illustrates this:

C#

```
olapReport.SlicerRangeFilters.Add(new SlicerRangeFiltersInfo { DimensionName = "TimeFlat", HierarchyName = "TimeFlat", LevelName = "TimeId", StartValue = "201010100031", EndValue = "201010100037" });
```


VB.NET

```
olapReport.SlicerRangeFilters.Add(New SlicerRangeFiltersInfo With {.DimensionName = "TimeFlat", .HierarchyName = "TimeFlat", .LevelName = "TimeId", .StartValue = "201010100031", .EndValue = "201010100037"})
```

Number of failed connections							
	Austria	Brasil	England	Germany	Poland	United States	Total
#null	2	0	1	1			4
Linux		0	1		1		2
Mac Os	1	1	1	1			4
Solaris					0	1	1
Windows		1	0	1	0		2
Total	2	2	2	2	1	1	14

Number of failed connections							
	Austria	Brasil	England	Germany	Poland	United States	Total
#null	0		1				1
Linux		0	1				1
Mac Os			1				1
Solaris						1	1
Windows		0		0			0
Total	0	0	2	0		1	3

Creating the OLAP report

To create a report:

1. Instantiate a new object for the OLAP report.
2. Create the required elements like dimension element and measure element.
3. Add the created element in the desired axis (column or categorical, row or series, and filter or slicer) elements.
4. Then, bind the created report to the OlapDataManager using the SetCurrentReport() method or assign the report to OlapDataManager's current report property.

Sample reports for OLAP data

This section gives you the code snippet to generate different types of report for OlapDataManager.

*Simple report***C#**

```
OlapReport olapReport = new OlapReport();
olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer";
dimensionElementColumn.HierarchyName = "Customer Geography";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
MeasureElements measureElementColumn = new MeasureElements();
measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet Sales Amount" });
DimensionElement dimensionElementRow = new DimensionElement();
//Specifying the Dimension Name
dimensionElementRow.Name = "Date";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
/// Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn);
```

```

///Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn);
///Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow);

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
dimensionElementColumn.HierarchyName = "Customer Geography"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the Dimension Name
dimensionElementRow.Name = "Date"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
''' Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn)
'''Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn)
'''Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow)

```

*Report with slicing operation***C#**

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the dimension Name
dimensionElementColumn.Name = "Customer";
//Adding the Level Name along with the Hierarchy Name
dimensionElementColumn.AddLevel("Customer Geography", "Country");
DimensionElement dimensionElementRow = new DimensionElement();
//Specifying the dimension Name
dimensionElementRow.Name = "Date";
//Adding the Level Name along with the Hierarchy Name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
DimensionElement dimensionElementSlicer = new DimensionElement();
dimensionElementSlicer.Name = "Sales Channel";
dimensionElementSlicer.AddLevel("Sales Channel", "Sales Channel");
MeasureElements measureElementRow = new MeasureElements();
measureElementRow.Elements.Add(new MeasureElement { Name = "Internet Sales A
mount" });
olapReport.CategoricalElements.Add(dimensionElementColumn);
olapReport.SeriesElements.Add(dimensionElementRow);
olapReport.SlicerElements.Add(dimensionElementSlicer);
olapReport.SeriesElements.Add(measureElementRow);

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the dimension Name
dimensionElementColumn.Name = "Customer"
'Adding the Level Name along with the Hierarchy Name
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the dimension Name
dimensionElementRow.Name = "Date"
'Adding the Level Name along with the Hierarchy Name
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
Dim dimensionElementSlicer As DimensionElement = New DimensionElement()
dimensionElementSlicer.Name = "Sales Channel"
dimensionElementSlicer.AddLevel("Sales Channel", "Sales Channel")
Dim measureElementRow As MeasureElements = New MeasureElements()
measureElementRow.Elements.Add(New MeasureElement { Name = "Internet Sales Amount" });
measureElementRow.Elements.Add(New MeasureElement With {.Name = "Internet Sales Amount"})
olapReport.CategoricalElements.Add(dimensionElementColumn)
olapReport.SeriesElements.Add(dimensionElementRow)
olapReport.SlicerElements.Add(dimensionElementSlicer)
olapReport.SeriesElements.Add(measureElementRow)

```

*Report with dicing operation***C#**

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer";
//Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
MeasureElements measureElementColumn = new MeasureElements();
measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet Sales Amount" });
DimensionElement dimensionElementRow = new DimensionElement();
//Specifying the Dimension Name
dimensionElementRow.Name = "Date";
dimensionElementRow.HierarchyName = "Fiscal";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
//Specifying Excluded column elements
DimensionElement excludedColumnElement = new DimensionElement();
excludedColumnElement.Name = "Customer";
excludedColumnElement.HierarchyName = "Customer Geography";
excludedColumnElement.AddLevel("Customer Geography", "Country");
excludedColumnElement.Hierarchy.LevelElements["Country"].Add("Canada");

```

```

excludedColumnElement.Hierarchy.LevelElements["Country"].Add("France");
excludedColumnElement.Hierarchy.LevelElements["Country"].Add("United Kingdom");
excludedColumnElement.Hierarchy.LevelElements["Country"].Add("United States");
//Specifying the Excluded row elements
DimensionElement excludedRowElement = new DimensionElement();
excludedRowElement.Name = "Date";
excludedRowElement.AddLevel("Fiscal", "Fiscal Year");
excludedRowElement.AddLevel("Fiscal", "Month");
excludedRowElement.AddLevel("Fiscal", "Fiscal Quarter");
excludedRowElement.AddLevel("Fiscal", "Fiscal Semester");
excludedRowElement.Hierarchy.LevelElements["Fiscal Year"].Add("FY 2002");
excludedRowElement.Hierarchy.LevelElements["Fiscal Year"].Add("FY 2004");
excludedRowElement.Hierarchy.LevelElements["Fiscal Year"].Add("FY 2005");
excludedRowElement.Hierarchy.LevelElements["Fiscal Semester"].Add("H2 FY 2003");
excludedRowElement.Hierarchy.LevelElements["Month"].Add("August 2002");
excludedRowElement.Hierarchy.LevelElements["Month"].Add("September 2002");
excludedRowElement.Hierarchy.LevelElements["Fiscal Quarter"].Add("Q2 FY 2003");
excludedRowElement.Hierarchy.LevelElements["Fiscal Quarter"].Add("Q2 FY 2003");
///Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn,excludedColumnElement);
///Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn);
///Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow,excludedRowElement);

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
'Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
measureElementColumn.Elements.Add(New MeasureElement { Name = "Internet Sales Amount" });
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the Dimension Name
dimensionElementRow.Name = "Date"
dimensionElementRow.HierarchyName = "Fiscal"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
'Specifying Excluded column elements
Dim excludedColumnElement As DimensionElement = New DimensionElement()
excludedColumnElement.Name = "Customer"
excludedColumnElement.HierarchyName = "Customer Geography"

```

```

excludedColumnElement.AddLevel("Customer Geography", "Country")
excludedColumnElement.Hierarchy.LevelElements("Country").Add("Canada")
excludedColumnElement.Hierarchy.LevelElements("Country").Add("France")
excludedColumnElement.Hierarchy.LevelElements("Country").Add("United
Kingdom")
excludedColumnElement.Hierarchy.LevelElements("Country").Add("United
States")
'Specifying the Excluded row elements
Dim excludedRowElement As DimensionElement = New DimensionElement()
excludedRowElement.Name = "Date"
excludedRowElement.AddLevel("Fiscal", "Fiscal Year")
excludedRowElement.AddLevel("Fiscal", "Month")
excludedRowElement.AddLevel("Fiscal", "Fiscal Quarter")
excludedRowElement.AddLevel("Fiscal", "Fiscal Semester")
excludedRowElement.Hierarchy.LevelElements("Fiscal Year").Add("FY 2002")
excludedRowElement.Hierarchy.LevelElements("Fiscal Year").Add("FY 2004")
excludedRowElement.Hierarchy.LevelElements("Fiscal Year").Add("FY 2005")
excludedRowElement.Hierarchy.LevelElements("Fiscal Semester").Add("H2 FY
2003")
excludedRowElement.Hierarchy.LevelElements("Month").Add("August 2002")
excludedRowElement.Hierarchy.LevelElements("Month").Add("September 2002")
excludedRowElement.Hierarchy.LevelElements("Fiscal Quarter").Add("Q2 FY
2003")
excludedRowElement.Hierarchy.LevelElements("Fiscal Quarter").Add("Q2 FY
2003")
'''Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn,excludedColumnElem
ent)
'''Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn)
'''Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow,excludedRowElement)

```

Ordered report

C#

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer";
//Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
//Creating Measure Elements
MeasureElements measureElementColumn = new MeasureElements();
measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet Sale
s Amount" });
DimensionElement dimensionElementRow = new DimensionElement();
//Specifying the Dimension Name
dimensionElementRow.Name = "Date";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
SortElement sortElement = new SortElement(AxisPosition.Categorical, SortOrde
r.BDESC, true);
sortElement.Element.UniqueName = "[Measures].[Internet Sales Amount]";

```

```

///Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn);
///Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn);
//Adding Sort Element
olapReport.CategoricalElements.Add(sortElement);
///Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow);

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
'Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
'Creating Measure Elements
Dim measureElementColumn As MeasureElements = New MeasureElements()
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the Dimension Name
dimensionElementRow.Name = "Date"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
Dim sortElement As SortElement = New SortElement(AxisPosition.Categorical,
SortOrder.BDESC, True)
sortElement.Element.UniqueName = "[Measures].[Internet Sales Amount]"
'''Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn)
'''Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn)
'Adding Sort Element
olapReport.CategoricalElements.Add(sortElement)
'''Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow)

```

Report with filter

C#

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
//Creating Measure Element
MeasureElements measureElementColumn = new MeasureElements();
measureElementColumn.Elements.Add(new MeasureElement { Name =
    "Internet Sales Amount" });
DimensionElement dimensionElementRow = new DimensionElement();

```

```
//Specifying the Dimension Name
dimensionElementRow.Name = "Date";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
FilterElement filterElement = new FilterElement(AxisPosition.Categorical);
filterElement.Elements.Add(measureElementColumn);
filterElement.Elements.Add(dimensionElementColumn);
filterElement.FilterCase = FilterCase.GreaterThan;
filterElement.FilterValue.Add(new MeasureElement { Name =
"Internet Sales Amount", Visible = true });
filterElement.FilterValue.Add(new FilterValue { Filter_Value = 2700000.00 })
;
filterElement.IsFilterCondition = true;
/// Adding Column Members
olapReport.CategoricalElements.Add(new Item { ElementValue =
dimensionElementColumn });
olapReport.CategoricalElements.IsFilterOrSortOn = true;
olapReport.FilterElements.Add(new Item { ElementValue = filterElement });
olapReport.SeriesElements.Add(dimensionElementRow);
```

VB.NET

```
Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
'Creating Measure Element
Dim measureElementColumn As MeasureElements = New MeasureElements()
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the Dimension Name
dimensionElementRow.Name = "Date"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
Dim filterElement As FilterElement = New
FilterElement(AxisPosition.Categorical)
filterElement.Elements.Add(measureElementColumn)
filterElement.Elements.Add(dimensionElementColumn)
filterElement.FilterCase = FilterCase.GreaterThan
filterElement.FilterValue.Add(New MeasureElement With {.Name = "Internet
Sales Amount", .Visible = True})
filterElement.FilterValue.Add(New FilterValue With {.Filter_Value =
2700000.00})
filterElement.IsFilterCondition = True
olapReport.CategoricalElements.Add(New Item With {.ElementValue =
dimensionElementColumn})
olapReport.CategoricalElements.IsFilterOrSortOn = True
olapReport.FilterElements.Add(New Item With {.ElementValue = filterElement})
```

Report with subset

C#

```
OlapReport olapReport = new OlapReport();
```

```

olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer";
//Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
MeasureElements measureElementColumn = new MeasureElements();
measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet Sales Amount" });
DimensionElement dimensionElementRow = new DimensionElement();
//Specifying the Dimension Name
dimensionElementRow.Name = "Date";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
//Specifying the SubsetElement
//Specify the start index and end index to retrieve the records.
SubsetElement subSetElementColumn = new SubsetElement(5);
subSetElementColumn.Name = "Top 5 Elements";
SubsetElement subSetElementRow = new SubsetElement(3);
subSetElementRow.Name = "Top 3 Elements";
///Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn);
///Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn);
olapReport.CategoricalElements.SubSetElement = subSetElementColumn;
///Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow);
olapReport.SeriesElements.SubSetElement = subSetElementRow;

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
'Creating Measure Element
Dim olapReport As OlapReport = New OlapReport()
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
'Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the Dimension Name
dimensionElementRow.Name = "Date"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
'Specifying the SubsetElement

```



```

'Specify the start index and end index to retrieve the records.'
Dim subSetElementColumn As SubsetElement = New SubsetElement(5)
subSetElementColumn.Name = "Top 5 Elements"
Dim subSetElementRow As SubsetElement = New SubsetElement(3)
subSetElementRow.Name = "Top 3 Elements"
'''Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn)
'''Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn)
olapReport.CategoricalElements.SubSetElement = subSetElementColumn
'''Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow)
olapReport.SeriesElements.SubSetElement = subSetElementRow

```

Drill down report

C#

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer";
//Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
MeasureElements measureElementColumn = new MeasureElements();
measureElementColumn.Elements.Add(new MeasureElement { Name =
"Internet Sales Amount" });
DimensionElement dimensionElementRow = new DimensionElement();
//Specifying the Dimension Name
dimensionElementRow.Name = "Date";
dimensionElementRow.HierarchyName = "Fiscal";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].Add("FY 2002");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ShowChildMembers = true;
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].Add("H1 FY 2002");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].ShowChildMembers = true;
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].Add("Q1 FY 2002");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].
ChildMemberElements[0].ShowChildMembers = true;
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].
ChildMemberElements[0].Add("July 2001");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].
ChildMemberElements[0].ChildMemberElements[0].ShowChildMembers = true;

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
'Creating Measure Element
Dim olapReport As OlapReport = New OlapReport()
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
'Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography"
Dim olapReport As OlapReport = New OlapReport()
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
'Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the Dimension Name
dimensionElementRow.Name = "Date"
dimensionElementRow.HierarchyName = "Fiscal"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").Add("FY 2002")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ShowChildMembers = True
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).Add("H1 FY 2002")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).ShowChildMembers = True
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).Add("Q1 FY 2002")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).
ChildMemberElements(0).ShowChildMembers = True
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).ChildMemberElements(0).Add("July
2001")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).
ChildMemberElements(0).ChildMemberElements(0).ShowChildMembers = True

```

Drill down report

C#

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";

```

```

DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer";
//Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
MeasureElements measureElementColumn = new MeasureElements();
measureElementColumn.Elements.Add(new MeasureElement { Name =
"Internet Sales Amount" });
DimensionElement dimensionElementRow = new DimensionElement();
//Specifying the Dimension Name
dimensionElementRow.Name = "Date";
dimensionElementRow.HierarchyName = "Fiscal";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].Add("FY 2002");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ShowChildMembers = true;
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].Add("H1 FY 2002");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].ShowChildMembers = true;
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].Add("Q1 FY 2002");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].
ChildMemberElements[0].ShowChildMembers = true;
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].
ChildMemberElements[0].Add("July 2001");
dimensionElementRow.Hierarchy.LevelElements["Fiscal Year"].
MemberElements[0].ChildMemberElements[0].
ChildMemberElements[0].ChildMemberElements[0].ShowChildMembers = true;

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
'Creating Measure Element
Dim olapReport As OlapReport = New OlapReport()
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
'Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography"
Dim olapReport As OlapReport = New OlapReport()
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
'Specifying the Hierarchy Name

```

```

dimensionElementColumn.HierarchyName = "Customer Geography"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the Dimension Name
dimensionElementRow.Name = "Date"
dimensionElementRow.HierarchyName = "Fiscal"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").Add("FY 2002")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ShowChildMembers = True
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).Add("H1 FY 2002")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).ShowChildMembers = True
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).Add("Q1 FY 2002")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).
ChildMemberElements(0).ShowChildMembers = True
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).ChildMemberElements(0).Add("July
2001")
dimensionElementRow.Hierarchy.LevelElements("Fiscal Year").
MemberElements(0).ChildMemberElements(0).
ChildMemberElements(0).ChildMemberElements(0).ShowChildMembers = True

```

Report with top count filter

C#

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
//Creating Measure Element
MeasureElements measureElementColumn = new MeasureElements();
measureElementColumn.Elements.Add(new MeasureElement { Name =
"Internet Sales Amount" });
DimensionElement dimensionElementRow = new DimensionElement();
//Specifying the Dimension Name
dimensionElementRow.Name = "Date";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
//Filter the top 5 elements of "Internet Sales Amount".
TopCountElement topCountElement = new
TopCountElement(AxisPosition.Categorical, 5);
topCountElement.MeasureName = "Internet Sales Amount";
/// Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn);
///Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn);
///Adding Measure Element

```

```

olapReport.CategoricalElements.Add(topCountElement);
///Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow);

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
'Creating Measure Element
Dim measureElementColumn As MeasureElements = New MeasureElements()
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Internet
Sales Amount"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the Dimension Name
dimensionElementRow.Name = "Date"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
'Filter the top 5 elements of "Internet Sales Amount".
Dim topCountElement As TopCountElement = New
TopCountElement(AxisPosition.Categorical, 5)
topCountElement.MeasureName = "Internet Sales Amount"
''' Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn)
'''Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn)
'''Adding Measure Element
olapReport.CategoricalElements.Add(topCountElement)
'''Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow)

```

Report with named set

C#

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Customer Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
///Specifying the dimension Name
dimensionElementColumn.Name = "Customer";
///Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
MeasureElements measureElementColumn = new MeasureElements();
///Specifying the measure elements
measureElementColumn.Elements.Add(new MeasureElement { Name = "Internet Sale
s Amount" });
///Specifying the NamedSet Element
NamedSetElement dimensionElementRow = new NamedSetElement();
///Specifying the dimension name
dimensionElementRow.Name = "Core Product Group";
///Adding the Column members

```

```

olapReport.CategoricalElements.Add(dimensionElementColumn);
///Adding the Measure elements
olapReport.CategoricalElements.Add(measureElementColumn);
///Adding the Row members
olapReport.SeriesElements.Add(dimensionElementRow);

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Customer Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
'Specifying the dimension Name
dimensionElementColumn.Name = "Customer"
'Specifying the Hierarchy Name
dimensionElementColumn.HierarchyName = "Customer Geography"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim measureElementColumn As MeasureElements = New MeasureElements()
'Specifying the measure elements
measureElementColumn.Elements.Add(New MeasureElement With {.Name =
"Internet Sales Amount"})
'Specifying the NamedSet Element
Dim dimensionElementRow As NamedSetElement = New NamedSetElement()
'Specifying the dimension name
dimensionElementRow.Name = "Core Product Group"
'''Adding the Column members
olapReport.CategoricalElements.Add(dimensionElementColumn)
'''Adding the Measure elements
olapReport.CategoricalElements.Add(measureElementColumn)
'''Adding the Row members
olapReport.SeriesElements.Add(dimensionElementRow)

```

Report with calculated member

C#

```

olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
//Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer";
dimensionElementColumn.HierarchyName = "Customer Geography";
dimensionElementColumn.AddLevel("Customer Geography", "Country");
DimensionElement internalDimension = new DimensionElement();
internalDimension.Name = "Product";
internalDimension.AddLevel("Product Categories", "Category");
//// Calculated Measure
CalculatedMember calculatedMeasure1 = new CalculatedMember();
calculatedMeasure1.Name = "Oder on Discount";
calculatedMeasure1.Expression = "[Measures].[Order Quantity] + ([Measures].[
Order Quantity] * 0.10)";
calculatedMeasure1.AddElement(new MeasureElement { Name = "Order Quantity" }
);
//// Calculated Measure
CalculatedMember calculatedMeasure2 = new CalculatedMember();
calculatedMeasure2.Name = "Sales Range";
calculatedMeasure2.AddElement(new MeasureElement { Name = "Sales Amount" });

```

```

calculatedMeasure2.Expression = "IIF([Measures].[Sales Amount]>200000,\"High\" ,\"Low\")";
// Calculated Dimension
CalculatedMember calculateDimension = new CalculatedMember();
calculateDimension.Name = "Bikes & Components";
calculateDimension.Expression = "([Product].[Product Categories].[Category].[Bikes] + [Product].[Product Categories].[Category].[Components] )";
calculateDimension.AddElement(internalDimension);
MeasureElements measureElementColumn = new MeasureElements();
measureElementColumn.Elements.Add(new MeasureElement { Name = "Sales Amount" });
measureElementColumn.Elements.Add(new MeasureElement { Name = "Order Quantity" });
DimensionElement dimensionElementRow = new DimensionElement();
//Specifying the Dimension Name
dimensionElementRow.Name = "Date";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
//// Adding Calculated members in calculated member collection
olapReport.CalculatedMembers.Add(calculatedMeasure1);
olapReport.CalculatedMembers.Add(calculateDimension);
olapReport.CalculatedMembers.Add(calculatedMeasure2);
/// Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn);
olapReport.CategoricalElements.Add(calculateDimension);
///Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn);
olapReport.CategoricalElements.Add(calculatedMeasure1);
olapReport.CategoricalElements.Add(calculatedMeasure2);
///Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow);

```

VB.NET

```

Dim dimensionElementColumn As DimensionElement = New DimensionElement()
' Specifying the Name for the Dimension Element
dimensionElementColumn.Name = "Customer"
dimensionElementColumn.HierarchyName = "Customer Geography"
dimensionElementColumn.AddLevel("Customer Geography", "Country")
Dim internalDimension As DimensionElement = New DimensionElement()
internalDimension.Name = "Product"
internalDimension.AddLevel("Product Categories", "Category")
'// Calculated Measure
Dim calculatedMeasure1 As CalculatedMember = New CalculatedMember()
calculatedMeasure1.Name = "Order on Discount"
calculatedMeasure1.Expression = "[Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)"
calculatedMeasure1.AddElement(New MeasureElement With {.Name = "Order Quantity"})
'// Calculated Measure
Dim calculatedMeasure2 As CalculatedMember = New CalculatedMember()
calculatedMeasure2.Name = "Sales Range"
calculatedMeasure2.AddElement(New MeasureElement With {.Name = "Sales Amount"})
calculatedMeasure2.Expression = "IIF([Measures].[Sales Amount]>200000,\"High\",\"Low\")"
' Calculated Dimension

```

```

Dim calculateDimension As CalculatedMember = New CalculatedMember()
calculateDimension.Name = "Bikes & Components"
calculateDimension.Expression = "([Product].[Product
Categories].[Category].[Bikes] + [Product].[Product
Categories].[Category].[Components] )"
calculateDimension.AddElement(internalDimension)
Dim measureElementColumn As MeasureElements = New MeasureElements()
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Sales
Amount"})
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Order
Quantity"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
'Specifying the Dimension Name
dimensionElementRow.Name = "Date"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
'/// Adding Calculated members in calculated member collection
olapReport.CalculatedMembers.Add(calculatedMeasure1)
olapReport.CalculatedMembers.Add(calculateDimension)
olapReport.CalculatedMembers.Add(calculatedMeasure2)
'''' Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn)
olapReport.CategoricalElements.Add(calculateDimension)
''''Adding Measure Element
olapReport.CategoricalElements.Add(measureElementColumn)
olapReport.CategoricalElements.Add(calculatedMeasure1)
olapReport.CategoricalElements.Add(calculatedMeasure2)
''''Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow)

```

Report with KPI element

C#

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Products Sales Report";
olapReport.CurrentCubeName = "Adventure Works";
DimensionElement dimensionElementColumn = new DimensionElement();
// Specifying the name for Dimension Element for Column
dimensionElementColumn.Name = "Product";
dimensionElementColumn.AddLevel("Product Categories", "Category");
dimensionElementColumn.Hierarchy.LevelElements["Category"].Add(this.productName);
dimensionElementColumn.Hierarchy.LevelElements["Category"].IncludeAvailableMembers = true;
MeasureElements measureElementColumn = new MeasureElements();
// Specifying the name for Measure Element
measureElementColumn.Elements.Add(new MeasureElement { Name = "Gross Profit" });
DimensionElement dimensionElementRow = new DimensionElement();
// Specifying the Name for Row Dimension Element
dimensionElementRow.Name = "Date";
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year");
KpiElements kpiElement = new KpiElements();
kpiElement.Elements.Add(new KpiElement { Name = "Revenue", ShowKPIStatus = true, ShowKPIGoal = false, ShowKPITrend = true, ShowKPIValue = true });
// Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn);

```



```

olapReport.CategoricalElements.Add(kpiElement);
// Adding Measure Elements
olapReport.CategoricalElements.Add(measureElementColumn);
// Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow);

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Products Sales Report"
olapReport.CurrentCubeName = "Adventure Works"
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
' Specifying the name for Dimension Element for Column
dimensionElementColumn.Name = "Product"
dimensionElementColumn.AddLevel("Product Categories", "Category")
dimensionElementColumn.Hierarchy.LevelElements("Category").Add(Me.productName)
dimensionElementColumn.Hierarchy.LevelElements("Category").IncludeAvailableMembers = True
Dim measureElementColumn As MeasureElements = New MeasureElements()
' Specifying the name for Measure Element
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Gross Profit"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
' Specifying the Name for Row Dimension Element
dimensionElementRow.Name = "Date"
dimensionElementRow.AddLevel("Fiscal", "Fiscal Year")
Dim kpiElement As KpiElements = New KpiElements()
kpiElement.Elements.Add(New KpiElement With {.Name = "Revenue",
.ShowKPIStatus = True, .ShowKPIGoal = False, .ShowKPITrend = True,
.ShowKPIValue = True})
' Adding Column Members
olapReport.CategoricalElements.Add(dimensionElementColumn)
olapReport.CategoricalElements.Add(kpiElement)
' Adding Measure Elements
olapReport.CategoricalElements.Add(measureElementColumn)
' Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow)

```

Report with member properties

C#

```

OlapReport olapReport = new OlapReport();
olapReport.Name = "Employee Sales Report";
// Specifying the current cube name
olapReport.CurrentCubeName = "Adventure Works";
MeasureElements measureElementColumn = new MeasureElements();
// Specifying the Name for the Measure Element
measureElementColumn.Elements.Add(new MeasureElement { Name = "Sales Amount Quota" });
DimensionElement dimensionElementRow = new DimensionElement();
// Specifying the Dimension Name
dimensionElementRow.Name = "Employee";
// Specifying the Hierarchy and level name for the Dimension Element
dimensionElementRow.AddLevel("Employees", "Employee Level 02");

```

```

dimensionElementRow.Hierarchy.LevelElements["Employee Level 02"].
IncludeAvailableMembers = true;
// Adding the Member properties to the Dimension Element
dimensionElementRow.MemberProperties.Add(new MemberProperty("Title",
"[Employee].[Employees].[Title]"));
dimensionElementRow.MemberProperties.Add(new MemberProperty("Phone",
"[Employee].[Employees].[Phone]"));
dimensionElementRow.MemberProperties.Add(new MemberProperty("Email Address",
"[Employee].[Employees].[Email Address]"));
DimensionElement dimensionElementColumn = new DimensionElement();
// Specifying the Dimension Name
dimensionElementColumn.Name = "Product";
// Specifying the Hierarchy and level name for the Dimension Element
dimensionElementColumn.AddLevel("Product Categories", "Category");
dimensionElementColumn.MemberProperties.Add(new MemberProperty("Dealer Price",
"[Product].[Product Categories].[Dealer Price]"));
dimensionElementColumn.MemberProperties.Add(new MemberProperty("Standard Cost",
"[Product].[Product Categories].[Standard Cost]"));
// Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow);
///Adding Column Members
olapReport.CategoricalElements.Add(measureElementColumn);

```

VB.NET

```

Dim olapReport As OlapReport = New OlapReport()
olapReport.Name = "Employee Sales Report"
' Specifying the current cube name
olapReport.CurrentCubeName = "Adventure Works"
Dim measureElementColumn As MeasureElements = New MeasureElements()
' Specifying the Name for the Measure Element
measureElementColumn.Elements.Add(New MeasureElement With {.Name = "Sales
Amount Quota"})
Dim dimensionElementRow As DimensionElement = New DimensionElement()
' Specifying the Dimension Name
dimensionElementRow.Name = "Employee"
' Specifying the Hierarchy and level name for the Dimension Element
dimensionElementRow.AddLevel("Employees", "Employee Level 02")
dimensionElementRow.Hierarchy.LevelElements("Employee Level 02").
IncludeAvailableMembers = True
' Adding the Member properties to the Dimension Element
dimensionElementRow.MemberProperties.Add(New MemberProperty("Title",
"[Employee].[Employees].[Title]"))
dimensionElementRow.MemberProperties.Add(New MemberProperty("Phone",
"[Employee].[Employees].[Phone]"))
dimensionElementRow.MemberProperties.Add(New MemberProperty("Email Address",
"[Employee].[Employees].[Email Address]"))
Dim dimensionElementColumn As DimensionElement = New DimensionElement()
' Specifying the Dimension Name
dimensionElementColumn.Name = "Product"
' Specifying the Hierarchy and level name for the Dimension Element
dimensionElementColumn.AddLevel("Product Categories", "Category")
dimensionElementColumn.MemberProperties.Add(New MemberProperty("Dealer
Price", "[Product].[Product Categories].[Dealer Price]"))
dimensionElementColumn.MemberProperties.Add(New MemberProperty("Standard
Cost", "[Product].[Product Categories].[Standard Cost]"))

```

```
' Adding Row Members
olapReport.SeriesElements.Add(dimensionElementRow)
'''Adding Column Members
olapReport.CategoricalElements.Add(measureElementColumn)
```

Binding OlapReport to OlapDataManager in WPF OLAP Common

After creating the report, it is set to the current report property of the OLAP data manager. Further, data processing, query creation, and cell set creation are started from the OLAP data manager.

Binding the OLAP report to the OLAP data manager.

Virtual KPI Element in WPF OLAP Common

KPI can be virtually defined during runtime. This feature enables users to create KPIs without storing them in SSAS (SQL Server Analysis Services). This feature is useful when users want to define KPIs at runtime and also minimize the time necessary to create KPIs.

Properties

- **Name:** Gets or sets the name for VirtualKpiElement.
- **KpiValueExpression:** Gets or sets the value which indicates the value expression for VirtualKpiElement.
- **KpiGoalExpression:** Gets or sets the value which indicates the goal expression for VirtualKpiElement.
- **KpiStatusExpression:** Gets or sets the value which indicates the status expression for VirtualKpiElement.
- **KpiTrendExpression:** Gets or sets the value which indicates the trend expression for VirtualKpiElement.
- **KpiTrendGraphic:** Gets or sets the name of the graphic used to represent the result of the trend expression.
- **KpiStatusGraphic:** Gets or sets the name of the graphic used to represent the result of the status expression.

OlapReport class properties

- **VirtualKpiElements:** Gets or sets the collection of VirtualKpiElement.

Adding virtual KPI element to the OlapReport

OLAP report definition with VirtualKpiElement

C#

```
public OlapReport VirtualKPIReport()
{
    OlapReport olapReport = new OlapReport("Virtual KPI Report");
    olapReport.CurrentCubeName = "Adventure Works";
    MeasureElements measureElements = new MeasureElements();
    measureElements.Add(new MeasureElement { Name = "Internet Sales Amount" });
    olapReport.CategoricalElements.Add(measureElements);
    VirtualKpiElement virtualKPIElement = new VirtualKpiElement();
    virtualKPIElement.Name = "Sample KPI";
    virtualKPIElement.KpiValueExpression = ""; //Value expression
    virtualKPIElement.KpiGoalExpression = ""; //Goal expression
```

```

virtualKPIElement.KpiStatusExpression = ""; //Status expression
virtualKPIElement.KpiTrendExpression = ""; //Trend expression
virtualKPIElement.StatusGraphic = ""; //Status graphic
virtualKPIElement.TrendGraphic = ""; //Trend graphic
olapReport.VirtualKpiElements.Add(virtualKPIElement);
olapReport.CategoricalElements.Add(virtualKPIElement);
DimensionElement internalDimension = new DimensionElement();
internalDimension.Name = "Product";
internalDimension.AddLevel("Product Categories", "Category");
olapReport.SeriesElements.Add(internalDimension);
return olapReport;
}

```

VB.NET

```

Public Function VirtualKPIReport() As OlapReport
Dim olapReport As New OlapReport("Virtual KPI Report")
olapReport.CurrentCubeName = "Adventure Works"
Dim measureElements As New MeasureElements()
measureElements.Add(New MeasureElement With {.Name = "Internet Sales
Amount"})
olapReport.CategoricalElements.Add(measureElements)
Dim virtualKPIElement As New VirtualKpiElement()
virtualKPIElement.Name = "Sample KPI"
virtualKPIElement.KpiValueExpression = "" 'Value expression
virtualKPIElement.KpiGoalExpression = "" 'Goal expression
virtualKPIElement.KpiStatusExpression = "" 'Status expression
virtualKPIElement.KpiTrendExpression = "" 'Trend expression
virtualKPIElement.StatusGraphic = "" 'Status graphic
virtualKPIElement.TrendGraphic = "" 'Trend graphic
olapReport.VirtualKpiElements.Add(virtualKPIElement)
olapReport.CategoricalElements.Add(virtualKPIElement)
Dim internalDimension As New DimensionElement()
internalDimension.Name = "Product"
internalDimension.AddLevel("Product Categories", "Category")
olapReport.SeriesElements.Add(internalDimension)
Return olapReport
End Function
Sample of Value, Goal, Status, and Trend expressions
[Value Expression]
[Measures].[Reseller Sales Amount]
[Goal Expression]
[Measures].[Sales Amount Quota]
[Status Expression]
Case
When [Measures].[Reseller Sales Amount] / [Measures].[Sales Amount Quota] >
1
Then 1
When [Measures].[Reseller Sales Amount] / [Measures].[Sales Amount Quota] <=
1
And
[Measures].[Reseller Sales Amount] / [Measures].[Sales Amount Quota] >= .85
Then 0
Else -1
End
[Trend Expression]

```

```

Case
When IsEmpty
(
ParallelPeriod
(
[Date].[Fiscal].[Fiscal Year],
1,
[Date].[Fiscal].CurrentMember
)
)
Then 0
When VBA!Abs
(
(
[Measures].[Reseller Sales Amount]
-
(
[Measures].[Reseller Sales Amount],
ParallelPeriod
(
[Date].[Fiscal].[Fiscal Year],
1,
[Date].[Fiscal].CurrentMember
)
)
)
/
(
[Measures].[Reseller Sales Amount],
ParallelPeriod
(
[Date].[Fiscal].[Fiscal Year],
1,
[Date].[Fiscal].CurrentMember
)
)
) <=.02
Then 0
When (
[Measures].[Reseller Sales Amount]
-
(
[Measures].[Reseller Sales Amount],
ParallelPeriod
(
[Date].[Fiscal].[Fiscal Year],
1,
[Date].[Fiscal].CurrentMember
)
)
)
/
(
[Measures].[Reseller Sales Amount],
ParallelPeriod
(
[Date].[Fiscal].[Fiscal Year],

```

```
1,  
[Date].[Fiscal].CurrentMember  
)  
) >.02  
Then 1  
Else -1  
End
```

Drill Position in WPF OLAP Common

Drill position enables users to drill the current position of the selected member in the OLAP report. This excludes the drilled data of the selected member in other positions by using the MDX query.

The following code illustrates how to achieve drill position support in the current report.

C#

```
olapDataManager.CurrentReport.DrillType = DrillType.DrillPosition;
```

VB.NET

```
olapDataManager.CurrentReport.DrillType = DrillType.DrillPosition
```

Drill Replace in WPF OLAP Common

Drill replace displays only the immediate child members and ancestors on drill-down and drill-up respectively.

The following code illustrates how to achieve drill replace support in a current report.

C#

```
olapDataManager.CurrentReport.DrillType = DrillType.DrillReplace;
```

VB.NET

```
olapDataManager.CurrentReport.DrillType = DrillType.DrillReplace
```

Paging in WPF OLAP Common

Paging enables users to view large records by breaking them into smaller segments.

Paging feature can be achieved by setting the *EnablePaging* property to *true* in a report.

The following code snippet demonstrates how to enable paging in the current report.

C#

```
olapDataManager.CurrentReport.EnablePaging = true;
```

VB.NET

```
olapDataManager.CurrentReport.EnablePaging = True
```

The user can customize the page settings such as current page and page size for both row and column.

The following code explains how to customize current page and page size settings.

C#

```
olapDataManager.CurrentReport.PagerOptions.CategoricalCurrentPage = 1;  
olapDataManager.CurrentReport.PagerOptions.SeriesCurrentPage = 2;  
olapDataManager.CurrentReport.PagerOptions.CategoricalPageSize = 50;  
olapDataManager.CurrentReport.PagerOptions.SeriesPageSize = 50;
```

VB.NET

```
olapDataManager.CurrentReport.PagerOptions.CategoricalCurrentPage = 1  
olapDataManager.CurrentReport.PagerOptions.SeriesCurrentPage = 2  
olapDataManager.CurrentReport.PagerOptions.CategoricalPageSize = 50  
olapDataManager.CurrentReport.PagerOptions.SeriesPageSize = 50
```

QueryBuilderEngine in WPF OLAP Common

This class generates the query from the MDXQuerySpecification by invoking the GenerateQueryEx() static method of QueryBuilderEngineVersion3, the inner class of QueryBuilderEngine class.

MDX query specification

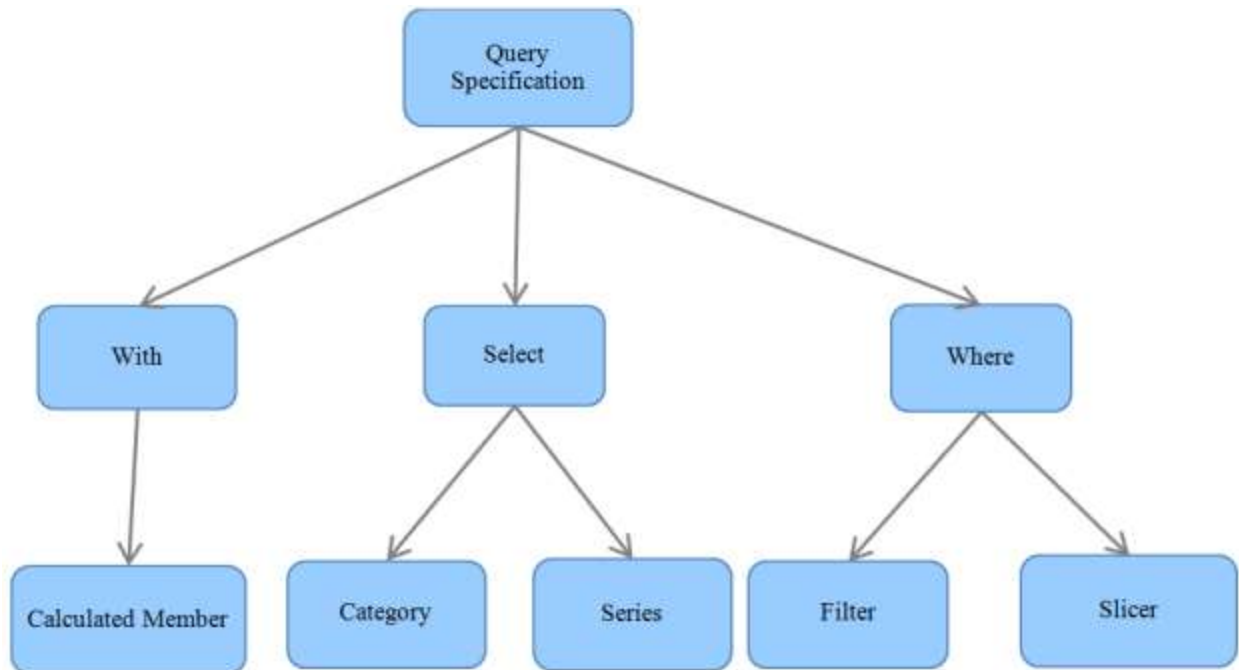
MDXQuerySpecification is the base for query creation. The MDXQuerySpecification categorizes the element into three clauses as follows:

- With
- Select
- Where

The elements in the given report are iterated and stored according to the specification.

- The calculated member element in the given report will be added under the With clause.
- The categorical and series items in the given report's categorical elements and series element will come under the Select category.
- The slicer and filter items will come under the Where category.

Based on the current report, the TogglePivot value and axis position of each item will be assigned before it is stored in the appropriate clauses.



MDXQuerySpecification

After creating the query specification, the `GenerateQueryEx()` method of `QueryBuilderEngine` is called by passing the created `MDXQuerySpecification` to generate the query.

Steps in query generation

To generate a query:

1. Get the query specification and iterate through the items in each category (With, Select and Where) of specification and separately store the column elements, row elements, filter elements, and slice elements.
2. The filter, slicer, and sort elements are moved to the appropriate axis based on their axis property.
3. Once the initial level categorizing of elements is completed, it starts creating a query string by writing using the With or Select category.
4. Now, it starts writing the query by checking each and every property.

By invoking the `BuildAxisElements()` method, the building query for the column axis elements and row axis elements are done.

The helper methods for the `BuildAxisElements()` are:

- `BuildAxisItems`.
- `BuildDimensionElement`.
- If no level is specified, the `GetDefaultLevel()` method will be called else the `BuildHierarchyElement()` will be called.
- `BuildLevelElements`.
- If the level member count is greater than zero, the `BuildMemberElement()` will be called else the `GetDefaultLevel()` method will be called.

The BuildAxisElements() will build the query for the column element when you pass the column items and will generate the query for the row elements when the row items is passed as an argument. The BuildAxisElements() method will return a Boolean value which represents whether the KPI element is existing in the given item list or not. Based on that return, the value of the KPI Element axis is fixed.

Upto this, the select section of the query was built. Now, the from section starts by invoking the BuildFilterCondition() method.

The helper method for the BuildFilterCondition() is given below:

1. The BuildFilterElements() method iterates through the elements and appends the parent details and child member details in the query based on the axis either in a row or a column.
2. Now, it comes to the Where section, by invoking the BuildSlicerElements() method.
3. Then, the BuildSlicerItem is invoked. This method checks whether the given item is dimension or measure or KPI or NamedSet or level and based on this, the appropriate query part will be appended with the query.

Finally, the cell properties will append with the created query and the query string will be returned to the OlapDataManager. By using the newly generated query, the OlapDataManager invokes the ExecuteCellSet of DataProvide, which will return the CellSet.

This CellSet can be used to create the PivotEngine, which will give the input to the controls.

MDX Query Parsing in WPF OLAP Common

MDX Query binding with drill-up and drill-down operations

This feature provides support for drill-up and drill-down operations for BI components such as BI grid, BI chart, and BI client to view the OLAP data through different levels using an MDX query instead of the OlapReport class. Previously, the drill-up and drill-down operations were supported by the OlapReport class only.

Properties

- **AllowMdxToReportParse:** Gets or sets a value indicating whether to parse the given MDX into the OlapReport class or not. The default value is true.

Sample links

OlapGrid [WPF]

<InstalledDrive>:\Users\<UserName>\AppData\Local\Syncfusion\EssentialStudio\<version>\WPF\OlapGrid.WPF\Samples\Defining Reports\MDX Query Demo

OlapChart [WPF]

<InstalledDrive>:\Users\<UserName>\AppData\Local\Syncfusion\EssentialStudio\<version>\WPF\OlapChart.WPF\Samples\Defining Reports\MDX Query Demo

OlapClient [WPF]

<InstalledDrive>:\Users\<UserName>\AppData\Local\Syncfusion\EssentialStudio\<Version>\WPF\OlapClient.WPF\Samples\Product Showcase\MDX Query Demo

OlapGrid [Silverlight]

```
<InstalledDrive>:\Users\<UserName>\AppData\Local\Syncfusion\EssentialStudio\<Version>\BI\Silverlight\OlapGrid.Silverlight\ReportDefinition\GridMDXQueryDemo
```

OlapChart [Silverlight]

```
<InstalledDrive>:\Users\<UserName>\AppData\Local\Syncfusion\EssentialStudio\<Version>\BI\Silverlight\OlapChart.Silverlight\DefiningReports\ChartMDXQueryDemo
```

OlapClient [Silverlight]

```
<InstalledDrive>:\Users\<UserName>\AppData\Local\Syncfusion\EssentialStudio\<Version>\BI\Silverlight\OlapClient.Silverlight\ProductShowcase\MDXQueryDemo
```

Adding MDX Query binding with drill-up and drill-down operations to an application

The following code samples are used to enable and disable the MDX to OLAP parsing and processing of an MDX query into OLAP data.

C#

```
//Enable MDX to OLAP parsing.
OlapDataManager olapDataManager = new OlapDataManager();
olapDataManager.MdxQuery = "MDX Query Here";
olapDataManager.ExecuteCellSet();
//Disable MDX to OLAP parsing.
OlapDataManager olapDataManager = new OlapDataManager();
olapDataManager.AllowMdxToOlapReportParse = false;
olapDataManager.MdxQuery = "MDX Query Here";
olapDataManager.ExecuteCellSet();
```

VB.NET

```
'Enable MDX to OLAP parsing.
Dim olapDataManager As New OlapDataManager()
olapDataManager.MdxQuery = "MDX Query Here"
olapDataManager.ExecuteCellSet()
'Enable MDX to OLAP parsing.
Dim olapDataManager As New OlapDataManager()
olapDataManager.AllowMdxToOlapReportParse = False
olapDataManager.MdxQuery = "MDX Query Here"
olapDataManager.ExecuteCellSet()
```

OLAP Data Processing in WPF OLAP Common

The OlapDataManager accepts an input from a user and processes the data based on the supplied input and generates the formatted output which Syncfusion OLAP controls can understand. The OlapDataManager can process the OLAP data (cube).

Steps in processing OLAP data

To process the OLAP data:

1. Provide an input to establish a connection to the specified data source. The connection information can be given as a connection string that contains information about the data source.

Example connection string: "DataSource = localhost; Initial Catalog = AdventureWorksDW";

2. Connect to a server or an offline cube as a data source.
3. After the connection is established, provide the input for data processing. You can give two types of inputs to process the OLAP data. They are:
 - MDX Query: You can write a MDX Query for the database and give that query as an input.
 - OlapReport: You can create an OLAP report and give that report as an input to the *OlapDataManager*.
4. The output will not differ, based on the input type. The processing method will differ in OLAP base, based on the input type.
5. If MDX query is given as an input then the query will be executed on the connected database.
6. If OlapReport is given as an input:
7. MDX query specification will be created based on the OlapReport.
8. From the MDX query specification, the MDX query will be generated with the help of *OlapQueryBuilderEngine*.
9. The generated query will be executed on the connection database.
10. After the query is executed either from the MDX query input or from the OlapReport input, the result is obtained.
11. This result set will be formatted to provide input for the controls. The formatted input should be passed to the controls.
12. The output will be displayed in the controls.

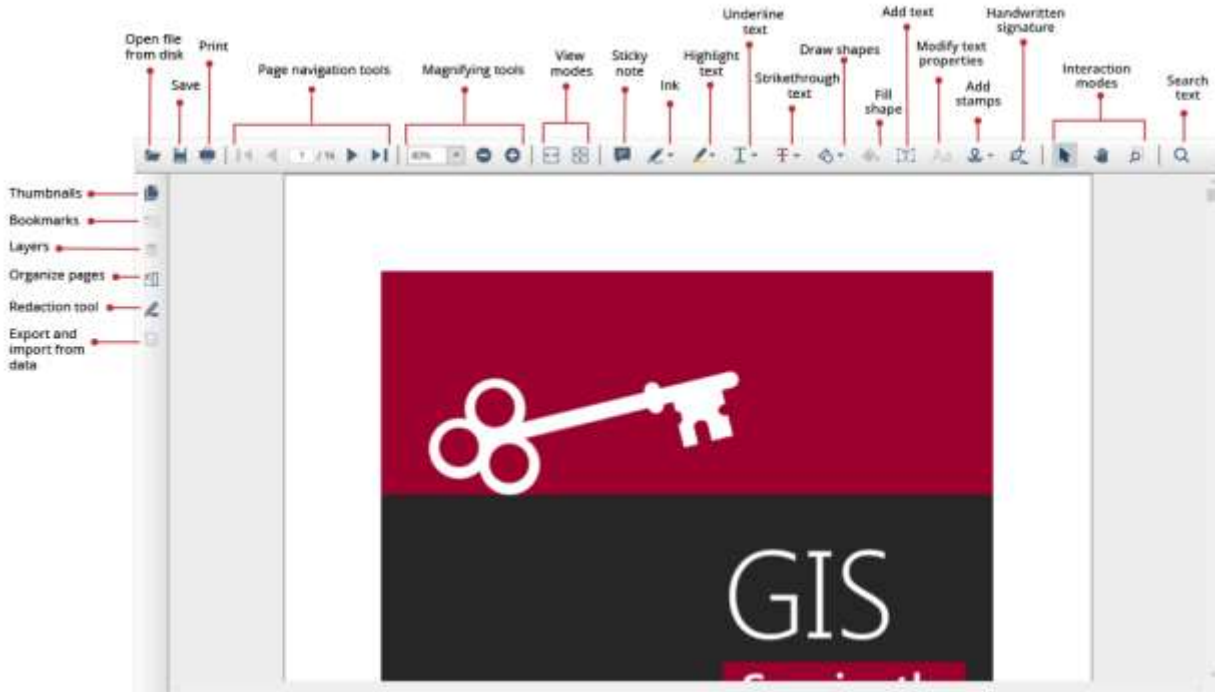
In Silverlight

1. The *OlapDataManager* requires an *OlapReport* and a virtual channel in the form of *OlapDataManager*.
2. The *OlapDataManager* is a set to control and in the *DataBind* method, the control will make an asynchronous call with the help of a virtual channel provided in *OlapDataManager*.
3. Now, with the help of WCF Service, the control communicates with OLAP base to retrieve *CellSet*.
4. The control passes the obtained *CellSet* to the *OlapDataManager* and in turn the *OlapDataManager* returns the *PivotEngine* to the control.
5. The output will reflect in the controls.

PdfViewer

WPF Pdf Viewer Overview

The PDF Viewer control supports viewing, reviewing, and printing PDF files in the WPF applications. The thumbnail, bookmark, hyperlink, and table of contents support provide easy navigation within and outside the PDF files. The form-filling support provides a platform to fill, flatten, save, and print PDF files with AcroForm. The PDF files can be reviewed with the abundantly available annotation tools.



Key features

- **Open PDF files:** Opens PDF files, both normal and protected, with AES and RC4 encryption algorithms (password-protected). Open PDF files from the stream, file path, and PdfLoadedDocument objects.
- **Instant loading:** Loads the PDF files with thousands of pages instantly.
- **Less runtime memory:** On-demand loading, and virtualization mean the control holds only the minimum required pages at runtime to reduce the memory consumption.
- **Virtualized pages:** Renders pages on demand to help reduce the initial load time when working with large documents.
- **Printing:** Supports both the silent printing as well as printing using the print dialog.
- **Select and copy text:** Allows you to select and copy text from the PDF files.
- **Search text:** Allows you to locate a word or phrase easily in a PDF file.
- **Annotations:** Allows you to review and annotate the PDF files using a rich set of annotating tools.
- **Navigations:** Supports many types of internal and external navigations such as bookmarks, thumbnails, hyperlinks, and table of contents navigations.
- **Form Filling:** Provides the ability to fill, edit, flatten, and save AcroForms fields in the PDF files.
- **Organize pages:** Allows you to rotate, rearrange, and delete pages in a PDF file using a miniature preview of the PDF pages.
- **Redaction:** Allows you to remove the sensitive text and graphics from a PDF file.
- **PDF Layers:** Supports displaying the layer contents (adding, modifying, or deleting annotations) over the layers and adding, modifying, or deleting layers in a PDF file.
- **Exporting:** Supports exporting the PDF pages to image and exporting form data.

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Getting Started with WPF Pdf Viewer

This section explains how to create an application that displays a PDF file using the [WPF PDF Viewer](#).

To get started quickly with WPF PDF Viewer, you can check on this video:

```
<style>#WPFPDFViewerVideoTutorial{width : 90% !important; height: 400px !important }</style>
```

```
<iframe id='WPFPDFViewerVideoTutorial' src='https://www.youtube.com/embed/H1YBX_-QWKc'></iframe>
```

Assemblies required

The following assemblies are required in your WPF application to use the PDF Viewer.

Required Assemblies	Description
Syncfusion.Compression.Base	This library handles various compression and decompression operations that are used in the PDF file internally.
Syncfusion.Pdf.Base	This library contains the PDF reader and creator that supports the PDF Viewer.
Syncfusion.PdfViewer.WPF	This component contains the rendering area and other related UI elements.
Syncfusion.Shared.WPF	This component contains various UI controls (ColorPickerPalette and Numeric UpDown) that are used in the PDF Viewer.

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.

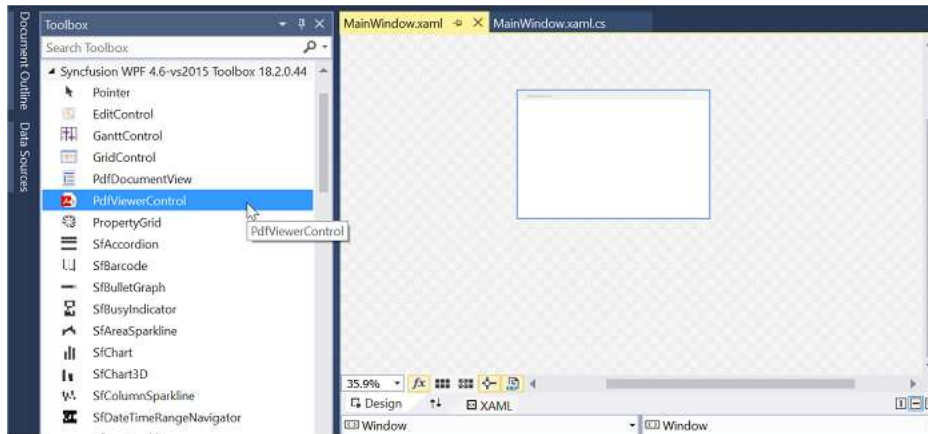
Create a simple PDF Viewer application

You can create a PDF Viewer application by simply drag the control from the Visual Studio toolbox and drop in the designer window of your application or by creating the control manually from code.

Drag and drop the PdfViewerControl from the toolbox

Follow these steps to drag and drop the PdfViewerControl from the toolbox.

1. After installing the Syncfusion Essential Studio in your machine, create a new WPF application in Visual Studio.
2. Open the Visual Studio toolbox.
3. Navigate to **Syncfusion WPF Toolbox** tab and drag the PdfViewerControl toolbox item to the Designer window, it automatically adds the required references to the current application.



PDF viewer control in toolbox

Adding control manually in XAML

To add control manually in XAML, do the following steps,

1. Add the required assemblies as a reference to the project.
2. Add the following Syncfusion namespace in XAML to make use of the [PdfViewerControl](#).

XML

```
<Window
xmlns:syncfusion="clr-
namespace:Syncfusion.Windows.PdfViewer;assembly=Syncfusion.PdfViewer.WPF"
/>
```

3. Declare the [PdfViewerControl](#) in the XAML page.

XML

```
<Window
x:Class="PdfViewerDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="PDF Viewer" WindowState="Maximized"
xmlns:syncfusion="clr-
namespace:Syncfusion.Windows.PdfViewer;assembly=Syncfusion.PdfViewer.WPF">
<Grid x:Name="HomeGrid">
<syncfusion:PdfViewerControl
x:Name="pdfViewer"></syncfusion:PdfViewerControl>
</Grid>
</Window>
```

Adding control manually from code

To add control manually from code, follow these steps,

1. Add the required assemblies as a reference to the project.
2. Add the following Syncfusion namespace class file.

CSHARP

```
using Syncfusion.Windows.PdfViewer;
```

3. Create a PdfViewerControl instance and add it to the main window.

CSHARP

```
using Syncfusion.Windows.PdfViewer;
using System.Windows;
namespace PdfViewerDemo
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        # region Constructor
        public MainWindow()
        {
            InitializeComponent();
            PdfViewerControl pdfViewer = new PdfViewerControl();
            HomeGrid.Children.Add(pdfViewer);
        }
        #endregion
    }
}
```

Display PDF file

The [PdfViewerControl](#)'s [ItemSource](#) property allows you to bind PDF documents in XAML. This property accepts a stream input that can be bounded to the viewer during initialization. The following steps explain how to display a PDF file using the [PdfViewerControl](#):

Note: From v16.3.0x onwards, PDF Viewer uses PDFium as a default rendering engine to render the PDF pages, which is a more robust and promising rendering engine. Refer to this [link](#) for more details.

1. Create a simple class in the application that implements [INotifyPropertyChanged](#) and declare a file stream property in the class as shown in the following code sample.

CSHARP

```
using System.ComponentModel;
using System.IO;
namespace PdfViewerDemo
{
    public class PdfReport : INotifyPropertyChanged
    {
        private Stream docStream;
        public event PropertyChangedEventHandler PropertyChanged;
        public Stream DocumentStream
        {
            get
            {
                return docStream;
            }
            set
            {
            }
        }
    }
}
```

```

{
    docStream = value;
    OnPropertyChanged(new PropertyChangedEventArgs("DocumentStream"));
}
}
public PdfReport()
{
    //Load the stream from the local system.
    docStream = new FileStream(@"../../Data/HTTP Succinctly.pdf",
        FileMode.OpenOrCreate);
}
public void OnPropertyChanged(PropertyChangedEventArgs e)
{
    if (PropertyChanged != null)
        PropertyChanged(this, e);
}
}
}

```

2. Set the [DataContext](#) to the Window for data binding. To add the [DataContext](#) in XAML, use the following code example.

XML

```

<Window.DataContext>
<pdfviewerdemo:PdfReport/>
</Window.DataContext>

```

3. After setting the [DataContext](#), bind the file stream property to the [ItemSource](#) dependency property of [PdfViewerControl](#) using the following code sample in XAML.

XML

```

<syncfusion:PdfViewerControl x:Name="pdfViewer" ItemSource="{Binding
    DocumentStream}"/>

```

The sample project for displaying PDF files using the PDF Viewer is available in the [GitHub](#).

Note: Alternatively, the Open button in the toolbar can also be used to load and display the PDF documents at runtime. Refer to this [link](#) for more details.

Theme

The WPF PdfViewer Control supports various built-in themes. Refer to the below links to apply themes for the PdfViewerControl,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

PDF Rendering Engines in WPF Pdf Viewer

Syncfusion WPF PDF Viewer renders the PDF pages through 2 different rendering engines.

- PDFium (Google Chrome's PDF rendering engine)
- SfPdf (Syncfusion's Own PDF rendering engine)

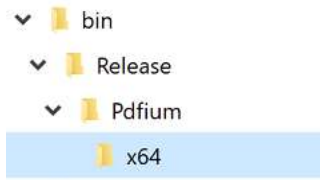
PDFium

PDFium is used in Google Chrome for rendering PDF files. It provides accurate and robust PDF rendering. It is the recommended PDF rendering engine.

Note: From v16.3.0.x onwards, this PDFium rendering engine is the default rendering engine of Syncfusion WPF PDF Viewer.

How PDFium works with Syncfusion's PDF Viewer

- On running your WPF application, Syncfusion PDF Viewer control generates a folder named **PDFium** in the application output path folder (for example: bin/release or bin/debug) at runtime.
- Syncfusion PDF Viewer control detects the architecture of the running machine automatically.
- Next, it creates another subfolder named "x64" or "x86" based on the machine architecture.
- Extracts the PDFium binary (PDFium.dll) into the subfolder (x64 or x86) and consumes it to render PDF files.



Note: PDFium rendering is not supported in Windows XP operating system.

How to run PDFium in a restricted access environment

If there is any access restriction applied to the application output folder, then the Syncfusion PDF Viewer control cannot be able to extract and consume the PDFium engine as mentioned above.

In that situation, you need to add the following steps to consume the PDFium rendering engine.

- Create a folder where your application can access, create & read files. For example, "d:\ThirdPartyBinaries\".
- Update the path of this folder to the [ReferencePath](#) property of PDF Viewer control, like shown in the following code sample.
- If [ReferencePath](#) is set, then PDF Viewer control extracts the PDFium binary inside that specified folder and consume the PDFium rendering engine.

C#

```
using System.Windows;
namespace PdfViewerDemo
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        # region Constructor
        public MainWindow()
        {
            InitializeComponent();
            pdfViewer.ReferencePath = @"D:\ThirdPartyBinaries\";
            pdfViewer.Load("Sample.pdf");
        }
        #endregion
    }
}
```

Note: In the run time, the PDF viewer will check the custom folder path provided in the [ReferencePath](#) property. If you already placed the Pdfium assemblies in the custom folder path, it will refer to the already available assemblies from the location. It won't generate the assemblies in the folder again.

You need to place the PDFium assembly in the correct folder structure as mentioned below.

- * ThirdPartyBinaries
- * Pdfium
- * x86
- * Pdfium.dll

* x64

* Pdfium.dll

SfPdf

SfPdf is the Syncfusion's own PDF rendering engine. Before v16.3.0.x, PDF Viewer control has used this rendering engine as default to rendering the PDF pages. If you wish to use **SfPdf** rendering engine or face any compatibility issues with **PDFium** rendering engine in your environment, you may set the [RenderingEngine](#) property to **SfPdf** as shown in the following code sample.

Note: The recommended PDF rendering engine is PDFium.

C#

```
using System.Windows;
using Syncfusion.Windows.PdfViewer;
namespace PdfViewerDemo
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        # region Constructor
        public MainWindow()
        {
            InitializeComponent();
            pdfViewer.RenderingEngine = PdfRenderingEngine.SfPdf;
            pdfViewer.Load("Sample.pdf");
        }
        #endregion
    }
}
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Viewing PDF Files in WPF Pdf Viewer

PDF Viewer allows you to easily open and view the PDF files interactively using the Open button in the built-in toolbar as well as from code behind using the available APIs.

Open PDF file from the local disk using toolbar

You can open a PDF file from the toolbar by browsing it from the local disk. You can open both the normal and password-protected PDF files. The Open button in the toolbar allows you to perform the same using the following steps.

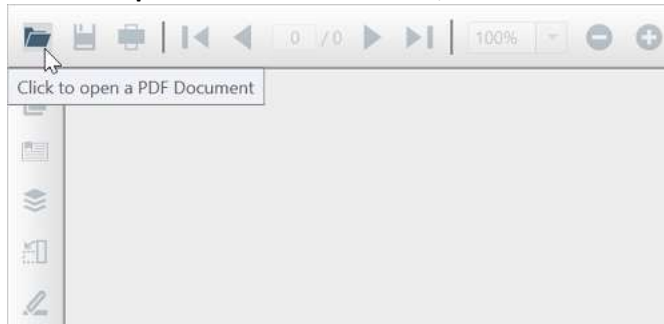
1. Add the [PdfViewerControl](#) in the MainWindow.xaml and run the project.

XML

```
<Window
x:Class="PdfViewerDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
WindowState="Maximized"
xmlns:syncfusion="clr-
namespace:Syncfusion.Windows.PdfViewer;assembly=Syncfusion.PdfViewer.WPF">
<Grid>
<syncfusion:PdfViewerControl
x:Name="pdfViewer"></syncfusion:PdfViewerControl>
</Grid>
</Window>
```

2. Click the **Open** button in the toolbar, as shown in the following picture.



3. In the open file dialog, enter the file name or browse the file from the local disk and select **Open**.

View PDF file using the file path

You can view the PDF file from code behind, by passing the file path as a parameter to the [Load](#) method of [PdfViewerControl](#). It accepts both the absolute and relative file paths. Refer to the following code to perform the same.

C#

```
using System.Windows;
namespace PdfViewerDemo
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        # region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Load PDF file using the file path.
            pdfViewer.Load(@"HTTP Succinctly.pdf");
        }
        #endregion
    }
}
```

View PDF file from stream

You can view the PDF file from code behind, by passing the [Stream](#) as a parameter to the Load method of [PdfViewerControl](#). Refer to the following code to perform the same.

C#

```
using System.IO;
using System.Windows;
namespace PdfViewerDemo
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        # region Constructor
        public MainWindow()
        {
            InitializeComponent();
            FileStream stream = new FileStream(@"HTTP Succinctly.pdf", FileMode.Open);
            //Load PDF file using stream.
            pdfViewer.Load(stream);
        }
        #endregion
    }
}
```

View PDF file using the ItemSource property

You can also view a PDF file using the [ItemSource](#) property of [PdfViewerControl](#). The property accepts a string file path, a [Stream](#), and a [PdfLoadedDocument](#) object.

C#

```
using System.Windows;
namespace PdfViewerDemo
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        # region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Load PDF file using the `ItemSource` property.
            pdfViewer.ItemSource = @"HTTP Succinctly.pdf";
        }
        #endregion
    }
}
```

Refer the below code snippet to load a [Stream](#) using the [ItemSource](#) property

C#

```
//Load PDF file as Stream using the `ItemSource` property.
pdfViewer.ItemSource = new FileStream(@"HTTP Succinctly.pdf", FileMode.Open);
```

Refer the below code snippet to load a [PdfLoadedDocument](#) object using the [ItemSource](#) property.

C#

```
//Load PDF file as PdfLoadedDocument object using the `ItemSource` property.
PdfLoadedDocument pdfLoadedDocument = new PdfLoadedDocument(@"HTTP Succinctly.pdf");
pdfViewer.ItemSource = pdfLoadedDocument;
```

View PDF files without using the toolbar

The [PdfDocumentView](#) control allows you to view the PDF files without toolbar using the [Load](#) methods from code behind. Refer to the following steps to perform the same.

1. Add the [PdfDocumentView](#) control in the MainWindow.xaml.

XML

```
<Window
x:Class="PdfViewerDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
WindowState="Maximized"
xmlns:syncfusion="clr-namespace:Syncfusion.Windows.PdfViewer;assembly=Syncfusion.PdfViewer.WPF">
<Grid>
<syncfusion:PdfDocumentView x:Name="pdfViewer"></syncfusion:PdfDocumentView>
</Grid>
</Window>
```

2. Load the file using the [Load](#) method as mentioned in the following code snippet in MainWindow.xaml.cs.

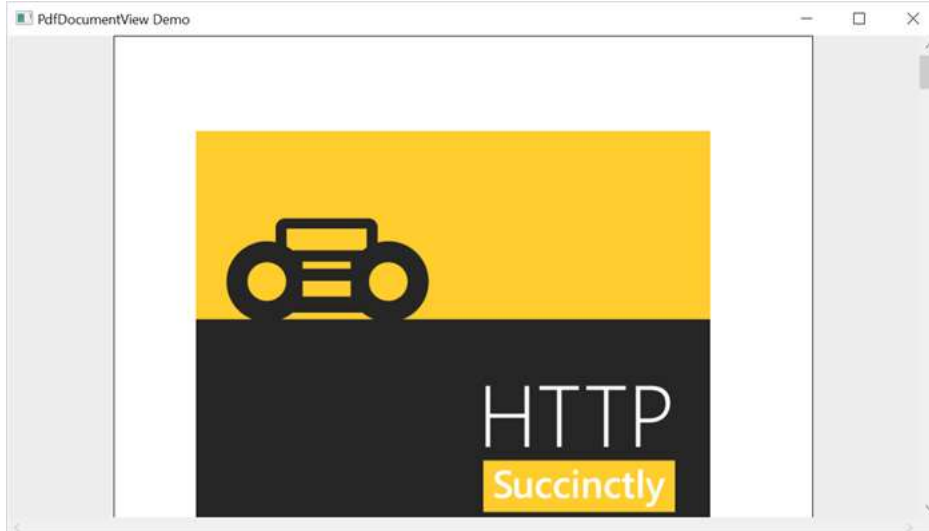
C#

```
using System.Windows;
namespace PdfViewerDemo
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        # region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Load PDF file using file path.
            pdfViewer.Load(@"HTTP Succinctly.pdf");
        }
        #endregion
    }
}
```

```
}  
}
```

3. Run the project.

The following picture illustrates how the PDF file being displayed in [PdfDocumentView](#) control.



Obtain the PDF file information

You can get the information on the PDF file that is being displayed in the control using the [DocumentInfo](#) property of [PdfViewerControl](#). This property provides you the information such as file name and the folder name from that the PDF file is opened using the [FileName](#) and [FilePath](#) properties respectively.

Refer to the following code to obtain the document information using the [DocumentInfo](#) property.

C#

```
using System.IO;  
using System.Windows;  
namespace PdfViewerDemo  
{  
    /// <summary>  
    /// Interaction logic for Window1.xaml  
    /// </summary>  
    public partial class MainWindow : Window  
    {  
        # region Constructor  
        public MainWindow()  
        {  
            InitializeComponent();  
            //Load PDF file with the absolute file path.  
            pdfViewer.Load(Path.GetFullPath(@"HTTP Succinctly.pdf"));  
            //Get the file name  
            string fileName = pdfViewer.DocumentInfo.FileName;  
            string folder = pdfViewer.DocumentInfo.FilePath;  
        }  
        #endregion  
    }  
}
```

```
}  
}
```

If you open a file using the toolbar, you can obtain the information in the [DocumentLoaded](#) event. Refer to the following code to achieve the same.

C#

```
using System.Windows;  
namespace PdfViewerDemo  
{  
    /// <summary>  
    /// Interaction logic for Window1.xaml  
    /// </summary>  
    public partial class MainWindow : Window  
    {  
        # region Constructor  
        public MainWindow()  
        {  
            InitializeComponent();  
            //wire the DocumentLoaded event  
            pdfViewer.DocumentLoaded += PdfViewer_DocumentLoaded;  
        }  
        #endregion  
        private void PdfViewer_DocumentLoaded(object sender, System.EventArgs args)  
        {  
            //Get the file name  
            string fileName = pdfViewer.DocumentInfo.FileName;  
            string folder = pdfViewer.DocumentInfo.FilePath;  
        }  
    }  
}
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Viewing Password protected PDF Files in WPF Pdf Viewer

PDF Viewer allows you to view the password-protected PDF files by passing the file name and the correct password as parameters to the [Load](#) method of [PdfViewerControl](#). Refer to the following code to perform the same.

C#

```
using System.Windows;  
namespace PdfViewerDemo  
{  
    /// <summary>  
    /// Interaction logic for MainWindow.xaml  
    /// </summary>  
    public partial class MainWindow : Window  
    {  
        # region Constructor  
        public MainWindow()  
        {  

```



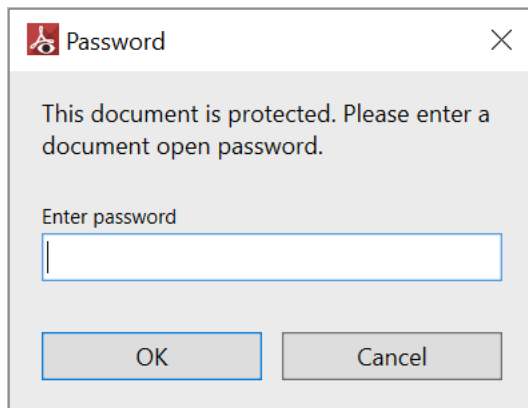
```

InitializeComponent();
//Load password protected PDF file using the file path and the password.
pdfViewer.Load(@"Template.pdf", "password");
}
#endregion
}
}

```

View password protected PDF files in run time

When opening a password protected PDF file in run time using the Open button available in the toolbar, the following built-in password dialog window helps to view the file contents, requesting the correct password from the user. In the password textbox, enter the correct password and click OK.



Hide the built-in password dialog

PDF Viewer helps to hide the built-in password using the `GetDocumentPassword` and gets the password using the `Password` property of the `GetDocumentPasswordEventArgs`. The event `GetDocumentPassword` occurs every time when you try to open a password protected PDF file in run-time. By setting the `Handled` property of `GetDocumentPasswordEventArgs` to true, the built-in password dialog will not appear. Refer to the following code to hide the password dialog and to provide password to open the file by wiring the event.

C#

```

using Syncfusion.Windows.PdfViewer;
using System.Collections.Generic;
using System.Windows;
namespace PasswordPDFDemo
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            pdfViewerControl.GetDocumentPassword += PdfViewer_GetDocumentPassword;
            string filePath = Path.GetFullPath(@"../../Data/syncfusion.pdf");
            pdfViewerControl.Load(filePath);
        }
        private void PdfViewer_GetDocumentPassword(object sender,
            GetDocumentPasswordEventArgs e)
        {

```

```

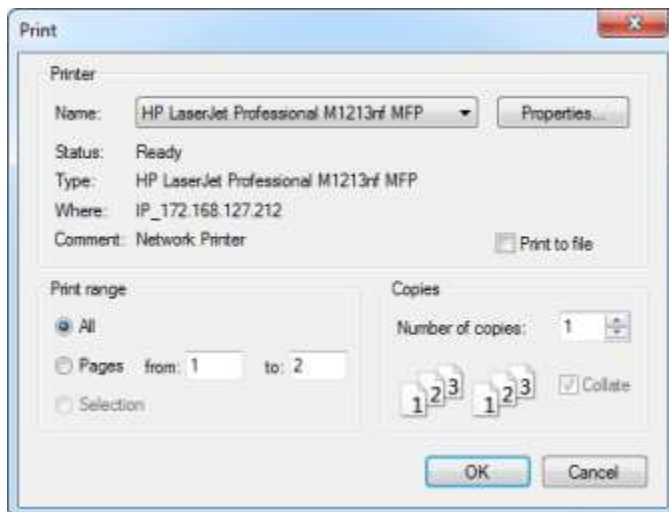
System.Security.SecureString secureString = new
System.Security.SecureString();
secureString.AppendChar('p');
secureString.AppendChar('a');
secureString.AppendChar('s');
secureString.AppendChar('s');
secureString.AppendChar('w');
secureString.AppendChar('o');
secureString.AppendChar('r');
secureString.AppendChar('d');
e.Password = secureString;
// Enabling handled to hide the password dialog.
e.Handled = true;
}
}
}

```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Printing PDF Files in WPF Pdf Viewer

PDF Viewer allows printing loaded PDFs using the Print button in the toolbar. The following Print dialog opens upon triggering the Print button.



Note: PdfDocumentView is used to view the PDF documents without the toolbar. So, make use of PdfViewerControl to print the document using Print button in the toolbar.

Silent Printing

The [Print](#) method of [PdfViewerControl](#) and [PdfDocumentView](#) allows you to print PDF files silently to the system's default printer, without any user interaction. You can enable the preferred settings for silent printing using the [PrinterSettings](#) property. The following code example shows how to perform silent printing in WPF PDF Viewer.

C#

```
pdfviewer1.Print();
```

VB.NET

```
pdfviewer1.Print()
```

Customizing print size

PDF viewer printer settings allows scaling PDF pages to shrink or enlarge while printing.

Actual Size

Actual size is the default value of print size option in printer settings. This prints the loaded PDF document without any scaling factors. The pages that do not fit on the paper will be cropped. The following code example illustrates how to print the document in Actual Size.

C#

```
// Prints the document in actual size.  
pdfviewer1.PrinterSettings.PageSize = PdfViewerPrintSize.ActualSize;
```

VB.NET

```
' Prints the document in actual size.  
pdfviewer1.PrinterSettings.PageSize = PdfViewerPrintSize.ActualSize
```

Fit

Fit option enlarges or reduces each page to fit the printable area of the selected paper size. The following code example illustrates the same.

C#

```
// Prints the document in fit size.  
pdfviewer1.PrinterSettings.PageSize = PdfViewerPrintSize.Fit;
```

VB.NET

```
' Prints the document in fit size.  
pdfviewer1.PrinterSettings.PageSize = PdfViewerPrintSize.Fit
```

Custom Scale

Custom Scale option resizes the page with the specified scale percentage. The following code example illustrates the same.

C#

```
// Prints the document with custom scaling.  
pdfviewer1.PrinterSettings.PageSize = PdfViewerPrintSize.CustomScale;  
// Scale percentage with the page to be resized and it is applicable only  
// for Custom Scale. The default value is 100.  
pdfviewer1.PrinterSettings.ScalePercentage = 120;
```

VB.NET

```
' Prints the document with custom scaling.  
pdfviewer1.PrinterSettings.PageSize = PdfViewerPrintSize.CustomScale
```

```
' Scale percentage with the page to be resized and it is applicable only for Custom Scale. The default value is 100.  
pdfviewer1.PrinterSettings.ScalePercentage = 120
```

Printing PDF document with orientation settings

PDF Viewer printer settings allows the user to print the document with a custom orientation.

Auto Portrait/Landscape

Auto Portrait/Landscape is the default option and it automatically selects the best orientation (Portrait or Landscape) based on the content and selected paper. The following code example illustrates how to print the document in Auto orientation.

C#

```
// Prints the document in auto orientation.  
pdfviewer1.PrinterSettings.PageOrientation = PdfViewerPrintOrientation.Auto;
```

VB.NET

```
' Prints the document in auto orientation.  
pdfviewer1.PrinterSettings.PageOrientation = PdfViewerPrintOrientation.Auto
```

Portrait

Portrait option prints the PDF document in portrait orientation and it overrides the orientation settings provided in the print dialog. The following code example illustrates the same.

C#

```
// Prints the document in portrait orientation.  
pdfviewer1.PrinterSettings.PageOrientation =  
PdfViewerPrintOrientation.Portrait;
```

VB.NET

```
' Prints the document in portrait orientation.  
pdfviewer1.PrinterSettings.PageOrientation =  
PdfViewerPrintOrientation.Portrait
```

Landscape

Landscape option prints the PDF document in landscape orientation and it overrides the orientation settings provided in print dialog. The following code example illustrates the same.

C#

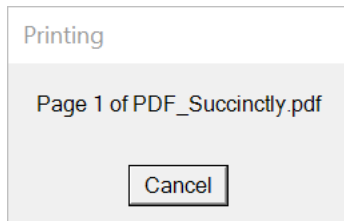
```
// Prints the document in landscape orientation.  
pdfviewer1.PrinterSettings.PageOrientation =  
PdfViewerPrintOrientation.Landscape;
```

VB.NET

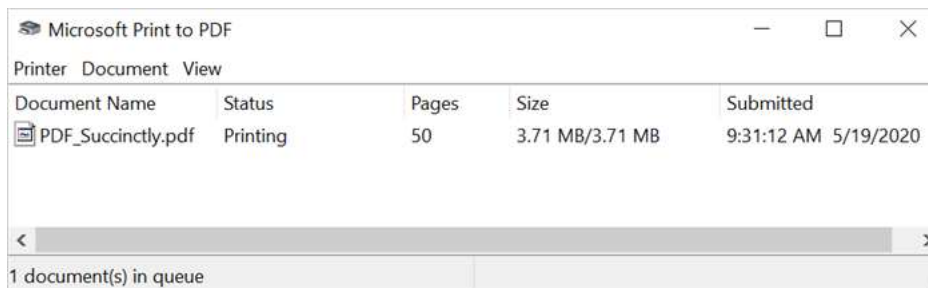
```
' Prints the document in landscape orientation.  
pdfviewer1.PrinterSettings.PageOrientation =  
PdfViewerPrintOrientation.Landscape
```

Set the custom document name to be displayed when printing

PDF viewer allows you to set the custom document name to be displayed when printing the PDF document using the [DocumentName](#) property that is available in the [PrinterSettings](#). You can either set the original file name or a custom text to the property for display. Setting the [DocumentName](#) property will affect the Document name details in the print status dialog.



The [DocumentName](#) property will also affect the Document name details in the print queue information window.



The following code shows how to set the original file name to the [DocumentName](#) property.

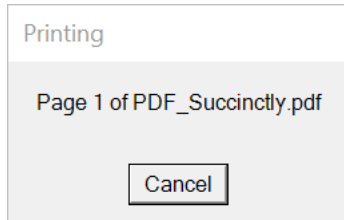
C#

```
using System.IO;
using System.Windows;
namespace PdfViewer
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        #region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Load PDF file with the absolute file path.
            string filePath = Path.GetFullPath(@"../../Data/PDF_Succinctly.pdf");
            pdfViewer.Load(filePath);
            //Get the file name
            string fileName = pdfViewer.DocumentInfo.FileName;
            //Set the document name to be displayed when printing the document.
            pdfViewer.PrinterSettings.DocumentName = fileName;
            //Print the file.
            pdfViewer.Print();
        }
        #endregion
    }
}
```

```
}
}
```

Hide Print Status dialog when printing the PDF files

PDF Viewer allows you to hide the following print status dialog when printing the PDF files by setting the value `false` to the [ShowPrintStatusDialog](#) property that is available in the [PrinterSettings](#).



It will be helpful if no UI interactions are required when printing. The following code shows how to hide the print status dialog using the [ShowPrintStatusDialog](#) property.

C#

```
using System.IO;
using System.Windows;
namespace PdfViewer
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        #region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Load PDF file.
            string filePath = Path.GetFullPath(@"../../Data/PDF_Succinctly.pdf");
            pdfViewer.Load(filePath);
            //Set the document name to be displayed when printing the document.
            pdfViewer.PrinterSettings.ShowPrintStatusDialog = false;
            //Print the file.
            pdfViewer.Print();
        }
        #endregion
    }
}
```

Customizing Paper source in Silent Printing

You can customize [PaperSource](#) in silent printing by changing the [PaperSource](#) property in [PageSettings](#) and passing it as a parameter with printer name to the `Print` API.

Note: The `System.Drawing` assembly is required.

The following code example illustrates how to set [PaperSource](#) for Your printer. In the [PageSettings](#), pass the required [PaperSource](#) value.

C#

```

PrinterSettings printerSettings = new PrinterSettings();
PageSettings pageSettings = new PageSettings();
private void Button_Click(object sender, RoutedEventArgs e)
{
    //Enter Your Printer's Name
    printerSettings.PrinterName = "Mention Your printer's name here";
    //Changes the papersource in pagesettings
    pageSettings.PaperSource = printerSettings.PaperSources[3];
    //Prints the PDF with required papersource
    pdfviewer.Print(printerSettings.PrinterName, pageSettings);
}

```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Exporting PDF pages in WPF Pdf Viewer

Essential PDF Viewer allows exporting pages of a PDF file into JPG, PNG, TIFF, and BMP formats using [ExportAsImage](#) methods. This option helps to convert PDF pages into images.

Exporting a single page

You can export a single page of the PDF file into an image by passing the page index as a parameter of the [ExportAsImage](#) method. Refer to the following code to export a single page of PDF into JPEG image.

C#

```

PdfViewerControl pdfViewer = new PdfViewerControl();
//Load the input PDF file
PdfLoadedDocument loadedDocument = new PdfLoadedDocument("Sample.pdf");
pdfViewer.Load(loadedDocument);
//Export the particular PDF page as image at the page index of 0
BitmapSource image = pdfViewer.ExportAsImage(0);
//Set up the output path
string output = @"..\..\Output\Image";
if (image != null)
{
    //Initialize the new Jpeg bitmap encoder
    BitmapEncoder encoder = new JpegBitmapEncoder();
    //Create the bitmap frame using the bitmap source and add it to the encoder
    encoder.Frames.Add(BitmapFrame.Create(image));
    //Create the file stream for the output in the desired image format
    FileStream stream = new FileStream(output + ".Jpeg", FileMode.Create);
    //Save the stream, so that the image will be generated in the output
    location
    encoder.Save(stream);
}
//Dispose the document
loadedDocument.Dispose();
loadedDocument = null;

```

VB.NET

```

Dim pdfViewer As PdfViewerControl = New PdfViewerControl()
'Load the input PDF file

```

```

Dim loadedDocument As PdfLoadedDocument = New
PdfLoadedDocument("Sample.pdf")
pdfViewer.Load(loadedDocument)
'Export the particular PDF page as image at the page index of 0
Dim image As BitmapSource = pdfViewer.ExportAsImage(0)
'Set up the output path
Dim output As String = "..\..\Output\Image"
If image IsNot Nothing Then
'Initialize the new Jpeg bitmap encoder
Dim encoder As BitmapEncoder = New JpegBitmapEncoder()
'Create the bitmap frame using the bitmap source and add it to the encoder.
encoder.Frames.Add(BitmapFrame.Create(image))
'Create the file stream for the output in the desired image format
Dim stream As FileStream = New FileStream(output & ".Jpeg", FileMode.Create)
'Save the stream, so that the image will be generated in the output location
encoder.Save(stream)
End If
'Dispose the document
loadedDocument.Dispose()
loadedDocument = Nothing

```

Note: You can follow a similar step for exporting PDF into images in all the other image formats.

Exporting a specific range of pages

You can export a specific range of PDF pages into images by passing the start and end page indexes as parameters of the [ExportAsImage](#) method. Refer to the following code to export the pages of PDF into JPEG images.

C#

```

PdfViewerControl pdfViewer = new PdfViewerControl();
//Load the input PDF file
PdfLoadedDocument loadedDocument = new PdfLoadedDocument("Sample.pdf");
pdfViewer.Load(loadedDocument);
//Export all the pages as images at the specific page range
BitmapSource[] image = pdfViewer.ExportAsImage(0, loadedDocument.Pages.Count - 1);
//Set up the output path
string output = @"..\..\Output\Image";
if (image != null)
{
for (int i = 0; i < image.Length; i++)
{
//Initialize the new Jpeg bitmap encoder
BitmapEncoder encoder = new JpegBitmapEncoder();
//Create the bitmap frame using the bitmap source and add it to the encoder
encoder.Frames.Add(BitmapFrame.Create(image[i]));
//Create the file stream for the output in the desired image format
FileStream stream = new FileStream(output + i.ToString() + ".Jpeg",
FileMode.Create);
//Save the stream, so that the image will be generated in the output
location
encoder.Save(stream);
}
}
//Dispose the document

```



```
loadedDocument.Dispose();
loadedDocument = null;
```

VB.NET

```
Dim pdfViewer As PdfViewerControl = New PdfViewerControl()
'Load the input PDF file
Dim loadedDocument As PdfLoadedDocument = New
PdfLoadedDocument("Sample.pdf")
pdfViewer.Load(loadedDocument)
'Export all the pages as images at the specific page range
Dim image As BitmapSource() = pdfViewer.ExportAsImage(0,
loadedDocument.Pages.Count - 1)
'Set up the output path
Dim output As String = "..\..\Output\Image"
If image IsNot Nothing Then
For i As Integer = 0 To image.Length - 1
'Initialize the new Jpeg bitmap encoder
Dim encoder As BitmapEncoder = New JpegBitmapEncoder()
'Create the bitmap frame using the bitmap source and add it to the encoder
encoder.Frames.Add(BitmapFrame.Create(image(i)))
'Create the file stream for the output in the desired image format
Dim stream As FileStream = New FileStream(output & i.ToString() & ".Jpeg",
FileMode.Create)
'Save the stream, so that the image will be generated in the output location
encoder.Save(stream)
Next
End If
'Dispose the document
loadedDocument.Dispose()
loadedDocument = Nothing
```

Exporting with a custom image size

You can export PDF pages as images with custom width and height by passing the required size as a parameter of the [ExportAsImage](#) method. Refer to the following code to export the pages of PDF into JPEG images. Refer to the following code to export the page at the index of 0 into JPEG image with the width and the height of 1836 and 2372 in pixels respectively.

C#

```
//Export the particular PDF page as image at the page index of 0
BitmapSource image = pdfViewerControl.ExportAsImage(0, new SizeF(1836,
2372), false);
//Set up the output path
string output = @"..\..\Output\Image";
if (image != null)
{
//Initialize the new Jpeg bitmap encoder
BitmapEncoder encoder = new JpegBitmapEncoder();
//Create the bitmap frame using the bitmap source and add it to the encoder
encoder.Frames.Add(BitmapFrame.Create(image));
//Create the file stream for the output in the desired image format
FileStream stream = new FileStream(output + ".Jpeg", FileMode.Create);
//Save the stream, so that the image will be generated in the output
location
```

```
encoder.Save(stream);  
}
```

Note: To maintain the aspect ratio of output images, you are required to pass the value `true` for the `keepAspectRatio` parameter.

Exporting with a custom image resolution

You can export PDF pages as images with a custom horizontal and vertical resolution by passing the required `DpiX` and `DpiY` values as parameters of the [ExportAsImage](#) method respectively. Refer to the following code to export the pages of PDF into JPEG images with the horizontal and vertical resolution of 200 respectively.

C#

```
int startPageIndex = 0;  
int endPageIndex = 3;  
float dpiX=200;  
float dpiY=200;  
BitmapSource[] images = pdfViewerControl.ExportAsImage(startPageIndex,  
endPageIndex, dpiX, dpiY);  
//Set up the output path  
string output = @"..\..\Output\Image";  
if (images != null)  
{  
    for (int i = 0; i < images.Length; i++)  
    {  
//Initialize the new Jpeg bitmap encoder  
        BitmapEncoder encoder = new JpegBitmapEncoder();  
//Create the bitmap frame using the bitmap source and add it to the encoder  
        encoder.Frames.Add(BitmapFrame.Create(images[i]));  
//Create the file stream for the output in the desired image format  
        FileStream stream = new FileStream(output + i.ToString() + ".Jpeg",  
        FileMode.Create);  
//Save the stream, so that the image will be generated in the output location  
        encoder.Save(stream);  
    }  
}
```

Note: The complete sample project of exporting the PDF pages to images using the Syncfusion PDF viewer is available in the [GitHub](#).

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Extract Text from PDF Files in WPF Pdf Viewer

PDF Viewer allows you to extract the text from a particular page or from the entire PDF file using the [ExtractText](#) methods of [PdfDocumentView](#).

Note: PDF Viewer uses PDFium as a default rendering engine to extract text from PDF files. Refer to this [link](#) for more details about the PDF rendering engines.

Extract text from a particular page

You can extract the text from a page using [ExtractText](#) method in [PdfDocumentView](#) class. The following code sample explains how to extract the text from the first page.

C#

```
using System.Windows;
using Syncfusion.Pdf;
using Syncfusion.Windows.PdfViewer;
namespace TextExtractionDemo
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        #region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Initialize the `PdfDocumentView` control.
            PdfDocumentView pdfDocumentView = new PdfDocumentView();
            //Load the PDF file.
            pdfDocumentView.Load(@"Sample.pdf");
            //Extract text from the first page.
            TextLines textLines = new TextLines();
            string extractedText = pdfDocumentView.ExtractText(0, out textLines);
        }
        #endregion
    }
}
```

Note: In this method, the text is extracted in the order in which it is written in the document stream and it may not be in the order in which it is viewed in the PDF reader application.

Extract text from an entire file

You can extract text from an entire file by using the following code sample.

C#

```
using System.Windows;
using Syncfusion.Pdf;
using Syncfusion.Windows.PdfViewer;
namespace TextExtractionDemo
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        #region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Initialize the `PdfDocumentView` control.
            PdfDocumentView pdfDocumentView = new PdfDocumentView();
```

```

//Load the PDF file.
pdfDocumentView.Load(@"Sample.pdf");
//Extract text from the file.
TextLines textLines = new TextLines();
string extractedText = string.Empty;
for (int i = 0; i < pdfDocumentView.PageCount; i++)
{
    extractedText += pdfDocumentView.ExtractText(i, out textLines);
}
}
#endregion
}
}

```

Extract text with bounds

Extract lines

You can get the text line by line along with the bounds using the [TextLines](#) property from the [ExtractText](#) method. Refer to the following code sample to perform the same.

C#

```

using System.Drawing;
using System.Windows;
using Syncfusion.Pdf;
using Syncfusion.Windows.PdfViewer;
namespace TextExtractionDemo
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        #region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Initialize the `PdfDocumentView` control.
            PdfDocumentView pdfDocumentView = new PdfDocumentView();
            //Load the PDF file.
            pdfDocumentView.Load(@"Sample.pdf");
            //Initialize the `TextLines`
            TextLines textLines = new TextLines();
            //Pass the `TextLines` as a parameter to the `ExtractText` method.
            pdfDocumentView.ExtractText(0, out textLines);
            //Gets specific line from the collection through the index.
            TextLine line = textLines[0];
            //Get text in the line.
            string text = line.Text;
            //Get bounds of the line.
            RectangleF lineBounds = line.Bounds;
        }
        #endregion
    }
}

```

Extract words

You can get the words in a line along with the bounds using the [WordCollection](#) property of the [TextLine](#) using [ExtractText](#) method. Refer to the following code sample to perform the same.

C#

```
using System.Collections.Generic;
using System.Drawing;
using System.Windows;
using Syncfusion.Pdf;
using Syncfusion.Windows.PdfViewer;
namespace TextExtractionDemo
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        #region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Initialize the `PdfDocumentView` control.
            PdfDocumentView pdfDocumentView = new PdfDocumentView();
            //Load the PDF file.
            pdfDocumentView.Load(@"Sample.pdf");
            //Initialize the `TextLines`
            TextLines textLines = new TextLines();
            //Pass the `TextLines` as a parameter to the `ExtractText` method.
            pdfDocumentView.ExtractText(0, out textLines);
            //Gets specific line from the collection through the index.
            TextLine line = textLines[0];
            //Get the word collection in a line.
            List<TextWord> wordCollection= line.WordCollection;
            //Get the word
            string word = wordCollection[0].Text;
            //Get the bounds of the word
            RectangleF bounds= wordCollection[0].Bounds;
        }
        #endregion
    }
}
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Form Filling in PDF Files in WPF Pdf Viewer

PDF Viewer provides the ability to Fill, Edit, Flatten, and Save the **AcroForm** fields in PDF files.

Supported form fields

You can load and fill the following form fields in a PDF document using the PDF Viewer.

1. Text box. 2. Password box. 3. Checkbox. 4. Radio button. 5. Combo box. 6. List box.

Retrieve the details of text box form field

You can retrieve the details of a text box form field through the [FormFieldClicked](#) event of [PdfViewerControl](#) by simply clicking on the field. The [FormField](#) property of the [FormFieldClickedEventArgs](#) needs to be typecast to the [TextBox](#) control as shown in the following code.

C#

```
//Wire the 'FormFieldClicked' event.
pdfViewer.FormFieldClicked += PdfViewer_FormFieldClicked;
private void PdfViewer_FormFieldClicked(object sender,
FormFieldClickedEventArgs args)
{
    //Typecast the 'FormField' property to 'System.Windows.Controls.TextBox'
    TextBox textBox = args.FormField as TextBox;
    //Retrive the 'Text' property
    string text = textBox.Text;
    //{Insert your code here}
}
```

Note: The sample project of PDF form filling using the Syncfusion PDF Viewer is available in the [GitHub](#).

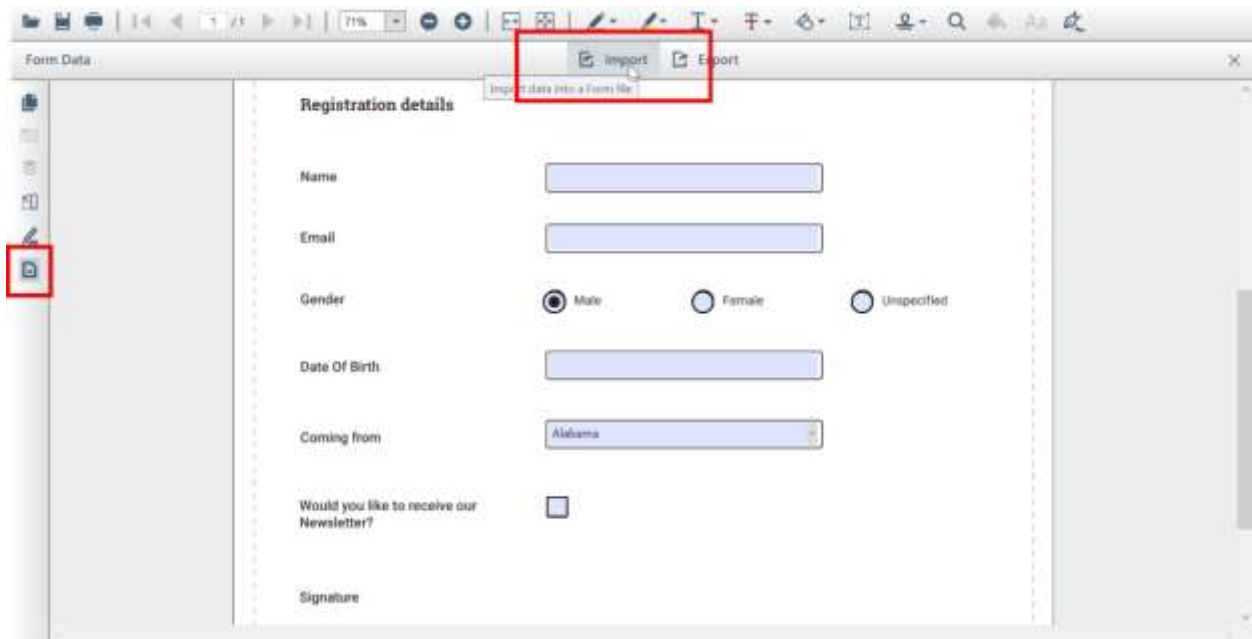
Import and Export Form Data

In WPF, the PDF viewer allows the users to import and export form data to and from the PDF documents.

Import Form Data

Follow the below steps to import date to PDF document with **AcroForm**.

1. Click the form data tool button in the left pane, the form data toolbar will appear as a secondary toolbar in the [PdfViewerControl](#). 2. Select **Import** option in form data toolbar to import the PDF form data.



The following code shows how to import form data in code behind.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Import PDF form data
    pdfviewer.ImportFormData("Import.fdf",
    Syncfusion.Pdf.Parsing.DataFormat.Fdf);
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Import PDF form data
    pdfviewer.ImportFormData("Import.fdf",
    Syncfusion.Pdf.Parsing.DataFormat.Fdf)
End Sub
```

Export Form Data

Follow the below steps to export data from PDF document

1. Select **Export** option in the form data toolbar, to save the completed PDF form data as a file in another file format.
2. In Export Form Data As dialog box, you can select the desired format to save the form data (FDF, XFDF, XML, and JSON).

Note: If the PDF document is loaded as a stream, the [PdfViewerControl](#) will request for the form name when exporting.

The following code shows how to export form data in code behind.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Export PDF form data
    pdfviewer.ExportFormData("Export.fdf",
    Syncfusion.Pdf.Parsing.DataFormat.Fdf, "SourceForm.pdf");
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Export PDF form data
    pdfviewer.ExportFormData("Export.fdf",
    Syncfusion.Pdf.Parsing.DataFormat.Fdf, "SourceForm.pdf")
End Sub
```

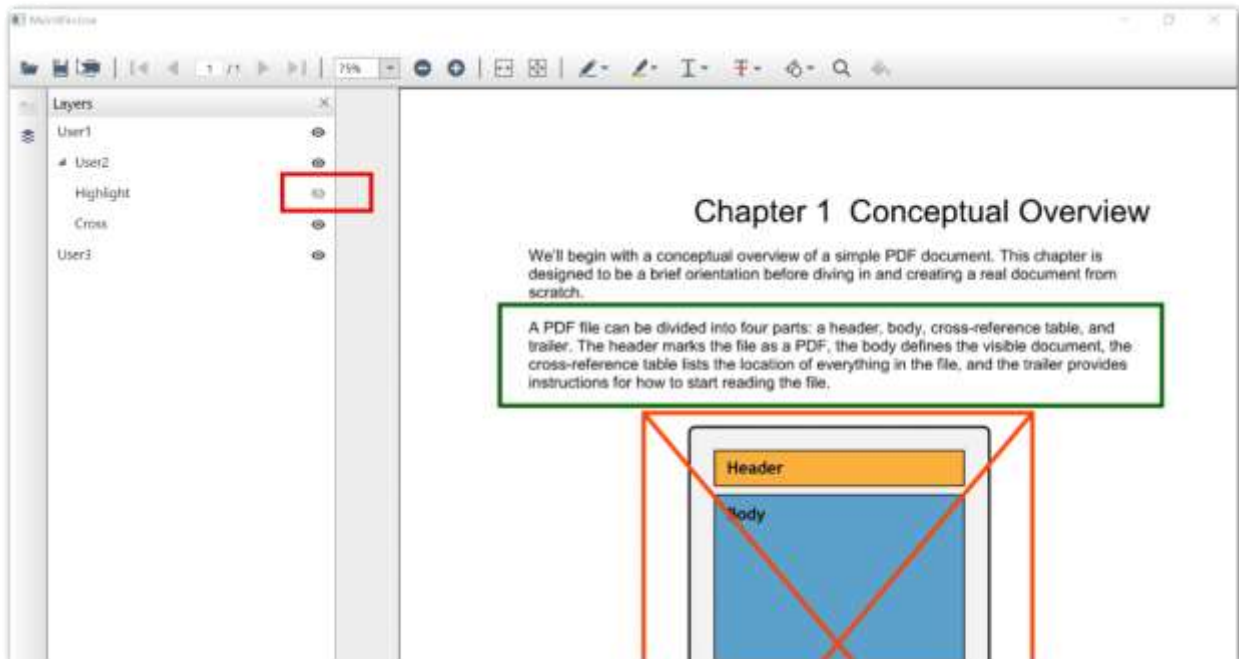
Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Working with PDF Layers in WPF Pdf Viewer

The layer support in PDF viewer allows users to toggle the visibility of individual and group of layers in the PDF document to view, print, save, and export as image.

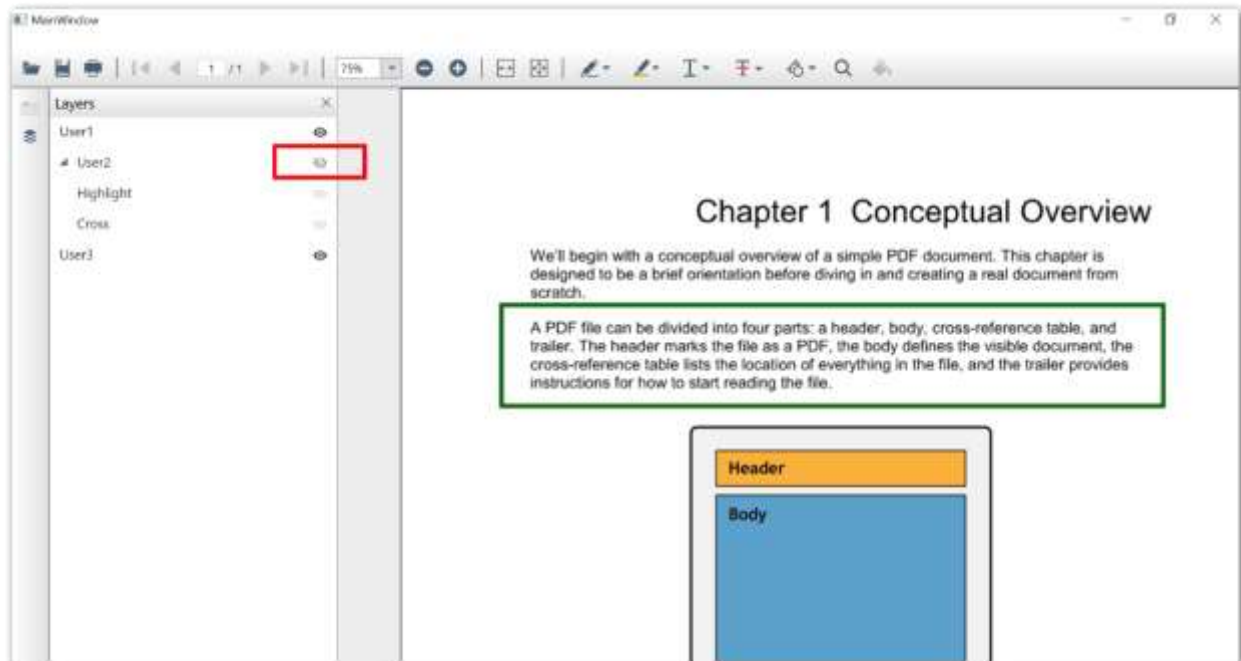
Toggleing the visibility of a PDF layer

To toggle the visibility of PDF layers individually, click the eye icon associated with each layer in the layers pane.



Toggleing the visibility of the group of layers

To toggle the visibility of a group of PDF layers, click the eye icon associated with parent layer in the layers pane.



Disabling the layers

You can disable the display of the layers present in the PDF document by setting the `EnableLayers` property to `false`. Refer to the following code example.

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    PdfLoadedDocument pdf = new PdfLoadedDocument("PdfLayers.pdf");
    pdfViewer.Load(pdf);
    pdfViewer.EnableLayers = false;
}
```

VB.NET

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    Dim pdf As New PdfLoadedDocument("PdfLayers.pdf")
    pdfViewer.Load(pdf)
    pdfViewer.EnableLayers = false;
End Sub
```

You can also achieve the same in XAML using the `DependencyProperty` illustrated as follows.

XML

```
<syncfusion:PdfViewerControl x:Name="pdfViewer" EnableLayers="False" />
```

Note: By default, the layer feature is enabled in PDF viewer.

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Redaction in WPF Pdf Viewer

Redaction support allows you to remove sensitive/confidential information in text, images, and graphics from a PDF document.

Enable redaction mode

The following code shows how to switch to redaction mode in code behind.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Enables redaction mode
    pdfviewer.PageRedactor.EnableRedactionMode = true;
}
```

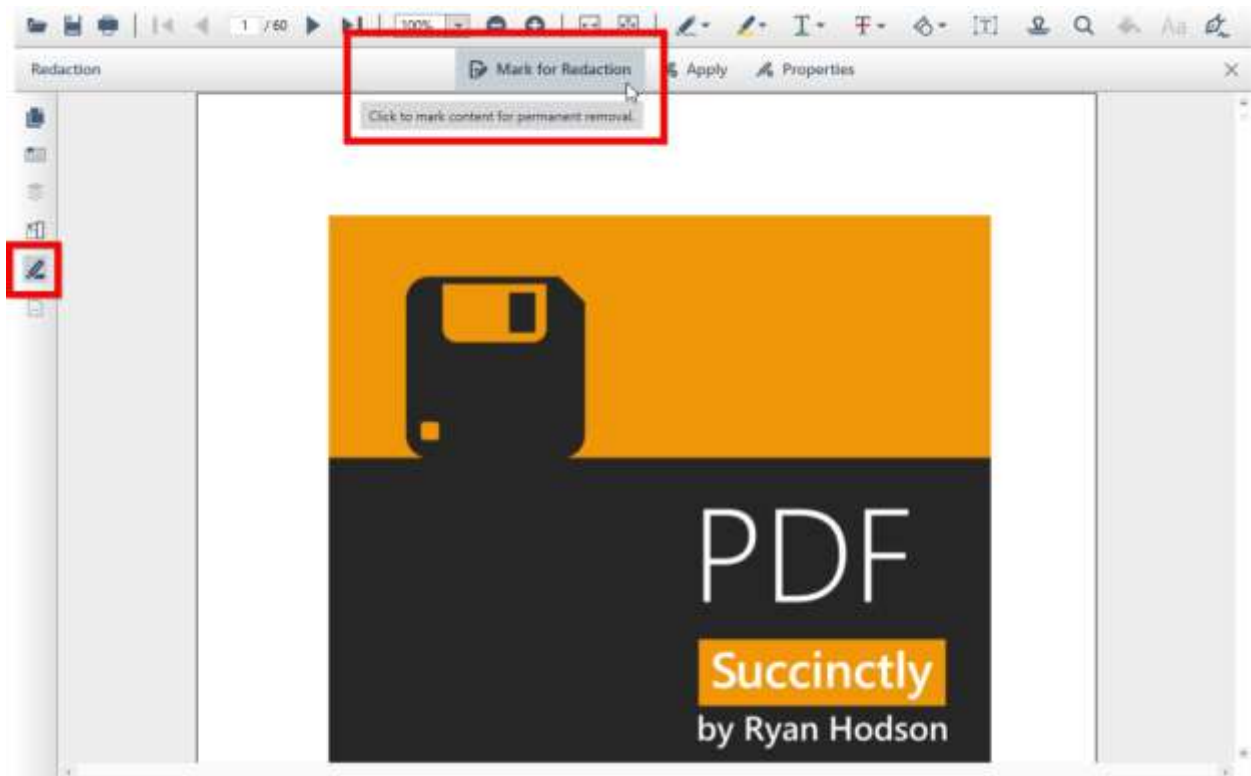
VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Enables redaction mode
    pdfviewer.PageRedactor.EnableRedactionMode = true
End Sub
```

Mark region to redact

Use the following steps to perform redaction in the PDF document page:

1. Click the redaction tool button in the left pane, this displays the redaction toolbar as a secondary toolbar in the PdfViewerControl.
2. Select **Mark for Redaction** from redaction toolbar to select the region or text to be redacted



3. Use the mouse pointer to select the region or the text to be redacted. This operation will create a mark in the region selected. 4. Choose Apply in the redaction toolbar to redact the marked region.

Note: Undo and redo can only be performed before applying the redaction, these operations cannot be performed after applying redaction.

Mark regions for redaction without UI interaction

You can also mark regions for redaction without UI interaction from the code behind using the `MarkRegions` method. Refer to the following code to mark the regions by passing the page index and the list of rectangle regions to be marked in the particular page for redaction.

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    List<RectangleF> rectangles = new List<RectangleF>();
    rectangles = new List<RectangleF>() { new RectangleF(100, 100, 100, 100) };
    // Appends regions to the existing marked regions, marking regions if it
    // does not already exist.
    pdfViewerControl.PageRedactor.MarkRegions(0, rectangles, false);
    // Enable the redaction mode.
    pdfViewerControl.PageRedactor.EnableRedactionMode = true;
}
```

Note: You can overwrite the existing marked regions with the new regions, By passing the value `true` for the `clearExisting` parameter of `MarkRegions` method.

Customizing redaction appearance

Choosing **Properties** from the redaction toolbar will show the default Redaction Tool Properties window, which contain two tabs:

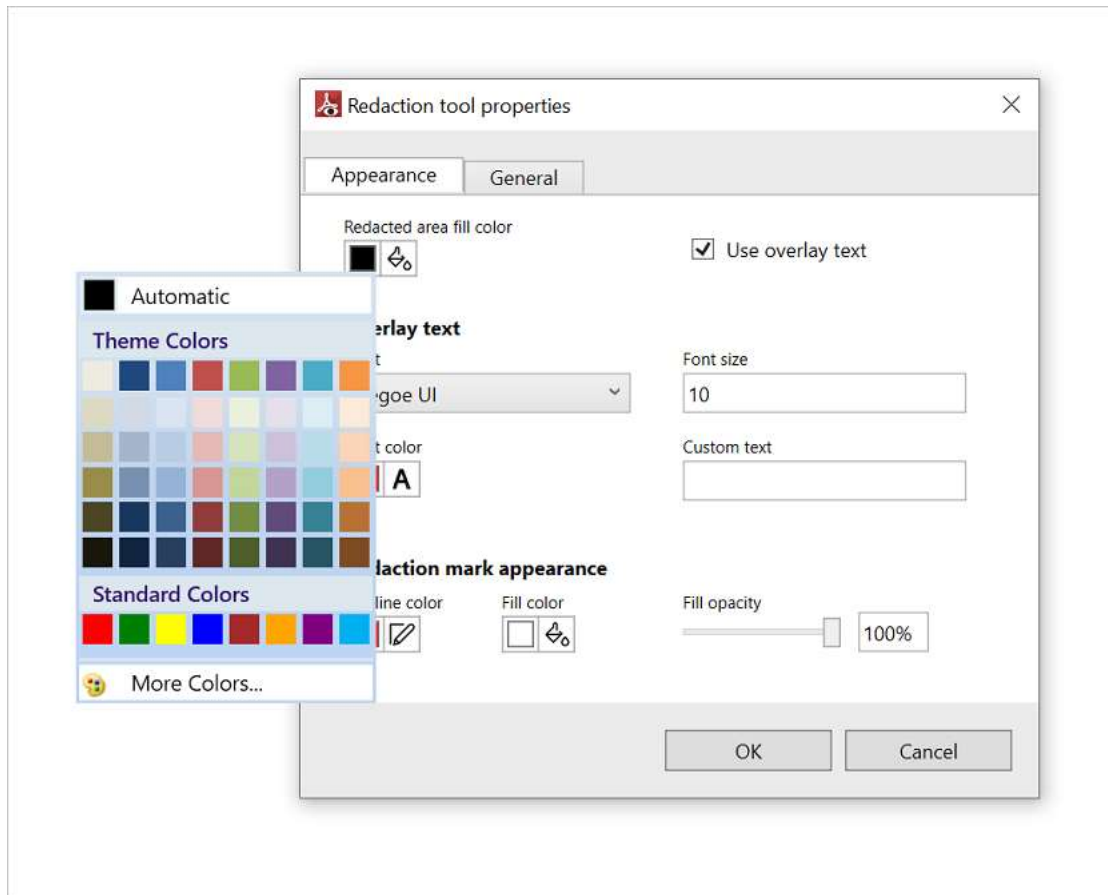
- Appearance
- General

Appearance tab

The redacted area fill color, overlay text, and redaction mark area appearances can be customized using the options in the Appearance tab.

Redaction appearance

The following image illustrates how to change the fill color of the redacted area.



The following code shows how to change the redacted area fill color from code behind.

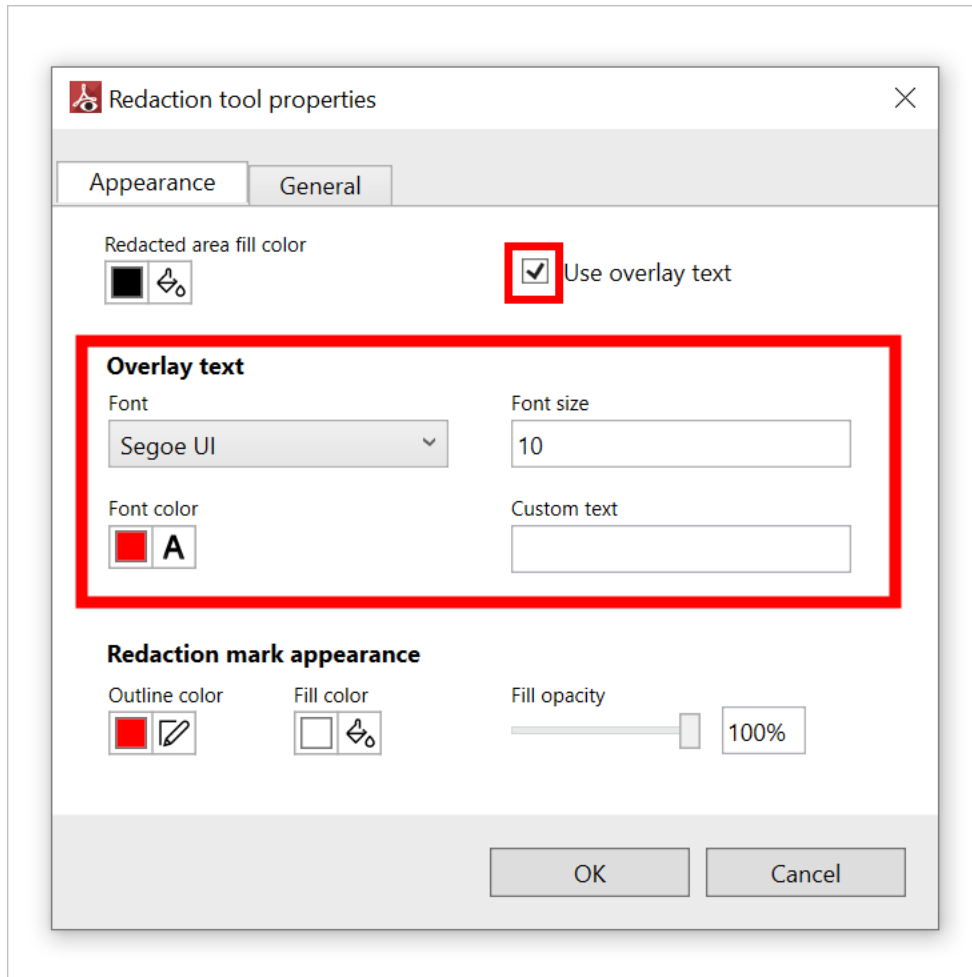
C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Sets the redacted area fill color
    pdfviewer.RedactionSettings.FillColor = Colors.AliceBlue;
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Sets the redacted area fill color
    pdfviewer.RedactionSettings.FillColor = Colors.AliceBlue
End Sub
```

The following image illustrates how to change the overlay text appearance of the redacted area. You can disable the overlay text appearance by unchecking the **Use Overlay Text** checkbox.



The following code shows how to customize the overlay text appearance from code behind.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Set the overlay text
    pdfviewer.RedactionSettings.OverlayText = "overlay text";
    //Sets the overlay text font color
    pdfviewer.RedactionSettings.FontColor = Colors.Blue;
    //Sets the overlay text font family
    pdfviewer.RedactionSettings.FontFamily = new
    System.Windows.Media.FontFamily("Times new Roman");
    //Sets the overlay text font size
    pdfviewer.RedactionSettings.FontSize = 17;
    //Enable the overlay text appearance
    pdfviewer.RedactionSettings.UseOverlayText = true;
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Set the overlay text
    pdfviewer.RedactionSettings.OverlayText = "overlay text"
```

```

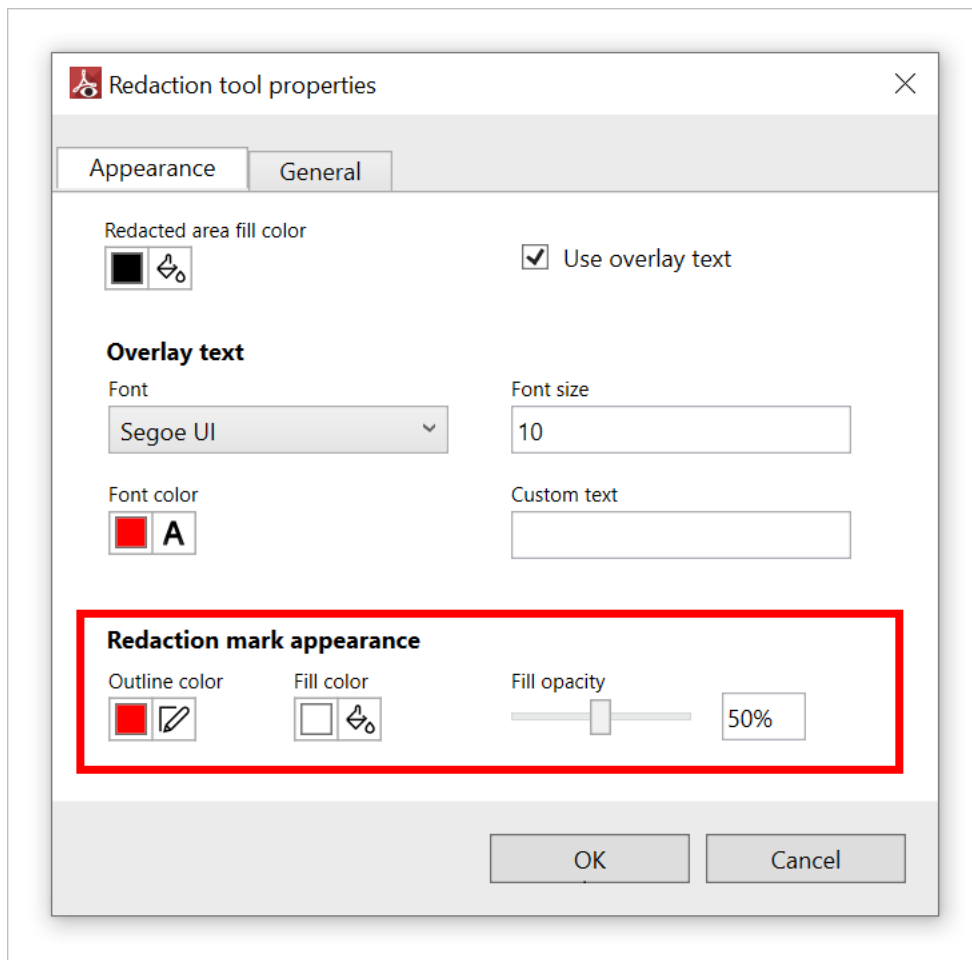
'Sets the overlay text font color
pdfviewer.RedactionSettings.FontColor = Colors.Blue
'Sets the overlay text font family
pdfviewer.RedactionSettings.FontFamily = new
System.Windows.Media.FontFamily("Times new Roman")
'Sets the overlay text font size
pdfviewer.RedactionSettings.FontSize = 17
'Enable the overlay text appearance
pdfviewer.RedactionSettings.UseOverlayText = true
End Sub

```

Marked redaction appearance

The marked redaction appearance represents the appearance of the region or text marked for redaction before the redaction is being applied. The fill color, border color, and fill color opacity of the redaction marked area can also be customized.

The following image illustrates how to customize the appearance of the redaction marked area.



The following code shows how to customize the appearance of mark for redaction area from code behind.

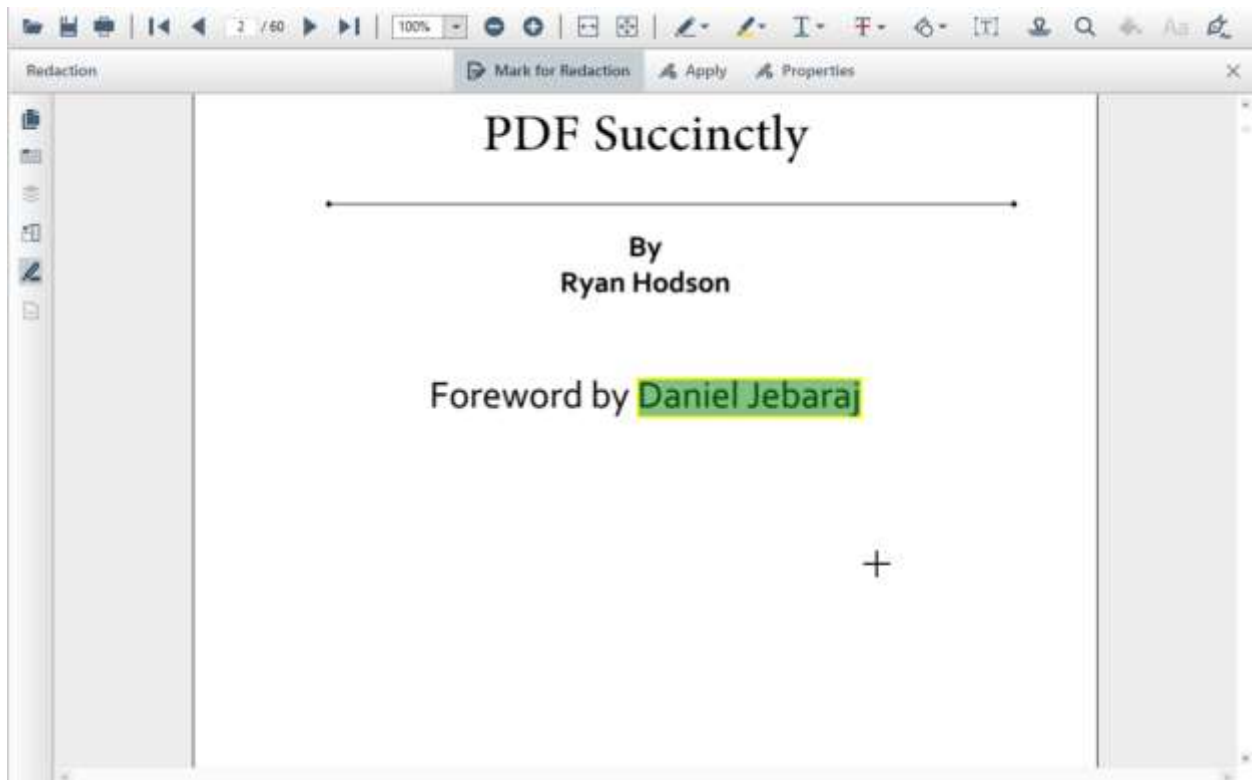
C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Sets the redact mark area fill color
    pdfviewer.RedactionSettings.MarkAppearance.FillColor = Colors.Green;
    //Sets the redact mark area fill color opacity
    pdfviewer.RedactionSettings.MarkAppearance.FillOpacity = 0.5f;
    //Sets the redact mark area outline color
    pdfviewer.RedactionSettings.MarkAppearance.OutlineColor = Colors.Yellow;
}
```

VB.NET

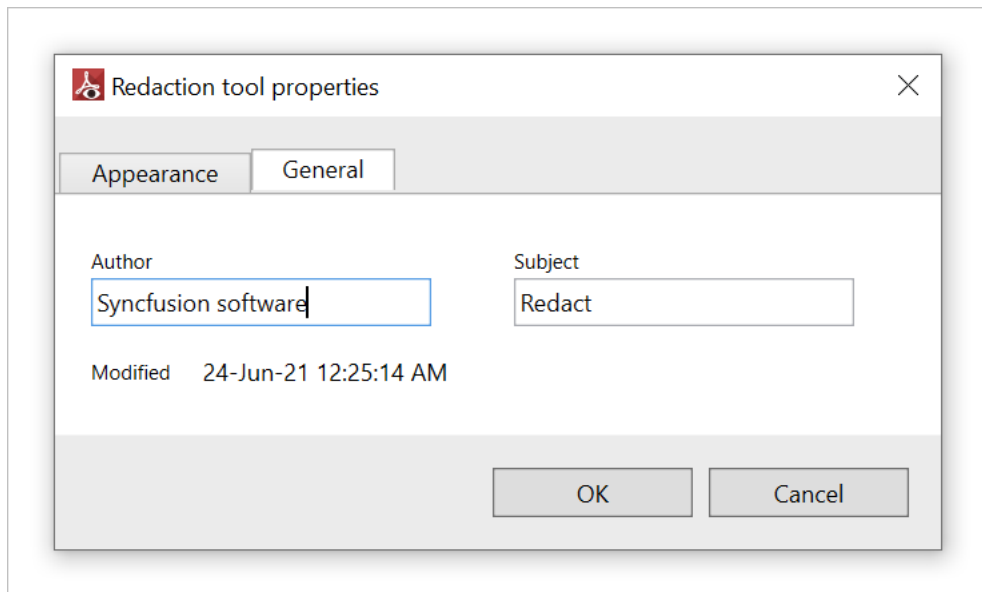
```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Sets the redact mark area fill color
    pdfviewer.RedactionSettings.MarkAppearance.FillColor = Colors.Green
    'Sets the redact mark area fill color opacity
    pdfviewer.RedactionSettings.MarkAppearance.FillOpacity = 0.5f
    'Sets the redact mark area outline color
    pdfviewer.RedactionSettings.MarkAppearance.OutlineColor = Colors.Yellow
End Sub
```

The following image illustrates the appearance changes in redaction marked region.

**General tab**

You can add or edit the Author and Subject of the redaction using General tab of the Redaction Tool Properties window.

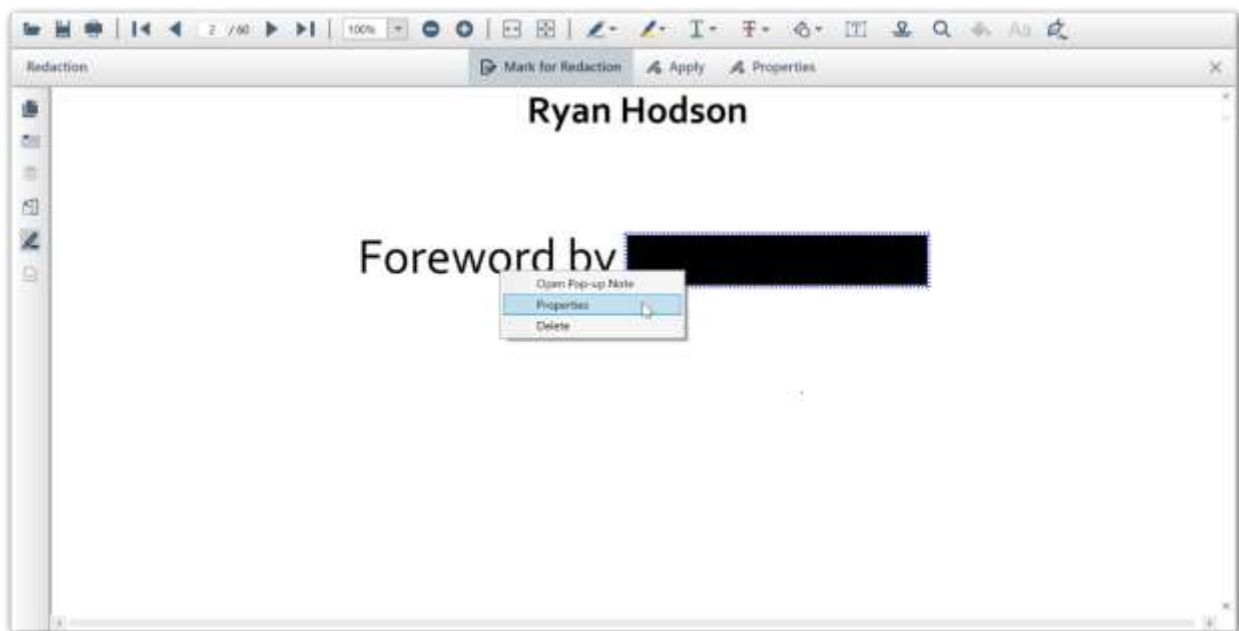
The following image illustrates the customization of Author and Subject of the redaction.



Appearance of the included redaction

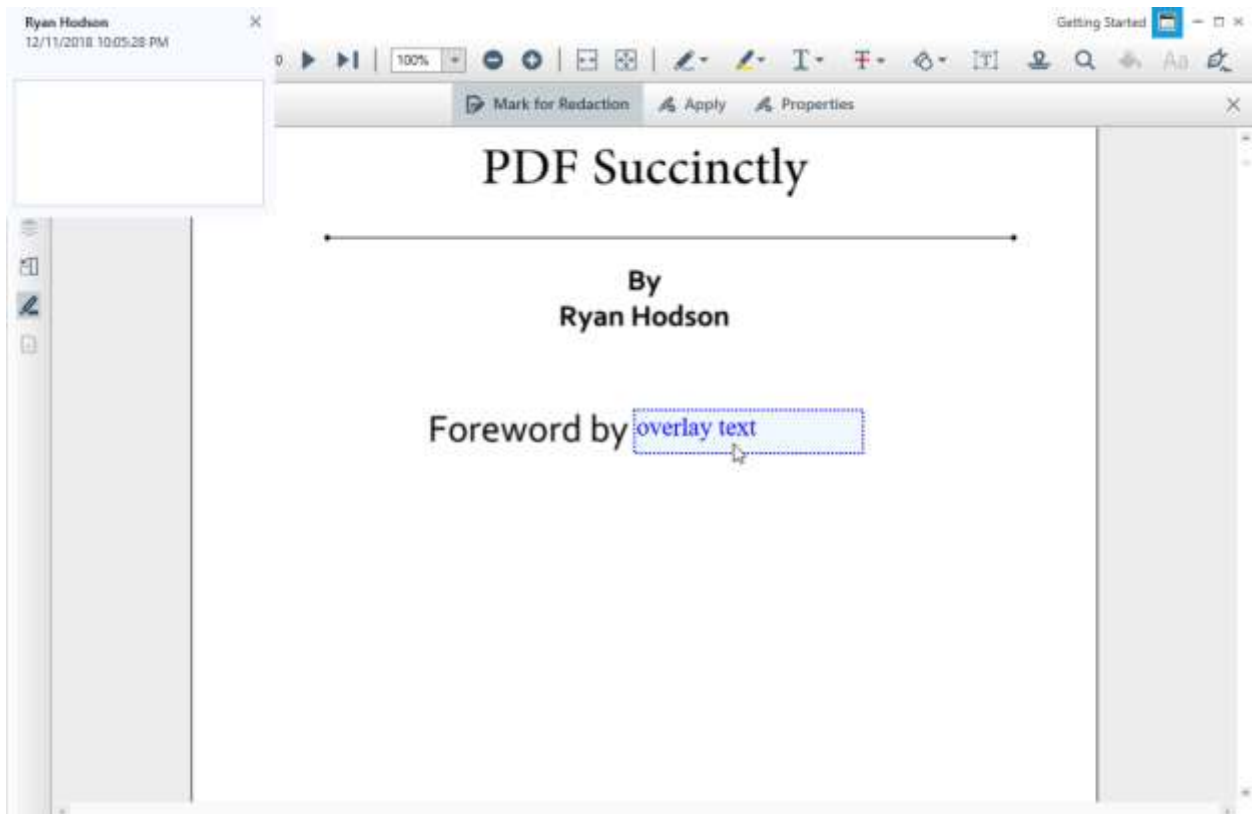
Selecting the properties option from the context menu that is displayed when right-clicking the selected redaction will show the Redaction Tool Properties window of selected redaction.

The following image illustrates how to customize the appearance of the included redaction.



Open pop-ups

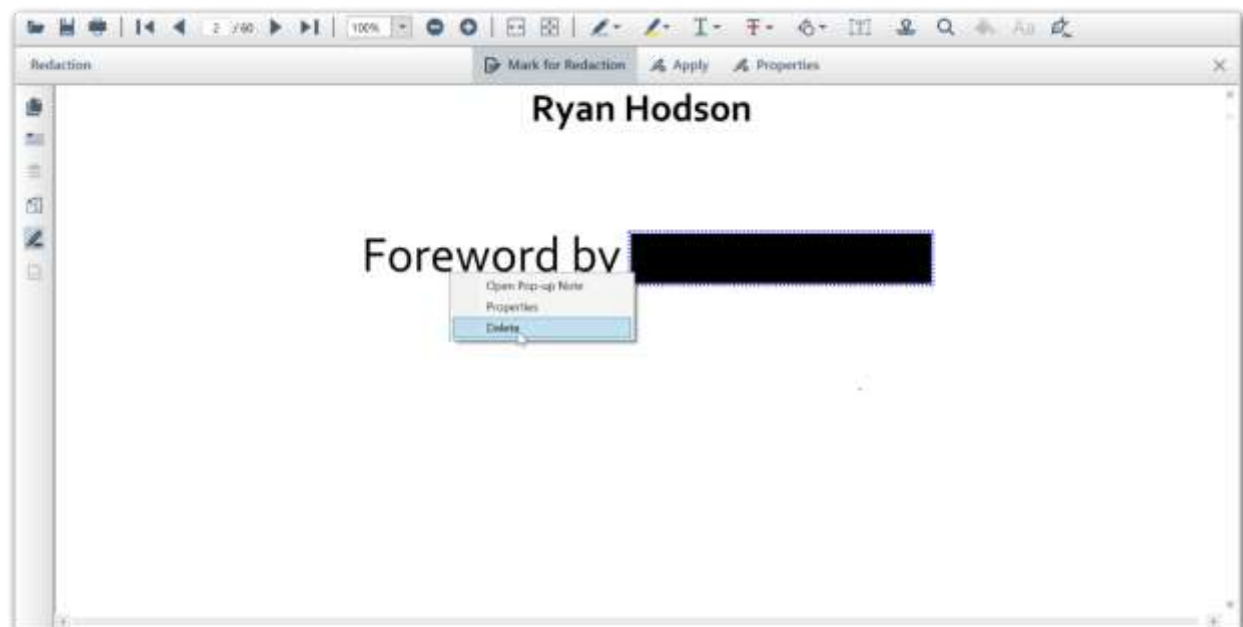
Notes can be added to the redaction marked region by choosing Open Pop-up note option from the context menu. The following image illustrates the notes added to the redaction marked region.



Delete redaction marked region

Select the delete option from the context menu that is displayed when right-clicking the selected redaction marked region will delete the respective redaction from the PDF document.

The following image illustrates how to delete the included redaction from the PDF document.



Events

Redaction applied notification

The `RedactionApplied` event notifies you when the marked regions of the page(s) are redacted. It provides the details of the redacted region through the `RedactionEventArgs`. The following code shows how to wire the event in [PdfViewerControl](#).

C#

```
using Syncfusion.Windows.PdfViewer;
using System.Collections.Generic;
using System.Windows;
namespace RedactionAppliedEventDemo
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            pdfViewerControl.PageRedactor.RedactionApplied +=
            PageRedactor_RedactionApplied;
            pdfViewerControl.Load(@"../../Data/PDF_Succinctly.pdf");
        }
        private void PageRedactor_RedactionApplied(object sender,
            Syncfusion.Windows.PdfViewer.RedactionEventArgs args)
        {
            //Collection of redacted regions for pdf pages
            List<RedactionMark> RedactedMarks = args.Marks;
            foreach (RedactionMark redactionMark in RedactedMarks)
            {
                //Bounds of the redacted region
                System.Drawing.RectangleF RedactedRegion = redactionMark.Bounds;
                //Brush color used to fill the redacted region
                System.Windows.Media.Color Fillcolor = redactionMark.Fill;
                //Page index of the redaction mark
                int pageIndex = redactionMark.PageIndex;
            }
        }
    }
}
```

Keyboard shortcuts

The following keyboard shortcuts are available to customize the marked redaction in the PDF document:

- Delete key: Deletes the selected mark redaction from the PDF document.
- Ctrl + Z: Performs undo functionality for recently performed operations.
- Ctrl + Y: Performs redo functionality for recently performed operations.

Note: All these operations can only be performed before applying the redaction.

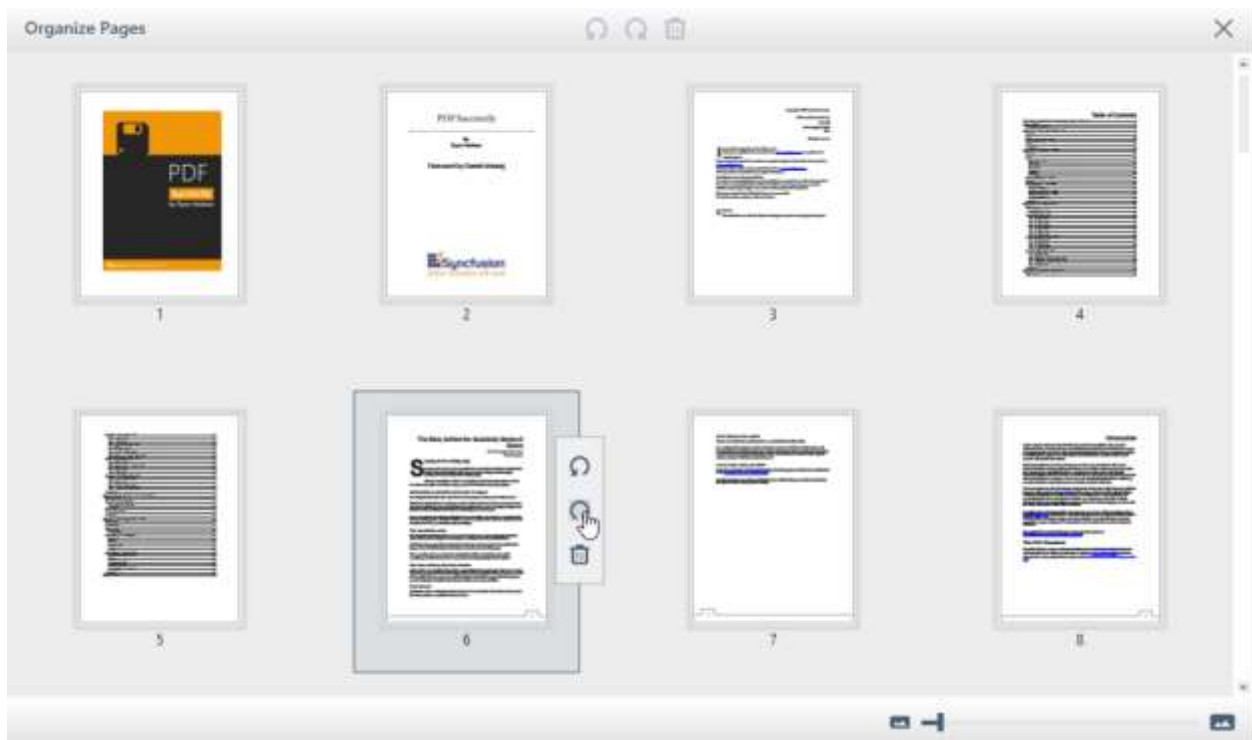
Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Organize Pages in WPF Pdf Viewer

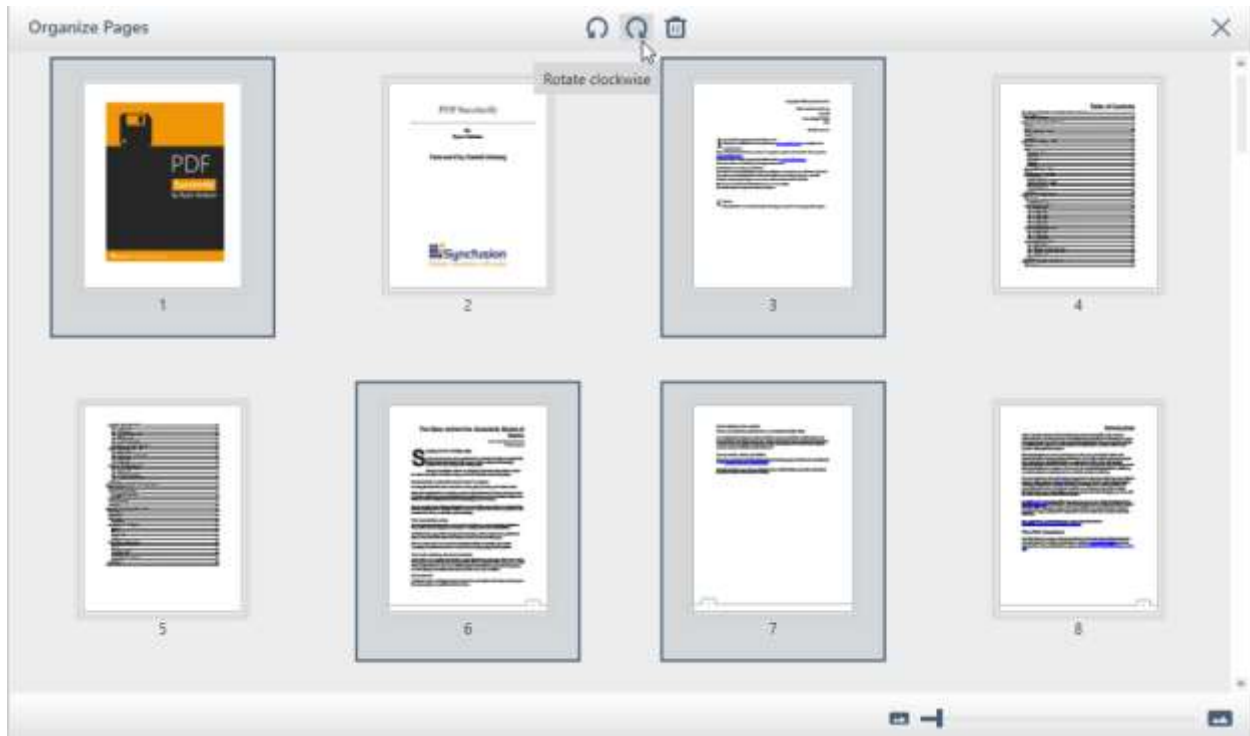
Organize pages support allows you to rotate, rearrange, and delete pages from a PDF document using a miniature preview of the PDF pages.

Use the following steps to organize the PDF page(s) in [PdfViewerControl](#):

1. Click the organize page button in the left pane, this displays the organize pages pane in the [PdfViewerControl](#).
2. You can rotate or delete a specific page using context menu that appears when hovering the mouse over the pages.

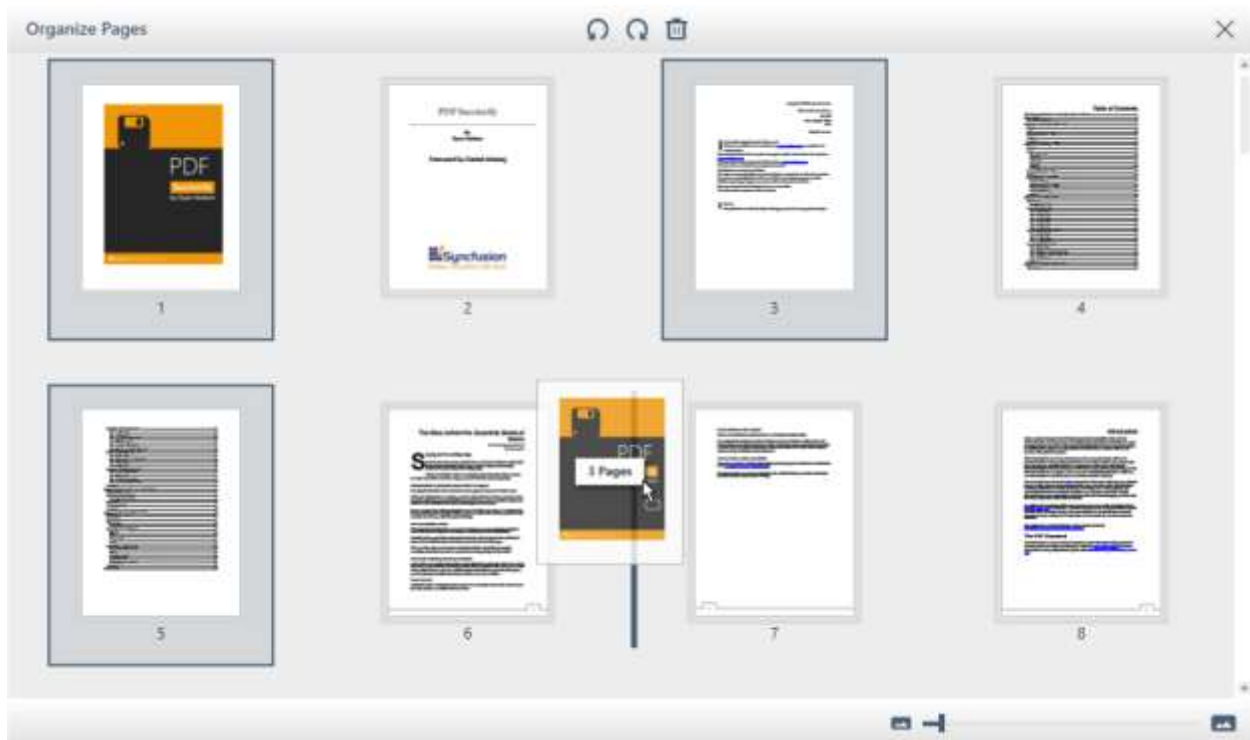


3. You can rotate or delete multiple pages using the organize pages toolbar.



Note: You can use Ctrl/Shift keys to select multiple pages. Also, you can select all pages using Ctrl+A shortcut key. You cannot delete all the pages from the document.

4. You can rearrange the page(s) by dragging and dropping them.



Rotating PDF page(s)

You can rotate PDF page(s) in clockwise, anti-clockwise or to a specific angle using the [Rotate](#) method. Refer to the following code example to rotate a page from code behind.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Rotating the pages with index 0 and 1 to 90 degree
    pdfviewer.PageOrganizer.Rotate(new
    int[] {0,1}, Syncfusion.Pdf.PdfPageRotateAngle.RotateAngle90);
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Rotating the pages with index 0 and 1 to 90 degree
    pdfviewer.PageOrganizer.Rotate(new int[] {0,1},
    Syncfusion.Pdf.PdfPageRotateAngle.RotateAngle90)
End Sub
```

Rotating PDF page(s) Clockwise

You can rotate PDF page(s) clockwise using the [PageOrganizer.RotateClockwise](#) method.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Rotating the pages with index 0 and 1 in clockwise
    pdfviewer.PageOrganizer.RotateClockwise(new int[] { 0, 1 });
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Rotating the pages with index 0 and 1 in clockwise
    pdfviewer.PageOrganizer.RotateClockwise(new int[] { 0, 1 })
End Sub
```

Rotating PDF page(s) Counterclockwise

You can rotate PDF page(s) in counterclockwise using the [PageOrganizer.RotateCounterclockwise](#) method.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Rotating the pages with index 0 and 1 in counterclockwise
    pdfviewer.PageOrganizer.RotateCounterclockwise (new int[] { 0, 1 });
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
```

```
'Rotating the pages with index 0 and 1 in counterclockwise
pdfviewer.PageOrganizer.RotateCounterclockwise (new int[] { 0, 1 })
End Sub
```

Rotating page(s) using command

You can rotate the specific range of pages using the [RotatePagesCommand](#) of [PdfViewerControl](#). The following code shows how to rotate pages by executing the command with the index of the pages to be rotated and the angle to which it is rotated as command parameter. In this example the pages at the index 0 and 1, are rotated through 180 degrees.

C#

```
pdfViewerControl.PageOrganizer.RotatePagesCommand.Execute(new object[] { new
int[] { 0, 1 }, PdfPageRotateAngle.RotateAngle180 });
```

Similarly, you can rotate the specific range of pages 90 degrees clockwise and counterclockwise with respect to the current angle using the [RotatePagesClockwiseCommand](#) and [RotatePagesCounterclockwiseCommand](#) of [PdfViewerControl](#). The following codes shows how to rotate pages clockwise and counterclockwise respectively, with the index of the pages to be rotated as command parameter.

C#

```
pdfViewerControl.PageOrganizer.RotatePagesClockwiseCommand.Execute( new
int[] { 0, 1 } );
```

C#

```
pdfViewerControl.PageOrganizer.RotatePagesCounterclockwiseCommand.Execute(
new int[] { 0, 1 } );
```

Acquiring page rotation

You can get the rotation angle of a PDF page in terms of [PdfPageRotateAngle](#) using [PageOrganizer.GetPageRotation\(\)](#) method.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Gets rotation angle of page with index 0
    PdfPageRotateAngle rotatedAngle =
    pdfviewer.PageOrganizer.GetPageRotation(0);
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Gets rotation angle of page with index 0
    Dim rotatedAngle As PdfPageRotateAngle =
    pdfviewer.PageOrganizer.GetPageRotation(0)
End Sub
```

Rearranging PDF pages

You can rearrange the existing PDF document pages using the [Rearrange\(int\[\]\)](#) method. This method uses zero based start index.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Rearrange the page by index
    pdfviewer.PageOrganizer.Rearrange(new int[] { 1, 0 });
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Rearrange the page by index
    pdfviewer.PageOrganizer.Rearrange(new int[] { 1, 0 })
End Sub
```

Note: If any of the already existing page index is not present in rearranged array, then that page will be removed.

Removing PDF page(s)

You can remove the PDF page(s) from the PDF document using the RemoveAt method in PageOrganizer.

To remove the page at specific index from PDF document, refer to the following code example.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Remove a page at index 0
    pdfviewer.PageOrganizer.RemoveAt(0);
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Remove a page at index 0
    pdfviewer.PageOrganizer.RemoveAt(0)
End Sub
```

To remove the set of pages from PDF document, refer to the following code example.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Removing the pages at index 0 and 1
    pdfviewer.PageOrganizer.RemovePages(new int[] { 0, 1 });
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
```

```
'Removing the pages at index 0 and 1
pdfviewer.PageOrganizer.RemovePages(new int[] { 0, 1 })
End Sub
```

Remove page(s) using command

You can remove the specific range of pages using the [RemovePagesCommand](#) of [PdfViewerControl](#). The following code shows how to remove pages by executing the command with the index of the pages as command parameter.

C#

```
pdfviewerControl.PageOrganizer.RemovePagesCommand.Execute(new int[] { 0, 1
});
```

Get the selected page indexes

You can get the selected page indexes of the PDF document in the organizing pages window. The [PageSelected](#) event indicates that a page(s) is selected and the [SelectedPages](#) property of the [PageSelectedEventArgs](#) provides you the index of the pages that are currently selected. The following code shows how to wire the event in [PdfViewerControl](#).

C#

```
// Hook the page selected event.
pdfviewer.PageSelected += PdfViewer_PageSelected;
private void PdfViewer_PageSelected(object sender, PageSelectedEventArgs e)
{
    // Get the selected pages.
    if (e.SelectedPages.Length != 0)
        selectedPages = e.SelectedPages;
}
```

Disabling page organizer

You can remove the page organizer icon from left pane of the 'PdfViewerControl' by setting the 'PageOrganizerSettings.IsIconVisible' property to false.

Refer to the following code example to disable the organize page.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Removing the page organizer icon from left pane.
    pdfviewer.PageOrganizerSettings.IsIconVisible = false;
}
```

VB.NET

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    'Removing the page organizer icon from left pane.
    pdfviewer.PageOrganizerSettings.IsIconVisible = false
End Sub
```


Note: The sample projects for organizing pages using the Syncfusion PDF Viewer are available in the [GitHub](#).

Keyboard shortcuts

The following keyboard shortcuts are available at miniature preview mode to organize the PDF pages:

- Delete key: Deletes the selected page from the PDF document.
- Ctrl+A: Selects all pages in the PDF document.
- Shift+Arrow/MouseButton: Selects a set of consecutive pages in the PDF document.
- Ctrl+MouseButton: Selects the clicked pages in the PDF document.

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Handwritten Signature in WPF Pdf Viewer

PDF viewer WPF allows the user to include handwritten signature in the PDF document and provides options to edit or remove the included handwritten signature in the PDF document.

The following code shows how to switch to handwritten signature mode in code behind.

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    PdfLoadedDocument pdf = new PdfLoadedDocument("Input.pdf");
    pdfviewer.Load(pdf);
    pdfviewer.AnnotationMode =
    PdfDocumentView.PdfViewerAnnotationMode.HandwrittenSignature;
}
```

VB.NET

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    Dim pdf As New PdfLoadedDocument("Input.pdf")
    pdfviewer.Load(pdf)
    pdfviewer.AnnotationMode =
    PdfDocumentView.PdfViewerAnnotationMode.HandwrittenSignature
End Sub
```

Moving to **HandwrittenSignature** mode will open the signature pad, requesting the user to draw the signature. Clicking on apply will add the signature in the PDF Document, this can then be resized and moved to appropriate location.



How to set the opacity of the handwritten signature?

The opacity of the handwritten signature can be customized at the time of inclusion itself. The following code shows how to set opacity value of the handwritten signature to be included.

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    PdfLoadedDocument pdf = new PdfLoadedDocument("Input.pdf");
    pdfviewer.Load(pdf);
    pdfviewer.HandwrittenSignatureSettings.Opacity = 0.5F;
}
```

VB.NET

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    Dim pdf As New PdfLoadedDocument("Input.pdf")
    pdfviewer.Load(pdf)
    pdfviewer.HandwrittenSignatureSettings.Opacity = 0.5F
End Sub
```

How to set the color of the handwritten signature?

The color of the handwritten signature can be customized at the time of inclusion itself.

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    PdfLoadedDocument pdf = new PdfLoadedDocument("Input.pdf");
    pdfviewer.Load(pdf);
    pdfviewer.HandwrittenSignatureSettings.Color = Colors.Green;
}
```

VB.NET

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    Dim pdf As New PdfLoadedDocument("Input.pdf")
    pdfviewer.Load(pdf)
    pdfviewer.HandwrittenSignatureSettings.Color = Colors.Green
End Sub
```

How to flatten handwritten signature on save?

The included handwritten signature can be flattened in the PDF document when saving it, setting `FlattenSignatureOnSave` Boolean to true in `HandwrittenSignatureSettings` will do this.

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    PdfLoadedDocument pdf = new PdfLoadedDocument("Input.pdf");
    pdfviewer.Load(pdf);
    pdfviewer.HandwrittenSignatureSettings.FlattenSignatureOnSave = true;
}
```

VB.NET

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    Dim pdf As New PdfLoadedDocument("Input.pdf")
    pdfViewer.Load(pdf)
    pdfviewer.HandwrittenSignatureSettings.FlattenSignatureOnSave = true
End Sub
```

Note: If the handwritten signature is not flattened, it will be saved as ink annotation in the PDF document.

Working with included handwritten signatures

The PDF viewer allows editing the color and opacity of the Handwritten signature included in the document in the UI.. To perform this, select the included handwritten signature and click right using mouse, over the selected handwritten signature, a pop-up context menu will appear with the following options,

Properties Delete

Properties

Selecting properties from the context menu will display the Handwritten Signature Properties window, which would have Appearance tab.

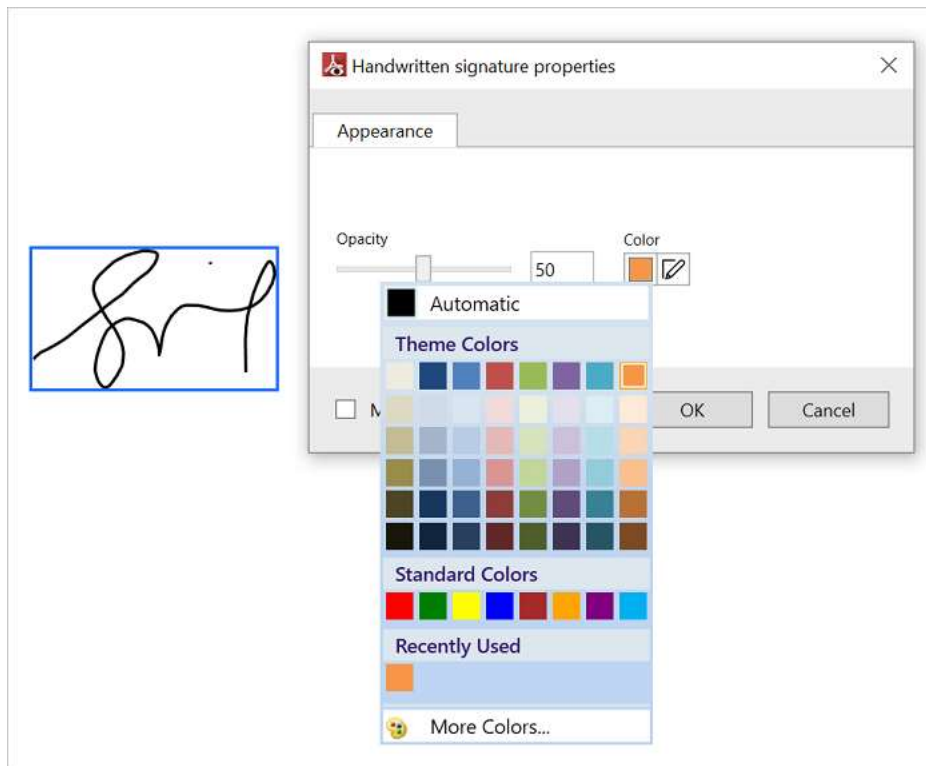
Appearance tab

The color, opacity of the handwritten signature can be edited using Appearance tab of Handwritten Signature Properties window.

Editing color of the signature

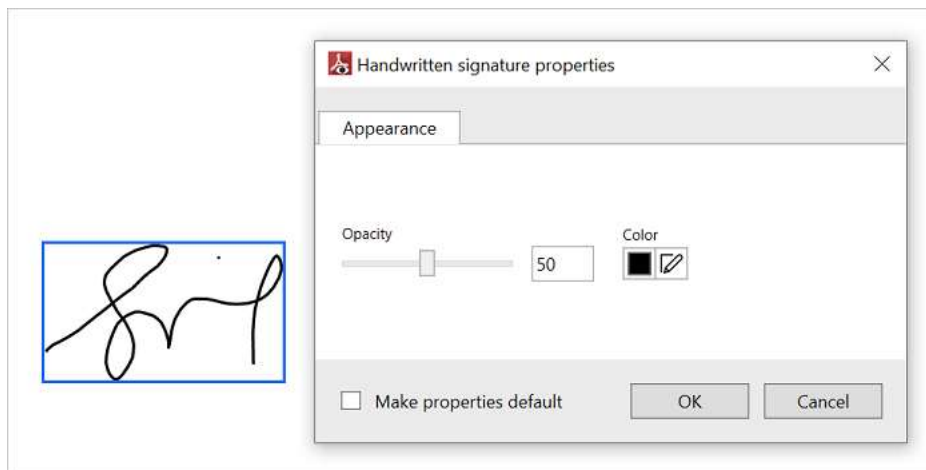
The color of the selected handwritten signature will be displayed in the color row in the appearance tab. Selecting the Color would displays the color palette, choosing a color from the color palette and clicking OK will apply the color to the handwritten signature.

The following image illustrates how to change the color of the handwritten signature.



Editing opacity of the signature

The slider displayed in the Appearance tab allows modifying opacity of the selected handwritten signature. You can also modify the opacity of the selected handwritten signature by giving numeric value in the opacity text box.



Deleting a signature

Selecting delete option from the context menu which is displayed when clicking right on the selected signature would delete the respective signature from the PDF document.



Track the changes in a handwritten signature

You can track the changes occurred in a handwritten signature such as add, select, deselect, move, resize, and remove. The `HandwrittenSignatureChanged` event indicates that a signature is changed in the current document and the `Action` property of the `HandwrittenSignatureChangedEventArgs` provides you the type of changes made in the signature. The following code shows how to wire the event in [PdfViewerControl](#).

C#

```
PdfViewerControl pdfViewer = new PdfViewerControl();
pdfViewer.Load("Barcode.pdf");
pdfViewer.HandwrittenSignatureChanged +=
PdfViewer_HandwrittenSignatureChanged;
private void PdfViewer_HandwrittenSignatureChanged(object sender,
HandwrittenSignatureChangedEventArgs e)
{
    //Insert your code that does something
}
```

VB.NET

```
Dim pdfViewer As PdfViewerControl = New PdfViewerControl()
pdfViewer.Load("Barcode.pdf")
AddHandler pdfViewer.HandwrittenSignatureChanged, AddressOf
PdfViewer_HandwrittenSignatureChanged
Private Sub PdfViewer_HandwrittenSignatureChanged(ByVal sender As Object,
ByVal e As HandwrittenSignatureChangedEventArgs)
    'Insert your code that does something
End Sub
```

Get the added handwritten signature

You can get the added handwritten signature as `System.Windows.Shapes.Path` using the `Signature` property of the `HandwrittenSignatureChangedEventArgs`. Refer to the following code example for more details.

C#

```
private void PdfViewer_HandwrittenSignatureChanged(object sender,
HandwrittenSignatureChangedEventArgs e)
{
    if (e.Action == AnnotationChangedAction.Add)
    {
        //Get the signature
    }
}
```

```

System.Windows.Shapes.Path signature = e.Signature as
System.Windows.Shapes.Path;
}
}

```

VB.NET

```

Private Sub PdfViewer_HandwrittenSignatureChanged(ByVal sender As Object,
ByVal e As HandwrittenSignatureChangedEventArgs)
If e.Action = AnnotationChangedAction.Add Then
'Get the signature
Dim signature As System.Windows.Shapes.Path = TryCast(e.Signature,
System.Windows.Shapes.Path)
End If

```

Track the changes in the location and size of a handwritten signature

You can track the changes in the location and size of a handwritten signature using the **OldBounds** and **NewBounds** properties of the **HandwrittenSignatureChangedEventArgs**. Refer to the following code example for more details.

C#

```

private void PdfViewer_HandwrittenSignatureChanged(object sender,
HandwrittenSignatureChangedEventArgs e)
{
if(e.Action== AnnotationChangedAction.Move)
{
//Get the old location
PointF oldLocation = e.OldBounds.Location;
//Get the new location
PointF newLocation = e.NewBounds.Location;
}
else if(e.Action == AnnotationChangedAction.Resize)
{
//Get the old size
SizeF oldSize = e.OldBounds.Size;
//Get the new size
SizeF newSize = e.NewBounds.Size;
}
}

```

VB.NET

```

Private Sub PdfViewer_HandwrittenSignatureChanged(ByVal sender As Object,
ByVal e As HandwrittenSignatureChangedEventArgs)
If e.Action = AnnotationChangedAction.Move Then
'Get the old location
Dim oldLocation As PointF = e.OldBounds.Location
'Get the new location
Dim newLocation As PointF = e.NewBounds.Location
ElseIf e.Action = AnnotationChangedAction.Resize Then
'Get the old size
Dim oldSize As SizeF = e.OldBounds.Size
'Get the new size
Dim newSize As SizeF = e.NewBounds.Size

```

```
End If
End Sub
```

Track the details of the selected handwritten signature

You can track the details such as location, size, and page number of the selected handwritten signature using `HandwrittenSignatureChangedEventArgs`. Refer to the following code example for more details.

C#

```
private void PdfViewer_HandwrittenSignatureChanged(object sender,
HandwrittenSignatureChangedEventArgs e)
{
    if(e.Action== AnnotationChangedAction.Select)
    {
        //Get the page number
        int pageNumber = e.PageNumber;
        //Get the signature location
        PointF location = e.NewBounds.Location;
        //Get the signature size
        SizeF size = e.NewBounds.Size;
    }
}
```

VB.NET

```
Private Sub PdfViewer_HandwrittenSignatureChanged(ByVal sender As Object,
ByVal e As HandwrittenSignatureChangedEventArgs)
If e.Action = AnnotationChangedAction.[Select] Then
    'Get the page number
    Dim pageNumber As Integer = e.PageNumber
    'Get the signature location
    Dim location As PointF = e.NewBounds.Location
    'Get the signature size
    Dim size As SizeF = e.NewBounds.Size
End If
End Sub
```

Add handwritten signature from code

PDF Viewer allows you to add a signature anywhere in the pages of a PDF document from code behind.

Add a handwritten signature in a single page

You can add signature in a particular page as `System.Windows.Shapes.Path` by using the `AddHandwrittenSignature` with the page number and the bounds as parameters. The following code explains how to add a handwritten signature at the first page of the PDF document.

C#

```
private void button_Click(object sender, RoutedEventArgs e)
{
    System.Windows.Shapes.Path signature = new System.Windows.Shapes.Path();
    //Note: Set your signature appearance as Data for the path here.
    //Add signature in the first page of the PDF document at the location: X=
    450, Y=750 and with the size: Width=100, Height=100.
```

```
pdfViewer.AddHandwrittenSignature(signature, 1, new RectangleF(450, 750, 100, 100));
}
```

VB.NET

```
Private Sub button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim signature As System.Windows.Shapes.Path = New
    System.Windows.Shapes.Path()
    'Note: Set your signature appearance as Data for the path here.
    'Add signature in the first page of the PDF document at the location: X=
    450, Y=750 and with the size: Width=100, Height=100.
    pdfViewer.AddHandwrittenSignature(signature, 1, New RectangleF(450, 750,
    100, 100))
End Sub
```

Add a handwritten signature in multiple pages

You can also add a signature in multiple pages of the PDF document at multiple places using the `AddHandwrittenSignature` method with the overload of dictionary with page numbers as keys and the list of bounds as values. The following code explains how to add a handwritten signature at the first and second page of the PDF document with different bounds.

C#

```
private void button_Click(object sender, RoutedEventArgs e)
{
    System.Windows.Shapes.Path signature = new System.Windows.Shapes.Path();
    //Note: Set your signature appearance using `Data` property of the path
    here.
    Dictionary<int, List<RectangleF>> bounds = new Dictionary<int,
    List<RectangleF>>();
    //Add signature bounds for the first page as: X= 450, Y=750, Width=100,
    Height=100.
    bounds.Add(1, new List<RectangleF>() { new RectangleF(450, 750, 100, 100)
    });
    //Add signature bounds for the second page as: X= 350, Y=550, Width=200,
    Height=200
    bounds.Add(2, new List<RectangleF>() { new RectangleF(350, 550, 200, 200)
    });
    pdfViewer.AddHandwrittenSignature(signature, bounds);
}
```

VB.NET

```
Private Sub button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim signature As System.Windows.Shapes.Path = New
    System.Windows.Shapes.Path()
    'Note: Set your signature appearance using `Data` property of the path here.
    Dim bounds As Dictionary(Of Integer, List(Of RectangleF)) = New
    Dictionary(Of Integer, List(Of RectangleF))()
    'Add signature bounds for the first page as: X= 450, Y=750, Width=100,
    Height=100.
    bounds.Add(1, New List(Of RectangleF)() From {
    New RectangleF(450, 750, 100, 100)
```



```
    })  
    'Add signature bounds for the second page as: X= 350, Y=550, Width=200,  
    Height=200.  
    bounds.Add(2, New List(Of RectangleF)() From {  
        New RectangleF(350, 550, 200, 200)  
    })  
    pdfViewer.AddHandwrittenSignature(signature, bounds)  
End Sub
```

Keyboard shortcuts

The below keyboard shortcuts are available to customize the handwritten signature in the PDF document.

Delete key – Deletes the selected signature from the PDF document. *Ctrl + Z* – Performs undo functionality for recently performed operations. * *Ctrl + Y* – Performs redo functionality for recently performed operations.

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Interaction Modes in WPF Pdf Viewer

The PDF Viewer supports following cursor modes for easy interaction with the PDF documents:

- Selection mode
- Panning mode
- Marquee zoom mode

Selection mode

In this mode, the text selection can be performed in the PDF document loaded in the PDF Viewer. It is the default mode of the control and allows users to select and copy text from the PDF files. This is helpful for copying and sharing text content. Refer to the following code to enable the selection mode in PdfViewerControl.

C#

```
PdfViewerControl pdfViewerControl = new PdfViewerControl();  
pdfViewerControl.CursorMode = PdfViewerCursorMode.SelectTool;  
pdfViewerControl.Load("Sample.pdf");
```

VB.NET

```
Dim pdfViewerControl As PdfViewerControl = New PdfViewerControl()  
pdfViewerControl.CursorMode = PdfViewerCursorMode.SelectTool  
pdfViewerControl.Load("Sample.pdf")
```

Panning mode

In this mode, the dragging and scrolling of the pages can be performed in any direction using mouse and touch interactions, but the text selection cannot be performed. Refer to the following code to enable the panning mode in PdfViewerControl.

C#

```
PdfViewerControl pdfViewerControl = new PdfViewerControl();  
pdfViewerControl.CursorMode = PdfViewerCursorMode.HandTool;  
pdfViewerControl.Load("Sample.pdf");
```

VB.NET

```
Dim pdfViewerControl As PdfViewerControl = New PdfViewerControl()  
pdfViewerControl.CursorMode = PdfViewerCursorMode.HandTool  
pdfViewerControl.Load("Sample.pdf")
```

Marquee zoom mode

In this mode, you can zoom a particular area of a PDF document by dragging the area using the zoom tool or keyboard actions. You can marquee a section by left click and drag using your mouse. By pressing the Ctrl key, you can decrease the zoom level. Refer to the following code to enable the marquee zoom mode in [PdfViewerControl](#).

C#

```
PdfViewerControl pdfViewerControl = new PdfViewerControl();  
pdfViewerControl.CursorMode = PdfViewerCursorMode.MarqueeZoom;  
pdfViewerControl.Load("Sample.pdf");
```

VB.NET

```
Dim pdfViewerControl As PdfViewerControl = New PdfViewerControl()  
pdfViewerControl.CursorMode = PdfViewerCursorMode.MarqueeZoom  
pdfViewerControl.Load("Sample.pdf")
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Working with PDF Pages in WPF Pdf Viewer**Page Border**

The [PageBorder](#) allows you to customize the border visibility and color of the pages that are being displayed in the PDF Viewer.

Note: This border customization applies only when viewing the PDF file in the PDF viewer and it will not affect the saved or printed file from the PDF Viewer.

Color

The [Color](#) property of the [PageBorder](#) allows you to customize the border color of the pages that are being displayed in the PDF Viewer. The default border color of the pages is **Black**. Refer to the following code sample to set a different color to the border of the pages.

C#

```
using System.Windows;  
using Syncfusion.Windows.PdfViewer;  
namespace PdfViewerDemo  
{
```

```

/// <summary>
/// Interaction logic for Window1.xaml
/// </summary>
public partial class MainWindow : Window
{
    #region Constructor
    public MainWindow()
    {
        InitializeComponent();
        //Create an instance for `PageBorder`.
        PageBorder pageBorder = new PageBorder();
        //Set the `Color` property.
        pageBorder.Color = System.Drawing.Color.Red;
        //Assign the 'PageBorder' property of PDF Viewer.
        pdfViewer.PageBorder = pageBorder;
        //Load the PDF file.
        pdfViewer.Load(@"Sample.pdf");
    }
    #endregion
}

```

Visibility

The [PageBorder](#) property allows you to show or hide the border of the pages that are being displayed in the PDF Viewer. By default, the border of the pages will be visible. Refer to the following code sample to hide the border of the pages.

C#

```

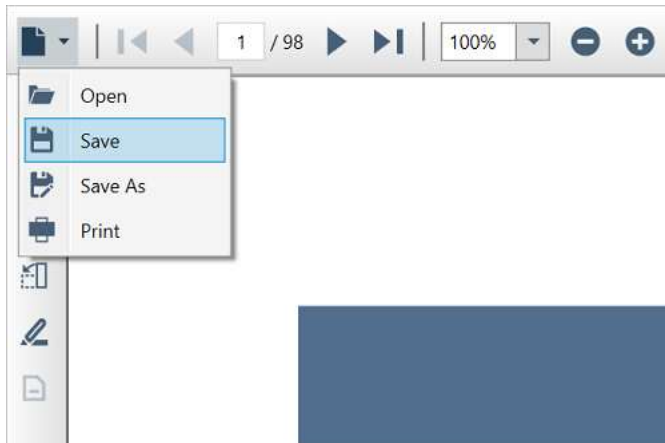
using System.Windows;
using Syncfusion.Windows.PdfViewer;
namespace PdfViewerDemo
{
    /// <summary>
/// Interaction logic for Window1.xaml
/// </summary>
    public partial class MainWindow : Window
    {
        #region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Create an instance for `PageBorder`.
            PageBorder pageBorder = new PageBorder();
            //Set the `IsVisible` property as false to hide the page border.
            pageBorder.IsVisible = false;
            //Assign the 'PageBorder' property of PDF Viewer.
            pdfViewer.PageBorder = pageBorder;
            //Load the PDF file.
            pdfViewer.Load(@"Sample.pdf");
        }
        #endregion
    }
}

```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Saving PDF Files in WPF Pdf Viewer

The PDF Viewer allows you to save the PDF document that is being displayed. It helps you to keep the file up to date with any modifications and prevent your work from being lost. You can save the changes in the PDF document using the "Save" and "Save As" options available in the toolbar.



Note: The "Save" option in the toolbar will not be enabled if the file is not edited, or if the file is loaded using the Stream and PdfLoadedDocument objects. In these cases, you can use the "Save As" option.

You can also call the [Save](#) method to silently save the modified PDF file to a specific path in the local disk. Refer to the following code to perform the same from the application level.

C#

```
Private void SavePDF()  
{  
    //Save the PDF file to a specific file path after doing any modifications by  
    //passing the file name as a parameter to the 'Save' method.  
    pdfViewer.Save("Saved.pdf");  
}
```

VB.NET

```
Private Sub SavePDF()  
    'Save the PDF file to a specific file path after doing any modifications by  
    'passing the file name as a parameter to the Save method.  
    pdfViewer.Save("Saved.pdf")  
End Sub
```

Events

The [PdfViewerControl](#) notifies you at the start and end of the save operation using the [BeginSave](#) and [EndSave](#) events respectively.

Begin Save

The [BeginSave](#) event occurs before initiating the save operation of the PDF file. It also allows you to cancel the save operation using the [Cancel](#) property of [BeginSaveEventArgs](#). The following code shows how to wire the event in the [PdfViewerControl](#).

C#

```
namespace SaveEvents
{
    public partial class MainWindow : Window
    {
        #region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Wire the `BeginSave` event.
            PdfViewer.BeginSave += PdfViewer_BeginSave;
            //Load the PDF file
            PdfViewer.Load("../Data/Windows Store Apps Succinctly.pdf");
        }
        #endregion
        #region Events
        private void PdfViewer_BeginSave(object sender, BeginSaveEventArgs e)
        {
            //Insert your code here
        }
        #endregion
    }
}
```

End Save

The [EndSave](#) event occurs after the completion of the save operation. The [IsCanceled](#) property of the [EndSaveEventArgs](#) helps you to know whether the save operation is canceled or not. The following code shows how to wire the event in the [PdfViewerControl](#).

C#

```
namespace SaveEvents
{
    public partial class MainWindow : Window
    {
        #region Constructor
        public MainWindow()
        {
            InitializeComponent();
            //Wire the `EndSave` event.
            PdfViewer.EndSave += PdfViewer_EndSave;
            //Load the PDF file
            PdfViewer.Load("../Data/Windows Store Apps Succinctly.pdf");
        }
        #endregion
        #region Events
        private void PdfViewer_EndSave(object sender, EndSaveEventArgs e)
        {
            //Insert your code here
        }
    }
}
```

```
#endregion  
}  
}
```

Keyboard shortcuts

The following keyboard shortcuts can be used to save the files when the toolbar is enabled:

- **Ctrl + S**: If the "Save" option is enabled in the PDF Viewer, it performs "Save" in the PDF document. Else, it performs "Save As" in the PDF document.
- **Shift + Ctrl + S**: Performs "Save As" in the PDF document.

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Adjust the magnification of PDF documents using the WPF PDF Viewer

The WPF PDF Viewer has predefined set of zoom tools in the built-in toolbar, that allows you to change magnifications of the PDF document that is being displayed.

![Zoom tools of WPF PDF Viewer](images/zoom tools.png)

1. **Combo box**: It allows you to select a zoom percentage from the pre-defined set of values listed in the dropdown. Since it is editable, you can also provide your zoom values by double editing the text area.
2. **Zoom out button**: It allows you to reduce the zoom value by 25% from the current zoom percentage.
3. **Zoom in button**: It allows you to increase the zoom value by 25% from the current zoom percentage.
4. **Fit to width button**: It allows you to fit the document to the width of the control.
5. **Fit to page button**: It allows you to fit a whole page in the window and to view only one page at a time.

Hide the zoom tools in the toolbar

You can hide the zoom tools in the toolbar by setting the [ShowZoomTools](#) property of [ToolbarSettings](#) to `false`. Refer to the following code sample to hide the zoom tools.

C#

```
// Hide the zoom tools  
pdfViewer.ToolbarSettings.ShowZoomTools = false;
```

Customization using APIs

If you are using [PdfDocumentView](#) control or do not want to use the built-in toolbar, you can still accomplish the functions, using the PDF Viewer's APIs.

Magnify the document to a given zoom percentage

You can magnify the document to a given zoom percentage by using the [ZoomTo](#) function of the PDF Viewer. Refer to the following code sample to apply the specific zoom percentage based on the page contents.

C#

```
//Zoom to 235 percentage.  
pdfViewer.ZoomTo(235);
```

The [ZoomPercentage](#) property can be used to get the control's current zoom percentage.

C#

```
//Get the current zoom percentage.  
int currentZoomPercentage = pdfViewer.ZoomPercentage;
```

Zoom modes

As stated above, the PDF Viewer supports the fit to page and fit to width zoom modes. You can change the zoom mode, using the [ZoomMode](#) property of the PDF Viewer. Refer to the following code sample to change the zoom mode.

C#

```
//To apply fit-to-page zoom mode.  
pdfViewer.ZoomMode = ZoomMode.FitPage;  
// To apply fit-to-width zoom mode.  
pdfViewer.ZoomMode = ZoomMode.FitWidth;
```

Define the minimum and maximum zoom percentage

The default minimum and maximum zoom percentage of the PDF Viewer is 50 and 400 respectively. However, the PDF Viewer allows you to customize the minimum and maximum zoom percentage values based on your requirement.

You can customize the minimum zoom percentage of the PDF viewer using the [MinimumZoomPercentage](#) property. Its default value is 50. Refer to the following code sample to set the property and apply the customized minimum zoom percent to the PDF Viewer

Note: The allowed value of this property ranges from 10 to the value of the [MaximumZoomPercentage](#) of PDF Viewer.

C#

```
//Load PDF document.  
pdfViewer.Load(@"../../Data/Barcode.pdf");  
//Set the minimum zoom percentage.  
pdfViewer.MinimumZoomPercentage = 20;  
//Magnify the document to minimum zoom percentage.  
pdfViewer.ZoomTo(pdfviewer.MinimumZoomPercentage);
```

Similarly, you can customize the maximum zoom percentage of the PDF Viewer using the [MaximumZoomPercentage](#) property. Its default value is 400. Refer to the following code sample to set the property and apply the customized maximum zoom percentage to PDF Viewer.

Note: The allowed value of this property ranges from the value of [MinimumZoomPercentage](#) of the PDF Viewer to 6400 for PDFium (default) rendering engine and 800 for SFPdf rendering engine, respectively.

C#

```
// Load PDF document.  
pdfviewer.Load(@"../../Data/Barcode.pdf");  
//Set the maximum zoom percentage.  
pdfviewer.MaximumZoomPercentage = 600;  
//Magnify the document to maximum zoom percentage.  
pdfviewer.ZoomTo(pdfviewer.MaximumZoomPercentage);
```

Zoom changes notification

You can be notified whenever the magnification is changed in the PDF Viewer using the [ZoomChanged](#) event. You can also obtain the current zoom percentage using the [ZoomEventArgs](#) event. Refer to the following code sample to wire and handle the event.

C#

```
void WireZoomChangedEvent()  
{  
    pdfViewer.ZoomChanged += PdfViewer_ZoomChanged;  
}  
private void PdfViewer_ZoomChanged(object sender, ZoomEventArgs args)  
{  
    int currentZoomPercentage = args.ZoomPercentage;  
}
```

Active view port rendering at higher zoom percentages

From the 19.3 version, the [MaximumZoomPercentage](#) of the PDF Viewer can be extended up to 6400% with the support of active view port rendering at higher zoom percentages. It renders only the part of the PDF file that is visible on-screen and ignoring the parts that are outside the viewport. The mode is automatically enabled when the page size or zoom increased beyond a specified limit on the zooming. This approach will be helpful to open the large-size pages containing PDF documents at higher zoom levels.

Note: Active view port rendering is supported only for the PDFium (default) rendering engine.

You need to set the [MaximumZoomPercentage](#) and call [ZoomTo](#) method to achieve it from the application level as shown in the following code example.

C#

```
//Set the maximum zoom percentage.  
pdfviewer.MaximumZoomPercentage = 6400;  
//Magnify the document to the zoom percentage you needed (less than or equal to 6400).  
//In the below example, control will magnify to 1200% zoom.  
pdfviewer.ZoomTo(1200);
```

Searching Text in WPF Pdf Viewer

Essential PDF Viewer allows you to search and highlight the text in the PDF document. The search box appears when Ctrl+F is pressed and searches the text in the PDF document as displayed in the following screenshot.

**Note:**

- PdfDocumentView is used to view the PDF documents without the toolbar. So, make use of PdfViewerControl to search the text using search box.

The PDF Viewer control also supports searching text in the PDF document using the following API. The [FindText](#) method returns true when the text given is found in the document. The dictionary contains the page index and the list of rectangular coordinates of the text found in that page. The following code example explains how text search can be achieved after [creating the control from the code](#) and handled in the Loaded event of the [application's MainWindow](#).

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    bool IsMatchFound;
    //Load the PDF.
    pdfViewerControl1.Load("../Data/Barcode.pdf");
    //Get the occurrences of the target text and location.
    Dictionary<int, List<RectangleF>>
    textSearch = new Dictionary<int, List<RectangleF>>();
    IsMatchFound = pdfViewerControl1.FindText("targetText", out textSearch);
}
```

VB.NET

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    Dim IsMatchFound As Boolean
    'Load the PDF.
    pdfViewerControl1.Load("../Data/Barcode.pdf")
    'Get the occurrences of the target text and location.
    Dim textSearch As New Dictionary(Of Integer, List(Of RectangleF))()
    IsMatchFound = pdfViewerControl1.FindText("targetText", textSearch)
End Sub
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Select and Copy Text in WPF Pdf Viewer

In PDF Viewer, text can be selected by clicking the mouse left button and dragging the mouse pointer over the text in any direction.

Detecting the completion of text selection

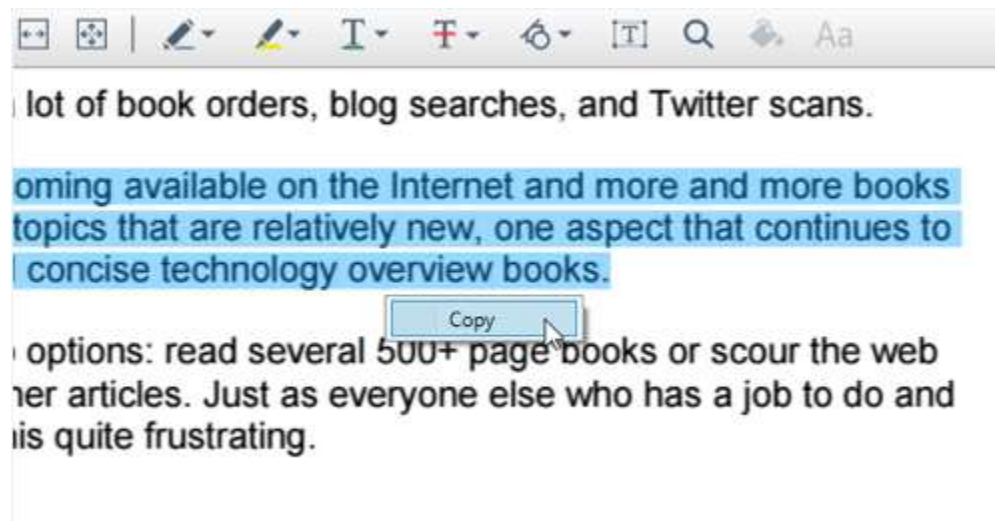
When the text selection is completed, the [TextSelectionCompleted](#) event will be raised. The selected text can be retrieved as string from the `args` parameter of the event handler.

C#

```
private void PdfViewer_TextSelectionCompleted(object sender,
TextSelectionCompletedEventArgs args)
{
    //Get the whole selected text
    string selectedText = args.SelectedText;
    //Get the selected text and its rectangular bounds for each page separately
    if the selection is made on multiple pages
    Dictionary<int, Dictionary<string, Rectangle>> selectedTextInformation =
    args.SelectedTextInformation;
}
```

Copying the selected text

The selected text can be copied by clicking the copy from the context menu, which appears when clicking the right mouse button after the text is selected.



The selected text can also be copied using the keyboard shortcut Ctrl + C.

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

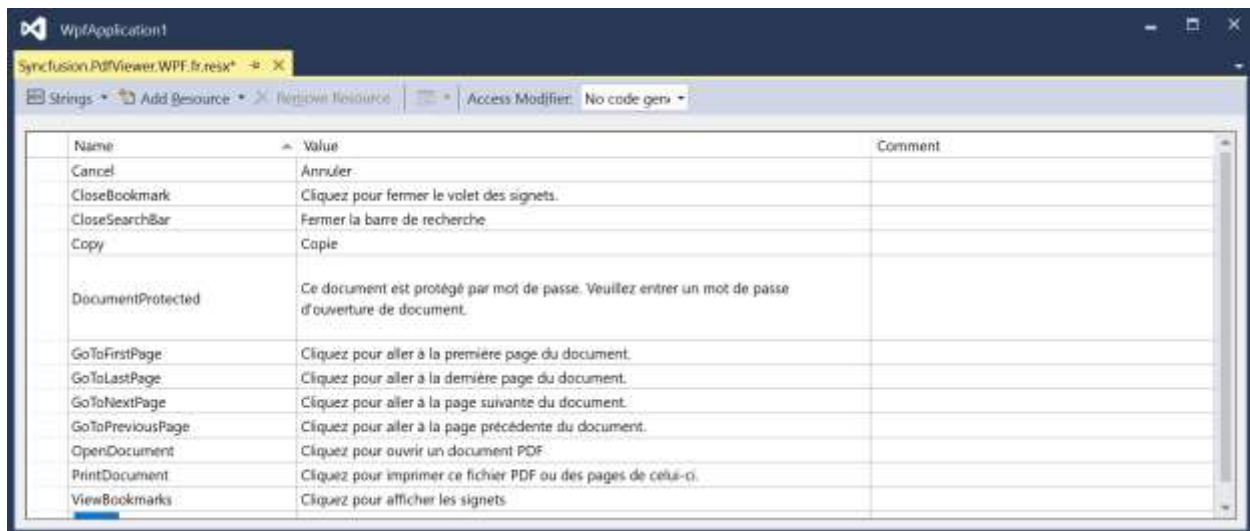
Localization in WPF Pdf Viewer

Localization is the process of configuring the application to a specific language. [PdfViewerControl](#) provides support to localize all the static text used for tooltip and context menu contents. Localization can be done by adding resource file (.resx) in the application.

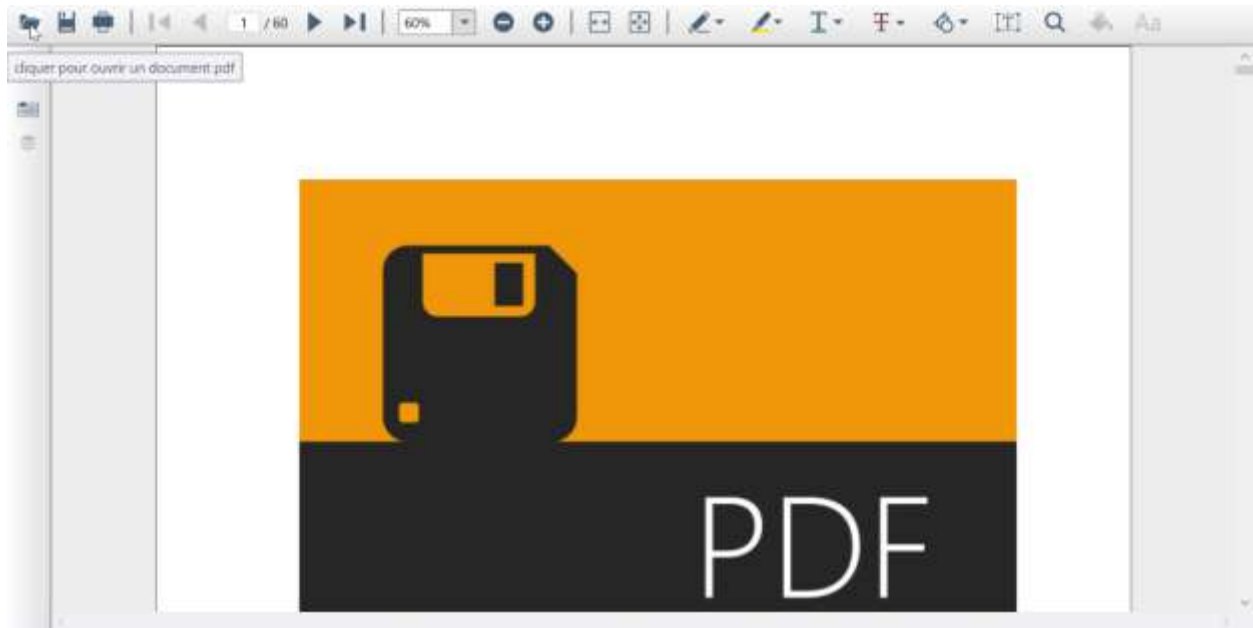
We are using **en-US** culture-based resource file as default in PDF Viewer. This can be downloaded from this [GitHub](#) location.

Adding Resource file

- Create a folder name as **Resources** in your application.
- Add default English ("en-US") [Resx](#) (resource) file of **PdfViewerControl** in **Resources** folder named as **Syncfusion.PdfViewer.WPF.resx**.
- Create a Resx (resource) files and name it as **Syncfusion.PdfViewer.WPF.[Culture name].resx**. For example, **Syncfusion.PdfViewer.WPF.fr.resx** for French culture.
- Add the resource key such as **OpenDocument** and its corresponding localized value in **Syncfusion.PdfViewer.WPF.fr.resx** file. Refer the below screenshot for the same.
- Execute the application in French culture to see the changes.



The following screenshot shows the **PdfViewerControl** in French language



Localize Resource File with Different Assembly or Namespace

In general, [PdfViewerControl](#) try to read the resource file from executing assembly and its default namespace can be done using [Assembly.GetExecuteAssembly](#) method. When the resource file is located at different assembly or namespace, it can set to the [PdfViewerControl](#) by using [LocalizationManager.SetResources](#) method.

C#

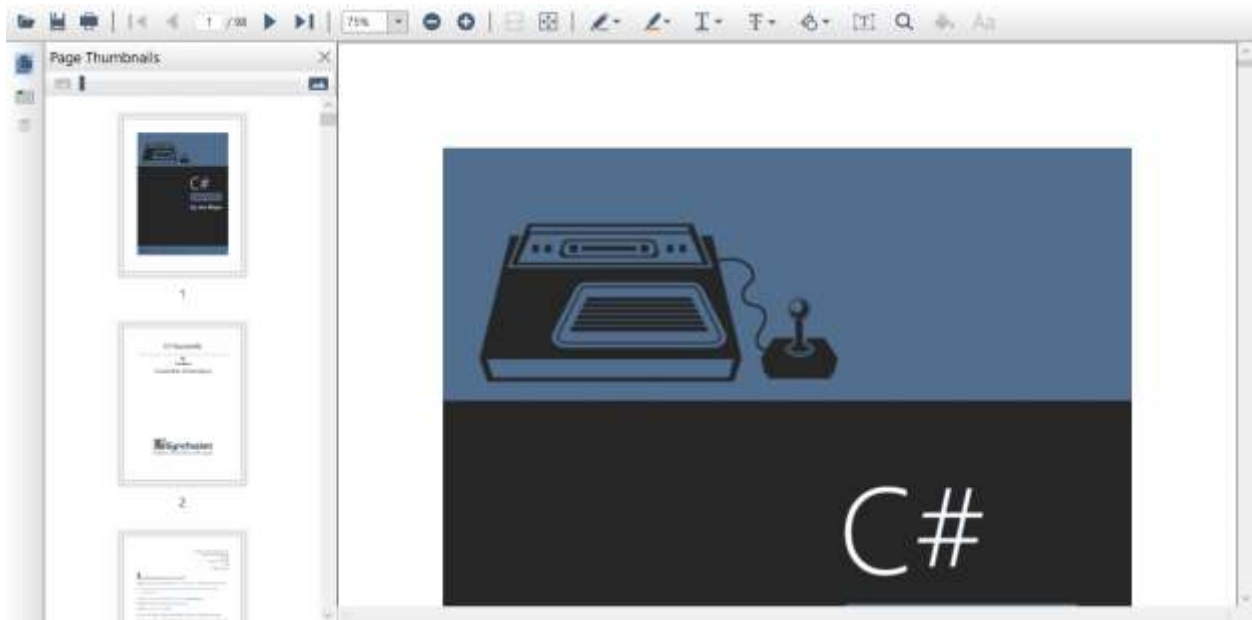
```
public MainWindow ()
{
    Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("fr-FR");
    // Set the Custom assembly and namespace for the localization.
    LocalizationManager.SetResources(typeof(Custom_Class).Assembly,
    "ClassLibrary");
    InitializeComponent();
}
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Navigation

Thumbnail Navigation in WPF Pdf Viewer

The thumbnail navigation support in PDF viewer allows users to view a miniature preview of the PDF pages for fast scrolling and easy navigation purpose.



Displaying page thumbnails

Page thumbnails are displayed by clicking the thumbnail icon in the left pane. To display thumbnails pane from code behind, use the following code example.

C#

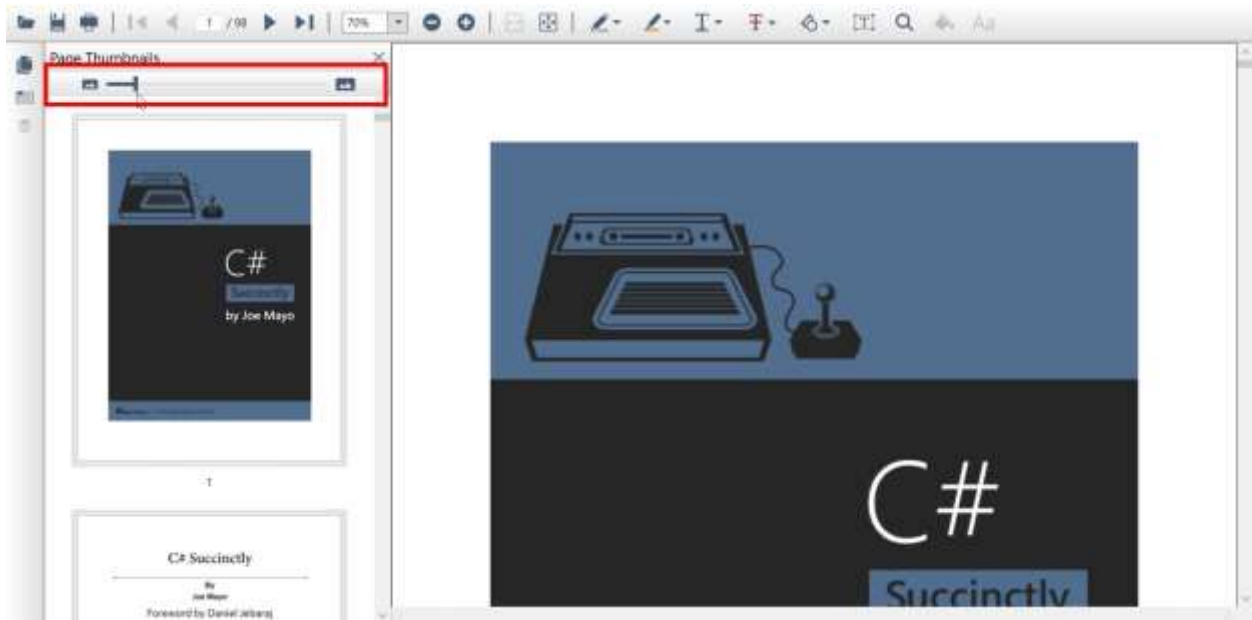
```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    PdfLoadedDocument pdf = new PdfLoadedDocument("Input.pdf");
    pdfviewer.Load(pdf);
    pdfviewer.ThumbnailSettings.IsExpanded = true;
}
```

VB.NET

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    Dim pdf As New PdfLoadedDocument("Input.pdf")
    pdfViewer.Load(pdf)
    pdfviewer.ThumbnailSettings.IsExpanded = true
End Sub
```

Enlarging and reducing size of page thumbnails

Page thumbnails size is enlarged and reduced using the zoom out and zoom in buttons in the Page Thumbnails pane, and also using the magnification slider.



Disabling thumbnails

Thumbnails are disabled by setting the `IsVisible` property of `ThumbnailSettings` in the PDF viewer control to false.

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    PdfLoadedDocument pdf = new PdfLoadedDocument("Input.pdf");
    pdfviewer.Load(pdf);
    pdfviewer.ThumbnailSettings.IsVisible = false;
}
```

VB.NET

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    Dim pdf As New PdfLoadedDocument("Input.pdf")
    pdfviewer.Load(pdf)
    pdfviewer.ThumbnailSettings.IsVisible = false
End Sub
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

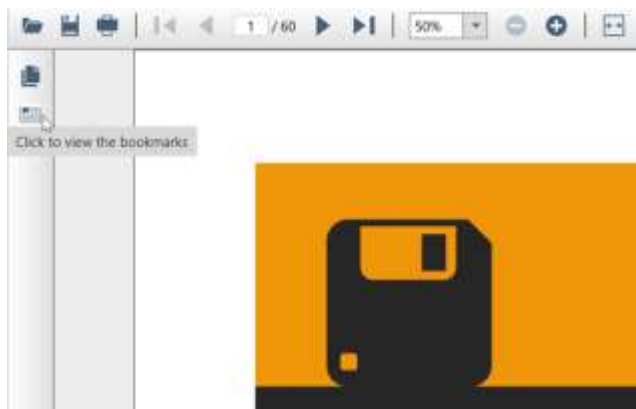
Bookmark Navigation in WPF Pdf Viewer

PDF Viewer control allows users to navigate to the bookmarks present in the loaded PDF document at UI level.

Steps to perform bookmark navigation in PdfViewerControl.

1. Open the bookmarks contained PDF document to enable the bookmark button in `PdfViewerControl`.
2. Clicking on the bookmark button from the left pane, will list the bookmarks present in the PDF

document. 3. To jump to a specific section, click its name in the bookmark pane. 4. If the bookmark has some children, you can explore them by clicking on the “+” button to the left of it.



Programmatically navigate to a bookmark destination

You can navigate to a desired bookmark destination using the `GotoBookmark(PdfBookmark)` method in `PdfDocumentView`. The target/destination bookmark should be provided as the parameter to this method. Refer to the following code sample.

C#

```
//Loads the PDF document in PdfLoadedDocument
PdfLoadedDocument loadedDocument = new PdfLoadedDocument(documentStream);
//Retrieves the bookmark collection from the loaded PDF document
PdfBookmarkBase bookmark = loadedDocument.Bookmarks;
//Navigate to the specified bookmark destination offset
pdfDocumentView.GotoBookmark(bookmark[0]);
```

VB.NET

```
Loads the PDF document in PdfLoadedDocument
Dim loadedDocument As PdfLoadedDocument = New
PdfLoadedDocument(documentStream)
`Retrieves the bookmark collection from the loaded PDF document
Dim bookmark As PdfBookmarkBase = loadedDocument.Bookmarks
`Navigate to the specified bookmark destination offset
pdfDocumentView.GotoBookmark(bookmark(0))
```

You can also perform the same in [PdfViewerControl](#) using the [GotoBookmark\(PdfBookmark\)](#) method. Refer to the following code sample.

C#

```
//Loads the PDF document in PdfLoadedDocument
PdfLoadedDocument loadedDocument = new PdfLoadedDocument(documentStream);
//Retrieves the bookmark collection from the loaded PDF document
PdfBookmarkBase bookmark = loadedDocument.Bookmarks;
//Navigate to the specified bookmark destination offset
pdfViewerControl.GotoBookmark(bookmark[0]);
```

VB.NET

```

`Loads the PDF document in PdfLoadedDocument
Dim loadedDocument As PdfLoadedDocument = New
PdfLoadedDocument(documentStream)
`Retrieves the bookmark collection from the loaded PDF document
Dim bookmark As PdfBookmarkBase = loadedDocument.Bookmarks
`Navigate to the specified bookmark destination offset
pdfViewerControl.GoToBookmark(bookmark(0))

```

Enabling and disabling bookmark feature

You can enable and disable the bookmark button from the built-in toolbar using the `IsBookmarkEnabled` property available in `PdfViewerControl`.

Property	Action
IsBookmarkEnabled	Enables or disables the bookmark feature.

This property removes the bookmark button and disable the bookmark feature, when it is set to false and vice versa.

C#

```

//Bookmark feature is disabled
pdfViewerControl.IsBookmarkEnabled = false;

```

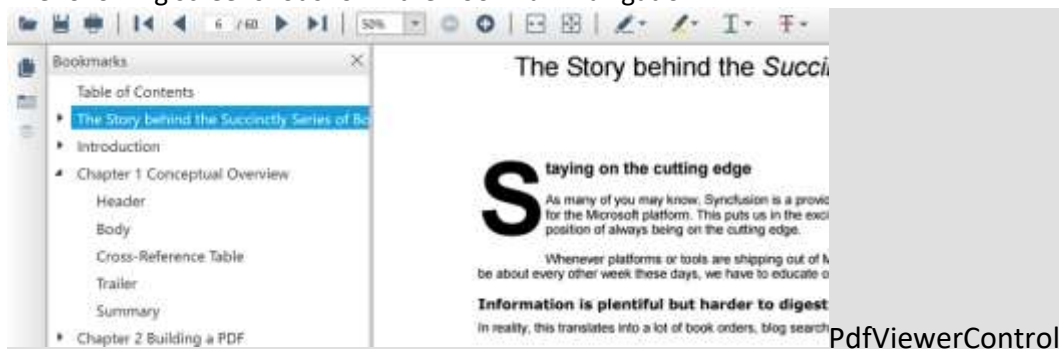
VB.NET

```

'Bookmark feature is disabled
pdfViewerControl.IsBookmarkEnabled = false

```

The following screenshot shown the Bookmark navigation in



`PdfViewerControl`,

![WPF PDF Viewer Enabling and Disabling Bookmark Feature](BookmarkNavigationimages/wpf-pdf-viewer-enabling-and-disabling-bookmark-feature.png)

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Page Navigation in WPF Pdf Viewer

PDF Viewer allows you to navigate through the pages of the PDF document using the [GotoPage](#) method. The following code example illustrates the navigation to page 2 of the PDF document.

C#

```
//Initialize PDF Viewer.
PdfViewerControl pdfViewer1 = new PdfViewerControl();
//Load the PDF.
pdfViewer1.Load("Sample.pdf");
//Navigate to page 2
pdfviewer1.GotoPage(2);
```

VB.NET

```
'Initialize PDF Viewer.
Private pdfViewer1 As New PdfViewerControl()
'Load the PDF.
pdfViewer1.Load("Sample.pdf")
'Navigate to page 2
pdfviewer1.GotoPage(2)
```

Navigate to the horizontal and vertical offset

You can now scroll to the given horizontal and vertical offset of the PDF document programmatically using the [ScrollTo](#) method of [PdfViewerControl](#). Refer to the following code to scroll the PDF document to the horizontal and vertical offset of 160 and 400 respectively.

C#

```
//Initialize PDF Viewer.
PdfViewerControl pdfViewerControl = new PdfViewerControl();
//Load the PDF.
pdfViewerControl.Load("Sample.pdf");
//Navigate to the horizontal and vertical offset of 160 and 400 respectively
pdfViewerControl.ScrollTo (160, 400);
```

VB.NET

```
'Initialize PDF Viewer.
Private pdfViewerControl As New PdfViewerControl()
'Load the PDF.
pdfViewerControl.Load("Sample.pdf")
'Navigate to the horizontal and vertical offset of 160 and 400 respectively
pdfViewerControl.ScrollTo (160, 400)
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Hyperlink Navigation in WPF Pdf Viewer

The WPF PDF Viewer supports URI annotations (hyperlinks) in the PDF document that enables the URI available in the PDF document to be opened in the default browser just by clicking on it. This also supports a few events that are listed in the following table.

Events Table

Event	Description	Arguments
-------	-------------	-----------

HyperlinkMouseOver	This event is triggered when the mouse pointer is placed over the hyperlink.	N/A
HyperlinkClicked	This event is triggered when the hyperlink in the PDF document is clicked.	HyperlinkClickedEventArgs

How to disable hyperlink navigation in PDF viewer control

You can disable the hyperlink navigation in PDF viewer control by setting the value of **Handled** in the **HyperlinkClickedEventArgs** parameter as true in the [HyperlinkClicked](#) event which is available in the PdfViewerControl and PdfDocumentView class.

Please refer to the following example for more details.

C#

```
// Wires the event handler for `HyperlinkClicked` event.
pdfViewerControl.HyperlinkClicked += PdfViewerControl_HyperlinkClicked;
private void Pdfviewer_HyperlinkClicked(object sender,
Syncfusion.Windows.PdfViewer.HyperlinkClickedEventArgs args)
{
    // Gets or sets the value to handle the navigation of hyperlink.
    args.Handled = true;
}
```

VB.NET

```
' Hooks the event handler for `HyperlinkClicked` event
'Initialize PDF Viewer.
Private pdfViewerControl As New PdfViewerControl()
'Load the PDF.
pdfViewerControl.Load("Sample.pdf")
AddHandler pdfviewer.HyperlinkClicked, AddressOf
PdfViewerControl_HyperlinkClicked
Private Sub PdfViewerControl_HyperlinkClicked(obj As Object, args As
Syncfusion.Windows.PdfViewer.AnnotEventArgs)
'Your code here...
End Sub
' Unhooks the event handler for `HyperlinkClicked` event.
RemoveHandler pdfviewer.HyperlinkClicked, AddressOf
PdfViewerControl_HyperlinkClicked
```

How to retrieve the clicked URI from PDF viewer

You can acquire the details of the hyperlink, which is clicked in the PDF file using the **HyperlinkClickedEventArgs** in the [HyperlinkClicked](#) event.

Please refer to the following example for more details.

C#

```
// Wires the event handler for `HyperlinkClicked` event.
pdfViewerControl.HyperlinkClicked += PdfViewerControl_HyperlinkClicked;
private void Pdfviewer_HyperlinkClicked(object sender,
Syncfusion.Windows.PdfViewer.HyperlinkClickedEventArgs args)
{
```

```
//Returns the URI clicked in the PDF viewer control.
string URI = args.Uri;
}
```

VB.NET

```
' Hooks the event handler for `HyperlinkClicked` event
AddHandler pdfviewer.HyperlinkClicked, AddressOf
PdfViewerControl_HyperlinkClicked
Private Sub PdfViewerControl_HyperlinkClicked(obj As Object, args As
Syncfusion.Windows.PdfViewer.AnnotEventArgs)
' Returns the URI clicked in the PDF viewer control.
Dim uri As String = args.URI
End Sub
```

Redirecting to a different Hyperlink

You can navigate to different hyperlink irrespective of the hyperlink clicked. To redirect a hyperlink, you need to set the value of **Handled** parameter as true to stop the navigation of the hyperlink clicked. Then open the desired hyperlink that you want to navigate instead of the hyperlink clicked.

Please refer to the following example for more details.

C#

```
// Wires the event handler for `HyperlinkClicked` event.
pdfViewerControl.HyperlinkClicked += PdfViewerControl_HyperlinkClicked;
private void Pdfviewer_HyperlinkClicked(object sender,
Syncfusion.Windows.PdfViewer.HyperlinkClickedEventArgs args)
{
// Gets or sets the value to handle the navigation of hyperlink.
args.Handled = true;
// Opens the URI (www.google.com) in a default browser.
System.Diagnostics.Process.Start("www.google.com");
}
```

VB.NET

```
' Hooks the event handler for `HyperlinkClicked` event.
AddHandler pdfviewer.HyperlinkClicked, AddressOf
PdfViewerControl_HyperlinkClicked
Private Sub PdfViewerControl_HyperlinkClicked(obj As Object, args As
Syncfusion.Windows.PdfViewer.AnnotEventArgs)
' Opens the URI (www.google.com) in a default browser.
System.Diagnostics.Process.Start("www.google.com")
End Sub
```

How to get notified when a mouse pointer moves over a hyperlink

[HyperlinkMouseOver](#) event is triggered when you place the mouse pointer over the URI in the PDF viewer control.

Please refer to the following example for more details.

C#

```
// Wire the event handler for `HyperlinkMouseOver` event.
```

```
pdfViewerControl.HyperlinkMouseOver += PdfViewerControl_HyperlinkMouseOver;
private void Pdfviewer_HyperlinkMouseOver(object sender, EventArgs args)
{
    //Your code here
}
// Unwire the event handler for `HyperlinkMouseOver` event.
pdfViewerControl.HyperlinkMouseOver -= PdfViewerControl_HyperlinkMouseOver;
```

VB.NET

```
' Hooks the event handler for `HyperlinkMouseOver` event
AddHandler pdfviewer.HyperlinkMouseOver, AddressOf
PdfViewerControl_HyperlinkMouseOver
Private Sub PdfViewerControl_HyperlinkMouseOver(obj As Object, args As
Syncfusion.Windows.PdfViewer.AnnotEventArgs)
End Sub
' Unhooks the event handler for `HyperlinkMouseOver` event.
RemoveHandler pdfviewer.HyperlinkMouseOver, AddressOf
PdfViewerControl_HyperlinkMouseOver
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

Annotations in WPF PDF Viewer

The WPF PDF Viewer allows you to review PDF Files with the rich set of annotating tools. You can add, edit or remove the following annotations in PDF files.

- Ink.
- Text markups (Highlight, Underline and Strikethrough).
- Shapes (Line, Circle, Rectangle, Arrow, Polygon and Polyline).
- Stamp.
- Text box.
- Sticky note.

Keyboard shortcuts

The below keyboard shortcuts are available to customize the annotation in the PDF document.

Delete key: Deletes the selected annotation from the PDF document. *Ctrl + Z:* Performs undo functionality for recently performed operations. *Ctrl + Y:* Performs redo functionality for recently performed operations. *Esc key:* To exit from the annotation mode. ** Up, down, right, left (direction keys):* Moves the annotations up, down, right, and left, respectively.

Note: You can find the example projects in the [GitHub example](#).

Working with Commands in WPF Pdf Viewer

Magnification operations

The PdfViewerControl provides the following set of commands to perform magnification.

- [IncreaseZoomCommand](#)
- [DecreaseZoomCommand](#)

Note:

- When executing the IncreaseZoomCommand, the 25% of zoom value has been increased from the current zoom percentage.
- When executing the DecreaseZoomCommand, the 25% of zoom value has been decreased from the current zoom percentage.

The following XAML code shows how to bind the IncreaseZoomCommand to a button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="ZoomIn" Height="20" Width="60" Command="{Binding
ElementName=pdfViewerControl, Path= IncreaseZoomCommand, Mode=OneWay}" />
```

The following XAML code shows how to bind the DecreaseZoomCommand to a button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="ZoomOut" Height="20" Width="60" Command="{Binding
ElementName=pdfViewerControl, Path= DecreaseZoomCommand, Mode= OneWay }" />
```

Page Navigations

The below list of commands will be helpful for performing the page navigation operations.

- [FirstPageCommand](#)
- [LastPageCommand](#)
- [PreviousPageCommand](#)
- [NextPageCommand](#)
- [GoToPageCommand](#)

FirstPageCommand

Navigation to the first page can be achieved by using the FirstPageCommand as shown below.

The following XAML code shows how to bind the FirstPageCommand to a button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Go to first page" Height="20" Width="100" Command="{Binding
ElementName=pdfViewerControl, Path=FirstPageCommand, Mode= OneWay }" />
```

Note:

- If the current page is already the first page of PDF document, then FirstPageCommand does not have any effect.

LastPageCommand

Navigation to the last page can be achieved by using the LastPageCommand as shown below.

The following XAML code shows how to bind the LastPageCommand to a button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Go to last page" Height="20" Width="100" Command="{Binding
ElementName=pdfViewerControl, Path=LastPageCommand, Mode= OneWay }" />
```

Note:

- If the current page is already the last page of PDF document, then LastPageCommand does not have any effect.

NextPageCommand

NextPageCommand displays the page next to the currently displayed page in the PDF Viewer.

The following XAML code shows how to bind the NextPageCommand to a button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Go to next page" Height="20" Width="100" Command="{Binding
ElementName=pdfViewerControl, Path=NextPageCommand, Mode= OneWay }" />
```

Note:

- If the current page is last page of the PDF document, then NextPageCommand does not have any effect.

PreviousPageCommand

PreviousPageCommand displays the page previous to the currently displayed page in the PDF Viewer.

The following XAML code shows how to bind the PreviousPageCommand to a button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Go to previous page" Height="20" Width="100"
Command="{Binding ElementName=pdfViewerControl, Path= PreviousPageCommand,
Mode= OneWay }" />
```

Note:

* If the current page is first page of the PDF document, then PreviousPageCommand does not have any effect.

GoToPageCommand

GoToPageCommand allows you to navigate through the pages of the PDF document. The following code example illustrates the navigation to page 2 of the PDF document.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Go to page" Height="20" Width="60" Command="{Binding
ElementName=pdfViewerControl, Path=GoToPageCommand, Mode= OneWay }"
CommandParameter="2" />
```

Note:

- If the command parameter is some other text or invalid page number, the `GoToPageCommand` does not have any effect.

Printing PDF

Printing can be performed by using the `Print` command.

The following XAML code shows how to bind the [PrintCommand](#) to a Button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Print" Height="20" Width="60" Command="{Binding
ElementName=pdfViewerControl,Path= PrintCommand, Mode= OneWay }" />
```

AnnotationCommand

The [AnnotationCommand](#) can be used to set the different types of annotation mode by using the `AnnotationParameter` in PdfViewer.

The following list of `AnnotationCommandParameter` values to the `AnnotationCommand` are used to set the annotation mode.

- Line
- Circle *Rectangle Arrow Ink Strikethrough Highlight Underline*

Note:

If the `AnnotationCommandParameter` value of `AnnotationCommand` is not equivalent to above mentioned any one of the text, then the annotation mode is set as none. The `AnnotationCommandParameter` is not a case sensitive.

Line Annotation

The following XAML code shows how to bind the annotation command for line annotation to a Button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Line" Height="20" Width="60" CommandParameter="Line"
Command="{Binding ElementName=pdfViewerControl, Path=AnnotationCommand,
Mode= OneWay }" />
```

Circle Annotation

The following XAML code shows how to bind the annotation command for circle annotation to a Button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Circle" Height="20" Width="60" CommandParameter="Circle "
Command="{Binding ElementName=pdfViewerControl, Path=AnnotationCommand,
Mode= OneWay }" />
```

Rectangle Annotation

The following XAML code shows how to bind the annotation command for rectangle annotation to a Button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Rectangle" Height="20" Width="60"
CommandParameter="Rectangle" Command="{Binding ElementName=pdfViewerControl,
Path=AnnotationCommand, Mode= OneWay }" />
```

Arrow Annotation

The following XAML code shows how to bind the annotation command for arrow annotation to a Button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl"/>
<Button Content="Arrow" CommandParameter="Arrow" Command="{Binding
ElementName=pdfViewerControl, Path=AnnotationCommand, Mode= OneWay }" />
```

Ink Annotation

The following XAML code shows how to bind the annotation command for ink annotation to a Button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Ink" Height="20" Width="60" CommandParameter="Ink"
Command="{Binding ElementName=pdfViewerControl, Path=AnnotationCommand,
Mode= OneWay }" />
```

Strikethrough Annotation

The following XAML code shows how to bind the annotation command for strikethrough annotation to a Button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Strikethrough" Height="20" Width="60"
CommandParameter="Strikethrough" Command="{Binding
ElementName=pdfViewerControl, Path=AnnotationCommand, Mode= OneWay }" />
```

Highlight Annotation

The following XAML code shows how to bind the annotation command for highlight annotation to a Button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Highlight" Height="20" Width="60"
CommandParameter="Highlight" Command="{Binding ElementName=pdfViewerControl,
Path=AnnotationCommand, Mode= OneWay }" />
```


Underline Annotation

The following XAML code shows how to bind the annotation command for underline annotation to a Button.

XML

```
<Syncfusion:PdfViewerControl x:Name="pdfViewerControl" />
<Button Content="Underline" Height="20" Width="60"
CommandParameter="Underline" Command="{Binding ElementName=pdfViewerControl,
Path=AnnotationCommand, Mode= OneWay }" />
```

Save Document Command

You can save the PDF file in the given file path using the `SaveDocumentCommand`. The following code shows how to save the PDF file by executing the command and to pass the file path as command parameter.

C#

```
pdfViewerControl.SaveDocumentCommand.Execute(@"C:\temp\Output.pdf");
```

Note: You can refer to our [WPF PDF Viewer](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF PDF Viewer example](#) to know how to render and configure the pdfviewer.

How to

Load a specific page

Navigation to a specific page, through code, is possible using [GoToPageAtIndex](#) method.

C#

```
//Initialize PDF Viewer.
PdfViewerControl pdfViewer1 = new PdfViewerControl();
//Load the PDF.
pdfViewer1.Load("Sample.pdf");
pdfViewer1.GoToPageAtIndex(2);
```

VB.NET

```
'Initialize PDF Viewer.
Private pdfViewer1 As New PdfViewerControl()
'Load the PDF.
pdfViewer1.Load("Sample.pdf")
pdfViewer1.GoToPageAtIndex(2)
```

Disable toolbar items

To remove the default toolbar completely, use the [PdfDocumentView](#) control instead of [PdfViewerControl](#) as described in the [section](#).

However, an individual toolbar item can also be removed from the default toolbar of PDF Viewer using the toolbar template. The following code sample explains disabling the text search tool from the default toolbar.

C#

```
private void HideTextSearchTool ()
{
    //Get the instance of the toolbar using its template name.
    DocumentToolbar toolbar = pdfViewer.Template.FindName("PART_Toolbar",
pdfViewer) as DocumentToolbar;
    //Get the instance of the open file button using its template name.
    Button textSearchButton =
(Button)toolbar.Template.FindName("PART_ButtonTextSearch", toolbar);
    //Set the visibility of the button to collapsed.
    textSearchButton.Visibility = System.Windows.Visibility.Collapsed;
}
```

Note: Apply the changes of hiding toolbar items, only after when the application window is loaded.

The sample project for disabling toolbar item is available in the [GitHub](#).

Similarly, other toolbar items also can be disabled. The following table lists the template names of the rest of the toolbar items along with their respective types in the order they appear in the toolbar.

Toolbar item	Template name	Type
File tool	PART_FileToggleButton	System.Windows.Controls.Primitives.ToggleButton
Navigation tools separator	Part_NavigationToolsSeparator	System.Windows.Shapes.Rectangle
First page tool	PART_ButtonGoToFirstPage	System.Windows.Controls.Button
Previous page tool	PART_ButtonGoToPreviousPage	System.Windows.Controls.Button
Current page number tool	PART_TextCurrentPageIndex	System.Windows.Controls.TextBox
Page count tool	PART_LabelTotalPageCount	System.Windows.Controls.TextBlock
Next page tool	PART_ButtonGoToNextPage	System.Windows.Controls.Button
Last page tool	PART_ButtonGoToLastPage	System.Windows.Controls.Button
Zoom tools separator	PartZoomToolsSeparator0	System.Windows.Shapes.Rectangle
Current zoom level tool	PART_ComboBoxCurrentZoomLevel	System.Windows.Controls.ComboBox
Zoom in tool	PART_ButtonZoomIn	System.Windows.Controls.Button
Zoom out tool	PART_ButtonZoomOut	System.Windows.Controls.Button
Zoom tools separator	PARTZoomToolsSeparator1	System.Windows.Shapes.Rectangle
Fit width tool	PART_ButtonFitWidth	System.Windows.Controls.Button
Fit page tool	PART_ButtonFitPage	System.Windows.Controls.Button

Annotation tools separator	PART_AnnotationToolsSeparator	System.Windows.Shapes.Rectangle
Sticky note tool	PART_StickyNote	System.Windows.Controls.Primitives.ToggleButton
Ink tool	PART_Ink	System.Windows.Controls.Primitives.ToggleButton
Ink eraser tool	PART_InkEraser	System.Windows.Controls.Primitives.ToggleButton
Highlight tool	PART_Highlight	System.Windows.Controls.Primitives.ToggleButton
Underline tool	PART_Underline	System.Windows.Controls.Primitives.ToggleButton
Strikethrough tool	PART_Strikethrough	System.Windows.Controls.Primitives.ToggleButton
Shapes tool	PART_Shapes	System.Windows.Controls.Primitives.ToggleButton
Fill tool	PART_Fill	System.Windows.Controls.Primitives.ToggleButton
Add textbox tool	PART_FreeText	System.Windows.Controls.Primitives.ToggleButton
Text properties tool	PART_ButtonTextBoxFont	System.Windows.Controls.Button
Separator between the annotation and cursor tools	PART_AnnotationsSeparator	System.Windows.Shapes.Rectangle
Stamp tool	PART_Stamp	System.Windows.Controls.Primitives.ToggleButton
Handwritten signature tool	PART_ButtonSignature	System.Windows.Controls.Button
Select tool	PART_SelectTool	System.Windows.Controls.Primitives.ToggleButton
Hand tool	PART_HandTool	System.Windows.Controls.Primitives.ToggleButton
Marquee zoom tool	PART_MarqueeZoom	System.Windows.Controls.Primitives.ToggleButton
Separator between the cursor tools and text search button	Part_CursorTools	System.Windows.Shapes.Rectangle
Text search tool	PART_ButtonTextSearch	System.Windows.Controls.Button

Note: From the v18.4.0.x onwards, the file menu items such as Open, Save, Save As, and Print are not directly present in the toolbar and they are present in the context menu of the File tools toggle button.

The following code sample explains disabling the Open tool from the menu.

C#

```
private void HideOpenTool(object sender, RoutedEventArgs e)
{
    //Get the instance of the toolbar using its template name.
    DocumentToolBar toolbar = pdfViewer.Template.FindName("PART_Toolbar",
pdfViewer) as DocumentToolBar;
    //Get the instance of the file menu button using its template name.
    ToggleButton FileButton =
    (ToggleButton)toolbar.Template.FindName("PART_FileToggleButton", toolbar);
    //Get the instance of the file menu button context menu and the item
    collection.
    ContextMenu FileContextMenu = FileButton.ContextMenu;
    foreach (MenuItem FileMenuItem in FileContextMenu.Items)
    {
        //Get the instance of the open menu item using its template name and disable
        its visibility.
        if (FileMenuItem.Name == "PART_OpenMenuItem")
        FileMenuItem.Visibility = System.Windows.Visibility.Collapsed;
    }
}
```

Similarly, other file menu items can be disabled. The following table lists the template names along with their respective types in the order they appear in the context menu.

Toolbar item	Template name	Type
Open tool	PART_OpenMenuItem	System.Windows.Controls.MenuItem
Save tool	PART_SaveMenuItem	System.Windows.Controls.MenuItem
SaveAs tool	PART_SaveAsMenuItem	System.Windows.Controls.MenuItem
Print tool	PART_PrintMenuItem	System.Windows.Controls.MenuItem

Note: The present UI design is subject to change, based on the inclusion of new features, enhancements, and user convenience.

Acquire total number of pages in WPF Pdf Viewer

PDF Viewer provides the total number of the pages in the PDF document that is being loaded into the PDF Viewer. This value can be acquired with the help of [PageCount](#) property in the PDF Viewer control, the following code example illustrates the same.

C#

```
//Initialize PDF Viewer.
PdfViewerControl pdfViewer1 = new PdfViewerControl();
//Load the PDF.
pdfViewer1.Load("Sample.pdf");
// Acquiring the total number of pages in the document being loaded
int pageCount = pdfviewer1.PageCount;
```

VB.NET

```
'Initialize PDF Viewer.
Private pdfViewer1 As New PdfViewerControl()
'Load the PDF.
pdfViewer1.Load("Sample.pdf")
'Acquiring the total number of pages in the document being loaded
Dim pageCount As Integer = pdfviewer1.PageCount
```

Change the color of the Loading Indicator in WPF Pdf Viewer

PDF Viewer allows you to change the color of the loading indicator displayed when loading the PDF document. The following code example illustrates the same.

C#

```
//Initialize PDF Viewer.
PdfViewerControl pdfViewer1 = new PdfViewerControl();
//Load the PDF.
pdfViewer1.Load("Sample.pdf");
// Changing the color of the loading indicator to Red
pdfviewer1.LoadingIndicator.LoaderColor =
System.Windows.Media.Color.FromArgb(255, 255, 0, 0);
```

VB.NET

```
'Initialize PDF Viewer.
Private pdfViewer1 As New PdfViewerControl()
'Load the PDF.
pdfViewer1.Load("Sample.pdf")
'Changing the color of the loading indicator to Red
pdfviewer1.LoadingIndicator.LoaderColor =
System.Windows.Media.Color.FromArgb(255, 255, 0, 0)
```

Change the text displayed in the loading indicator in WPF Pdf Viewer

PDF Viewer allows you to change the text displayed in the loading indicator. The following code example illustrates the same.

C#

```
// Changing the text displayed in the loading indicator
pdfviewer1.LoadingIndicator.LoadingMessage = "Document loading";
```

VB.NET

```
'Changing the text displayed in the loading indicator
pdfviewer1.LoadingIndicator.LoadingMessage = "Document loading"
```

Toggle visibility of the tool bar in WPF Pdf Viewer

PDF Viewer supports showing and hiding toolbar, when you feel to customize the toolbar, you can hide the default toolbar of the PDF Viewer using the [ShowToolbar](#) property. The following code example hides the default toolbar in the PDF Viewer control.

C#

```
// Hide the default (top) toolbar of the PDF Viewer
```

```
pdfViewer.ShowToolBar = false;
```

VB.NET

```
' Hiding the default toolbar of the PDF Viewer  
pdfViewer.ShowToolBar = False
```

Hide the vertical toolbar

You can hide the vertical toolbar which is present in the left side of PDF Viewer by disabling all the items present in the toolbar. Refer to the following code to hide the vertical toolbar.

C#

```
private void HideVerticalToolBar()  
{  
    // Hides the thumbnail icon.  
    pdfViewer.ThumbnailSettings.IsVisible = false;  
    // Hides the bookmark icon.  
    pdfViewer.IsBookmarkEnabled = false;  
    // Hides the layer icon.  
    pdfViewer.EnableLayers = false;  
    // Hides the organize page icon.  
    pdfViewer.PageOrganizerSettings.IsIconVisible = false;  
    // Hides the redaction icon.  
    pdfViewer.EnableRedactionTool = false;  
    // Hides the form icon.  
    pdfViewer.FormSettings.IsIconVisible = false;  
}
```

VB.NET

```
Private Sub HideVerticalToolBar()  
    pdfViewer.ThumbnailSettings.IsVisible = False  
    pdfViewer.IsBookmarkEnabled = False  
    pdfViewer.EnableLayers = False  
    pdfViewer.PageOrganizerSettings.IsIconVisible = False  
    pdfViewer.EnableRedactionTool = False  
    pdfViewer.FormSettings.IsIconVisible = False  
End Sub
```

Note: The sample project for disabling top and left toolbar is available in the [GitHub](#).

Toggle visibility of the scroll bar in WPF Pdf Viewer

PDF Viewer supports showing and hiding scrollbar, when you feel to use the PDF Viewer only with the touch support, you can hide the default scrollbars of the PDF Viewer using the [ShowScrollbar](#). The following code example hides the scrollbar in the PDF Viewer control.

C#

```
//Initialize PDF Viewer.  
PdfViewerControl pdfViewer1 = new PdfViewerControl();  
//Load the PDF.  
pdfViewer1.Load("Sample.pdf");  
// Hiding the scrollbar of the PDF Viewer
```

```
pdfviewer1.ShowScrollbar = false;
```

VB.NET

```
'Initialize PDF Viewer.
Private pdfViewer1 As New PdfViewerControl()
'Load the PDF.
pdfViewer1.Load("Sample.pdf")
' Hiding the scrollbar of the PDF Viewer
pdfviewer1.ShowScrollbar = False
```

Acquire current page being displayed in WPF Pdf Viewer

PDF Viewer supports acquiring the index of the page being displayed in the PDF Viewer at any moment using the [CurrentPageIndex](#) property. The following code example illustrates the same.

C#

```
//Initialize PDF Viewer.
PdfViewerControl pdfViewer1 = new PdfViewerControl();
//Load the PDF.
pdfViewer1.Load("Sample.pdf");
// Acquiring the number of page being displayed in the Viewer
int pageCount = pdfviewer1.CurrentPageIndex;
```

VB.NET

```
'Initialize PDF Viewer.
Private pdfViewer1 As New PdfViewerControl()
'Load the PDF.
pdfViewer1.Load("Sample.pdf")
' Acquiring the number of page being displayed in the Viewer
Dim pageCount As Integer = pdfviewer1. CurrentPageIndex
```

Enable and Disable Notification bar in WPF Pdf Viewer

Notification bar is a part of the PDF Viewer that is used to display when an unexpected error occurs in the PDF Viewer control. You can suppress the display of the Notification bar by setting the [EnableNotificationBar](#) property to false. The following code example illustrate the same.

C#

```
//Initialize PDF Viewer.
PdfViewerControl pdfViewer1 = new PdfViewerControl();
//Load the PDF.
pdfViewer1.Load("Sample.pdf");
// Hiding the scrollbar of the PDF Viewer
pdfviewer1.EnableNotificationBar= false;
```

VB.NET

```
'Initialize PDF Viewer.
Private pdfViewer1 As New PdfViewerControl()
'Load the PDF.
pdfViewer1.Load("Sample.pdf")
```

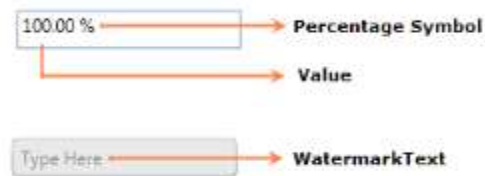
```
' Hiding the scrollbar of the PDF Viewer  
pdfviewer1.EnableNotificationBar= False
```

PercentTextBox

WPF Percent TextBox Overview

The [PercentTextBox](#) control restricts text box input to only double values and displays the percentage of the given value with support for data binding, Watermark, Null Value, and culture support. It provides many customization options to enhance the appearance and to suit the applications.

Control structure



Features

The core features of the PercentTextBox are as follows:

- Provides the ability to control the range of the 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 Null Value support.
- Provides culture support.

Getting Started with WPF Percent TextBox

This section explains how to create a WPF **PercentTextBox** 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.

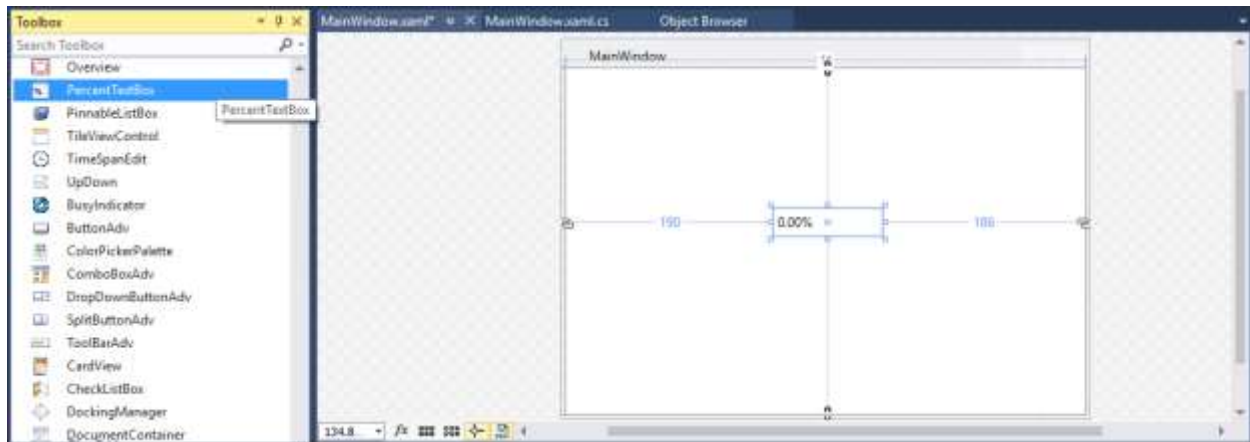
You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

Adding WPF PercentTextBox via designer

You can add the [PercentTextBox](#) 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 PercentTextBox via XAML

To add the PercentTextBox 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 **PercentTextBox** 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="PercentTextBoxSample.MainWindow"
Title="PercentTextBox Sample" Height="350" Width="525">
<Grid>
<!--Adding PercentTextBox control -->
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100" Height="25"
VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
</Window>
```

Adding WPF PercentTextBox via C\#

To add the PercentTextBox 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 PercentTextBox and add it to the window.

C#

```
//Creating an instance of PercentTextBox control
PercentTextBox percentTextBox = new PercentTextBox();
// Setting height and width to PercentTextBox
percentTextBox.Height = 25;
percentTextBox.Width = 100;
//Adding PercentTextBox as window content
this.Content = percentTextBox;
```



Setting Value

The percent value of the `PercentTextBox` can be set by using the `PercentValue` property.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100" Height="23"
PercentValue="100"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 23;
percentTextBox.PercentValue = 100;
```



Note: Do not use the `Text` property to set the percent value for the `PercentTextBox`. Use only the `PercentValue` 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 `PercentTextBox` using the `Value` Property.

The following code snippets illustrate the percent value binding from one `PercentTextBox` to another.

XML

```
<StackPanel>
<syncfusion:PercentTextBox x:Name="percentTextBox1" Height="25" Width="100"
PercentValue="{Binding MyValue, UpdateSourceTrigger=PropertyChanged}"/>
<syncfusion:PercentTextBox x:Name="percentTextBox2" Width="100" Height="25"
PercentValue="{Binding MyValue, UpdateSourceTrigger=PropertyChanged}" />
</StackPanel>
```

ViewModel.cs

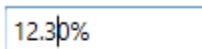
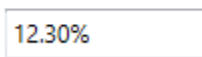
C#

```
class ViewModel : NotificationObject
{
private double myValue;
public double MyValue
```

```

{
    get
    {
        return myValue;
    }
    set
    {
        myValue = value;
        RaisePropertyChanged("MyValue");
    }
}

```

Value Changed Notification

The **PercentTextBox** control can notify the percent value changes through the [PercentValueChanged](#) event. You can get old percent value and new percent value from **OldValue** and **NewValue** properties in **PercentValueChanged** event.

XML

```

<syncfusion:PercentTextBox
    PercentValueChanged="PercentTextBox_PercentValueChanged"/>

```

C#

```

PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.PercentValueChanged += new
PropertyChangedCallback(PercentTextBox_PercentValueChanged);

```

You can handle the event as follows:

C#

```

private void PercentTextBox_PercentValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    // Get old and new percent value
    var newValue = e.NewValue;
    var oldValue = e.OldValue;
}

```

Min Max Value Restriction

The **PercentValue** of **PercentTextBox** 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 percent value between **MinValue** and **MaxValue**.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100" Height="25"
PercentValue="100" MaxValue="999.99" MinValue="-999.99"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
//Setting minimum value
percentTextBox.MinValue = -999.99;
//Setting maximum value
percentTextBox.MaxValue = 999.99;
percentTextBox.PercentValue = 100;
```


Step Interval to increase or decrease the value

The **PercentTextBox** control allows to increase or decrease the percent 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:PercentTextBox x:Name="percentTextBox" Width="150" Height="25"
PercentValue="8"
IsScrollingOnCircle="True" ScrollInterval="4"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.MinValue = 0;
percentTextBox.MaxValue = 100;
percentTextBox.PercentValue = 8;
percentTextBox.IsScrollingOnCircle = true;
percentTextBox.ScrollInterval = 4;
```


Formatting the value

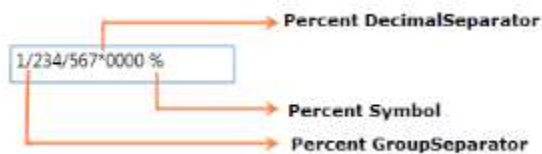
You can customize the number format by either setting the [NumberFormat](#) property or the [PercentGroupSeparator](#), [PercentGroupSizes](#), [PercentDecimalDigits](#), [PercentDecimalSeparator](#), [PercentNegativePattern](#), [PercentPositivePattern](#), and [PercentageSymbol](#) properties of **PercentTextBox**.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Height="25" Width="150"
PercentValue="1234567" PercentageSymbol="%" PercentDecimalDigits="4"
PercentDecimalSeparator="*" PercentGroupSeparator="/" />
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 1234567;
percentTextBox.PercentageSymbol = "%";
percentTextBox.PercentDecimalDigits = 4;
percentTextBox.PercentDecimalSeparator = "/";
percentTextBox.PercentGroupSeparator = "*";
```



Setting the Culture

The **PercentTextBox** provides support for globalization by using the [Culture](#) property. The **Culture** is used to format the decimal separator and group separator of the **PercentTextBox** percent value based on the respective culture.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Height="25" Width="150"
Culture="en-US" PercentValue="1234567"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 1234567;
percentTextBox.Culture = new System.Globalization.CultureInfo("en-US");
```

Note: When you use both **NumberFormat** and **Culture**, the **NumberFormat** will have a higher priority.

Theme

PercentTextBox supports various built-in themes. Refer to the below links to apply themes for the PercentTextBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

Changing Percent Value in WPF Percent TextBox

The [PercentTextBox](#) allows the user to change the percent value using the [PercentValue](#) property.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Height="25"
Width="150" PercentValue="10"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 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 percent value to the [PercentValue](#) property by binding, you can change the [PercentTextBox](#) percent value.

The following code snippets illustrate the percent value binding from one [PercentTextBox](#) to another.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox1" PercentValue="{Binding
MyValue, UpdateSourceTrigger=PropertyChanged}" Height="25" Width="100"/>
<syncfusion:PercentTextBox x:Name="percentTextBox2" PercentValue="{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 percent value by pasting the clipboard's text

By default, **PercentTextBox** 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,

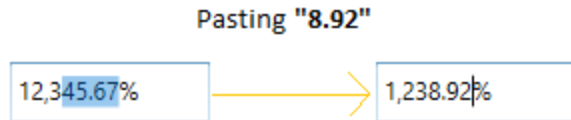
S.No	Action	Pasting behaviour in Advanced paste mode
1	When the whole value is selected	It simply replaces the whole value by copied value with the current number format.
2	When the cursor is at some position and the copied value does not contain a number decimal separator	It inserts the copied value 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:PercentTextBox PasteMode="Advanced"
PercentValue="12345.67"
Name="percentTextBox"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.PasteMode = PasteMode.Advanced;
percentTextBox.PercentValue = 12345.67;
```



Show UpDown Button

You can increment or decrement the percent value of `PercentTextBox` by setting the `ShowSpinButton` property value as `true`. Click `UpButton` to increment or `DownButton` to decrement the percent value. The default value of `ShowSpinButton` property is `false`.

XML

```
<syncfusion:PercentTextBox Height="30" Width="150" ShowSpinButton="True" />
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.ShowSpinButton = true;
```



Value Changed Event

The `PercentTextBox` control can notify changes in percent value through the [PercentValueChanged](#) event. In `PercentValueChanged` event, you can get old percent value and new percent value from the `OldValue` and `NewValue` properties.

XML

```
<syncfusion:PercentTextBox
PercentValueChanged="PercentTextBox_PercentValueChanged"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.PercentValueChanged += new
PropertyChangedCallback(PercentTextBox_PercentValueChanged);
```

You can handle the event as follows:

C#

```
private void PercentTextBox_PercentValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    // Get old and new percent value
    var newValue = e.NewValue;
    var oldValue = e.OldValue;
}
```


Setting the Null value

By default, the `PercentTextBox` control will display zero value when the `PercentValue` is set to `null`. You can use the [NullValue](#) and [UseNullOption](#) properties to show the null or any other percent value instead of zero.

The default value of the `NullValue` property is `null`, you can reset this to any other percent value. It will display only on setting the `UseNullOption` property is set to `true`.

NullValue = Null

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Height="25"
Width="100" UseNullOption="True" NullValue="{x:Null}"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.NullValue = null;
percentTextBox.UseNullOption = true;
```



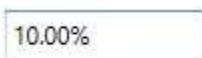
NullValue = 10

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Height="25"
Width="100" UseNullOption="True" NullValue="10"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.NullValue = 10;
percentTextBox.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 `Percent` value is `null` or empty, the control is not in focus and the `UseNullOption` property is `true`.

Setting the WatermarkText Foreground

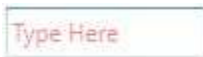
The `PercentTextBox` allows you to set the desired brush as a foreground for `WaterMarkText` using [WaterMarkTextForeground](#) property. The default color of `WaterMarkTextForeground` is Black.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100"
Height="25" UseNullOption="True" WatermarkText="Type here"
WatermarkTextIsVisible="True" WatermarkTextForeground="Red"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.UseNullOption = true;
percentTextBox.WatermarkText = "Type Here";
percentTextBox.WatermarkTextIsVisible = true;
percentTextBox.WatermarkTextForeground = Brushes.Red;
```



Setting Watermark Template

You can customize the Visual appearance of the `WatermarkText` by using the [WatermarkTemplate](#) property.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100" Height="25"
WatermarkText="Type Here" CornerRadius="3"
WatermarkTextIsVisible="True" WatermarkOpacity="0.5"
UseNullOption="True">
  <syncfusion:PercentTextBox.WatermarkTemplate >
    <DataTemplate>
      <Border Background="Red">
        <TextBlock Text="{Binding}" VerticalAlignment="Center" Margin="5,0,0,0"/>
      </Border>
    </DataTemplate>
  </syncfusion:PercentTextBox.WatermarkTemplate>
</syncfusion:PercentTextBox>
```



Note: The `UseNullOption` property must be enabled if you want to see `NullValue` or `WaterMarkText` in `PercentTextBox` control.

Note: If both `NullValue` and `WaterMarkText` are specified, you will only see `NullValue` but not `WaterMarkText`.

Step Interval in WPF Percent TextBox

The [PercentTextBox](#) control allows you to increase or decrease the percent 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 `PercentTextBox` control [PercentValue](#) 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 `PercentTextBox` control allows you to increase or decrease the `PercentValue` of `PercentTextBox` based on the `ScrollInterval` by pressing the up arrow and down arrow keys on the keyboard.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="150"
Height="25" PercentValue ="10" ScrollInterval="2"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 10;
percentTextBox.ScrollInterval = 2;
```



Change Value on Mouse Wheel


The `PercentTextBox` allows you to increase or decrease the `PercentValue` 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:PercentTextBox x:Name="percentTextBox" Width="150" Height="25"
PercentValue ="34"
IsScrollingOnCircle="True" ScrollInterval="3"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 34;
percentTextBox.IsScrollingOnCircle = true;
percentTextBox.ScrollInterval = 3;
```



Change Value on Click and Drag

The `PercentTextBox` allows you to increase or decrease the percent value based on the `ScrollInterval` by clicking and dragging the mouse when the `EnableExtendedScrolling` property is `true`.

`PercentTextBox` percent 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:PercentTextBox x:Name="percentTextBox" Width="120" Height="25"
PercentValue ="10"
ScrollInterval="5" EnableExtendedScrolling="True"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 120;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 10;
percentTextBox.ScrollInterval = 5;
percentTextBox.EnableExtendedScrolling = true;
```

**Allow or restrict selection on focus**

PercentTextBox 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:PercentTextBox x:Name="percentTextBox"
TextSelectionOnFocus="False"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.TextSelectionOnFocus = true;
```

TextSelectionOnFocus = "False"**TextSelectionOnFocus = "True"****Restriction or Validation in WPF Percent TextBox**

This section explains how to validate or restrict the [PercentTextBox](#) control value.

Restrict the value within minimum and maximum value

The [PercentValue](#) of the [PercentTextBox](#) can be restricted within the maximum and minimum limits. Once the percent 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 percent values by using the [MinValidation](#) and [MaxValidation](#) properties.

- **OnKeyPress** — When setting the **MaxValidation** or **MinValidation** to **OnKeyPress**, the percent value in the **PercentTextBox** will be validated shortly after pressing a key. So, it is not possible to provide any invalid input at all and the percent value does not exceed the maximum and minimum limits.
- **OnLostFocus** - When setting **MaxValidation** or **MinValidation** to **OnLostFocus**, the percent value in the **PercentTextBox** is validated, when the **PercentTextBox** loses the focus. That is, the **PercentTextBox** will accept any percent value, validation will only take place after the **PercentTextBox** has lost its keyboard focus. After validation, when the percent value of the **PercentTextBox** is greater than the **MaxValue** or less than the **MinValue**, the percent 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 percent value or reset to maximum limit that is specified. For example, if **MaxValue** is set to 100 and you are trying to input 200. **PercentValue** 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 percent value or reset to minimum limit that is specified. For example, if **MinValue** is set to 200 and the **PercentValue** is 205 and you are trying to change the percent value to 20. **PercentValue** will be changed to 200 when **MinValueOnExceedMinDigit** is true. When **MinValueOnExceedMinDigit** is false, Old percent value 205 will be retained.

Note: **MinValueOnExceedMinDigit** will be enabled only when the **MinValidation** is set to **OnKeyPress**.

XML

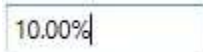
```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="150"
MaxValue="100" MinValue="10"
MinValueOnExceedMinDigit="True" MaxValueOnExceedMaxDigit="True"
MinValidation="OnKeyPress" MaxValidation="OnLostFocus"/>
```

C#

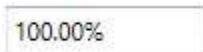
```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.MinValue = 10;
percentTextBox.MaxValue = 100;
percentTextBox.MinValidation = MinValidation.OnKeyPress;
```

```
percentTextBox.MaxValueValidation = MaxValidation.OnLostFocus;
percentTextBox.MinValueOnExceedMinDigit = true;
percentTextBox.MaxValueOnExceedMaxDigit = true;
```

MinValidation is set to **OnKeyPress**, it cannot let to enter a percent value less than the **MinValue**. If try to enter a percent value less than the **MinValue**, then the **MinValue** will set to the **PercentValue** 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 digit

You can format the decimal digits in the [PercentTextBox](#) control using [PercentDecimalDigits](#) property. You can restrict the decimal digits of text within maximum and minimum limits in [PercentTextBox](#) control using [MinPercentDecimalDigits](#) and [MaxPercentDecimalDigits](#) properties. The default value of [MinPercentDecimalDigits](#), [MaxPercentDecimalDigits](#) and [DoubleDecimalDigits](#) properties is -1.

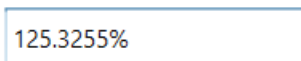
Note: If the value of [MinPercentDecimalDigits](#) property is greater than value of the [MaxPercentDecimalDigits](#) property, the text of [PercentTextBox](#) will be updated based on value of [MinPercentDecimalDigits](#) property.

XML

```
<syncfusion:PercentTextBox Value="125.32545" HorizontalAlignment="Center"
VerticalAlignment="Center"
MaxPercentDecimalDigits="4"
MinPercentDecimalDigits="1" />
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Value = 125.32545;
percentTextBox.MaxPercentDecimalDigits = 4;
percentTextBox.MinPercentDecimalDigits = 1;
```



When the value of [MinPercentDecimalDigits](#), [MaxPercentDecimalDigits](#) and [PercentDecimalDigits](#) properties are specified, [PercentDecimalDigits](#) property takes high precedence and updates the text of [PercentTextBox](#) property.


XML

```
<syncfusion:PercentTextBox Value="125.32545" HorizontalAlignment="Center"
VerticalAlignment="Center"
MaxPercentDecimalDigits="4"
MinPercentDecimalDigits="1" />
```

```
PercentDecimalDigits="3"
/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Value = 125.32545;
percentTextBox.MaxPercentDecimalDigits = 4;
percentTextBox.MinPercentDecimalDigits = 1;
percentTextBox.PercentDecimalDigits = 3;
```


Read only mode

The **PercentTextBox** 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 **PercentTextBox** by setting the [IsReadOnlyCaretVisible](#) property to **true**. However, value can be changed programmatically in readonly mode.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" IsReadOnly="True"
PercentValue="10" IsReadOnlyCaretVisible="True"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.PercentValue = 10;
percentTextBox.IsReadOnly = true;
percentTextBox.IsReadOnlyCaretVisible = true;
```


Culture and Formatting in WPF Percent TextBox

Value of **PercentTextBox** can be formatted in following ways:

- Culture
- NumberFormatInfo
- Dedicated properties (PercentGroupSeparator, PercentGroupSizes, PercentDecimalDigits, PercentDecimalSeparator)

Culture based formatting

The [PercentTextBox](#) provides support for globalization by using the [Culture](#) property. The **Culture** property is used to format the decimal separator and group separator of the **PercentTextBox** value based on the respective culture.

XML

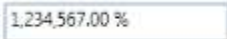
```
<syncfusion:PercentTextBox x:Name="percentTextBox" Height="25" Width="150"
```

```
Culture="bs-Latn" PercentValue="1234567"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 1234567;
//Setting Latin culture for percent textbox.
percentTextBox.Culture = new CultureInfo("bs-Latn");
```

By default the US culture uses “,” as the **PercentGroupSeparator** and “.” as the **PercentDecimalSeparator** where as the Latin culture uses “.” as the **PercentGroupSeparator** and “,” as the **PercentDecimalSeparator**.

Default Culture

Latin Culture

NumberFormatInfo based formatting

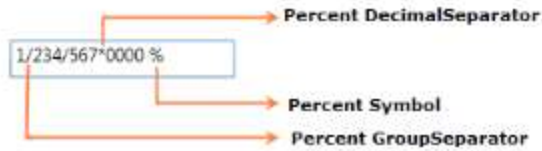
The number formatting of **PercentTextBox** can be customized by setting [NumberFormat](#) property.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Height="25" Width="150"
PercentValue="1234567">
  <syncfusion:PercentTextBox.NumberFormat >
    <numberformat:NumberFormatInfo PercentGroupSeparator="/"
    PercentSymbol="%"
    PercentDecimalDigits="4"
    PercentDecimalSeparator="*" />
  </syncfusion:PercentTextBox.NumberFormat>
</syncfusion:PercentTextBox>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 1234567;
percentTextBox.NumberFormat = new NumberFormatInfo()
{
    PercentGroupSeparator = "/",
    PercentDecimalDigits = 4,
    PercentDecimalSeparator = "*",
    PercentSymbol = "%"
};
```

The following code illustrate how to set percent group size by using the `NumberFormat` property.

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 123456789;
percentTextBox.NumberFormat = new NumberFormatInfo()
{
    PercentGroupSeparator = "/",
    PercentDecimalDigits = 4,
    PercentDecimalSeparator = "*",
    PercentSymbol = "%",
    // Adding the Number group size via NumberFormat property.
    PercentGroupSizes = new int[] { 2, 3, 4 }
};
```

12/345/67*0000 %

Formatting with dedicated properties

The number formatting of `PercentTextBox` can also be customized by setting the [PercentGroupSeparator](#), [PercentGroupSizes](#), [PercentDecimalDigits](#), [PercentDecimalSeparator](#), [PercentNegativePattern](#), [PercentPositivePattern](#), and [PercentageSymbol](#) properties of `PercentTextBox`. You can show the group separator by setting the [GroupSeperatorEnabled](#) property to `true`.

The following code illustrate how to format using the `PercentDecimalSeparator`, `PercentDecimalDigits`, `PercentGroupSeparator`, `PercentGroupSizes` property of the `PercentTextBox`.

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 123456789;
percentTextBox.GroupSeperatorEnabled = true;
percentTextBox.PercentageSymbol = "%";
percentTextBox.PercentDecimalDigits = 4;
percentTextBox.PercentDecimalSeparator = "/";
percentTextBox.PercentGroupSeparator = "*";
// Adding the percent group size via NumberGroupSizes property.
percentTextBox.PercentGroupSizes = new Int32Collection() { 4, 3, 2};
```

1234*567*89/0000%

Note: When you use both the `NumberFormat` and the dedicated properties (`PercentGroupSeparator`, `PercentageSymbol`, `PercentDecimalDigits`, `PercentDecimalSeparator` and `PercentGroupSizes`) to format the value of `PercentTextBox`, the `PercentGroupSeparator` and `PercentGroupSizes` 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 [PercentPositivePattern](#) property to customize the location of the percent symbol and the positive percent values. In the table below, "%" denotes the symbol of the percent, and "n" denotes the number.

PercentPositivePattern table

Value	Associated Pattern
0	n %
1	n%
2	%n
3	% n

`PercentPositivePattern = 0`

12.34 %

`PercentPositivePattern = 1`

12.34%

`PercentPositivePattern = 2`

%12.34

`PercentPositivePattern = 3`

% 12.34

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Height="25" Width="150"
PercentValue="1234" PercentPositivePattern="3"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.Value = 1234;
percentTextBox.PercentPositivePattern = 3;
```

% 1,234.00

Negative Value Pattern

You can use the [PercentNegativePattern](#) property to customize the location of the percent symbol and the negative percent values. In the table below, "%" denotes the symbol of the percent, and "n" denotes the number.

PercentNegativePattern 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- %

PercentNegativePattern = 0

-12.34 %

PercentNegativePattern = 1

-12.34%

PercentNegativePattern = 2

-%12.34

PercentNegativePattern = 3

%-12.34

PercentNegativePattern = 4

%12.34-

PercentNegativePattern = 5

12.34-%

PercentNegativePattern = 6

12.34%-

PercentNegativePattern = 7

-% 12.34

PercentNegativePattern = 8

12.34 %-

PercentNegativePattern = 9

% 12.34-

PercentNegativePattern = 10

% -12.34

PercentNegativePattern = 11

12.34- %

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Height="25" Width="150"
PercentValue="1234" PercentNegativePattern="7"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 150;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 1234;
percentTextBox.PercentNegativePattern = 7;
```



Appearance in WPF Percent TextBox

This section deals with the appearance of **PercentTextBox** control and contains the following topics.

Setting the Foreground

The **PercentTextBox** control **Foreground** can be modified based on the percent value of the control. The following are the foreground for **PercentTextBox** control.

Foreground for Positive Value

We can change a positive color for the percent value of **PercentTextBox** by setting the **PositiveForeground** property and it will be applied when the **PercentValue** is positive. The default color of **PositiveForeground** is Black.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100" Height="25"
PercentValue = "10" PositiveForeground="Blue" />
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 10;
percentTextBox.PositiveForeground = Brushes.Blue;
```


Foreground for Negative Value

We can change a negative color for the value of **PercentTextBox** by setting the **NegativeForeground** property and it will be applied when the **ApplyNegativeForeground** property is **true** and the **PercentValue** is negative. The default color of **NegativeForeground** is Red.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" PercentValue="10"
Width="100" Height="25" PercentValue = "-10"
NegativeForeground="SpringGreen" ApplyNegativeForeground="True" />
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.PercentValue = -10;
percentTextBox.ApplyNegativeForeground = true;
percentTextBox.NegativeForeground = Brushes.SpringGreen;
```


Foreground for Zero Value

We can change a zero color for the percent value of `PercentTextBox` by setting the [ZeroColor](#) property and it will be applied when the [ApplyZeroColor](#) property is `true` and the `PercentValue` is zero.

The default color of `ZeroColor` is `Green`.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" PercentValue="0"
Width="100" Height="25"
ApplyZeroColor="True" ZeroColor="DarkGoldenrod"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.PercentValue = 0;
percentTextBox.ApplyZeroColor = true;
percentTextBox.ZeroColor = Brushes.DarkGoldenrod;
```


Setting the Background

`PercentTextBox` 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:PercentTextBox x:Name="percentTextBox" Width="100"
Height="25" Background="Cyan"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.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 PercentTextBox, use [CornerRadius](#) property. The default value of [CornerRadius](#) property is 1.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100" Height="25"
    CornerRadius="5"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.CnerRadius = new CornerRadius(5);
```



Apply Background for Selection

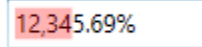
PercentTextBox allows different brushes to highlight the selected text by setting the [SelectionBrush](#) and [SelectionOpacity](#) properties. The [SelectionOpacity](#) property specifies the opacity of the [SelectionBrush](#).

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100" Height="25"
    SelectionBrush="Red" SelectionOpacity="0.5"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.SelectionBrush = Brushes.Red;
percentTextBox.SelectionOpacity = 0.3;
```



Align Value

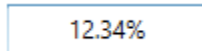
PercentTextBox allows to display the value from right or center or left side by setting the [TextAlignment](#) property to [Right](#) or [Left](#) or [Center](#). The Default value of [TextAlignment](#) is [Left](#).

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100" Height="25"
    TextAlignment="Center"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.TextAlignment = TextAlignment.Center;
```



Setting ToolTip

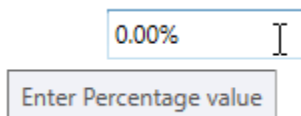
PercentTextBox provides support for ToolTip to display certain information when the mouse hovers on the PercentTextBox. You can customize the tooltip information by setting the [ToolTip](#) property.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" Width="100" Height="25"
ToolTip="Enter Percentage Value"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.Width = 100;
percentTextBox.Height = 25;
percentTextBox.ToolTip = "Enter Percentage value";
```



Theme

PercentTextBox supports various built-in themes. Refer to the below links to apply themes for the PercentTextBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Range Adorner in WPF Percent TextBox

[Value](#) of [PercentTextBox](#) can be visually indicated like a progress bar using range-adorner feature, this feature is disabled by default. You can show the adorner over PercentTextBox 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:PercentTextBox x:Name="percentTextBox" MinValue="0"
PercentValue="63" MaxValue="100" EnableRangeAdorner="True" />
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.MinValue = 0;
percentTextBox.MaxValue = 100;
percentTextBox.PercentValue = 63;
percentTextBox.EnableRangeAdorner = true;
```



Changing background of range-adorner

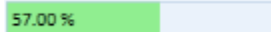
You can change the background color of the range adorner using [RangeAdornerBackground](#) property.

XML

```
<syncfusion:PercentTextBox x:Name="percentTextBox" MinValue="0"
MaxValue="100" PercentValue="57" EnableRangeAdorner="True"
RangeAdornerBackground="LightGreen"/>
```

C#

```
PercentTextBox percentTextBox = new PercentTextBox();
percentTextBox.MinValue = 0;
percentTextBox.MaxValue = 100;
percentTextBox.PercentValue = 57;
percentTextBox.EnableRangeAdorner = true;
percentTextBox.RangeAdornerBackground = Brushes.LightGreen;
```



PivotGrid

WPF Pivot Grid Overview

The [WPF Pivot Grid](#) is a powerful cell-oriented, extensible grid control which works with relational data. It simulates the pivot table feature of Microsoft Excel. The pivot grid pivots, summarizes, and groups data to organize it in a cross-tabulated form. It is used in financial domains to organize and analyze the business data.

Note: You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and displays the result in a cross-table format.

Getting Started with WPF Pivot Grid

Important

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

Adding pivot grid through designer

Open Visual Studio IDE and navigate to File > New > Project > WPF Application (inside Visual C# Templates) to create a new WPF application.

After creating the WPF application, go to View menu and select Toolbox option. Now, the toolbox will appear inside the Visual Studio IDE. Drag the pivot grid control from **Syncfusion BI WPF** category in the

toolbox to the designer page. The pivot grid control will be added along with the dependency assemblies automatically to the application.

![Adding pivot grid through designer](PivotGrid-Getting-Started-images/Adding PivotGrid through designer.png)

Adding pivot grid through XAML

Open Visual Studio IDE and navigate to File > New > Project > WPF Application (inside Visual C# Templates) to create a new WPF application.

Next, you should add the following dependency assemblies to your WPF application. To add them to your WPF application, right-click **References** in the Solution Explorer and select **Add Reference**. Now, in the **Reference Manager** dialog, under **Assemblies > Extension**, the following mentioned Syncfusion assemblies are found.

- Syncfusion.Grid.Wpf
- Syncfusion.GridCommon.Wpf
- Syncfusion.Linq.Base
- Syncfusion.PivotAnalysis.Base
- Syncfusion.PivotAnalysis.Wpf
- Syncfusion.Shared.Wpf

Note: You can also get the assemblies by browsing to the default assembly location.

{System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\\{version number}\precompiledassemblies\\{version number}

Then, define the pivot grid control in MainWindow.xaml as shown in the following code sample.

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:PivotGridControl Name="pivotGrid" HorizontalAlignment="Left"
Margin="5,0,0,0" VerticalAlignment="Top" Height="50" Width="100"/>
</Grid>
</Window>
```

Adding pivot grid through code-behind

Open Visual Studio IDE and navigate to File > New > Project > WPF Application (inside Visual C# Templates) to create a new WPF application.

Add the following dependency assemblies to your WPF application. To add them to your WPF application, right-click **References** in the Solution Explorer and select **Add Reference**. Now, in the **Reference Manager** dialog, under **Assemblies > Extension**, the following mentioned Syncfusion assemblies are found.

- Syncfusion.Grid.Wpf

- Syncfusion.GridCommon.Wpf
- Syncfusion.Linq.Base
- Syncfusion.PivotAnalysis.Base
- Syncfusion.PivotAnalysis.Wpf
- Syncfusion.Shared.Wpf

Note: You can also get the assemblies by browsing to the default assembly location

{System Drive}:\Program Files (x86)\Syncfusion\Essential Studio\\{version number}\precompiledassemblies\\{version number}

Then, add a grid container in MainWindow.xaml file to hold the pivot grid control created and added from code-behind.

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 Name="grid1">
</Grid>
</Window>
```

Include the *Syncfusion.Windows.Controls.PivotGrid* namespace in the MainWindow.xaml.cs file to create a pivot grid control in code-behind.

C#

```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Syncfusion.Windows.Controls.PivotGrid;
```

Raise the **Loaded** event of the MainWindow and within that event, create a new instance of the PivotGridControl class. This instance should be added as a child of the grid present inside the MainWindow.

C#

```
public MainWindow() {
InitializeComponent();
this.Loaded += MainWindow_Loaded;
}
void MainWindow_Loaded(object sender, RoutedEventArgs e) {
PivotGridControl pivotGrid = new PivotGridControl();
this.grid1.Children.Add(pivotGrid);
}
```

Adding pivot grid through expression blend

Open Blend for Visual Studio and navigate to File > New Project > WPF > WPF Application to create a new WPF application.

Select the **Projects** tab available in the top-left corner of the Blend IDE, right-click the **References**, and select **Add Reference**. Now, browse and add the following assemblies to the project.

- Syncfusion.Grid.Wpf
- Syncfusion.GridCommon.Wpf
- Syncfusion.Linq.Base
- Syncfusion.PivotAnalysis.Base
- Syncfusion.PivotAnalysis.Wpf
- Syncfusion.Shared.Wpf

Note: You can also get the assemblies by browsing to the default assembly location

{System Drive}\Program Files (x86)\Syncfusion\Essential Studio\\{version number}\precompiledassemblies\\{version number}

After adding the above assemblies, the pivot grid control will be automatically added under **Assets** tab.

Now, choose the **Assets** tab and drag the pivot grid control from the toolbox to designer section.

![Adding pivot grid through expression blend](PivotGrid-Getting-Started-images/Adding PivotGridControl through expression blend.png)

Binding a data source to pivot grid control

Right-click the project in the solution explorer and select **Add > New item**. In the **Add New Item** window, choose **Class** option and provide the name of the class as *ProductSales.cs* and click **OK**.

The ItemSource for the pivot grid control will be initialized in this file. Refer to the following code snippet.

C#

```
public class ProductSales {
    public string Product {
        get;
        set;
    }
    public string Date {
        get;
        set;
    }
    public string Country {
        get;
        set;
    }
    public string State {
        get;
        set;
    }
    public int Quantity {
        get;
        set;
    }
    public double Amount {
        get;
        set;
    }
}
```

```

public static ProductSalesCollection GetSalesData() {
    /// Geography
    string[] countries = new string[] {
        "Canada"
    };
    string[] canadaStates = new string[] {
        "Alberta",
        "British Columbia",
        "Ontario"
    };
    /// Time
    string[] dates = new string[] {
        "FY 2005",
        "FY 2006",
        "FY 2007"
    };
    /// Products
    string[] products = new string[] {
        "Bike",
        "Car"
    };
    Random r = new Random(123345345);
    int numberOfRecords = 2000;
    ProductSalesCollection listOfProductSales = new ProductSalesCollection();
    for (int i = 0; i < numberOfRecords; i++) {
        ProductSales sales = new ProductSales();
        sales.Country = countries[r.Next(0, countries.GetLength(0))];
        sales.Quantity = r.Next(1, 12);
        /// 1 percent discount for 1 quantity
        double discount =
            (30000 * sales.Quantity) * (double.Parse(sales.Quantity.ToString()) / 100);
        sales.Amount = (30000 * sales.Quantity) - discount;
        sales.Date = dates[r.Next(r.Next(dates.GetLength(0) + 1))];
        sales.Product = products[r.Next(r.Next(products.GetLength(0) + 1))];
        sales.State = canadaStates[r.Next(canadaStates.GetLength(0))];
        listOfProductSales.Add(sales);
    }
    return listOfProductSales;
}

public override string ToString() {
    return string.Format("{0}-{1}-{2}", this.Country, this.State, this.Product);
}

public class ProductSalesCollection: List < ProductSales > {
}
}

```

Above mentioned GetSalesData method is used to get the collection that needs to be populated in the pivot grid control. Now, bind the collection to the pivot grid control as its ItemSource. It can be done through XAML or code-behind.

To initialize the ItemSource through **XAML**, ObjectDataProvider is used. Refer to the following code.

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" Title="MainWindow" Height="350"
Width="525" xmlns:local="clr-namespace:WpfApplication1">
<Window.Resources>
<ResourceDictionary>
<ObjectDataProvider x:Key="data" ObjectType="{x:Type local:ProductSales}"
MethodName="GetSalesData" />
</ResourceDictionary>
</Window.Resources>
<Grid>
<syncfusion:PivotGridControl Name="pivotGrid" HorizontalAlignment="Left"
VerticalAlignment="Top" ItemSource="{Binding Source={StaticResource data}}"
VisualStyle="Metro">
</syncfusion:PivotGridControl>
</Grid>
</Window>

```

To initialize the ItemsSource through code-behind, refer to the following code snippet.

C#

```

public MainWindow() {
InitializeComponent();
this.Loaded += MainWindow_Loaded;
}
void MainWindow_Loaded(object sender, RoutedEventArgs e) {
PivotGridControl pivotGrid = new PivotGridControl();
this.grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
}

```

Defining the PivotItems and PivotComputations

PivotItem is a container for items in the pivot grid control, and an item can be either a **PivotRow** or **PivotColumn**. Refer to the topic [1.3.1 Pivot Item](#) for more details. **PivotComputationInfo** holds the value fields for the pivot grid control, and it does have different types of custom calculations. You can summarize the values depends on your requirement and refer to the topic [1.3.2 PivotComputationInfo](#) for more details. PivotItems and PivotComputations can be defined through XAML and code-behind.

To define the PivotItems and PivotComputations through XAML, refer to the following code snippet.

XML

```

<Grid>
<syncfusion:PivotGridControl Name="pivotGrid" HorizontalAlignment="Left"
VerticalAlignment="Top" ItemSource="{Binding Source={StaticResource data}}"
VisualStyle="Metro">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />

```

```

<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

To define through code-behind, include the *Syncfusion.PivotAnalysis.Base* namespace in *MainWindow.xaml.cs*. Refer to the following code snippet.

C#

```

using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Syncfusion.Windows.Controls.PivotGrid;
using Syncfusion.PivotAnalysis.Base;

```

C#

```

public MainWindow() {
InitializeComponent();
this.Loaded += MainWindow_Loaded;
}
void MainWindow_Loaded(object sender, RoutedEventArgs e) {
PivotGridControl pivotGrid = new PivotGridControl();
grid1.Children.Add(pivotGrid);
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", SummaryType =
SummaryType.Count
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {

```

```
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.ItemSource = ProductSales.GetSalesData();
}
```

Run the application, the following output will be generated.

![Pivot grid loaded with values](PivotGrid-Getting-Started-images/PivotGrid with Populated values.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Data Binding in WPF Pivot Grid

Binding pivot grid to list

The [WPF Pivot Grid](#) allows binding a list based collection as its item source. Refer to this topic [Binding a data source to pivot grid control](#) to learn more.

Binding pivot grid to data table

The pivot grid allows binding a DataSet from the data table as its item source.

To do so, initially you should create a connection to the database. Then, retrieve and store the required data to a DataSet through SQL queries. Refer to the following code sample.

C#

```
public static DataView GetOrderDetails() {
    try {
        DataSet ds = new DataSet();
        using(SqlCeConnection con = new SqlCeConnection(@"
DataSource=Data\Northwind.sdf")) {
            con.Open();
            SqlCeCommand cmd = new SqlCeCommand(@"SELECT Top(100) [Order ID], [Product
ID], [Unit Price], Quantity FROM [Order Details] ", con);
            SqlCeDataAdapter da = new SqlCeDataAdapter(cmd);
            da.Fill(ds);
            //Creation of DataView
            DataView dataView = new DataView(ds.Tables[0], "[Unit Price] >= 20", "[Order
ID]", DataViewRowState.CurrentRows);
            return dataView;
        }
    }
}
```

Then, bind the DataSet as ItemSource to the pivot grid by invoking the appropriate method. It can be done through XAML or code-behind.

For XAML, refer to the following code sample.

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" xmlns:local="clr-namespace:WpfApplication1">
<Window.Resources>
<ResourceDictionary>
<ObjectDataProvider x:Key="data" ObjectType="{x:Type local:OrderDetails}"
MethodName="GetOrderDetails" />
</ResourceDictionary>
</Window.Resources>
<Grid>
<syncfusion:PivotGridControl Margin="5" Grid.Row="2" x:Name="pivotGrid1"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="OID" FieldMappingName="Order ID"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="PID" FieldMappingName="Product ID"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo FieldName="Unit Price"
CalculationName="UnitTotal" SummaryType="DoubleTotalSum" Format="#.00" />
<syncfusion:PivotComputationInfo FieldName="Quantity"
CalculationName="QuantityCalc" SummaryType="Count" Format="C" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
</Window>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
public MainWindow() {
InitializeComponent();
//Binding DataView to the ItemSource
this.pivotGrid1.ItemSource = OrderDetails.GetOrderDetails();
this.pivotGrid1.PivotColumns.Add(new PivotItem {
FieldHeader = "OrderID", FieldMappingName = "Order ID", TotalHeader =
"Total"
});
this.pivotGrid1.PivotRows.Add(new PivotItem {
FieldHeader = "ProductID", FieldMappingName = "Product ID", TotalHeader =
"Total"
});
this.pivotGrid1.PivotCalculations.Add(new PivotComputationInfo {
CalculationName = "UnitPrice", FieldName = "Unit Price", Format = "C",
SummaryType = SummaryType.DecimalTotalSum
});
this.pivotGrid1.PivotCalculations.Add(new PivotComputationInfo {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
});
}
}

```



```
}

```

Note: You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

PivotItem in WPF Pivot Grid

A pivot item is a container for items in the pivot grid control. An item in a pivot table field provides the information needed to define an item. The items are individual data entries in a field category. The pivot item object is a member of the pivot items collection and consists of the following fields.

- **FieldHeader:** Gets or sets the title you want to view in the header for the pivot item.
- **FieldMappingName:** Gets or sets the pivot item property's mapping name.
- **TotalHeader:** Gets or sets the string you want to append to the pivot item's summary cells.
- **Comparer:** Gets or sets the IComparer object used for sorting. If this value is null, then sorting will be performed under the assumption that this field is IComparable.
- **Format:** Gets or sets the format string for the specified field.
- **ShowSubTotal:** Gets or sets whether the subtotal for this item can be shown or hidden.
- **AllowRunTimeGroupByField:** Gets or sets a value to enable or disable grouping for the pivot item.
- **AllowFilter:** Enables or disables filtering for the pivot item.
- **AllowSort:** Enables or disables sorting for the pivot item.

Defining pivot item in XAML

Create a new pivot item by using the `PivotGridControl.PivotItem` class. A pivot item can be either a `PivotRow` or `PivotColumn`. Refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl Name="pivotGrid" HorizontalAlignment="Left"
VerticalAlignment="Top" ItemSource="{Binding Source={StaticResource data}}"
VisualStyle="Metro">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
</syncfusion:PivotGridControl>
</Grid>
```

Defining pivot item in code-behind

Include the `Syncfusion.PivotAnalysis.Base` in the `MainWindow.xaml.cs` file, and then define the pivot item for row and column. Add the defined pivot item to the pivot rows and pivot columns collections of the pivot grid control. Refer to the following code sample.

C#

```
public MainWindow() {
```

```

InitializeComponent();
this.Loaded += MainWindow_Loaded;
}
void MainWindow_Loaded(object sender, RoutedEventArgs e) {
PivotGridControl pivotGrid = new PivotGridControl();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
}

```

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

PivotComputationInfo in WPF Pivot Grid

Pivot computation holds the information needed for calculations that appear in the pivot grid control. For each calculation, there is an associated **PivotComputationInfo** object added to the PivotCalculations collection.

Properties

- **CalculationName:** Gets or sets what needs to be displayed in the pivot table if more than one calculation is included in the pivot grid.
- **Description:** Gets or sets the description of the calculation.
- **FieldName:** Gets or sets the name of the property to be used in this calculation.
- **FieldHeader:** Gets or sets the title you want to view in the header.
- **Format:** Gets or sets the format string to be used, to format the calculation results in the pivot grid.
- **Summary:** Gets or sets the SummaryBase object used to define the calculation. When you specify SummaryType.Custom, set summary as an instance of your custom SummaryBase-derived object.
- **SummaryType:** Gets or sets the SummaryType enumeration for the calculation. You can set some enumeration values to the summary type; otherwise, by default it is set as count.
- **AllowRunTimeGroupByField:** Gets or sets the value to enable/disable grouping for the pivot item.
- **DisplayOption:** Gets or sets the calculation values to be displayed in pivot grid.
- **CalculationType:** Gets or sets the CalculationType enumeration for this computation object and defines how to make the computational objects in pivot grid visible.
- **AllowFilter:** Gets or sets whether the calculation column can be filtered when RowPivotsOnly is true in the PivotEngine.
- **AllowSort:** Gets or sets whether the calculation column can be sorted when RowPivotsOnly is true in the PivotEngine.
- **BaseItem:** Gets or sets the value of the BaseItem for calculations.

- **BaseField:** Gets or sets the base field value for calculations.
- **EnableHyperlinks:** Gets or sets whether the calculation column should be hyperlinked when RowPivotsOnly is true in the PivotEngine.
- **InnerMostComputationsOnly:** Gets or sets whether the aggregation results appear only for the innermost level.
- **PadString:** Gets or sets the PadString that is used when SummaryType is DisplayIfDiscreteValuesEqual.

Defining PivotComputationInfo in XAML

Create a new PivotComputationInfo item using the `PivotGridControl.PivotComputationInfo` class and it can be added to a PivotCalculations collection.

Refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl Name="pivotGrid" HorizontalAlignment="Left"
Margin="131,131,0,0" VerticalAlignment="Top" Height="48" Width="117">
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldHeader="Quantity" FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

Defining PivotComputationInfo in code-behind

Include the *Syncfusion.PivotAnalysis.Base* in the MainWindow.xaml.cs file, and then define the PivotComputationInfo's for each calculation items. Add the defined PivotComputationInfo's to PivotCalculations collection of the pivot grid control.

Refer to the following code sample.

C#

```
public MainWindow() {
InitializeComponent();
this.Loaded += MainWindow_Loaded;
}
void MainWindow_Loaded(object sender, RoutedEventArgs e) {
PivotGridControl pivotGrid = new PivotGridControl();
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Total", FieldName = "Quantity", FieldHeader = "Quantity",
SummaryType = SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
}
```

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Summary Types in WPF Pivot Grid

The pivot grid summarizes the data for various data types by using the `SummaryType` enumerator. `SummaryType` should be defined while defining the pivot calculation using `PivotComputationInfo` class to specify the type of summary. It holds the following summary types.

- **DoubleTotalSum**: Computes the sum of double or integer in pivot values for corresponding pivot item.
- **DoubleAverage**: Computes average of double or integer in pivot values for corresponding pivot item.
- **DoubleMaximum**: Computes maximum of double or integer in pivot values for corresponding pivot item.
- **DoubleMinimum**: Computes minimum of double or integer in pivot values for corresponding pivot item.
- **DoubleStandardDeviation**: Computes the standard deviation of double or integer in pivot values for corresponding pivot item.
- **DoubleVariance**: Computes the variance of double or integer in pivot values for corresponding pivot item.
- **Count**: Computes count of double or integer from pivot values for corresponding pivot item.
- **DecimalTotalSum**: Computes the sum of decimal in pivot values for corresponding pivot item.
- **IntTotalSum**: Computes the sum of integer in pivot values for corresponding pivot item.
- **Custom**: Specifies that you are using a custom summary base object to define the calculation.
- **DisplayIfDiscreteValuesEqual**: Displays the aggregated value in the pivot computation column if all the values are common.

`SummaryType` property can be set for the corresponding pivot calculation item through `PivotComputationInfo` class. It can be set through XAML or code-behind.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" ItemSource="{Binding Source={StaticResource
data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

```
</syncfusion:PivotGridControl.PivotCalculations>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", SummaryType =
            SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
    }
}
```

![PivotGrid shows the values as double total sum and count format](Summary-Images/PivotGrid shows summary type Double variance.png)

[DisplayIfDiscreteValuesEqual](#) summary type in pivot grid

DisplayIfDiscreteValuesEqual is a new summary type added to the **SummaryType** enumerator of the pivot grid control. This summary type displays the aggregated value in the pivot calculation column if all the values are common, else the default value will be displayed as "**".

You can change the default value to any custom string of your choice by using the **PadString** property.

Set the SummaryType as **DisplayIfDiscreteValuesEqual** along with value for the **PadString** while defining pivot calculations in the pivot grid control.

For setting these properties through XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="DisplayIfDiscreteValuesEqual"
PadString="****" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For setting these properties through code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
```

```

pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", SummaryType =
    SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    SummaryType.DisplayIfDiscreteValuesEqual, PadString = "****"
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
}
}

```

![[PivotGrid Shows DisplayIfDiscretevaluesequal summary type](Summary-Images/PivotGrid Shows DisplayIfDiscretevaluesequal summary type.png)]

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Custom Summary in WPF Pivot Grid

The pivot grid enables you to set the custom summaries for the PivotItem values at both load time and runtime (using pivot computation info dialog).

To create a custom SummaryBase class, add a new class and inherit the abstract class **SummaryBase**. Implement your summary logics by overriding Combine(), CombineSummary(), GetResult(), GetInstance(), and Reset() methods.

Refer to the following code sample.

C#

```

public class MyCustomSummaryBase1: SummaryBase {
    public MyCustomSummaryBase1() {}
    private double mTotalValue;
    public override void Combine(object other) {
        mTotalValue += (double) other;
    }
    public override void CombineSummary(SummaryBase other) {
        MyCustomSummaryBase1 dpsb = other as MyCustomSummaryBase1;
        if (null != dpsb) {
            mTotalValue += dpsb.mTotalValue;
        }
    }
    public override SummaryBase GetInstance() {
        return new MyCustomSummaryBase1();
    }
    public override object GetResult() {
        return mTotalValue / 3.33333;
    }
    public override void Reset() {
        mTotalValue = 0;
    }
}

```

```

}
}
public class MyCustomSummaryBase2: SummaryBase {
private double mTotalValue;
public override void Combine(object other) {
mTotalValue += (double) other;
}
public override void CombineSummary(SummaryBase other) {
MyCustomSummaryBase2 dpsb = other as MyCustomSummaryBase2;
if (null != dpsb) {
mTotalValue += dpsb.mTotalValue;
}
}
public override SummaryBase GetInstance() {
return new MyCustomSummaryBase2();
}
public override object GetResult() {
return mTotalValue / 5.5555;
}
public override void Reset() {
mTotalValue = 0;
}
}
}

```

Defining custom summary in load time

You can define your own custom SummaryBase to PivotCalculations in pivot grid by setting the instance of the custom summary in the Summary property. This custom summary can be used only if you set the SummaryType of that PivotCalculation as "Custom". It can be done through XAML or code-behind.

For XAML, refer to the following code sample.

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" xmlns:local="clr-namespace:WpfApplication1">
<Window.Resources>
<ResourceDictionary>
<ObjectDataProvider x:Key="data" ObjectType="{x:Type local:ProductSales}"
MethodName="GetSalesData" />
<local:MyCustomSummaryBase1 x:Key="summary1" />
</ResourceDictionary>
</Window.Resources>
<Grid Name="grid1">
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>

```



```

<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="Custom" Summary="{StaticResource summary1}" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
</Window>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.Custom, Summary = new MyCustomSummaryBase1()
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
    }
}

```

Defining custom summary in runtime

Create an object for all the custom SummaryBase class and set the add objects to the CustomSummaryBaseCollection property of the pivot schema designer. This property is an ObservableCollection type of SummaryBase that enables users to add more than one class object. Each object is considered as a unique custom SummaryBase.

Using the CustomSummaryBaseCollection property, you can set the summary for the respective columns by its Summary property.

Refer to the following code sample.

C#

```
public partial class MainWindow: Window {
    public MainWindow() {
        InitializeComponent();
        this.pivotGrid.Loaded += pivotGrid_Loaded;
    }
    void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
        /// Adding Custom SummaryBases to the CustomSummaryBaseCollection property
        this.pivotSchemaDesigner.CustomSummaryBaseCollection = new
        System.Collections.ObjectModel.ObservableCollection < SummaryBase > {
            new MyCustomSummaryBase1(),
            new MyCustomSummaryBase2()
        };
    }
}
```

To set custom summary at runtime, double-click the items from the pivot schema designer that will pop-up the pivot computation information dialog box. In the summarize value by combo box, you can select the predefined custom summaries as required.

![PivotGrid with custom summaries](RunTime-custom-summary-images/PivotGrid with custom summaries.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Custom Calculations in WPF Pivot Grid

CalculationType is an enumerator defined in the PivotComputationInfo class that is used to specify the type of calculation. The various types of calculations are:

- **NoCalculation:** Removes the custom calculations and restores to original values (default value). Displays the pivot values as default value.
- **PercentageOfGrandTotal:** Displays a value cell as a percentage of grand total of all value cells of PivotEngine.
- **PercentageOfColumnTotal:** Displays all value cells in each column as a percentage of its corresponding column total.
- **PercentageOfRowTotal:** Displays all value cells in each row as a percentage of its corresponding row total.
- **PercentageOfParentColumnTotal:** Displays a value cell as a percentage of parent column item values.

- **PercentageOfParentRowTotal:** Displays a value cell as a percentage of parent row item values.
- **PercentageOfParentTotal:** Displays a value cell as a percentage of base field (parent row/column total).
- **Index:** Displays a value cell as an index value based on PivotEngine generation.
- **Formula:** Displays a calculation based on a well formed algebraic expression involving other calculations.
- **PercentageOf:** Displays values as a percentage of the value of the base item in the base field.
- **DifferenceFrom:** Displays values as the difference from the value of the base item in the base field.
- **PercentageOfDifferenceFrom:** Displays values as the percentage difference from the value of the base item in the base field.
- **RunningTotalIn:** Displays the value for successive items in the base field as a running total.
- **PercentageOfRunningTotalIn:** Calculates the value for successive items in the base field that are displayed as a running total as a percentage.
- **RankSmallestToLargest:** Displays the rank of selected values in a specific field and lists the smallest item in the field as 1 and each larger value as a higher rank value.
- **RankLargestToSmallest:** Displays the rank of selected values in a specific field and lists the largest item in the field as 1 and each smaller value as a higher rank value.
- **Distinct:** Displays the subtotals based on the distinct values of BaseItem defined for the calculation item.

To achieve this, the `CalculationType` property is set for the corresponding pivot calculation item through the `PivotComputationInfo` class. It can be set through XAML or code-behind.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum"
CalculationType="PercentageOfColumnTotal" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.DoubleTotalSum, CalculationType =
            CalculationType.PercentageOfColumnTotal
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
    }
}
```

![To display the pivot calculation as percentage of column total format](Calculation-Type-images/PivotGrid shows percentageofcolumn total calculation type.png)

Providing expression field calculation for summaries

To provide a calculated field support to summary cells in the pivot grid and make it behave accordingly when options such as sum, count, maximum, and minimum are provided in summaries, set the calculation type to "Formula" and specify the appropriate formula. Refer to the following code samples and screenshots.

XML

```
<Grid>
```

```

<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="UnitPrice" CalculationType="Formula" Formula="[Amount] /
[Quantity]" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum

```

```

};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
PivotComputationInfo m_PivotComputationInfo2 = new PivotComputationInfo() {
    CalculationName = "Total", FieldName = "UnitPrice", CalculationType =
    CalculationType.Formula, Formula = "[Amount] / [Quantity]"
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo2);
}
}

```

! [To display the pivot calculation values based on given formula] (Calculation-Type-images/PivotGrid shows formula calculation type.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Expression Fields in WPF Pivot Grid

The [WPF Pivot Grid](#) supports adding expression fields and handles other features similar to normal fields. You can add new expression field information by using the `AllowedFields` class in the pivot grid control and you can use the following property to load the allowed fields in the pivot grid:

- **LoadWithDefaultPropertyFields:** Gets or sets a value indicating whether the pivot grid control loads with default property fields for expression support.

This property can be defined either in XAML or code-behind. For XAML, refer to the following code sample.

XAML

```

<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
    VerticalAlignment="Top" VisualStyle="Metro"
    LoadWithDefaultPropertyFields="True" ItemSource="{Binding
    Source={StaticResource data}}" >
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
    TotalHeader="Total" ShowSubTotal="False" />
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
    TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
    TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
    TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>

```

```

<syncfusion:PivotComputationInfo CalculationName = "Total" FieldName =
"Amount" Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName = "Total" FieldName =
"Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.LoadWithDefaultPropertyFields = true;
    }
}

```

Creating the expression fields

FieldInfo class is used to create the expression fields that are added to the **AllowedFields** collection of pivot grid control. This can be done either in XAML or code-behind.

For XAML, refer to the following code sample.

XML

```

<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro"
LoadWithDefaultPropertyFields="True" ItemSource="{Binding
Source={StaticResource data}}" >
<syncfusion:PivotGridControl.AllowedFields>
<syncfusion:FieldInfo Name="UnitPrice" Expression="[Amount] + [Quantity]"
FieldType="Expression" />
</syncfusion:PivotGridControl.AllowedFields>
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total"/>
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total"/>
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName = "Total" FieldName =
"Amount" Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName = "Total" FieldName =
"Quantity" SummaryType="Count" />
<syncfusion:PivotComputationInfo CalculationName = "Total" Description =
"Summation of values" FieldName = "UnitPrice" FieldHeader="Unit Price"
Format = "C" SummaryType="DoubleTotalSum"/>
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", ShowSubTotal = false,
TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows

```



```

pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
    = Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    Syncfusion.PivotAnalysis.Base.SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
FieldInfo field1 = new FieldInfo {
    Name = "UnitPrice", Expression = "[Amount] + [Quantity]", FieldType =
    FieldTypes.Expression
};
pivotGrid.AllowedFields.Add(field1);
PivotComputationInfo m_PivotComputationInfo2 = new PivotComputationInfo() {
    CalculationName = "UnitPrice", FieldName = "Unit Price", Format = "C",
    SummaryType = Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo2);
}
}

```

![Adding the expression fields in PivotGrid](Expression-Field-Images/PivotGrid shows expression fields.png)

Note: You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Normal Mode

Filtering in WPF Pivot Grid

Filtered data displays only a subset of data that meets a specified criterion. **Pivot grid filters** are automatically reapplied every time the pivot grid is refreshed or updated until you remove those filters. In the pivot grid, filters are additive, which means that each additional filter is based on the current filter and reduces the subset of data. You can apply 'n' number of filtering conditions to the pivot grid at a time. While applying filters, a **FilterExpression** is created and data is filtered based on that specified FilterExpression.

Filtering by FilterExpression

FilterExpression class is used to encapsulate the information needed to define a filter. It contains the following properties:

- **Expression:** Gets or sets the well-formed logical expression that defines this FilterExpression.
- **Name:** Gets or sets the name of the FilterExpression.
- **DimensionName:** Gets or sets the dimension name for the FilterExpression.
- **DimensionHeader:** Gets or sets the dimension header for the FilterExpression.
- **Format:** Gets or sets the format of FilterExpression.
- **Evaluator:** Evaluates the given value.

Defining filters in XAML

Create a new **FilterExpression** under the **PivotGridControl.Filters** class. Refer to the following code snippet.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" ShowFieldList="True" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotGridControl.Filters>
<syncfusion:FilterExpression DimensionHeader="Product"
DimensionName="Product" Name="Product" Expression="Product = Bike" />
</syncfusion:PivotGridControl.Filters>
</syncfusion:PivotGridControl>
</Grid>
```

Defining filters in code-behind

Create a new **FilterExpression** by using the **FilterExpression** class and add that **FilterExpression** to the **Filters** collection of the pivot grid control.

Refer to the following code snippet.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
```

```

FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
FilterExpression m_FilterExpression = new FilterExpression() {
DimensionHeader = "Product", Name = "Product", DimensionName = "Product",
Expression = "Product = Bike"
};
//Adding FilterExpression to Filters
pivotGrid.Filters.Add(m_FilterExpression);
}
}

```

![To filter the data values by using filter expression](Filtering-by-values-images/PivotGrid shows the filtered values.png)

Filtering by using filter pop-up

Filters can also be applied to pivot grid control at runtime using the filter pop-up. This filter pop-up can be opened by clicking the filter button in the grouping bar item. The filter pop-up contains the filter list in which you can uncheck the items that are to be filtered and click OK.

This, in turns, creates a FilterExpression at runtime by using the unchecked items and applies the filters to the pivot grid control.

![To filter the data values by using filter popup](Filtering-by-values-images/PivotGrid with FilterPopup.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and displays the result in a cross-table format.

Excel-Like Filtering in WPF Pivot Grid

The [WPF Pivot Grid](#) control provides Excel-like sorting and filtering features applied to a pivot item. You can enable or disable the Excel-like sorting and filtering pop-up in the pivot grid by setting the `AllowMultiFunctionalSortFilter` property of grouping bar in the pivot grid control.

To do so, define the pivot grid control and raise the loaded event for pivot grid. Inside the *PivotGridLoaded()* event, *raise the loaded event for the grouping bar*. Inside the *GroupingBarLoaded()* event, set the value of the *AllowMultiFunctionalSortFilter* property to "true".

Refer to the following code sample.

C#

```
public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.Loaded += pivotGrid_Loaded;
    }
    void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
        pivotGrid.GroupingBar.Loaded += GroupingBar_Loaded;
    }
    void GroupingBar_Loaded(object sender, RoutedEventArgs e) {
        pivotGrid.GroupingBar.AllowMultiFunctionalSortFilter = true;
    }
}
```

![Excel like Filter Popup](Excel-Like-Filtering-images/Excel like Filter Popup.png)

Various features of MultiFunctional pop-up

Sort A to Z

It can be used to sort the corresponding pivot item in the ascending order.

Sort Z to A

It can be used to sort the corresponding pivot item in the descending order.

Clear filters

The clear filters are used to clear all filter changes applied to the corresponding pivot item and bring back the pivot grid to the normal state.

Label filters

The label filters are used to filter the pivot item filter labels of the pivot grid by using the following various options.

- Equals
- Does not equals
- Greater than
- Greater than or equal to
- Less than
- Less than or equal to
- Begins with
- Does not begins with
- Ends with
- Does not ends with
- Contains
- Does not contains
- Between
- Not between

![[Label Filter window](Excel-Like-Filtering-images/Label filter window.png)]

Label Filter window for filtering “Alberta” in State

Value filters

The value filters are used to filter the pivot item field values of the pivot grid by using the following various options.

- Equals
- Does not equals
- Greater than
- Greater than or equal to
- Less than
- Less than or equal to
- Between
- Not between
- Top 10

![[Value Filter window]](Excel-Like-Filtering-images/Value filter window.png)]

Value Filter window for filtering “Alberta” using its Quantity value “677” in State

![PivotGrid with Label Filter or Value Filter applied](Excel-Like-Filtering-images/Filtered PivotGrid by using label value.png)

PivotGrid with Label Filter or Value Filter applied

Note: You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Sorting

Sorting in WPF Pivot Grid Control

Sorting enables you to quickly visualize, understand, organize, and find data that you want. By default, the pivot grid holds built-in comparers for all data types so that it will populate data in ascending/descending order according to its data type. You can also define your own custom comparer to view the data.

Sorting using custom comparer

Sorting the data with your own custom comparer can be achieved by defining the custom comparer and initializing its instance to the `Comparer` property of the corresponding `PivotItem`.

For example, a custom `ReverseOrderComparer` is defined for the `PivotItem`. Find the following appropriate code sample.

C#

```
// ReverseOrderComparer for descending sort order.
public class ReverseOrderComparer: IComparer
{
    public int Compare(object x, object y)
    {
        if (x == null && y == null)
            return 0;
        else if (y == null)
            return 1;
        else if (x == null)
            return -1;
        else
            return -x.ToString().CompareTo(y.ToString());
    }
}
```

To apply this comparer to `PivotItem`, an instance for the `ReverseOrderComparer` is created and assigned it to the `Comparer` property of the `Product` `PivotItem`. Refer to the following code sample.

C#

```
public MainWindow()
{
    InitializeComponent();
    this.Loaded += MainWindow_Loaded;
}

void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    PivotGridControl pivotGrid = new PivotGridControl();
    grid1.Children.Add(pivotGrid);
    PivotItem m_PivotItem = new PivotItem() {
```

```

FieldHeader = "Product", FieldMappingName = "Product", TotalHeader =
"Total", Comparer = new ReverseOrderComparer()
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", SummaryType =
SummaryType.Count
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.ItemSource = ProductSales.GetSalesData();
}

```

![[PivotGrid without sorting]](Sorting-Images/Not Sorted PivotGrid.png)

PivotGrid without ReverseOrderComparer

![[Sorted PivotGrid]](Sorting-Images/Sorted PivotGrid.png)

PivotGrid with ReverseOrderComparer

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Sorting by Values in WPF Pivot Grid control

Sorting-by-Values provides the following sort options:

- Sort all the columns.
- Sort all columns other than total and grand total columns.
- Sort only total columns.
- Sort only grand total columns.
- Disable the sort.

Properties:

- **SortDirection:** Gets or sets the sort order to ascending or descending.
- **SortOption:** Gets or sets the sorting option as all, none, column sorting, total sorting, or grand total sorting.

The **SortOption** property of pivot grid control can be defined both in XAML and code-behind. Refer to the following code sample and screenshots.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" SortOption="All " ItemSource="{Binding
Source={StaticResource data}} " >
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product " FieldMappingName="Product "
TotalHeader="Total " />
<syncfusion:PivotItem FieldHeader="Date " FieldMappingName="Date "
TotalHeader="Total " />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country " FieldMappingName="Country "
TotalHeader="Total " />
<syncfusion:PivotItem FieldHeader="State " FieldMappingName="State "
TotalHeader="Total " />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName = "Total " FieldName =
"Amount " Format="C " SummaryType="DoubleTotalSum "/>
<syncfusion:PivotComputationInfo CalculationName = "Total " FieldName =
"Quantity " SummaryType="Count "/>
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
```



```

pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
    = SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
this.pivotGrid.SortOption = PivotSortOption.All;
}
}

```

Sorting all the value columns

All is used to enable sorting for all the value columns of pivot grid.

![To enable sorting for all columns in pivot grid](Sorting-Images/Sorted PivotGrid when using All option.png)

Sorting all columns other than total and grand total columns

ColumnSorting is used to enable sorting for all the value columns other than subtotal and grand total of the pivot grid.

![To enable sorting for value columns only in pivot grid](Sorting-Images/Sorted PivotGrid when using Column sorting option.png)

Sorting only subtotal columns

TotalSorting is used to enable sorting only for the subtotal columns of pivot grid.

![To enable sorting for subtotal columns only in pivot grid](Sorting-Images/Sorted PivotGrid when using Total Sorting.png)

Sorting only grand total columns

GrandTotalSorting is used to enable sorting only for the grand total columns of the pivot grid.

![To enable sorting for grand total columns only in pivot grid](Sorting-Images/Sorted PivotGrid when using GrandTotal Sorting option.png)

Disable sorting

None is the default option and it disables sorting on all the value columns of pivot grid.

Multi-column sorting

Please [Multi column sorting](#) for more details.

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and displays the result in a cross-table format.

Sorting Events in WPF Pivot Grid Control

The pivot grid provides certain sorting events to keep track of the sorted value fields, which is invoked before and after the sort operation.

- **SortBegin:** An event that notifies before the sorting action begins and returns the index or position of value fields with its corresponding values in the pivot grid.
- **SortCompleted:** An event that notifies after the sort option is completed and returns the index or position of the sorted value fields with its corresponding values in the pivot grid.

Using SortBegin and SortCompleted Events

To do so, create a new pivot grid, bind the `ItemSource`, and then define the `PivotItems` and `PivotComputations`. Invoke the `SortBegin` and `SortCompleted` events and then save the indexes to a separate list as illustrated below.

C#

```
public MainWindow() {
    InitializeComponent();
    this.Loaded += MainWindow_Loaded;
}

void MainWindow_Loaded(object sender, RoutedEventArgs e) {
    PivotGridControl pivotGrid = new PivotGridControl();
    grid1.Children.Add(pivotGrid);
    PivotItem m_PivotItem = new PivotItem() {
        FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
    };
    PivotItem m_PivotItem1 = new PivotItem() {
        FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
    };
    PivotItem n_PivotItem = new PivotItem() {
        FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
    };
    PivotItem n_PivotItem1 = new PivotItem() {
        FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
    };
    // Adding PivotItem to PivotRows
    pivotGrid.PivotRows.Add(m_PivotItem);
    pivotGrid.PivotRows.Add(m_PivotItem1);
    // Adding PivotItem to PivotColumns
    pivotGrid.PivotColumns.Add(n_PivotItem);
    pivotGrid.PivotColumns.Add(n_PivotItem1);
    PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
        CalculationName = "Amount", FieldName = "Amount", SummaryType =
        SummaryType.Count
    };
    PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
        CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
        SummaryType.Count
    };
    pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
    pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
    pivotGrid.ItemSource = ProductSales.GetSalesData();
    pivotGrid.SortBegin += pivotGrid_SortBegin;
    pivotGrid.SortCompleted += pivotGrid_SortCompleted;
}

void pivotGrid_SortCompleted(object sender, OnSortActionCompleted e) {
    Dictionary < int, object > indexesAfterSort = new Dictionary < int, object >
    ();
    indexesAfterSort = e.List;
}
```

```

}
void pivotGrid_SortBegin(object sender, OnSortActionStarted e) {
Dictionary < int, object > indexesBeforeSort = new Dictionary < int, object
> ();
indexesBeforeSort = e.List;
}

```

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Grouping Bar in WPF Pivot Grid

Grouping bar allows you to slice and dice the fields between column, row, value, and filter. This allows you to add, rearrange, or remove the fields to show data in the pivot grid as desired. It consists of the following areas:

- Filter header area: Holds the filter items of the pivot grid control.
- Data header area: Holds the pivot calculation items of the pivot grid control.
- Column header area: Holds the pivot column items of the pivot grid control.
- Row header area: Holds the pivot row items of the pivot grid control.

By default, the grouping bar is enabled in the pivot grid control. You can show or hide it by using the `ShowGroupingBar` property. It can be done both in XAML and code-behind.

For XAML, refer to the following code sample.

XML

```

<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
syncfusion:SkinStorage.VisualStudio="Metro"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="WpfApplication1.MainWindow" Title="MainWindow" Height="350"
Width="525" xmlns:local="clr-namespace:WpfApplication1">
<Window.Resources>
<ResourceDictionary>
<ObjectDataProvider x:Key="data" ObjectType="{x:Type local:ProductSales}"
MethodName="GetSalesData" />
</ResourceDictionary>
</Window.Resources>
<Grid Name="grid1">
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" ShowGroupingBar="True" ItemSource="{Binding
Source={StaticResource data}}">
</syncfusion:PivotGridControl>
</Grid>
</Window>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
public MainWindow() {

```

```
InitializeComponent();
this.Loaded += MainWindow_Loaded;
}
void MainWindow_Loaded(object sender, RoutedEventArgs e) {
    PivotGridControl pivotGrid = new PivotGridControl();
    grid1.Children.Add(pivotGrid);
    pivotGrid.ItemSource = ProductSales.GetSalesData();
    pivotGrid.ShowGroupingBar = true;
}
}
```

![PivotGrid with Grouping Bar](Grouping-Bar-Images/PivotGrid Shows Grouping Bar.png)

PivotGrid with Grouping Bar

![PivotGrid without Grouping Bar](Grouping-Bar-Images/PivotGrid without grouping bar.png)

PivotGrid without Grouping Bar

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Grouping Bar Context Menu in WPF Pivot Grid

The grouping bar context menu consists of the following menu items:

- **Reload Data:** Refreshes the grid with the current item source.
- **Show Field List:** Launches the pivot grid field list.
- **Order:** Changes the position of the item present in the grouping bar. It contains the following submenu items:
 - Move to Beginning: Moves the current item to the first position.
 - Move to Left: Moves the current item one step toward its left.
 - Move to Right: Moves the current item one step toward its right.
 - Move to End: Moves the current item to the last position.
- **Smallest to Largest:** Arranges the pivot fields based on the field header from first letter to the last.
- **Largest to Smallest:** Arranges the pivot fields based on the field header from last letter to the first.
- **Calculated field:** Adds a new calculation field at runtime.

By default, the context menu is enabled in all areas of the grouping bar. The `DisableContextMenu` property should be set individually for row, column, and data header area in the grouping bar to alter their visibility.

After defining the pivot grid control, raise the loaded event for the pivot grid. Inside the `PivotGridLoaded()` event, raise the loaded event for the grouping bar. Inside the `GroupingBarLoaded()` event, set the value for the `DisableContextMenu` property.

Refer to the following code sample.

C#

```
public partial class MainWindow: Window {
```

```

PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
    InitializeComponent();
    grid1.Children.Add(pivotGrid);
    pivotGrid.ItemSource = ProductSales.GetSalesData();
    PivotItem m_PivotItem = new PivotItem() {
        FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
    };
    PivotItem m_PivotItem1 = new PivotItem() {
        FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
    };
    PivotItem n_PivotItem = new PivotItem() {
        FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
    };
    PivotItem n_PivotItem1 = new PivotItem() {
        FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
    };
    // Adding PivotItem to PivotRows
    pivotGrid.PivotRows.Add(m_PivotItem);
    pivotGrid.PivotRows.Add(m_PivotItem1);
    // Adding PivotItem to PivotColumns
    pivotGrid.PivotColumns.Add(n_PivotItem);
    pivotGrid.PivotColumns.Add(n_PivotItem1);
    PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
        CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
        = SummaryType.DoubleTotalSum
    };
    PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
        CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
        SummaryType.Count
    };
    pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
    pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
    pivotGrid.Loaded += pivotGrid_Loaded;
}
void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
    pivotGrid.GroupingBar.Loaded += GroupingBar_Loaded;
}
void GroupingBar_Loaded(object sender, RoutedEventArgs e) {
    pivotGrid.GroupingBar.ColumnHeaderArea.DisableContextMenu = false;
    pivotGrid.GroupingBar.RowHeaderArea.DisableContextMenu = true;
    pivotGrid.GroupingBar.DataHeaderArea.DisableContextMenu = false;
}
}

```

![To enable the context menu options for grouping bar items](Grouping-Bar-Images/Grouping bar context menu.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Calculated Field in WPF Pivot Grid

The pivot grid inserts a new calculated field based on the existing calculated items using the calculated field window. This option is available as a menu item in the grouping bar context menu.

Inserting a new calculated field in pivot grid

To insert a new calculated field, open the calculated field window using the grouping bar context menu. Then, define a **Name** for the new calculated field.

![Defining Name for the Calculated Field](Calculated-Field-images/Defining a name to the new calculated field.png)

Defining Name for the Calculated Field

Note that, the **Formula** can be entered by inserting calculation fields through the **Fields** section. For inserting numerical operator, you can use the formula pop-up as shown in the following screenshot.

![Entering Formula for the Calculated Field](Calculated-Field-images/Formula has been given to the newly created calculated field.png)

Entering Formula for the Calculated Field

Click **Add** to add the calculated field and **OK** to populate the pivot grid control.

![PivotGrid with newly added Calculated Field](Calculated-Field-images/PivotGrid shows the newly added calculated field.png)

PivotGrid with newly added Calculated Field

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

PivotGrid Field List in WPF Pivot Grid

With the current implementation of grouping bar, you cannot add the deleted items to the pivot grid. The delete operation can be easily performed in the grouping bar by using its "AllowRemoving" feature, but there is no possibility to add again the deleted items. To achieve this, maintain a separate window called **PivotTable Field List**, which holds the fields that are not present in the pivot grid but available in the ItemSource. You can bind a collection of PivotItems as **PivotFields** which gets included in the field list window but not present in the pivot grid.

The pivot table field list (or dynamic field list) can be launched by setting the **ShowFieldList** property to "true" or clicking the **ShowFieldList** menu item of grouping bar context menu. The field list is bound to **PivotFields** property of pivot grid control, which is a collection of PivotItems. This property can be set through XAML or code-behind.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" ShowFieldList="True" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
```

```

<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.ShowFieldList = true;
    }
}

```

![Pivot grid Shows Field List](PivotGridFieldlist-images/PivotGrid Shows Field List.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Restrict Resizing of Row Header in WPF Pivot Grid

The pivot grid supports restricting the row header items from being stretched when there are too many PivotCalculation items in the data header area. When you click the **ShowFields** button in the data header area of the grouping bar, the pivot computation list window appears with the PivotCalculation fields.

The `AllowRowHeaderAreaAutoSizing` property is set to "false" to display the computation button (ShowFields button) and restrict the row header items from being stretched when more items are added to the computation area. By default, this property is set to "true", and it can be defined both in XAML and code-behind.

For XAML, refer to the following code sample.

XAML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" AllowRowHeaderAreaAutoSizing="False"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window
{
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
}
```



```

pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
    FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
    FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
    FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
    FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.AllowRowHeaderAreaAutoSizing = false;
}
}

```

![Pivot grid restrict the row header items](Grouping-Bar-Images/Grouping bar while disabling the auto resizing feature.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Expand/Collapse Headers in WPF Pivot Grid

Expand/collapse operations can be done at both UI and programmatic level.

UI level expand/collapse

The header cell context menu will be shown while right-clicking the expander cell. Expand/collapse operations can be handled through this context menu at the row level and column level individually.

EnableContextMenu property allows you to enable or disable the context menu for row or column headers individually.

To do so, define the pivot grid control and raise the loaded event for the pivot grid. Inside the `PivotGrid_Loaded()` event, set the visibility of `EnableContextMenu` of each row and column header areas.

Refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.Loaded += pivotGrid_Loaded;
    }
    void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
        pivotGrid.ColumnHeaderCellStyle.EnableContextMenu = true;
        pivotGrid.RowHeaderCellStyle.EnableContextMenu = true;
    }
}

```

Programmatic level expand/collapse

You can expand/collapse any number of rows or columns, programmatically. This can be done using the following methods.

Method	Description	Parameters	Return Type
ExpandRow(string)	Expands the group for the given row unique text.	string	void
ExpandColumn(string)	Expands the group for the given column unique text.	string	void

CollapseRow(string)	Collapses the group for the given row unique text.	string	void
CollapseColumn(string)	Collapses the group for the given column unique text.	string	void
ExpandRow(List)	Expands the group for the given list of row unique text.	List	void
ExpandColumn(List)	Expands the group for the given list of column unique text.	List	void
CollapseRow(List)	Collapses the group for the given list of row unique text.	List	void
CollapseColumn(List)	Collapses the group for the given list of column unique text.	List	void
ExpandAllGroup	Expands all the group.	-	void
CollapseAllGroup	Collapses all the group.	-	void

To expand/collapse operations in code behind programmatically, use the previously mentioned methods as needed. By passing the unique text as a parameter, you can expand/collapse one or more columns/rows as desired.

To do so, define the pivot grid control and raise the loaded event for pivot grid. Inside the `PivotGrid_Loaded()` event, use the appropriate methods for expand/collapse operations.

Refer to the following code sample.

C#

```
public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
    }
}
```

```

pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
    = SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.Loaded += pivotGrid_Loaded;
}
void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
    /// Expands the Bike from row
    pivotGrid.ExpandRow("Bike");
    /// Collapses the Bike from row
    pivotGrid.CollapseRow("Bike");
    /// Expands the Canada from column
    pivotGrid.ExpandColumn("Canada");
    /// Collapses the Canada from column
    pivotGrid.CollapseColumn("Canada");
    /// Collapses the given collection of UniqueText string values for row
    pivotGrid.CollapseRow(new List < string > {
        "Bike",
        "Car"
    });
    /// Expands the given collection of UniqueText string values for row
    pivotGrid.ExpandRow(new List < string > {
        "Bike",
        "Car"
    });
    /// Collapses the given collection of UniqueText string values for Column
    pivotGrid.CollapseColumn(new List < string > {
        "Canada",
        "France"
    });
    /// Expands the given collection of UniqueText string values for Column
    pivotGrid.ExpandColumn(new List < string > {
        "Canada",
        "France"
    });
    /// Expands entire group in both row and column.
    pivotGrid.ExpandAllGroup();
    /// Collapse entire group in both row and column.
    pivotGrid.CollapseAllGroup();
}
}

```

![Context menu for Row Headers](Header-Cell-context-menu-images/Context menu for Row headers.png)

Context menu for Row Headers

![Context menu for Column Headers](Header-Cell-context-menu-images/Context menu for Column headers.png)

Context menu for Column Headers

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Display Options in WPF Pivot Grid

The [WPF Pivot Grid](#) control provides support for the PivotComputationInfo to display calculation values in preferred areas of the pivot grid using the `DisplayOption` property.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" ItemSource="{Binding Source={StaticResource
data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
DisplayOption="Calculations" Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" DisplayOption="Summary" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
```

```

};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum, DisplayOption = DisplayOption.Calculations
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count, DisplayOption = DisplayOption.Summary
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
}
}

```

All

Displays the calculation values in all columns of the pivot grid.

![To display the calculation values in all columns](Display-options/Images/PivotComputationinfo using All option.png)

Summary

Data actually presents in the PivotEngine, but displays the calculation values only in the summary columns of the pivot grid.

![To display the calculation values only in summary columns](Display-options/Images/PivotComputationinfo using summary option.png)

Calculations

Data actually presents in the PivotEngine, but displays the calculation values only in the calculation columns and does not display in the summary and grand total columns of the pivot grid.

![To display the calculation values only in calculation columns](Display-options/Images/PivotComputationinfo using Calculations option.png)

GrandTotal

Data actually presents in the PivotEngine, but displays the calculation values only in the grand total columns of the pivot grid.

![To display the calculation values only in grand total columns](Display-options/Images/PivotComputationinfo using Grand Totals option.png)

None

Data actually presents in the PivotEngine, but hides all the calculation values in all columns of the pivot grid.

![To hide the calculation values in all columns](Display-options-Images/PivotComputationinfo using none option.png)

Note: You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and displays the result in a cross-table format.

Grid Layout in WPF Pivot Grid

The pivot grid supports for two different types of layout options with respect to summaries. That is, whether to display the summary data of each pivot item at the top or at the bottom of value cells.

The `GridLayout` property is used to set the layout options, and this can be defined in both XAML and code-behind.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" GridLayout="TopSummary" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
```

```

};
PivotItem m_PivotItem1 = new PivotItem() {
    FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
    FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
    FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.GridLayout = GridLayout.TopSummary;
}
}

```

Normal

Displays the summary data of rows or columns at the end of value cells.

![Displaying the PivotGrid in Normal layout](Grid-Layout-Images/PivotGrid with normal layout.png)

Top summary

Displays the summary data of rows or columns at the beginning of value cells.

![Displaying the PivotGrid in Top summary layout](Grid-Layout-Images/PivotGrid with top summary layout.png)

Excel-like layout

Displays the summary cells at the bottom alone and child members that appear below the parent member with some indent space.

![Displaying the PivotGrid in Excel-like layout](Grid-Layout-Images/PivotGrid with excel like layout.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

State Persistence in WPF Pivot Grid

The pivot grid allows you to maintain the collapsed or expanded state when the corresponding schema items are changed. This can be achieved using the `StatePersistence` property of the pivot grid control.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" StatePersistenceEnabled="True" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
}
```

```

PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
    = SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.StatePersistenceEnabled = true;
}
}

```

![PivotGrid with collapsed Bike](State-Persistence-images/PivotGrid with state persistence enabled.png)

PivotGrid with collapsed "Bike"

![PivotGrid maintaining collapsed state of Bike after Schema change](State-Persistence-images/PivotGrid with state persistence enabled 1.png)

PivotGrid maintaining collapsed state of "Bike" after Schema change

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and displays the result in a cross-table format.

Sub-Totals for Child Elements in WPF Pivot Grid

By default, the pivot grid calculates summaries based on the parent nodes available in rows and columns. You can also display the subtotals based on the child elements available in rows and columns using the `ShowSubTotalsForChildren` property. This property can be set through XAML or code-behind.

For XAML, refer to the following code sample.

XAML

```

<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid1"
VerticalAlignment="Top" VisualStyle="Metro" ShowSubTotalsForChildren="True"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>

```

```

<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum"
CalculationType="PercentageOfColumnTotal" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        pivotGrid.PivotRows.Add(n_PivotItem);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.PivotEngine.ShowSubTotalsForChildren = true;
    }
}

```

Drop Filter fields here

Amount Quantity			State ▾ ▲									
			Alberta		Bayern		British Columbia		Brunswick		Grand Total	
Prod... ▾ ▲	D... ▾ ▼	Cou... ▾ ▲	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Q
* Bike	* FY 2009	Canada	\$416,100.00	2			\$202,500.00	2	\$279,600.00	2	\$898,200.00	
	FY 2009 Total		\$416,100.00	2			\$202,500.00	2	\$279,600.00	2	\$898,200.00	
	* FY 2008	Canada	\$372,300.00	3			\$353,100.00	3	\$941,400.00	5	\$1,666,800.00	
		Germany			\$994,200.00	5					\$994,200.00	
	FY 2008 Total		\$372,300.00	3	\$994,200.00	5	\$353,100.00	3	\$941,400.00	5	\$2,661,000.00	
	* FY 2007	Canada	\$1,126,500.00	6			\$1,549,200.00	8	\$1,189,500.00	6	\$3,865,200.00	
		Germany			\$883,200.00	5					\$883,200.00	
	FY 2007 Total		\$1,126,500.00	6	\$883,200.00	5	\$1,549,200.00	8	\$1,189,500.00	6	\$4,748,400.00	
	* FY 2006	Canada	\$1,695,900.00	11			\$2,647,200.00	15	\$3,162,600.00	17	\$7,505,700.00	
		Germany			\$2,452,500.00	13					\$2,452,500.00	
	FY 2006 Total		\$1,695,900.00	11	\$2,452,500.00	13	\$2,647,200.00	15	\$3,162,600.00	17	\$9,958,200.00	
	* FY 2005	Canada	\$5,409,300.00	34			\$4,605,600.00	29	\$6,063,300.00	35	\$16,078,200.00	
	Germany			\$5,967,600.00	33					\$5,967,600.00		
	FY 2005 Total		\$5,409,300.00	34	\$5,967,600.00	33	\$4,605,600.00	29	\$6,063,300.00	35	\$22,045,800.00	
Bike Total		Canada	\$9,020,100.00	56			\$9,357,600.00	57	\$11,636,400.00	65	\$30,014,100.00	
		Germany			\$10,297,500.00	56					\$10,297,500.00	
		Total	\$9,020,100.00	56	\$10,297,500.00	56	\$9,357,600.00	57	\$11,636,400.00	65	\$40,311,600.00	
Grand Total			\$9,020,100.00	56	\$10,297,500.00	56	\$9,357,600.00	57	\$11,636,400.00	65	\$40,311,600.00	

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

RowPivotsOnly Mode in WPF Pivot Grid

RowPivotsOnly mode allows the pivot grid to display only the column values.

When displaying the pivot grid in RowPivotsOnly mode, it shows the row header cells and column header cells as value cells. Also, it supports column filtering, column sorting, and so on.

The `RowPivotsOnly` property is used to achieve this and it can be mentioned in XAML or code-behind.

For XAML, refer to the following code sample.

XAML

```
<Grid>
<syncfusion:PivotGridControl Name="pivotGrid" RowPivotsOnly="True">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem AllowFilter="False" FieldHeader="PID"
FieldMappingName="PID" TotalHeader="Total" />
<syncfusion:PivotItem AllowFilter="False" FieldHeader="Location"
FieldMappingName="Location" TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo FieldHeader="Color" FieldName="Color"
Format="0.0" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo FieldHeader="Class" FieldName="Class"
Format="0.0" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo FieldHeader="PID" FieldName="PID"
Format="0.0" SummaryType="DoubleTotalSum" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

```

<syncfusion:PivotComputationInfo FieldHeader="Units" FieldName="Units"
Format="0.0" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo FieldHeader="Retail" FieldName="Retail"
Format="0.0" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo FieldHeader="Cost" FieldName="Cost"
Format="0.0" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo FieldHeader="TestStr" FieldName="TestStr"
Format="0.0" PadString="***" SummaryType="DisplayIfDiscreteValuesEqual" />
<syncfusion:PivotComputationInfo FieldHeader="TestInt" FieldName="TestInt"
Format="0.0" PadString="***" SummaryType="DisplayIfDiscreteValuesEqual" />
<syncfusion:PivotComputationInfo FieldHeader="TestDouble"
FieldName="TestDouble" Format="0.00" PadString="***"
SummaryType="DisplayIfDiscreteValuesEqual" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
public MainWindow() {
InitializeComponent();
pivotGrid.RowPivotsOnly = true;
}
}

```

![RowPivotsOnly mode enabled in the pivotGrid](Features-in-RowPivotsOnly-images/PivotGrid when RowPivotsOnly mode is enabled.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Multi-Column Sorting in WPF Pivot Grid

Multicolumn sorting allows you to sort the fields one after the other. For example, if a user has three different fields rendered in the pivot grid, then it is possible to sort like:

OrderBy(field1).ThenBy(field2).ThenBy(field3)..

A separate index number is used at the top of each column to maintain the order of the sorted columns while doing the multicolumn sorting. Also, multicolumn sorting can be applied for both **Normal** and **Row Pivots Only** modes of the pivot grid control.

Multicolumn sorting in normal mode

You can enable the sorting through XAML or code-behind using the `SortOption` property of the pivot grid control.

For XAML, refer to the following code sample.

XML

```

<Grid>

```

```

<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" SortOption="All" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
};

```

```
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.SortOption = PivotSortOption.All;
}
}
```

- The following screenshot shows the normal pivot grid.

![Pivot grid without sorting](Sorting-Images/Normal PivotGrid.png)

- The **Quantity** column has been sorted in the pivot grid.

![Sorting the quantity column in pivot grid](Sorting-Images/PivotGrid sorted by single column.png)

- By pressing the Ctrl key, the **Amount** column is sorted. Now, the pivot grid is sorted by multiple columns.

![Sorting the amount column in pivot grid](Sorting-Images/PivotGrid sorted by multiple columns.png)

- Then, by pressing the Ctrl key again, you can sort the pivot grid for 'n' number of columns.

![Sorting the n number of columns in pivot grid](Sorting-Images/PivotGrid sorted by N columns.png)

Multicolumn sorting in row pivots only mode

You can enable the sorting through XAML or code-behind using the **SortOption** property of the pivot grid control in RowPivotsOnly mode.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl Name="pivotGrid" SortOption="All"
RowPivotsOnly="True">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem AllowFilter="False" FieldHeader="PID"
FieldMappingName="PID" TotalHeader="Total" />
<syncfusion:PivotItem AllowFilter="False" FieldHeader="Location"
FieldMappingName="Location" TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo FieldHeader="Color" FieldName="Color"
Format="0.0" SummaryType="DoubleTotalSum" AllowSort="True" />
<syncfusion:PivotComputationInfo FieldHeader="Class" FieldName="Class"
Format="0.0" SummaryType="DoubleTotalSum" AllowSort="True" />
<syncfusion:PivotComputationInfo FieldHeader="PID" FieldName="PID"
Format="0.0" SummaryType="DoubleTotalSum" AllowSort="True" />
<syncfusion:PivotComputationInfo FieldHeader="Units" FieldName="Units"
Format="0.0" SummaryType="DoubleTotalSum" AllowSort="True" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

```

<syncfusion:PivotComputationInfo FieldHeader="Retail" FieldName="Retail"
Format="0.0" SummaryType="DoubleTotalSum" AllowSort="True" />
<syncfusion:PivotComputationInfo FieldHeader="Cost" FieldName="Cost"
Format="0.0" SummaryType="DoubleTotalSum" AllowSort="True" />
<syncfusion:PivotComputationInfo FieldHeader="TestStr" FieldName="TestStr"
Format="0.0" PadString="***" SummaryType="DisplayIfDiscreteValuesEqual"
AllowSort="True" />
<syncfusion:PivotComputationInfo FieldHeader="TestInt" FieldName="TestInt"
Format="0.0" PadString="***" SummaryType="DisplayIfDiscreteValuesEqual"
AllowSort="True" />
<syncfusion:PivotComputationInfo FieldHeader="TestDouble"
FieldName="TestDouble" Format="0.00" PadString="***"
SummaryType="DisplayIfDiscreteValuesEqual" AllowSort="True" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
public MainWindow() {
InitializeComponent();
pivotGrid.SortOption = PivotSortOption.All;
}
}

```

Similarly, you can perform 'n' number of sorting in RowPivotsOnly mode by pressing the Ctrl key.

![Multicolumn sorting in row pivots only mode](Features-in-RowPivotsOnly-images/Multi column sort in RowPivots.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Show/Hide Sub-Totals in WPF Pivot Grid

The subtotal hiding feature is used to show or hide subtotals in the pivot grid. It provides an abstract view of data as desired and provides the following levels of hiding.

- Hiding all the subtotals.
- Hiding only the row subtotals.
- Hiding only the column subtotals.
- Hiding the subtotals for the specific PivotItem.

Hiding all the subtotals

The `ShowSubTotals` property is used to hide all subtotals and can be mentioned in XAML or code-behind.

For XAML, refer to the following code sample.

XML


```

<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ShowSubTotals="False"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum

```

```

};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.ShowSubTotals = false;
}
}

```

![To hides both row and column subtotals](Hiding-SubTotals-images/PivotGrid when sub totals hidden.png)

Hiding only the row subtotals

The `ShowRowSubTotals` property is used to hide the row subtotals and can be mentioned in XAML or code-behind.

For XAML, refer to the following code sample.

XAML

```

<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ShowRowSubTotals="False"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
}
}

```

```

pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
    FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
    FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
    FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
    FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.ShowRowSubTotals = false;
}
}

```

![[To hides only row subtotals]](Hiding-SubTotals-images/PivotGrid when row sub totals hidden.png)

Hiding only the column subtotals

The `ShowColumnSubTotals` property is used to hide the column subtotals and can be mentioned in XAML or code-behind.

For XAML, refer to the following code sample.

XAML

```

<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ShowColumnSubTotals="False"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />

```

```

<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.ShowColumnSubTotals = false;
}
}

```

![To hides only column subtotals](Hiding-SubTotals-images/PivotGrid when column sub totals hidden.png)

Hiding the subtotals for the specific pivot item

You can hide the subtotals for specific pivot item by setting the **ShowSubTotal** property to false, and it can be mentioned in XAML or code-behind.

For XAML, refer to the following code sample.

XAML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" ShowSubTotal="False" />
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader =
"Total", ShowSubTotal = false
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem m_PivotItem2 = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
```

```

FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem2);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
}
}

```

![To hides the subtotals for specific pivot item](Hiding-SubTotals-images/PivotGrid shows the subtotal hidden for specific pivot item.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Asynchronous Data Loading in WPF Pivot Grid

The [WPF Pivot Grid](#) control supports loading data in a unique UI thread. That is, it can perform long running operations asynchronously on a background thread. It also loads asynchronously for every layout change operation, such as filtering, sorting, drag and drop, manipulating the field list, or changing the pivot schema designer. This can be achieved by setting the `LoadInBackground` property of the pivot grid control.

The `LoadInBackground` property can be defined in XAML or code-behind. For XAML, refer to the following code sample.

XAML

```

<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" LoadInBackground="True"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />

```

```

</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.LoadInBackground = true;
}
}

```

![UI thread loading](Asynchronous-data-loading-images/UI thread loading.png)

UI Thread Loading

![Asynchronously loaded data](Asynchronous-data-loading-images/Asynchronously loaded data.png)

Asynchronously Loaded Data

Note: You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and displays the result in a cross-table format.

Virtualized Binding (Performance Improvement) in WPF Pivot Grid

It provides high loading performance for a large set of records. This feature helps to load data in the pivot grid on-demand using the **Index Engine**.

- **UseIndexedEngine:** Gets or sets whether an optimized algorithm that relies on indexing the raw data should be used to compute the pivot information.
- **EnableOnDemandCalculations:** Gets or sets whether the calculations are postponed until the value is requested through the indexer in the PivotEngine.

To achieve this, after defining the pivot grid control, set the **EnableOnDemandCalculations** and **UseIndexedEngine** properties of PivotEngine to true. The values for IndexEngine can be obtained by using the **ItemObjectLookup()** method and the time span should be calculated by using the **Dispatcher.BeginInvoke()** method. Refer to the following code sample.

C#

```
public partial class MainWindow : Window
{
    DateTime _startIndex = DateTime.Now;
    public MainWindow()
    {
        InitializeComponent();
        pivotGrid.PivotEngine.EnableOnDemandCalculations =
        pivotGrid.PivotEngine.UseIndexedEngine = true;
        pivotGrid.PivotEngine.GetValue = ItemObjectLookup;
        ObservableCollection<ItemObject> itemsSourceObject =
        (pivotGrid.DataContext as ViewModel.ViewModel).ItemObjectCollection;
        pivotGrid.PivotEngine.PivotSchemaChanged +=
        PivotEngine_PivotSchemaChanged;
        pivotGrid.ItemSource = itemsSourceObject;
    }
    private void PivotEngine_PivotSchemaChanged(object sender,
    PivotSchemaChangedEventArgs e)
    {
        _startIndex = DateTime.Now;
        ScrollViewer _scrollViewer = e.OriginalSource as ScrollViewer;
        pivotGrid.Dispatcher.BeginInvoke(DispatcherPriority.SystemIdle, new
        Action(() =>
        {
            if (!pivotGrid.IgnoreRefresh)
            {
                if ((_scrollViewer.Content as TextBlock) != null)
                && (_scrollViewer.Content as TextBlock).Text == string.Empty);
                CheckTime(_startIndex, "Initial part done in");
                ContinueLoadingAsyncholonously(
                pivotGrid.PivotEngine.IndexEngine, _startIndex);
            }
        }));
    }
}
```



```

}
private void ContinueLoadingAsynchonolously(IndexEngine engine, DateTime
startIndex)
{
pivotGrid.Dispatcher.BeginInvoke(new Action(() =>
{
if (engine != null && engine.HighRowLevel < engine.RowCount - 1)
{
int cutOff = Math.Min(engine.HighRowLevel + 800, engine.RowCount - 1);
object o = engine[cutOff, 0]; ////Gets 800 more rows from the pivot engine
(on demand calculation).
if ((_scrollViewer.Content as TextBlock) != null)
{
(_scrollViewer.Content as TextBlock).Text +=
string.Format("\n{0}/{1}", engine.HighRowLevel, engine.RowCount - 1);
ContinueLoadingAsynchonolously(engine, startIndex); //Recursive call to
update the rows of the PivotEngine until they reach high row level.
}
}
else
{
CheckTime(startIndex, "Load Completed");
}
}), DispatcherPriority.SystemIdle);
}
private void CheckTime(DateTime start, string label)
{
if (_textBlock != null)
_textBlock.Text += string.Format("\n{0} {1:0.0000} seconds.", label,
DateTime.Now.Subtract(start).TotalSeconds);
}
public IComparable ItemObjectLookup(object o, string name)
{
IComparable c = null;
var io = o as ItemObject;
if (io != null)
{
switch (name)
{
case "Date":
c = io.Date;
break;
case "Client":
c = io.Client;
break;
case "Campaign":
c = io.Campaign;
break;
case "Color":
c = io.Color;
break;
case "Shape":
c = io.Shape;
break;
case "Price":
c = io.Price;
break;

```

```

case "Spend":
c = io.Spend;
break;
case "ColH":
c = io.ColH;
break;
case "ColI":
c = io.ColI;
break;
case "ColJ":
c = io.ColJ;
break;
}
}
return c;
}
}

```

![Loads the pivot grid using Index Engine](Virtualized-Binding-images/PivotGrid loaded with OnDemand index engine.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Improved (Deferred) Scrolling in WPF Pivot Grid

A support has been provided to improve the scrolling performance of pivot grid by updating the data only when user releases the thumb on scrolling. This can be achieved by the `EnableDeferredScrolling` property of the pivot grid control.

For XAML, refer to the following code sample.

XML

```

<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
FreezeHeaders="False" VerticalAlignment="Top" VisualStyle="Metro"
EnableDeferredScrolling="True" ItemSource="{Binding Source={StaticResource
data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />

```

```

</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.EnableDeferredScrolling = true;
    }
}

```

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Resize to Fit in WPF Pivot Grid

The pivot grid supports resizing the grid to fit its content while expanding and collapsing the groups. The grid will be resized after refreshing the page.

The `ResizePivotGridToFit` property is used to resize the grid, and it can be defined in both XAML and code-behind.

For XAML, refer to the following code sample.

XAML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" ResizePivotGridToFit="True" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
}
```

```
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
    = SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.ResizePivotGridToFit = true;
}
}
```

![[Resized PivotGrid]](Resizing-image/Resized PivotGrid during collapsed state.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Editing in WPF Pivot Grid

The [WPF Pivot Grid](#) supports editing the value and total cells, and this can be achieved by using the `EnableValueEditing` property. While handling the editing operations, the values will be calculated automatically and its total values will be adjusted accordingly.

Enable editing for value cells

The `EnableValueEditing` property of the pivot grid control is used to edit the value cells and it can be mentioned in XAML or code-behind.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" EnableValueEditing="True"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

```
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            Syncfusion.PivotAnalysis.Base.SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.EnableValueEditing = true;
    }
}
```

![To edit the value cells in PivotGrid](Editing-Images/PivotGrid shows edited value cell.png)

Enable editing for total cells

The `AllowEditingOfTotalCells` property of `PivotEditingManager` is used to edit the total cells and it can be mentioned in code-behind.

To achieve this, define the pivot grid control and raise its loaded event. Inside the `PivotGrid_Loaded()` event, set the `AllowEditingIfTotalCells` property of the pivot grid control.

C#

```
public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType =
            Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            Syncfusion.PivotAnalysis.Base.SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.Loaded += pivotGrid_Loaded;
    }
    void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
        pivotGrid.EditManager.AllowEditingOfTotalCells = true;
    }
}
```

![To edit the total cell values in PivotGrid](Editing-Images/PivotGrid shows edited total cell.png)

Custom editing manager

After editing the value, you can further customize the value using the custom editing manager, which is used to format the `PivotCellInfo` in the pivot grid.

To do so, create a class named `CustomEditManager` as illustrated in the following code snippet. Dispose the existing edit manager and set the created custom edit manager to the pivot grid by using the edit manager.

C#

```

public class CustomEditManager: PivotEditingManager {
public CustomEditManager(PivotGridControl pg): base(pg) {}
protected override void ChangeValue(object oldValue, object newValue, int
row1, int coll, PivotCellInfo pi) {
//do the base change
base.ChangeValue(oldValue, newValue, row1, coll, pi);
//mark all the adjusted cell contents
pi.FormattedText += "*";
}
}

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
Syncfusion.PivotAnalysis.Base.SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.Loaded += pivotGrid_Loaded;
}
void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
pivotGrid.EditManager.Dispose();
pivotGrid.EditManager = new CustomEditManager(pivotGrid);
}
}

```

! [To customize the edited cell values in PivotGrid] (Editing-Images/PivotGrid shows edited Custom Editing.png)

Note: You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and displays the result in a cross-table format.

Updating in WPF Pivot Grid

The pivot grid supports updating the values in the pivot grid in real time and it can be achieved through the `EnableUpdating` property. While handling the updating operations, the values will be calculated automatically and its total values will be reflected.

- **EnableUpdating:** Gets/sets the Boolean value to enable updating.
- **ThrottleUpdateRate:** Gets/sets a millisecond value for the time taken between UI refreshes. Zero indicates immediate refresh of the UI without delays. Throttling the refresh rate can minimize CPU usage. Depending upon your updating rate, values from 300 to 500 milliseconds may give lower CPU usage.

The `EnableUpdating` property can be mentioned in XAML or code-behind.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" EnableUpdating="True"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
}
```

```

PivotItem m_PivotItem = new PivotItem() {
    FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
    FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
    FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
    FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
Syncfusion.PivotAnalysis.Base.SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.EnableUpdating = true;
}
}

```

While updating the pivot grid control, user can throttle its updating speed by setting the **ThrottleUpdateRate** property. It gets the value in milliseconds as the time interval for UI refreshes to take place. Zero indicates immediate refresh of the UI without any delays. By throttling the refresh rate, you can minimize CPU usage. The default value is zero, but depending upon the user updating rate, values can be given from 300 to 500 milliseconds and it may achieve lower CPU usage.

To achieve this, define the pivot grid control and raise its loaded event. Inside the **PivotGrid_Loaded()** event, set the **ThrottleUpdateRate** property of the pivot grid control.

Refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
    }
}

```

```

};
PivotItem n_PivotItem = new PivotItem() {
    FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
    FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType =
    Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    Syncfusion.PivotAnalysis.Base.SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.EnableUpdating = true;
pivotGrid.Loaded += pivotGrid1_Loaded;
}
void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
    pivotGrid.UpdateManager.ThrottleUpdateRate = 300;
}
}

```

![PivotGrid with updated values](Updating-Images/PivotGrid with updated values.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Single Calculation Header in WPF Pivot Grid

By default, the pivot grid does not show the calculation headers when there is only one PivotCalculation. To show the calculation headers even when there is only one PivotCalculation, use the `ShowSingleCalculationHeader` property of pivot grid control.

Refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {

```

```

FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.PivotEngine.ShowSingleCalculationHeader = true;
}
}

```

![Pivot grid shows the field header for single calculation fields](Show-Single-calculation-header-images/PivotGrid shows single calculation header.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

GetRawItem in WPF Pivot Grid

GetRawItemFor method is used to obtain the list of raw items for value cell, total cell, or grand total cell in the pivot grid.

To achieve this, define the pivot grid control and enable the hyperlink option of the value and summary cells using the `IsHyperlinkCell` property. Then, invoke the hyperlink cell click event and call the `GetRawItemFor` method as illustrated in the following code snippet.

C#

```

public partial class MainWindow: Window {
public MainWindow() {
InitializeComponent();
pivotGrid.ValueCellStyle.IsHyperlinkCell = true;
pivotGrid.SummaryCellStyle.IsHyperlinkCell = true;
pivotGrid.HyperlinkCellClick += pivotGrid_HyperlinkCellClick;
}
void pivotGrid_HyperlinkCellClick(object sender, HyperlinkCellClickEventArgs
e) {

```

```
pivotGrid.PivotEngine.GetRawItemsFor(e.RowColumnIndex.RowIndex,
e.RowColumnIndex.ColumnIndex);
}
}
```

![Getting the list of raw items for specific cells](GetRawItem-images/PivotGrid shows Raw Item.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Conditional Formatting

Formatting using data conditions in WPF Pivot Grid

Conditional formatting is a process of applying customized styles to any object based on specified conditions.

It can be defined by using the `PivotGridControl.ConditionalFormats`, which is an observable collection of `PivotGridDataConditionalFormat` type. The criteria for filtering the cells are specified by using the `PivotGridDataConditionalFormat.Conditions` property, which is a collection of `PivotGridDataCondition` objects. The style for each `ConditionalFormat` can be specified by using the `PivotGridDataConditionalFormat.CellStyle` property, which should be the `PivotGridCellStyle` type. It can be defined in XAML or code-behind.

For **XAML**, refer to the following code snippet.

XAML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotGridControl.ConditionalFormats>
<!-- Adding Conditions. -->
<syncfusion:PivotGridDataConditionalFormat Name="C1">
<!-- Specifying the Cell Style. -->
<syncfusion:PivotGridDataConditionalFormat.CellStyle>
```

```

<syncfusion:PivotGridCellStyle Background="Red" FontFamily="Calibri"
FontSize="12" />
</syncfusion:PivotGridDataConditionalFormat.CellStyle>
<!-- Specifying Conditions. -->
<syncfusion:PivotGridDataConditionalFormat.Conditions>
<syncfusion:PivotGridDataCondition ConditionType="GreaterThan"
Value="50000000" SummaryElement="Amount" PredicateType="And" />
</syncfusion:PivotGridDataConditionalFormat.Conditions>
</syncfusion:PivotGridDataConditionalFormat>
<syncfusion:PivotGridDataConditionalFormat Name="C2">
<!-- Specifying the Cell Style. -->
<syncfusion:PivotGridDataConditionalFormat.CellStyle>
<syncfusion:PivotGridCellStyle Background="Yellow" FontFamily="Calibri"
FontSize="12" />
</syncfusion:PivotGridDataConditionalFormat.CellStyle>
<!-- Specifying Conditions. -->
<syncfusion:PivotGridDataConditionalFormat.Conditions>
<syncfusion:PivotGridDataCondition ConditionType="LessThan" Value="50"
SummaryElement="Quantity" PredicateType="And" />
</syncfusion:PivotGridDataConditionalFormat.Conditions>
</syncfusion:PivotGridDataConditionalFormat>
</syncfusion:PivotGridControl.ConditionalFormats>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code snippet.

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum

```

```

};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    Syncfusion.PivotAnalysis.Base.SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
PivotGridDataConditionalFormat m_PivotGridDataConditionalFormat = new
PivotGridDataConditionalFormat();
//Adding Conditions
m_PivotGridDataConditionalFormat.Conditions.Add(new PivotGridDataCondition()
{
    ConditionType = PivotGridDataConditionType.GreaterThan,
    SummaryElement = "Amount",
    Value = "50000000",
    PredicateType = PredicateType.And
});
// Specifying the Cell Style.
m_PivotGridDataConditionalFormat.CellStyle = new PivotGridCellStyle() {
    Background = Brushes.Red, FontFamily = new FontFamily("Calibri"), FontSize =
    12
};
PivotGridDataConditionalFormat n_PivotGridDataConditionalFormat = new
PivotGridDataConditionalFormat();
//Adding Conditions
n_PivotGridDataConditionalFormat.Conditions.Add(new PivotGridDataCondition()
{
    ConditionType = PivotGridDataConditionType.GreaterThan,
    SummaryElement = "Quantity",
    Value = "50",
    PredicateType = PredicateType.And
});
// Specifying the Cell Style.
n_PivotGridDataConditionalFormat.CellStyle = new PivotGridCellStyle() {
    Background = Brushes.Yellow, FontFamily = new FontFamily("Calibri"),
    FontSize = 12
};
//Adding Conditional Formats to PivotGrid
pivotGrid.ConditionalFormats.Add(m_PivotGridDataConditionalFormat);
pivotGrid.ConditionalFormats.Add(n_PivotGridDataConditionalFormat);
}
}

```

![Apply conditional formatting using data conditions.](Conditional-format-images/PivotGrid shows conditionally formatted values.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Formatting using gradient color scales in WPF Pivot Grid

Conditional formatting is a process of applying customized styles to any object based on specified conditions. In this scenario, styles are applied in the form of gradient color scales throughout the control.

It can be defined by using the `PivotGridControl.ColorScaleConditionalFormats`, which is an observable collection of `PivotGridColorScaleConditionalFormat` type. Meanwhile, it can be done in the following ways.

- Apply format using predefined color scales
- Apply format using custom colors

Note: To apply conditional formatting based on gradient color scale values, make sure that `ApplyToAllLevels` property is set to **true**.

Apply format using predefined color scales

It can be applied by setting the `ColorScaleName` property with pre-defined color scale value selected from `ColorScale` enumerator. These properties can be defined either in XAML or code-behind.

For **XAML**, refer to the following code snippet.

XAML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
<!-- Conditional Formatting Cells -->
<syncfusion:PivotGridControl.ColorScaleConditionalFormats>
<syncfusion:PivotGridColorScaleConditionalFormat Name="C1"
CalculationName="Amount" ApplyToAllLevels="True"
ColorScaleName="BlueWhiteRed" />
</syncfusion:PivotGridControl.ColorScaleConditionalFormats>
</syncfusion:PivotGridControl>
</syncfusion:PivotGridControl>
</Grid>
```

For **code-behind**, refer to the following code snippet.

C#

```
public partial class MainWindow: Window {
```



```

PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
    InitializeComponent();
    grid1.Children.Add(pivotGrid);
    pivotGrid.ItemSource = ProductSales.GetSalesData();
    PivotItem m_PivotItem = new PivotItem() {
        FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
    };
    PivotItem m_PivotItem1 = new PivotItem() {
        FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
    };
    PivotItem n_PivotItem = new PivotItem() {
        FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
    };
    PivotItem n_PivotItem1 = new PivotItem() {
        FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
    };
    // Adding PivotItem to PivotRows
    pivotGrid.PivotRows.Add(m_PivotItem);
    pivotGrid.PivotRows.Add(m_PivotItem1);
    // Adding PivotItem to PivotColumns
    pivotGrid.PivotColumns.Add(n_PivotItem);
    pivotGrid.PivotColumns.Add(n_PivotItem1);
    PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
        CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
        = Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
    };
    PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
        CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
        Syncfusion.PivotAnalysis.Base.SummaryType.Count
    };
    pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
    pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
    // Conditional formatting cells.
    PivotGridColorScaleConditionalFormat m_ColorScaleFormat = new
    PivotGridColorScaleConditionalFormat()
    {
        Name = "C1",
        CalculationName = "Amount",
        ColorScaleName = ColorScale.BlueWhiteRed,
        ApplyToAllLevels = true
    };
    //Adding Conditional Formats to PivotGrid
    pivotGrid.ColorScaleConditionalFormats.Add(m_ColorScaleFormat);
}
}

```

		▼ Canada						
		Alberta		British Columbia		Brunswick		Manitoba
		Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount
▼ Bike	FY 2005	\$5,409,300.00	34	\$4,605,600.00	29	\$6,063,300.00	35	\$3,970,200.00
	FY 2006	\$1,695,900.00	11	\$2,647,200.00	15	\$3,162,600.00	17	\$2,914,500.00
	FY 2007	\$1,126,500.00	6	\$1,549,200.00	8	\$1,189,500.00	6	\$1,050,300.00
	FY 2008	\$372,300.00	3	\$353,100.00	3	\$941,400.00	5	\$723,000.00
	FY 2009	\$416,100.00	2	\$202,500.00	2	\$279,600.00	2	\$195,300.00
Bike Total		\$9,020,100.00	56	\$9,357,600.00	57	\$11,636,400.00	65	\$8,853,300.00
▼ Car	FY 2005	\$1,410,300.00	8	\$1,482,900.00	10	\$304,500.00	2	\$1,149,600.00
	FY 2006	\$1,003,500.00	4	\$629,700.00	3	\$245,700.00	1	\$337,800.00
	FY 2007			\$87,300.00	1	\$328,800.00	2	\$169,200.00
	FY 2008	\$270,000.00	1			\$169,200.00	1	
	FY 2009							
Car Total		\$2,683,800.00	13	\$2,199,900.00	14	\$1,048,200.00	6	\$1,656,600.00
Grand Total		\$11,703,900.00	69	\$11,557,500.00	71	\$12,684,600.00	71	\$10,509,900.00

Apply format using custom colors

It can be applied by providing custom colors to the following properties - **StartColor**, **EndColor**, **MiddleColor** in the application. Also, we can apply conditional formatting based on two level of gradient color scales by defining **StartColor** and **EndColor** properties alone. These properties can be defined either in XAML or code-behind.

For **XAML**, refer to the following code snippet.

XAML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

```

</syncfusion:PivotGridControl.PivotCalculations>
<!-- Conditional Formatting Cells -->
<syncfusion:PivotGridControl.ColorScaleConditionalFormats>
<syncfusion:PivotGridColorScaleConditionalFormat Name="C1"
CalculationName="Amount" ApplyToAllLevels="True" StartColor="Green"
MiddleColor="Yellow" EndColor="Red" />
</syncfusion:PivotGridControl.ColorScaleConditionalFormats>
</syncfusion:PivotGridControl>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code snippet.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            Syncfusion.PivotAnalysis.Base.SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        // Conditional formatting cells.
        PivotGridColorScaleConditionalFormat m_ColorScaleFormat = new
        PivotGridColorScaleConditionalFormat()
        {
            Name = "C1",
            CalculationName= "Amount",
            ApplyToAllLevels = true,

```

```

StartColor = System.Drawing.Color.Green,
MiddleColor = System.Drawing.Color.Yellow,
EndColor = System.Drawing.Color.Red
};
//Adding Conditional Formats to PivotGrid
pivotGrid.ColorScaleConditionalFormats.Add(m_ColorScaleFormat);
}
}

```

		▼ Canada							
		Alberta		British Columbia		Brunswick		Manitoba	
		Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	
▼ Bike	FY 2005	\$5,409,300.00	34	\$4,605,600.00	29	\$6,063,300.00	35	\$3,970,200.00	
	FY 2006	\$1,695,900.00	11	\$2,647,200.00	15	\$3,162,600.00	17	\$2,914,500.00	
	FY 2007	\$1,126,500.00	6	\$1,549,200.00	8	\$1,189,500.00	6	\$1,050,300.00	
	FY 2008	\$372,300.00	3	\$353,100.00	3	\$941,400.00	5	\$723,000.00	
	FY 2009	\$416,100.00	2	\$202,500.00	2	\$279,600.00	2	\$195,300.00	
Bike Total		\$9,020,100.00	56	\$9,357,600.00	57	\$11,636,400.00	65	\$8,853,300.00	
▼ Car	FY 2005	\$1,410,300.00	8	\$1,482,900.00	10	\$304,500.00	2	\$1,149,600.00	
	FY 2006	\$1,003,500.00	4	\$629,700.00	3	\$245,700.00	1	\$337,800.00	
	FY 2007			\$87,300.00	1	\$328,800.00	2	\$169,200.00	
	FY 2008	\$270,000.00	1			\$169,200.00	1		
	FY 2009								
Car Total		\$2,683,800.00	13	\$2,199,900.00	14	\$1,048,200.00	6	\$1,656,600.00	
Grand Total		\$11,703,900.00	69	\$11,557,500.00	71	\$12,684,600.00	71	\$10,509,900.00	

Apply formatting to specific level

ApplyToSpecificLevel property specifies whether to apply conditional formatting to data cells placed at the intersection of specific row and column fields. It can be achieved by defining RowName and ColumnName properties with appropriate pivot field name.

Note: To apply conditional formatting to specific level, make sure that ApplyToAllLevels property is set to false.

To achieve this through XAML, refer to the following code snippet.

XAML

```

<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />

```

```

</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
<!-- Conditional Formatting Cells -->
<syncfusion:PivotGridControl.ColorScaleConditionalFormats>
<syncfusion:PivotGridColorScaleConditionalFormat Name="C1"
CalculationName="Quantity" ApplyToSpecificLevel="True" RowName="Date"
ColumnName="Country" ColorScaleName="GreenYellow" />
</syncfusion:PivotGridControl.ColorScaleConditionalFormats>
</syncfusion:PivotGridControl>
</syncfusion:PivotGridControl>
</Grid>

```

To achieve this through code-behind, refer to the following code snippet.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {

```

```

CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
Syncfusion.PivotAnalysis.Base.SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
// Conditional formatting cells.
PivotGridColorScaleConditionalFormat m_ColorScaleFormat = new
PivotGridColorScaleConditionalFormat()
{
    Name = "C1",
    CalculationName = "Quantity",
    ApplyToSpecificLevel = true,
    RowName = "Date",
    ColumnName = "Country",
    ColorScaleName = ColorScale.GreenYellow
};
//Adding Conditional Formats to PivotGrid
pivotGrid.ColorScaleConditionalFormats.Add(m_ColorScaleFormat);
}
}

```

		▼ Canada		Canada Total	▼ France		France Total	▼ Germany	
		Alberta	Ontario		Garonne (Haute)	Gers		Bayern	Brandenburg
▼ Bike	FY 2005	34	32	66	39	54	93	33	37
	FY 2006	11	10	21	20	18	38	13	13
	FY 2007	6	6	12	9	8	17	5	5
	FY 2008	3	3	6	4	3	7	5	8
	FY 2009	2	1	3	2	4	6		4
Bike Total		56	52	108	74	87	161	56	67
▼ Car	FY 2005	8	5	13	12	9	21	5	5
	FY 2006	4	4	8	1	4	5	1	2
	FY 2007		3	3	3	1	4		2
	FY 2008	1	3	4	1	1	2		2
	FY 2009				1	1	2	1	2
Car Total		13	15	28	18	16	34	7	13
Grand Total		69	67	136	92	103	195	63	80

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Cell Style and Template in WPF Pivot Grid

Cell styles

Column, row, summary, and value cells of a grid can be formatted independently by using the specific style properties as listed below:

- **ColumnHeaderCellStyle:** Specifies the style for column headers.

- **RowHeaderCellStyle**: Specifies the style for row headers.
- **SummaryCellStyle**: Specifies the style for summary cells.
- **ValueCellStyle**: Specifies the style for value cells.

The following properties of the cell can be customized.

Property Name	Description	Type
Background	Gets or sets the background color of a grid cell.	Brush
FontFamily	Gets or sets the font family of a grid cell.	FontFamily
FontSize	Gets or sets the font size of a grid cell.	int
FontWeight	Gets or sets the font weight of a grid cell.	FontWeight
Foreground	Gets or sets the foreground color of a grid cell.	Brush

Defining the cell styles in pivot grid

After defining the pivot grid control, raise the loaded event of pivot grid control. Inside the **PivotGrid_Loaded()** event, set the properties of cell styles of the pivot grid control.

Refer to the following code snippet.

C#

```
public partial class MainWindow: Window {
    public MainWindow() {
        InitializeComponent();
        pivotGrid += pivotGrid_Loaded;
    }
    void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
        // Specifying the Background color for Grid column header
        pivotGrid.ColumnHeaderCellStyle.Background = new
        SolidColorBrush(Color.FromRgb(175, 209, 255));
        // Specifying the Background color for Grid row header
        pivotGrid.RowHeaderCellStyle.Background = new
        SolidColorBrush(Color.FromRgb(175, 209, 255));
        // Specifying the Background color for Grid summary cell
        pivotGrid.SummaryCellStyle.Background = new
        SolidColorBrush(Color.FromRgb(206, 225, 248));
        // Specifying the Font Family for Grid column header
        pivotGrid.ColumnHeaderCellStyle.FontFamily = new FontFamily("Arial");
        // Specifying the Font Family for Grid row header
        pivotGrid.RowHeaderCellStyle.FontFamily = new FontFamily("Arial");
        // Specifying the Font Family for Grid summary cell
        pivotGrid.SummaryCellStyle.FontFamily = new FontFamily("Calibri");
    }
}
```

![PivotGrid shows customized styles](Styles-and-Templates-images/PivotGrid shows customized styles.png)

Cell templates

Cell templates feature of pivot grid allows you to define the templates to change the appearance of elements, such as column, row, summary, and value cells that are present in the grid. The style for each element in the grid should be defined of `PivotGridTemplateCell` type. The customized template can be defined for the following properties of pivot grid:

- `ColumnHeaderCellStyle`
- `RowHeaderCellStyle`
- `SummaryHeaderStyle`
- `SummaryCellStyle`
- `ValueCellStyle`

The expander's in the grid can also be customized with any `UIElement` and it should be named as "PART_Expander" to perform drill-up and drill-down operations.

Defining cell templates in pivot grid

After defining the pivot grid control, define your own style for row, column, value, and summary cells and assign that style to the corresponding property in the pivot grid control.

Here, the row header cells are defined with our own style by overriding an expander icon and cell's `TextBlock`. After defining the style, it is applied to `RowHeaderCellStyle` of the pivot grid control.

Refer to the following code snippet.

XML

```
<Window.Resources>
<ResourceDictionary>
<ObjectDataProvider x:Key="data" ObjectType="{x:Type local:ProductSales}"
MethodName="GetSalesData"/>
<Style x:Key="rowStyle" TargetType="{x:Type
syncfusion:PivotGridTemplateCell}">
<Setter Property="MinHeight" Value="25"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type syncfusion:PivotGridTemplateCell}">
<StackPanel Grid.Column="1" Orientation="Horizontal"
Background="#FF119EDA">
<!--Expander control style, to enable expand/collapse control name should be
"PART_Expander" -->
<Expander Margin="1" x:Name="PART_Expander" IsExpanded="{Binding
IsExpanded,RelativeSource={RelativeSource TemplatedParent}}"
Visibility="{Binding Converter={StaticResource expanderVisiblityConverter}}"
Grid.Column="0" />
<!--Image block-->
<Image Margin="2,4,1,0" Grid.Column="1" VerticalAlignment="Top"
HorizontalAlignment="Center">
<Image.Style>
<Style TargetType="{x:Type Image}">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Text, RelativeSource={RelativeSource
TemplatedParent}}" Value="Car">
<Setter Property="Source" Value="{StaticResource Car}"/>
<Setter Property="Width" Value="32"/>
<Setter Property="Height" Value="32"/>
```



```

</DataTrigger>
<DataTrigger Binding="{Binding Path=Text, RelativeSource={RelativeSource
TemplatedParent}}" Value="Bike">
<Setter Property="Source" Value="{StaticResource Bike}"/>
<Setter Property="Width" Value="32"/>
<Setter Property="Height" Value="32"/>
</DataTrigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<TextBlock Grid.Column="1" Margin="3,4,2,0"
Text="{Binding Path=Text, RelativeSource={RelativeSource TemplatedParent}}"
TextWrapping="Wrap"
VerticalAlignment="Top" FontFamily="Segoe UI" FontSize="12"
/>
</StackPanel>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>
</Window.Resources>
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotGridControl.RowHeaderCellStyle>
<syncfusion:PivotGridCellStyle Style="{StaticResource rowStyle}" />
</syncfusion:PivotGridControl.RowHeaderCellStyle>
</syncfusion:PivotGridControl>
</Grid>

```

![[PivotGrid shows templated cells](Styles-and-Templates-images/PivotGrid shows templated cells.png)]

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Cell Selection in WPF Pivot Grid

The pivot grid supports selecting entire row or column and group of cells with or without header cells. It is similar to cell selection in the Microsoft Excel.

Cell selection

The `AllowSelection` property is used to achieve selection behavior in the pivot grid. It can be done through XAML or code-behind.

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" AllowSelection="True" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
};
```

```

PivotItem n_PivotItem1 = new PivotItem() {
    FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
    = SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
    SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.AllowSelection = true;
}
}

```

![PivotGrid shows selected cells](Selection-Images/PivotGrid shows selected cells.png)

To select a row

You can select a whole row in the pivot grid by clicking the specific row header and you can select multiple rows by clicking a row header and dragging it through the other row headers as per requirement.

![PivotGrid shows selection of multiple rows](Selection-Images/PivotGrid shows selection of multiple rows.png)

To select a column

You can select a whole column in the pivot grid by clicking the specific column header and you can select multiple columns by clicking a column header and dragging it through the other column headers as per requirement.

![PivotGrid shows selection of multiple columns](Selection-Images/PivotGrid shows selection of multiple columns.png)

Cell selection with headers

The `AllowSelectionWithHeaders` property is used to achieve the selection behavior along with headers. It can be achieved through XAML or code-behind.

For XAML, refer to the following code sample.

XAML

```

<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" AllowSelectionWithHeaders="True"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />

```

```

<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);

```

```
pivotGrid.AllowSelectionWithHeaders = true;
}
}
```

![PivotGrid selection along with Row Headers](Selection-Images/PivotGrid shows selection along with row headers.png)

PivotGrid selection along with Row Headers

![PivotGrid selection along with Column Headers](Selection-Images/PivotGrid shows selection along with column headers.png)

PivotGrid selection along with Column Headers

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and displays the result in a cross-table format.

Freeze Headers in WPF Pivot Grid

The [WPF Pivot Grid](#) provides built-in support for freezing column and row headers for better viewing of value cells.

The `FreezeHeaders` property in the pivot grid control is used to achieve the same. This can be mentioned either in XAML or code-behind.

For XAML, refer to the following code sample.

XAML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" FreezeHeaders="False" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.FreezeHeaders = false;
    }
}

```

![Freeze headers enabled in PivotGrid](Freeze-headers-images/PivotGrid with freezed headers.png)

PivotGrid when freeze headers enabled

![Freeze headers disabled in PivotGrid](Freeze-headers-images/PivotGrid when freeze headers disabled.png)

PivotGrid when freeze headers disabled

Note: You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Hyperlink Cells in WPF Pivot Grid

The pivot grid allows hyperlinking of cells to retrieve a detailed information about a particular cell. It generates a separate event called `HyperLinkCellClick` for the clicked hyperlink cell, and the `HyperLinkCellClickEventArgs` returns the clicked `PivotCellInfo`.

The **Hyperlink Cell** property can be applied individually to the following cells:

- Column header
- Row header
- Summary header
- Summary cell
- Value cell

Property

- **IsHyperlinkCell**: Gets or sets whether to enable or disable the grid cells of pivot grid as hyperlink cells.

Defining the property in pivot grid

You can enable or disable the hyperlink for the cells separately for all row header, column header, summary header, and value cells through the **IsHyperlinkCell** property. It can be mentioned in code-behind.

C#

```
public partial class MainWindow: Window {
public MainWindow() {
InitializeComponent();
// Enabling Column Header cells as Hyperlink cell
pivotGrid.ColumnHeaderCellStyle.IsHyperlinkCell = true;
// Enabling Row Header cells as Hyperlink cell
pivotGrid.RowHeaderCellStyle.IsHyperlinkCell = true;
// Enabling both Row and Column summary Header cells as Hyperlink cell
pivotGrid.SummaryHeaderStyle.IsHyperlinkCell = true;
// Enabling both Row and Column summary cells as Hyperlink cell
pivotGrid.SummaryCellStyle.IsHyperlinkCell = true;
// Enabling Value cells as Hyperlink cell
pivotGrid.ValueCellStyle.IsHyperlinkCell = true;
}
}
```

![PivotGrid with Column header hyperlink](Hyperlink-Cells-Images/PivotGrid shows column header hyperlink.png)

PivotGrid with Column header hyperlink

![PivotGrid with Row header hyperlink](Hyperlink-Cells-Images/PivotGrid shows row header hyperlink.png)

PivotGrid with Row header hyperlink

![PivotGrid with Summary cell hyperlink](Hyperlink-Cells-Images/PivotGrid shows summary cell hyperlink.png)

PivotGrid with Summary cell hyperlink

![PivotGrid with Summary header hyperlink](Hyperlink-Cells-Images/PivotGrid shows summary header hyperlink.png)

PivotGrid with Summary header hyperlink

![PivotGrid with Value cell hyperlink](Hyperlink-Cells-Images/PivotGrid shows value cell hyperlink.png)

PivotGrid with Value cell hyperlink

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Tooltip in WPF Pivot Grid

Tooltip holds the respective cell value and its row and column information. It can be enabled or disabled using the `ToolTipEnabled` Boolean property. User can customize the tooltip skin at the sample level and also can set the custom text.

Adding tooltip for entire pivot grid

The `ToolTipEnabled` property can be used to achieve this requirement and it can be mentioned in XAML or code-behind.

For XAML, refer to the following code snippet.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" ToolTipEnabled="True" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code snippet.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
}
```



```

pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
    FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
    FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
    FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
    FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.ToolTipEnabled = true;
}
}

```

![PivotGrid Shows ToolTip for value cells](ToolTip-Images/PivotGrid Shows ToolTip.png)

Adding tooltip to specific areas in pivot grid

This can be achieved by setting the appearance of tooltip with respect to each cell styles individually. Each style has its own `ToolTipEnabled` property.

After defining the pivot grid control, raise the loaded event of pivot grid control. Inside the `PivotGrid_Loaded()` event, set the `ToolTipEnabled` property of each cell styles in the pivot grid control.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
    }
}

```

```

};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid += pivotGrid_Loaded;
}
void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
//Enable Tooltip for RowHeaderCellStyle
this.pivotGrid.RowHeaderCellStyle.ToolTipEnabled = true;
//Enable Tooltip for ColumnHeaderCellStyle
this.pivotGrid.ColumnHeaderCellStyle.ToolTipEnabled = true;
//Enable Tooltip for ValueCellStyle
this.pivotGrid.ValueCellStyle.ToolTipEnabled = true;
//Enable Tooltip for SummaryHeaderStyle
this.pivotGrid.SummaryHeaderStyle.ToolTipEnabled = true;
//Enable Tooltip for SummaryCellStyle
this.pivotGrid.SummaryCellStyle.ToolTipEnabled = true;
}
}

```

![PivotGrid shows Tooltip for summary header](ToolTip-Images/PivotGrid shows Tooltip for summary header.png)

PivotGrid shows tooltip for summary header

![PivotGrid shows Tooltip for column header](ToolTip-Images/PivotGrid shows Tooltip for column header.png)

PivotGrid shows tooltip for column header

![PivotGrid shows Tooltip for row header](ToolTip-Images/PivotGrid shows Tooltip for row header.png)

PivotGrid shows tooltip for row header

![PivotGrid shows Tooltip for summary values](ToolTip-Images/PivotGrid shows Tooltip for summary value.png)

PivotGrid shows tooltip for summary values

Adding custom tooltip for pivot grid

Custom data templates can be set to the pivot grid controls tooltip using the `CustomToolTipTemplateKey` property. To do so, write a data template and bind the style's `Tag` property, and then set the key to the pivot grid control's `CustomToolTipTemplateKey` property.

Refer to the following code snippets.

XML

```
<Window.Resources>
<ResourceDictionary>
<ObjectDataProvider x:Key="data" ObjectType="{x:Type local:ProductSales}"
MethodName="GetSalesData" />
<DataTemplate x:Key="CustomTemplateTooltip">
<Border BorderThickness="1" BorderBrush="Black" Background="LightGreen"
CornerRadius="2">
<StackPanel Margin="5" Orientation="Horizontal" VerticalAlignment="Center">
<TextBlock Text="{ Binding Tag }" VerticalAlignment="Center" />
</StackPanel>
</Border>
</DataTemplate>
</ResourceDictionary>
</Window.Resources>
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" ToolTipEnabled="True"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

C#

```
public partial class MainWindow: Window {
public MainWindow() {
InitializeComponent();
pivotGrid.CustomToolTipTemplateKey = "CustomTemplateTooltip";
}
}
```

Similarly, you can define the custom tooltip to specific areas with respect to individual cell styles using the `CustomToolTipTemplateKey` property of row, column, summary header, and value cell styles.

![PivotGrid shows custom ToolTip](ToolTip-Images/PivotGrid shows CustomToolTip.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Skin Customization in WPF Pivot Grid

Skin customization applies particular style settings to the visual elements of a component. The pivot grid provides the following skin options:

- Office 2010 Blue
- Office 2010 Black
- Office 2010 Silver
- Transparent
- Office 2007 Blue
- Office 2007 Black
- Office 2007 Silver
- Blend
- Metro
- Office 2003
- Default

The `VisualStyle` property is used to customize the skin. It can be mentioned in XAML or code-behind.

For XAML, refer to the following code snippet.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top"
VisualStyle="Office2010Blue" ItemSource="{Binding Source={StaticResource
data}}" >
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName = "Total" FieldName =
"Amount" Format="C" SummaryType="DoubleTotalSum" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

```
<syncfusion:PivotComputationInfo CalculationName = "Total" FieldName =
"Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code snippet.

C#

```
public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid.VisualStudio = PivotGridVisualStyle.Office2010Blue;
    }
}
```

![PivotGrid shows customized visual style](Skin-Customization-Images/PivotGrid shows customized visual style.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Serialization and Deserialization in WPF Pivot Grid

Serialization can be implemented for applications that need to save their data and structure after closed. Serialization supports saving the structure and data of pivot grid control to an XML file and it can be loaded at any time.

The following are the properties of pivot grid that can be serialized.

Property Name	Type
AllowResizeColumns	bool
AllowResizeRows	bool
AllowSelection	bool
AllowSelectionWithHeaders	bool
AutoSizeColumnCount	int
AutoSizeOption	GridAutoSizeOption
AutoSizeRowCount	int
DeferLayoutUpdate	bool
Filters	ObservableCollection()
FreezeHeaders	bool
IsDynamicData	bool
PivotCalculations	ObservableCollection()
PivotColumns	ObservableCollection()
PivotFields	ObservableCollection()
PivotRows	ObservableCollection()
ShowCalculationsAsColumns	bool
ShowFieldList	bool
ShowGrandTotals	bool
ShowGroupingBar	bool
GroupingBar.AllowFiltering	bool
GroupingBar.AllowSorting	bool
EnableColumnHeader	bool
EnableRowHeader	bool
AllowMultiFunctionalSortFilter	bool

VisibleRecords	int
----------------	-----

Using the serialization/deserialization in pivot grid

On serialization, the expand and collapse states of pivot grid cells are maintained. So when deserializing, the item source specified for the grid should be same as the item source used in serialization. This can be ignored by setting the `IgnoreExpandCollapseOnSerialization` property of pivot grid control to "False".

The following are the methods used in pivot grid for serialization/deserialization.

Methods table

Method	Description	Parameters	Return Type
Serialize()	Serializes the pivot grid into an XML file format using the save file dialog.	-	void
Deserialize()	Deserialize the pivot grid from the saved XML file using the open file dialog.	-	void
Serialize(string fileName)	Serializes the pivot grid into an XML file format and saves it in the specified location.	string fileName	void
Deserialize(string filename)	Deserialize the pivot grid from the specified XML file.	string fileName	void
SerializedXmlString()	Serializes some specific properties of pivot grid control in string format.	-	string
Deserialize(string XmlString)	Deserialize the XML string format in the pivot grid control.	string XmlString	void

Serialization

After defining a pivot grid control, call the `Serialize()` method in the separate event handler. Refer to the following code snippet.

C#

```
public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        this.grid1.Children.Add(pivotGrid);
    }
    void button1_Click(object sender, RoutedEventArgs e) {
        // Serialize the PivotGrid into XML file format.
        pivotGrid.Serialize();
        // Serialize the PivotGrid into XML file format and saves it in the
        // specified location.
        pivotGrid.Serialize(@"C:/PivotGrid.xml");
        // Serializes some specific properties in PivotGridControl into string
        // format
    }
}
```

```
pivotGrid.SerializedXmlString();
}
}
```

Deserialization

After defining a pivot grid control, call the De-Serialize() method in the separate event handler. Refer to the following code snippet.

C#

```
public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow() {
        InitializeComponent();
        this.grid1.Children.Add(pivotGrid);
    }
    void button1_Click(object sender, RoutedEventArgs e) {
        // De-Serialize the PivotGrid into XML file format.
        pivotGrid.Deserialize();
        // De-serializes the PivotGrid from the specified XML file
        pivotGrid.Deserialize(@"C:/PivotGrid.xml");
        // De-serializes the XML string format into PivotGridControl
        pivotGrid.DeserializeXmlString(pivotGrid.SerializedXmlString());
    }
}
```

![[Pivot grid serialized as xml string]](Serialization-images/Serialized PivotGrid.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Exporting in WPF Pivot Grid

The [WPF Pivot Grid](#) provides an option to export its content to various formats. It provides the following exporting options.

- Microsoft Excel format - Cell and pivot table mode
- Microsoft Word format
- PDF format
- CSV format

Export to Excel

The pivot grid can be exported to an Excel document using Syncfusion XlsIO libraries. It consists of two exporting modes, and you can switch from one mode to another mode using the **ExportMode** property.

- **Pivot Table:** Exports the pivot grid directly as a pivot table like in Microsoft Excel.
- **Cell:** Exports the pivot grid as individual cell values.

To export the pivot grid contents to Excel, add **Syncfusion.PivotGridConverter.Wpf.dll** and **Syncfusion.XlsIO.Base** assemblies to your application and include the namespace **Syncfusion.Windows.Controls.PivotGrid.Converter** to access the **GridExcelExport** class.

Now, define the pivot grid control and bind the data source along with pivot rows, pivot columns, and pivot calculations. Create an instance of `GridExcelExport` class and then invoke the `Export` method in it. `OpenFileDialog` and `SaveFileDialog` options are used to save the exported file in the preferred location.

Refer to the following code sample.

C#

```
private void Button_Export(object sender, RoutedEventArgs e) {
    //////Export to Excel document
    SaveFileDialog savedialog = new SaveFileDialog();
    savedialog.AddExtension = true;
    savedialog.FileName = "Sample";
    savedialog.DefaultExt = "xlsx";
    savedialog.Filter = "Excel file (.xlsx)|*.xlsx";
    if (savedialog.ShowDialog() == true) {
        fileName = savedialog.FileName;
        GridExcelExport excelExport = new GridExcelExport(pivotGrid,
            Syncfusion.XlsIO.ExcelVersion.Excel2007);
        excelExport.ExportMode = ExportModes.PivotTable;
        excelExport.Export(fileName);
    }
}
```

![Excel Export in PivotTable Mode](Exporting-Images/Export to excel in PivotTable mode.png)

Excel Export in PivotTable Mode

![Excel Export in Cell Mode](Exporting-Images/Export to excel in Cell mode.png)

Excel Export in Cell Mode

Export to Word

To export the pivot grid contents to Word, add the **Syncfusion.PivotGridConverter.Wpf.dll** assembly to your application and include the namespace **Syncfusion.Windows.Controls.PivotGrid.Converter** to access the `GridWordExport` class.

Now, define the pivot grid control and bind the data source along with pivot rows, pivot columns, and pivot calculations. Create an instance for `GridWordExport` class and then invoke the `Export` method in it. `OpenFileDialog` and `SaveFileDialog` options are used to save the exported file in the preferred location.

Refer to the following code sample.

C#

```
private void Button_Export(object sender, RoutedEventArgs e) {
    ////// Export to Word document
    SaveFileDialog savedialog = new SaveFileDialog();
    savedialog.AddExtension = true;
    savedialog.FileName = "Sample";
    savedialog.DefaultExt = "Doc";
    savedialog.Filter = "Word file (.Doc)|*.Doc";
    if (savedialog.ShowDialog() == true) {
        fileName = savedialog.FileName;
        GridWordExport wordExport = new GridWordExport(pivotGrid);
    }
}
```

```
wordExport.Export(fileName);  
}  
}
```

![Exports the PivotGrid into word document](Exporting-Images/Export to word.png)

Export to PDF

To export the pivot grid contents to PDF, add **Syncfusion.PivotGridConverter.Wpf.dll** assembly to your application and include the namespace **Syncfusion.Windows.Controls.PivotGrid.Converter** to access the **GridPdfExport** class.

Now, define the pivot grid control and bind the data source along with pivot rows, pivot columns, and pivot calculations. Create an instance for **GridPdfExport** class and then, invoke the **Export** method in it. **OpenFileDialog** and **SaveFileDialog** options are used to save the exported file in the preferred location.

Refer to the following code sample.

C#

```
private void Button_Export(object sender, RoutedEventArgs e) {  
    ////Export to PDF document  
    SaveFileDialog savedialog = new SaveFileDialog();  
    savedialog.AddExtension = true;  
    savedialog.FileName = "Sample";  
    savedialog.DefaultExt = "pdf";  
    savedialog.Filter = "Pdf file (.pdf) | *.pdf";  
    if (savedialog.ShowDialog() == true) {  
        fileName = savedialog.FileName;  
        GridPdfExport pdfExport = new GridPdfExport(pivotGrid);  
        pdfExport.Export(fileName);  
    }  
}
```

![Exports the PivotGrid into PDF document](Exporting-Images/Export to pdf.png)

Export to CSV

To export the pivot grid contents to CSV, add the **Syncfusion.PivotGridConverter.Wpf.dll** assembly to your application and include the namespace **Syncfusion.Windows.Controls.PivotGrid.Converter** to access the **GridCsvExport** class.

Now, define the pivot grid control and bind the data source along with pivot rows, pivot columns, and pivot calculations. Create an instance of **GridCsvExport** class and then invoke the **Export** method in it. **OpenFileDialog** and **SaveFileDialog** options are used to save the exported file in the preferred location.

Refer to the following code sample.

C#

```
private void Button_Export(object sender, RoutedEventArgs e) {  
    ////Export to CSV document  
    SaveFileDialog savedialog = new SaveFileDialog();  
    savedialog.AddExtension = true;  
    savedialog.FileName = "Sample";
```

```

savedialog.DefaultExt = "CSV";
savedialog.Filter = "CSV file (*.csv)|*.csv";
if (savedialog.ShowDialog() == true) {
    fileName = savedialog.FileName;
    GridCsvExport csvExport = new GridCsvExport(pivotGrid);
    csvExport.Delimiter = ",";
    csvExport.Export(fileName);
}
}

```

![Exports the PivotGrid into CSV document](Exporting-Images/Export to csv.png)

Threshold limitations

The following table represents the number of rows and columns to be taken in account while exporting the pivot grid control without affecting its performance.

Functionality	Row Count	Column Count
Exporting PivotGrid contents to Excel document in Cell mode	300	70
Exporting PivotGrid contents to Excel document in PivotTable mode	No threshold limit	No threshold limit
Exporting PivotGrid contents to Word document	1800	70
Exporting PivotGrid contents to PDF document	800	70
Exporting PivotGrid contents to CSV document	No threshold limit	No threshold limit

Note: You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Printing in WPF Pivot Grid

The printing support for pivot grid control is provided along with grouping bar. You can use the `ShowPrintPreview` method to preview the pivot grid control before printing. The `PrintHeader` and `PrintFooter` properties are used to add or remove the header and footer information while printing.

Defining header and footer in XAML

After defining the pivot grid control, define the `DataTemplate` for header and footer and set the `PrintHeader` and `PrintFooter` properties of the pivot grid control.

Refer to the following code sample to learn about the header template, footer template, and setting its properties in the pivot grid control.

XML

```

<Window.Resources>
<ResourceDictionary>
<ObjectDataProvider x:Key="data" ObjectType="{x:Type local:ProductSales}"
    MethodName="GetSalesData" />
<!--Creating Template for Header -->
<DataTemplate x:Key="HeaderTemplate">
<Grid Height="30">

```

```

<TextBlock Text="Header" FontSize="15" FontWeight="Bold"
HorizontalAlignment="Center"></TextBlock>
</Grid>
</DataTemplate>
<!--Creating Template for Footer -->
<DataTemplate x:Key="FooterTemplate">
<Grid Height="30">
<TextBlock Text="Footer" FontSize="15" FontWeight="Bold"
HorizontalAlignment="Center"></TextBlock>
</Grid>
</DataTemplate>
</ResourceDictionary>
</Window.Resources>
<Grid Name="grid1">
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro"
syncfusion:PrintSettings.PrintHeader="True"
syncfusion:PrintSettings.PrintFooter="True" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

Invoking print preview window in pivot grid

After defining the pivot grid control, raise the loaded event of pivot grid. Inside the `pivotGrid_Loaded` event, invoke the `ShowPrintPreview` method to enable the printing behavior.

C#

```

public partial class MainWindow: Window {
public MainWindow() {
InitializeComponent();
this.pivotGrid.Loaded += pivotGrid_Loaded;
}
void pivotGrid_Loaded(object sender, RoutedEventArgs e) {
//Shows the Print Preview window with templates for header, footer and with
user defined title.
}
}

```

```

this.pivotGrid.ShowPrintPreview((DataTemplate)
this.Resources["HeaderTemplate"], (DataTemplate)
this.Resources["FooterTemplate"], "Printing PivotGrid", this);
//Shows the Print Preview window with empty template and default title.
this.pivotGrid.ShowPrintPreview(this);
//Shows the Print Preview window with templates for header, footer and with
default title.
this.pivotGrid.ShowPrintPreview((DataTemplate)
this.Resources["HeaderTemplate"], (DataTemplate)
this.Resources["FooterTemplate"], this);
}
}

```

![[Printing pivot grid](Print-Images/Print preview.png)]

Print preview options

The print preview window provides the following options:

- Zooming
- Page settings
- Print

Zooming

Click the **Zoom** drop-down button in the print preview window and select the desired percentage to magnify the pivot grid view. You can choose from various preset zoom level options such as 50%, 100%, 200% or 400%.

Page settings

Click the **PageSettings** in the print preview window to change the pages while printing.

Print

Click the **Print** in the print preview window to print the pivot grid content.

![[Print preview window](Print-Images/Print and Zooming options.png)]

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Localization in WPF Pivot Grid

Localization deals with customizing data and resources for specific culture or language. The built-in localization and globalization mechanism in WPF allows you to localize any string resource used by pivot grid controls. Localization can be done in the following ways.

- Localization using resource file.
- Localization using satellite assembly.

Localization using resource file

Create and place the resource files in a separate location inside the user application. Then, access the culture specific resources from the current application assembly.

For this, first create a resource file for the pivot grid control and translate the strings to your culture. After the strings are translated, you might use the resources in your projects by setting the corresponding culture in your application.

Refer to the following code sample.

C#

```
public MainWindow() {
    //Set the current thread culture to load the localization resource file.
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("ar-AE");
    InitializeComponent();
    if (System.Globalization.CultureInfo.CurrentUICulture.ToString() == "ar-AE")
    {
        this.FlowDirection = System.Windows.FlowDirection.RightToLeft;
    }
}
```

Localization using satellite assembly

Here, the resource file should be embedded in a satellite assembly and can be used in the application for applying localization.

The following steps illustrate the process of creating satellite assembly:

Open the **Visual Studio Command prompt** and navigate to the folder that contains the .resx* file in your application.

Generate a .resources* file that is to be embedded in the satellite assembly by entering the following command in the command prompt.

system drive> Resgen MyResource.ar-AE.resx

After executing the above command, a .resources* file will be generated in the same folder. Then, enter the following command in the command prompt.

system drive> AL.exe

Create a satellite assembly for your culture by embedding the .resources* file using the following command.

**system drive> al /target:lib /embed:MyResource.ar-AE.resources /culture:ar-AE
/out:MyApp.resources.dll**

Now, MyResource.ar-AE.resources is the resource file embedded in the MyApp.resources.dll.

After executing the above command, the satellite assembly will be generated in the same folder. Finally, place the assembly in the following path:

{Location of the application}\bin\debug\ar-AE\MayAppName.resources.dll

Note: Make sure that the name of satellite assembly should be in the format *MyAppName.resources.dll* and the name of the .resx* file should be like *Syncfusion.PivotAnalysis.Wpf.ar-AE.resx* (Arabic). If the name of dll's differs from your application name, then localization will not work.

![Localized pivot grid and display the data from right to left](Localization-Images/PivotGrid Shows localization behaviour.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

PivotSchemaDesigner

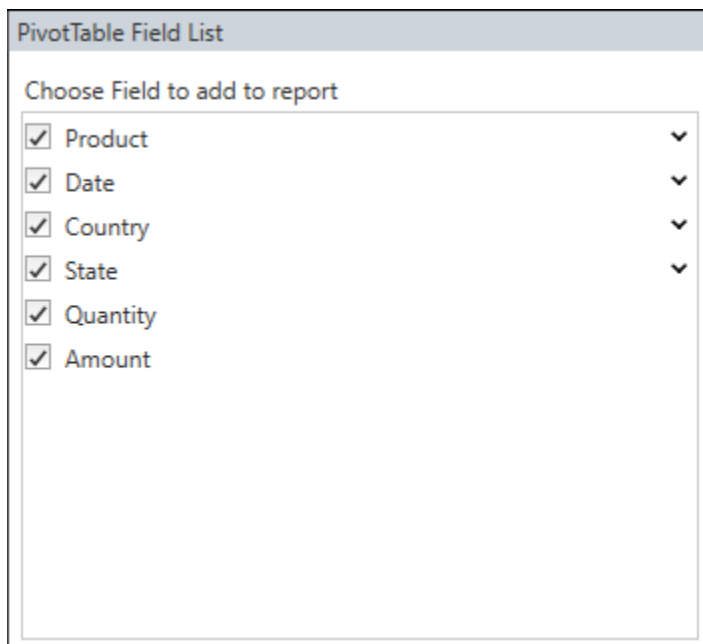
Overview in WPF Pivot Grid

The pivot schema designer can be supported in the pivot grid, so that it can be presented like an **ExcelPivotTable**. By using the pivot schema designer, you can add, rearrange, or remove fields to show data in the pivot grid exactly the way you want. It contains two sections, consisting of the following items:

- A **field section** at the top is used for adding and removing fields from the pivot grid.
- A **layout section** at the bottom is used for rearranging and repositioning the fields in the pivot grid.

Fields section

The **Fields section** consists of a list of fields present in the pivot grid including row, column, and summary elements, which is called as **PivotTable Field List**. A field will be added to the pivot grid if it is checked, or it will be removed from the pivot grid if it is unchecked. By default, fields will be added to the row label if checked, and added to the column label by simply dragging the field to the column label area. The filtering process is also supported in the pivot table field list. Filter pop-up will be opened while clicking the expander icon in the right-corner of each pivot table field list items.



Layout section

The layout section is used to rearrange and reposition the fields in a pivot grid. It has the following areas:

- Report filter
- Column label
- Row label

- Values

![Layout section](PivotSchemaDesigner-Images/Layout section.png)

Report filter

The report filter is used to filter the entire report based on the selected item in the report filter. The report filter pop-up window can be launched by clicking the expander icon available in the right-corner of each filter item.

![Filter Popup window](PivotSchemaDesigner-Images/Filter Pop up window.png)

Column label

The column label is used to display fields as columns at the top of a report. A column at the lower position is nested within another column positioned immediately above it in the pivot grid.

Row label

Row label is used to display fields as rows at the top of a report. A row at the lower position is nested within another row positioned immediately above it in the pivot grid.

Values

The values section is used to display the summary fields of the pivot grid.

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Binding PivotSchemaDesigner in WPF Pivot Grid

The pivot schema designer can be bound to the pivot grid with the help of the `PivotControl` property. After creating a pivot grid, a new pivot schema designer is created using the `PivotSchemaDesigner` class and bound to the pivot grid control using the `PivotControl` property. This can be achieved either in XAML or code-behind.

For XAML, refer to the following code sample.

XML

```
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width=".35*" />
</Grid.ColumnDefinitions>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro"
EnableValueEditing="True" ItemSource="{Binding Source={StaticResource
data}}" >
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
</syncfusion:PivotGridControl>
</Grid>
```



```

</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName = "Total" FieldName =
"Amount" Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName = "Total" FieldName =
"Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
<syncfusion:PivotSchemaDesigner Grid.Column="1" Name="shemaDesginer"
VisualStyle="Metro" PivotControl="{Binding
ElementName=pivotGrid}"></syncfusion:PivotSchemaDesigner>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow : Window
{
    PivotGridControl pivotGrid = new PivotGridControl();
    PivotSchemaDesigner schemaDesigner = new PivotSchemaDesigner();
    public MainWindow()
    {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() { FieldHeader = "Product",
        FieldMappingName = "Product", TotalHeader = "Total" };
        PivotItem m_PivotItem1 = new PivotItem() { FieldHeader = "Date",
        FieldMappingName = "Date", TotalHeader = "Total" };
        PivotItem n_PivotItem = new PivotItem() { FieldHeader = "Country",
        FieldMappingName = "Country", TotalHeader = "Total" };
        PivotItem n_PivotItem1 = new PivotItem() { FieldHeader = "State",
        FieldMappingName = "State", TotalHeader = "Total" };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
        CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
        = Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
        CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
        Syncfusion.PivotAnalysis.Base.SummaryType.Count };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        schemaDesigner.PivotControl = pivotGrid;
        grid1.Children.Add(schemaDesigner);
        Grid.SetColumn(schemaDesigner, 1);
    }
}

```

PivotTable Field List

Choose Field to add to report

☒ Product
 ☒ Date
 ☒ Country
 ☒ State
 ☒ Quantity
 ☒ Amount
 ☐ UnitPrice
 ☐ TotalPrice

Drag fields between areas below:

Report Filter

Column Label

Country

State

Row Label

Product

Date

Σ Values

Amount

Quantity

☒ Show Calculations as column
 ☐ Defer Layout Update

Update

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Setting Caption in PivotTableFieldList in WPF Pivot Grid

The pivot grid supports duplicating the same fields in different names. This can be achieved by using the `FieldCaption` property of `PivotItem` and `PivotComputationInfo`. This support allows you to define multiple items in same underlying type for the pivot grid control.

The following code sample illustrates how to set the field caption for `PivotItem` and `PivotComputationInfo`.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
FreezeHeaders="False" VerticalAlignment="Top" VisualStyle="Metro"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldMappingName="Product" FieldCaption="Product_1"
TotalHeader="Total"/>

```

```

<syncfusion:PivotItem FieldMappingName="Product" FieldCaption="Product_2"
TotalHeader="Total"/>
<syncfusion:PivotItem FieldMappingName="Date" FieldHeader="Date"
FieldCaption="Date" TotalHeader="Total"/>
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldMappingName="Country" FieldHeader="Country"
FieldCaption="Country" TotalHeader="Total"/>
<syncfusion:PivotItem FieldMappingName="State" FieldHeader="State"
FieldCaption="State" TotalHeader="Total"/>
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total"
Description="Summation of values" FieldName="Amount"
FieldCaption="Amount_1" Format="C" SummaryType="DoubleTotalSum"/>
<syncfusion:PivotComputationInfo CalculationName="Total"
Description="Summation of values" FieldName="Amount"
FieldCaption="Amount_2" Format="#,##" SummaryType="IntTotalSum"/>
<syncfusion:PivotComputationInfo CalculationName="Total"
Description="Summation of values" FieldName="Quantity"
FieldCaption="Quantity" Format="#,##0"/>
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

C#

```

public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem() {
FieldCaption = "Product_1", FieldMappingName = "Product", TotalHeader =
"Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
FieldCaption = "Product_2", FieldMappingName = "Product", TotalHeader =
"Total"
};
PivotItem m_PivotItem2 = new PivotItem() {
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
pivotGrid.PivotRows.Add(m_PivotItem2);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);

```

```
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", FieldCaption = "Amount_1",
    Format = "C", SummaryType = SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", FieldCaption = "Amount_2",
    Format="#,##", SummaryType = SummaryType.IntTotalSum
};
PivotComputationInfo m_PivotComputationInfo2 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", Format="#,##0",
    SummaryType = SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo2);
}
}
```

PivotTable Field List

Choose Field to add to report

☒ Product_1
☒ Date
☒ Country
☒ State
☒ Quantity
☒ Amount_1
☒ Amount_2
☒ Product_2

Drag fields between areas below:

Report Filter

Column Label

Country
State

Row Label

Product_1
Product_2
Date

Values

Amount_1
Amount_2
Quantity

☒ Show Calculations as column

☐ Defer Layout Update

Update

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Pivot Computation Information Dialog in WPF Pivot Grid

The pivot computation information dialog is used to change or edit value formats, summary types, calculation types, and the field header.

Field Name	Amount
Filter Header	Amount
Description	Summation of values
Format	C
Summarize Value By	DoubleTotalSum
Show Value As	NoCalculation
Base Field	
Base Item	

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Hiding Fields in WPF Pivot Grid

You can hide the unnecessary fields from the pivot table field list using the `ShowDisplayFieldsOnly` property.

For XAML, refer to the following code sample.

XML

```
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width=".35*" />
</Grid.ColumnDefinitions>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" VisualStyle="Metro" EnableValueEditing="True"
ItemSource="{Binding Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

```

</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
<syncfusion:PivotSchemaDesigner Grid.Column="1" Name="shemaDesginer"
ShowDisplayFieldsOnly="True" VisualStyle="Metro" PivotControl="{Binding
ElementName=pivotGrid}"></syncfusion:PivotSchemaDesigner>
</Grid>

```

For code-behind, refer to the following code sample.

C#

```

public partial class MainWindow: Window {
    PivotGridControl pivotGrid = new PivotGridControl();
    PivotSchemaDesigner schemaDesigner = new PivotSchemaDesigner();
    public MainWindow() {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem() {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem() {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem() {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem() {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
            CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
            = Syncfusion.PivotAnalysis.Base.SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
            CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
            Syncfusion.PivotAnalysis.Base.SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        schemaDesigner.PivotControl = pivotGrid;
        schemaDesigner.ShowDisplayFieldsOnly = true;
        grid1.Children.Add(schemaDesigner);
        Grid.SetColumn(schemaDesigner, 1);
    }
}

```

![[PivotSchemaDesigner with ShowDisplayFieldsOnly](PivotSchemaDesigner-Images/PivotTableFieldList when enabled showsdisplayfields only.png)]

PivotSchemaDesigner with ShowDisplayFieldsOnly

![PivotSchemaDesigner without ShowDisplayFieldsOnly](PivotSchemaDesigner-Images/PivotTableFieldList when disabled showsdisplayfields only.png)

PivotSchemaDesigner without ShowDisplayFieldsOnly

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

Displaying Calculations in WPF Pivot Grid

The pivot grid provides support to show the calculation values in column or row. This can be achieved by using the `ShowCalculationsAsColumns` property of pivot grid control or through simple uncheck or check option in the pivot schema designer.

Setting ShowCalculationsAsColumns property through code

For XAML, refer to the following code sample.

XML

```
<Grid>
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
FreezeHeaders="False" VerticalAlignment="Top" VisualStyle="Metro"
ShowCalculationsAsColumns="False" ItemSource="{Binding
Source={StaticResource data}}">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity" SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

For code-behind, refer to the following code sample.

C#

```
public partial class MainWindow: Window {
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow() {
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
}
```



```

PivotItem m_PivotItem = new PivotItem() {
    FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem() {
    FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem() {
    FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem() {
    FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo() {
    CalculationName = "Amount", FieldName = "Amount", Format = "C", SummaryType
= SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo() {
    CalculationName = "Quantity", FieldName = "Quantity", SummaryType =
SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid.ShowCalculationsAsColumns = false;
}
}

```

Setting ShowCalculationsAsColumns property through pivot schema designer

You can enable or disable the `ShowCalculationsAsColumns` property through the "Show Calculations as column" checkbox of the pivot schema designer. To view the calculation values in row, uncheck the "Show Calculations as column" checkbox.

![[Layout section]](PivotSchemaDesigner-Images/Layout section.png)

Note: You can refer to our [WPF Pivot Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Pivot Grid example](#) to know how to organize and summarize business data and display the result in a cross-table format.

How to

How to load an existing PivotEngine into PivotGrid?

After defining the PivotGrid control, set the property `IsExternalEngine` as true. Now create a new PivotEngine using `PivotEngine` class and assign it to your PivotGrid control's PivotEngine.

Please refer the below code sample.

C#

```

public MainWindow()
{
    InitializeComponent();
}

```

```
PivotGridControl pivotGrid = new PivotGridControl();
this.grid1.Children.Add(pivotGrid);
///Set IsExternalEngine to true to use the external PivotEngine.
pivotGrid.IsExternalEngine = true;
///Create a new instance of the PivotEngine.
PivotEngine pivotEngine = new PivotEngine();
///Assign the new engine to PivotGrid control.
pivotGrid.PivotEngine = pivotEngine;
}
```

How to improve loading and scrolling performance in PivotGrid?

The performance of the PivotGrid control can be improved by enabling the on-demand calculation on the value cells and by disabling the auto-sizing option. This refreshes the calculation only while loading or scrolling the PivotGrid control. This can be achieved by using `EnableOnDemandCalculations` and `AutoSizeOption` properties of PivotGrid control.

Please refer the below code snippet.

C#

```
public MainWindow()
{
    InitializeComponent();
    pivotGrid.AutoSizeOption = GridAutoSizeOption.None;
    pivotGrid.PivotEngine.EnableOnDemandCalculations = true;
}
```

How to expand and collapse entire group in PivotGrid?

Expanding entire group in PivotGrid

After defining PivotGrid control, invoke the method `ExpandAllGroup()` to expand entire group in the PivotGrid control.

Please refer the below code sample.

C#

```
public MainWindow()
{
    InitializeComponent();
    ///To expand entire group in PivotGrid
    pivotGrid.ExpandAllGroup();
}
```

Collapsing entire group in PivotGrid

After defining PivotGrid control, invoke the method `CollapseAllGroup()` to collapse entire group in the PivotGrid control.

Please refer the below code sample.

C#

```
public MainWindow()
{
    InitializeComponent();
}
```

```
//To collapse entire group in PivotGrid
pivotGrid.CollapseAllGroup();
}
```

How to get the count of records in WPF PivotGridControl

It is possible to get the count of records which are currently visible in PivotGrid by using **VisibleRecords** property.

After defining the PivotGrid control, raise its loaded event. Inside the **pivotGrid_Loaded** event, you can get the list of **VisibleRecords** from the PivotEngine and store it as a separate collection.

Please refer the below code sample.

C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        pivotGrid.Loaded += pivotGrid_Loaded;
    }
    void pivotGrid_Loaded(object sender, RoutedEventArgs e)
    {
        List<object> VisibleRecords = pivotGrid.PivotEngine.VisibleRecords;
        int _count = VisibleRecords.Count;
    }
}
```

How to hide the Grand Total present in the PivotGrid?

It can be achieved by setting the property **ShowGrandTotals** to false. By default, PivotGrid displays Grand Total for both column and row headers.

It can be mentioned either in *XAML* or in *Code-Behind*.

If through *XAML*, please refer the below code sample.

XAML

```
<Grid>
<syncfusion:PivotGridControl x:Name="pivotGrid" ShowGrandTotals="False">
</syncfusion:PivotGridControl>
</Grid>
```

Else if through *Code-Behind*, please refer the below code sample.

C#

```
public MainWindow()
{
    InitializeComponent();
    this.pivotGrid.ShowGrandTotals = false;
}
```

How to change the GridLine color and thickness?

GridLine color can be changed by using the `GridLineStroke` property of PivotGrid and it can be mentioned either in XAML or in Code-Behind.

If through XAML, please refer the below code sample.

XML

```
<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top" GridLineStroke="#2F5BB7" ItemSource="{Binding
Source={StaticResource data}}">
</syncfusion:PivotGridControl>
```

Else if through Code-Behind, please refer the below code sample.

C#

```
public MainWindow() {
    InitializeComponent();
    pivotGrid.GridLineStroke = new SolidColorBrush(Colors.Black);
}
```

How to customize the appearance of expanders in PivotGrid?

Define your own style for the expander and assign that style to the `ExpanderStyle` property of the PivotGrid control.

Please refer the below code sample. Here we have defined our own style for expander and assigned it to the `ExpanderStyle` property of the PivotGrid control.

```
{}
```

XML

```
<Window.Resources>
<Style TargetType="ToggleButton" x:Key="GridExpanderStyle">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type ToggleButton}">
<Border Padding="{TemplateBinding Padding}">
<Grid SnapsToDevicePixels="False" Background="Transparent">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="13"/>
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Ellipse x:Name="circle" Stroke="DarkGray" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="13" Height="13"/>
<Path x:Name="arrow" Stroke="#666" StrokeThickness="2"
HorizontalAlignment="Center" VerticalAlignment="Center"
SnapsToDevicePixels="false" Data="M 1,4.5 L 4.5,1 L 8,4.5"/>
<ContentPresenter HorizontalAlignment="Left" Margin="2,0,0,0"
VerticalAlignment="Center" SnapsToDevicePixels="True" Grid.Column="1"
RecognizesAccessKey="True"/>
</Grid>
</Border>
<ControlTemplate.Triggers>
<Trigger Property="IsChecked" Value="true">
```

```

<Setter Property="Data" TargetName="arrow" Value="M 1,1.5 L 4.5,5 L 8,1.5"/>
</Trigger>
<Trigger Property="IsMouseOver" Value="true">
<Setter Property="Stroke" TargetName="circle" Value="#FF3C7FB1"/>
<Setter Property="Stroke" TargetName="arrow" Value="#222"/>
</Trigger>
<Trigger Property="IsPressed" Value="true">
<Setter Property="Stroke" TargetName="circle" Value="#FF526C7B"/>
<Setter Property="StrokeThickness" TargetName="circle" Value="1.5"/>
<Setter Property="Stroke" TargetName="arrow" Value="#FF003366"/>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
<!--Binding the expander style to the grid-->
<syncfusion:PivotGridControl Name="pivotGrid" ExpanderStyle="{StaticResource
GridExpanderStyle}" HorizontalAlignment="Left" VerticalAlignment="Top"/>
</Grid>

```

How to set defer layout update in PivotGrid?

It can be achieved by setting the property `DeferLayoutUpdate` to true via code or through the `DeferLayoutUpdate` check-box (UI option) inside `PivotSchemaDesigner` control.

If through *XAML*, please refer the below code sample.

XML

```

<syncfusion:PivotGridControl HorizontalAlignment="Left" Name="pivotGrid"
VerticalAlignment="Top"
DeferLayoutUpdate="True" ItemSource="{Binding Source={StaticResource
data}}" >
</syncfusion:PivotGridControl>

```

Else if through *Code-behind*, please refer the below code sample.

C#

```

public MainWindow()
{
    InitializeComponent();
    pivotGrid.DeferLayoutUpdate = true;
}

```

How to resizing the columns and rows in PivotGrid?

`PivotGridControl` supports resizing of rows and columns dynamically. In this topic, on-demand resizing of rows and columns is discussed.

Resizing the columns

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.

To enable the column resizing dynamically, the [AllowResizeColumns](#) property should be enabled. The below code snippet shows the resizing of columns in pivot grid is being enabled.

XML

```
<Grid>
<syncfusion:PivotGridControl Name="pivotGrid" ItemSource="{Binding
Source={StaticResource data}}"
AllowResizeColumns="True">
<syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotRows>
<syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
<syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
</syncfusion:PivotGridControl.PivotColumns>
<syncfusion:PivotGridControl.PivotCalculations>
<syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"
Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity"
SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>
```

C#

```
public partial class MainWindow: Window
{
PivotGridControl pivotGrid = new PivotGridControl();
public MainWindow()
{
InitializeComponent();
grid1.Children.Add(pivotGrid);
pivotGrid.ItemSource = ProductSales.GetSalesData();
PivotItem m_PivotItem = new PivotItem()
{
FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
};
PivotItem m_PivotItem1 = new PivotItem()
{
FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
};
PivotItem n_PivotItem = new PivotItem()
{
FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
};
PivotItem n_PivotItem1 = new PivotItem()
{
FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
```

```

};
// Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem);
pivotGrid.PivotRows.Add(m_PivotItem1);
// Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem);
pivotGrid.PivotColumns.Add(n_PivotItem1);
PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo()
{
    CalculationName = "Amount",
    FieldName = "Amount",
    Format = "C",
    SummaryType = SummaryType.DoubleTotalSum
};
PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo()
{
    CalculationName = "Quantity",
    FieldName = "Quantity",
    SummaryType = SummaryType.Count
};
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
pivotGrid1.AllowResizeColumns = true;
}
}

```

VB.NET

```

Partial Public Class MainWindow
Inherits Window
Private pivotGrid As New PivotGridControl()
Public Sub New()
InitializeComponent()
grid1.Children.Add(pivotGrid)
pivotGrid.ItemSource = ProductSales.GetSalesData()
Dim m_PivotItem As New PivotItem() With {.FieldHeader = "Product",
    .FieldMappingName = "Product", .TotalHeader = "Total"}
Dim m_PivotItem1 As New PivotItem() With {.FieldHeader = "Date",
    .FieldMappingName = "Date", .TotalHeader = "Total"}
Dim n_PivotItem As New PivotItem() With {.FieldHeader = "Country",
    .FieldMappingName = "Country", .TotalHeader = "Total"}
Dim n_PivotItem1 As New PivotItem() With {.FieldHeader = "State",
    .FieldMappingName = "State", .TotalHeader = "Total"}
' Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem)
pivotGrid.PivotRows.Add(m_PivotItem1)
' Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem)
pivotGrid.PivotColumns.Add(n_PivotItem1)
Dim m_PivotComputationInfo As New PivotComputationInfo() With
{.CalculationName = "Amount", .FieldName = "Amount", .Format = "C",
    .SummaryType = SummaryType.DoubleTotalSum}
Dim m_PivotComputationInfo1 As New PivotComputationInfo() With
{.CalculationName = "Quantity", .FieldName = "Quantity", .SummaryType =
    SummaryType.Count}
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo)

```

```

pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1)
pivotGrid1.AllowResizeColumns = True
End Sub
End Class

```

Drop Filter Fields Here									
Amount Quantity		Country ▼ ▲ State ▼ ▲							
		▼ Canada							
		Alberta		British Columbia		Brunswick		Manitoba	
Prod... ▼ ▲	D... ▼ ▲	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
▼ Bike	FY 2005	\$5,409,300.00	34	\$4,605,600.00	29	\$6,063,300.00	35	\$3,970,200.00	22
	FY 2006	\$1,695,900.00	11	\$2,647,200.00	15	\$3,162,600.00	17	\$2,914,500.00	17
	FY 2007	\$1,126,500.00	6	\$1,549,200.00	8	\$1,189,500.00	6	\$1,050,300.00	8
	FY 2008	\$372,300.00	3	\$353,100.00	3	\$941,400.00	5	\$723,000.00	4
	FY 2009	\$416,100.00	2	\$202,500.00	2	\$279,600.00	2	\$195,300.00	1
Bike Total		\$9,020,100.00	56	\$9,357,600.00	57	\$11,636,400.00	65	\$8,853,300.00	52
▼ Car	FY 2005	\$1,410,300.00	8	\$1,482,900.00	10	\$304,500.00	2	\$1,149,600.00	7
	FY 2006	\$1,003,500.00	4	\$629,700.00	3	\$245,700.00	1	\$337,800.00	2
	FY 2007			\$87,300.00	1	\$328,800.00	2	\$169,200.00	1
	FY 2008	\$270,000.00	1			\$169,200.00	1		
	FY 2009								
Car Total		\$2,683,800.00	13	\$2,199,900.00	14	\$1,048,200.00	6	\$1,656,600.00	10
Grand Total		\$11,703,900.00	69	\$11,557,500.00	71	\$12,684,600.00	71	\$10,509,900.00	62

Resizing the Rows

You can change the row height by clicking and dragging the resizing cursor at the edge of row header. The resizing cursor appears when you hover the grid line exists between two rows.

To enable the row resizing dynamically, the [AllowResizeRows](#) property should be enabled. The below code snippet shows the resizing of rows in pivot grid is being enabled.

XML

```

<Grid>
  <syncfusion:PivotGridControl Name="pivotGrid" ItemSource="{Binding
Source={StaticResource data}}"
AllowResizeRows="True">
    <syncfusion:PivotGridControl.PivotRows>
      <syncfusion:PivotItem FieldHeader="Product" FieldMappingName="Product"
TotalHeader="Total" />
      <syncfusion:PivotItem FieldHeader="Date" FieldMappingName="Date"
TotalHeader="Total" />
    </syncfusion:PivotGridControl.PivotRows>
    <syncfusion:PivotGridControl.PivotColumns>
      <syncfusion:PivotItem FieldHeader="Country" FieldMappingName="Country"
TotalHeader="Total" />
      <syncfusion:PivotItem FieldHeader="State" FieldMappingName="State"
TotalHeader="Total" />
    </syncfusion:PivotGridControl.PivotColumns>
    <syncfusion:PivotGridControl.PivotCalculations>
      <syncfusion:PivotComputationInfo CalculationName="Total" FieldName="Amount"

```



```

Format="C" SummaryType="DoubleTotalSum" />
<syncfusion:PivotComputationInfo CalculationName="Total"
FieldName="Quantity"
SummaryType="Count" />
</syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
</Grid>

```

C#

```

public partial class MainWindow: Window
{
    PivotGridControl pivotGrid = new PivotGridControl();
    public MainWindow()
    {
        InitializeComponent();
        grid1.Children.Add(pivotGrid);
        pivotGrid.ItemSource = ProductSales.GetSalesData();
        PivotItem m_PivotItem = new PivotItem()
        {
            FieldHeader = "Product", FieldMappingName = "Product", TotalHeader = "Total"
        };
        PivotItem m_PivotItem1 = new PivotItem()
        {
            FieldHeader = "Date", FieldMappingName = "Date", TotalHeader = "Total"
        };
        PivotItem n_PivotItem = new PivotItem()
        {
            FieldHeader = "Country", FieldMappingName = "Country", TotalHeader = "Total"
        };
        PivotItem n_PivotItem1 = new PivotItem()
        {
            FieldHeader = "State", FieldMappingName = "State", TotalHeader = "Total"
        };
        // Adding PivotItem to PivotRows
        pivotGrid.PivotRows.Add(m_PivotItem);
        pivotGrid.PivotRows.Add(m_PivotItem1);
        // Adding PivotItem to PivotColumns
        pivotGrid.PivotColumns.Add(n_PivotItem);
        pivotGrid.PivotColumns.Add(n_PivotItem1);
        PivotComputationInfo m_PivotComputationInfo = new PivotComputationInfo()
        {
            CalculationName = "Amount",
            FieldName = "Amount",
            Format = "C",
            SummaryType = SummaryType.DoubleTotalSum
        };
        PivotComputationInfo m_PivotComputationInfo1 = new PivotComputationInfo()
        {
            CalculationName = "Quantity",
            FieldName = "Quantity",
            SummaryType = SummaryType.Count
        };
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo);
        pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1);
        pivotGrid1.AllowResizeRows = true;
    }
}

```

```
}
}
```

VB.NET

```
Partial Public Class MainWindow
Inherits Window
Private pivotGrid As New PivotGridControl()
Public Sub New()
InitializeComponent()
grid1.Children.Add(pivotGrid)
pivotGrid.ItemSource = ProductSales.GetSalesData()
Dim m_PivotItem As New PivotItem() With {.FieldHeader = "Product",
.FieldMappingName = "Product", .TotalHeader = "Total"}
Dim m_PivotItem1 As New PivotItem() With {.FieldHeader = "Date",
.FieldMappingName = "Date", .TotalHeader = "Total"}
Dim n_PivotItem As New PivotItem() With {.FieldHeader = "Country",
.FieldMappingName = "Country", .TotalHeader = "Total"}
Dim n_PivotItem1 As New PivotItem() With {.FieldHeader = "State",
.FieldMappingName = "State", .TotalHeader = "Total"}
' Adding PivotItem to PivotRows
pivotGrid.PivotRows.Add(m_PivotItem)
pivotGrid.PivotRows.Add(m_PivotItem1)
' Adding PivotItem to PivotColumns
pivotGrid.PivotColumns.Add(n_PivotItem)
pivotGrid.PivotColumns.Add(n_PivotItem1)
Dim m_PivotComputationInfo As New PivotComputationInfo() With
{.CalculationName = "Amount", .FieldName = "Amount", .Format = "C",
.SummaryType = SummaryType.DoubleTotalSum}
Dim m_PivotComputationInfo1 As New PivotComputationInfo() With
{.CalculationName = "Quantity", .FieldName = "Quantity", .SummaryType =
SummaryType.Count}
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo)
pivotGrid.PivotCalculations.Add(m_PivotComputationInfo1)
pivotGrid1.AllowResizeRows = True
End Sub
End Class
```

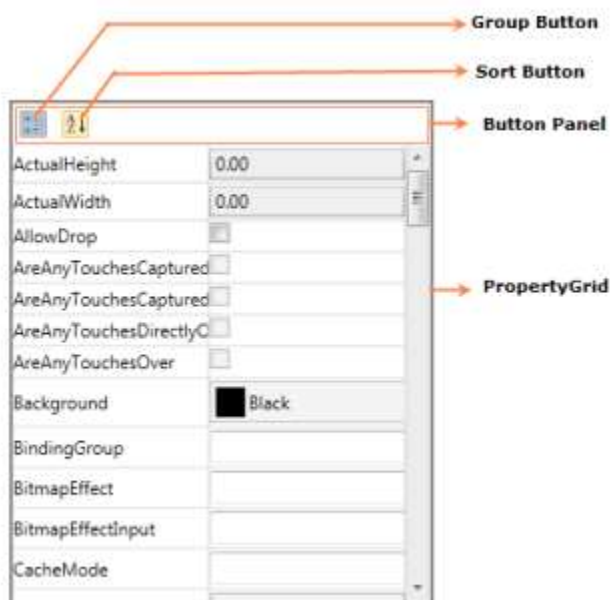
Drop Filter Fields Here									
Amount Quantity		Country ▼ ▲ State ▼ ▲							
		▼ Canada							
		Alberta		British Columbia		Brunswick		Manitoba	
Prod... ▼ ▲	D... ▼ ▲	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
▼ Bike	FY 2005	\$5,409,300.00	34	\$4,605,600.00	29	\$6,063,300.00	35	\$3,970,200.00	22
	FY 2006	\$1,695,900.00	11	\$2,647,200.00	15	\$3,162,600.00	17	\$2,914,500.00	17
	FY 2007	\$1,126,500.00	6	\$1,549,200.00	8	\$1,189,500.00	6	\$1,050,300.00	8
	FY 2008	\$372,300.00	3	\$353,100.00	3	\$941,400.00	5	\$723,000.00	4
	FY 2009	\$416,100.00	2	\$202,500.00	2	\$279,600.00	2	\$195,300.00	1
Bike Total		\$9,020,100.00	56	\$9,357,600.00	57	\$11,636,400.00	65	\$8,853,300.00	52
▼ Car	FY 2005	\$1,410,300.00	8	\$1,482,900.00	10	\$304,500.00	2	\$1,149,600.00	7
	FY 2006	\$1,003,500.00	4	\$629,700.00	3	\$245,700.00	1	\$337,800.00	2
	FY 2007			\$87,300.00	1	\$328,800.00	2	\$169,200.00	1
	FY 2008	\$270,000.00	1			\$169,200.00	1		
	FY 2009								
Car Total		\$2,683,800.00	13	\$2,199,900.00	14	\$1,048,200.00	6	\$1,656,600.00	10
Grand Total		\$11,703,900.00	69	\$11,557,500.00	71	\$12,684,600.00	71	\$10,509,900.00	62

PropertyGrid

WPF PropertyGrid Overview

The [PropertyGrid](#) control provides an interface for browsing and editing an object's properties with Blendability support, custom editors, category editors, sorting and grouping supports. The WPF [PropertyGrid](#) control provides similar features to the Windows Forms [PropertyGrid](#) control.

Control structure



Features

- Binding with any objects — Denotes the object that properties displayed in the `PropertyGrid`.
- Custom Editor — `CustomEditor` support enables you to set custom value editors for particular properties, instead of default editors.
- Category Editor — `CategoryEditor` support allows you to set the related properties (one or more properties) to be categorized.
- Grouping — `PropertyGrid` groups the properties based on the property's `Category` attribute.
- Sorting — `PropertyGrid` supports both ascending and descending order sorting.
- Skin - `PropertyGrid` provides a variety of skin themes.
- Blendability — `PropertyGrid` control can be easily customized in a blend.
- Code sharing — Provides code sharing with `Silverlight`.

Getting Started with WPF PropertyGrid

This section explains how to explore and edit the properties of an object using WPF `PropertyGrid` 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.

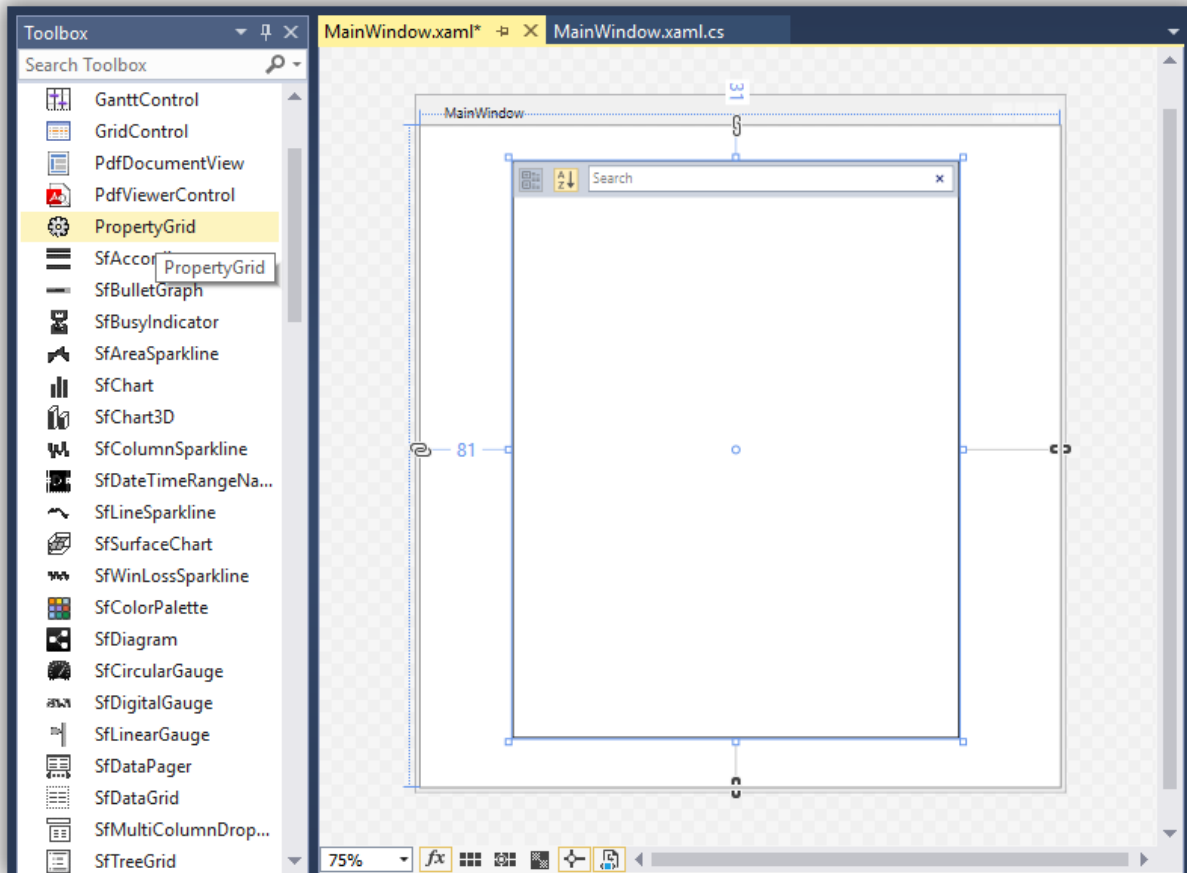
You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

Adding WPF PropertyGrid via designer

You can add the `PropertyGrid` 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.PropertyGrid.Wpf
- Syncfusion.Shared.WPF
- Syncfusion.Tools.Wpf



Adding WPF PropertyGrid via XAML

To add the **PropertyGrid** 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,
 - Syncfusion.PropertyGrid.Wpf
 - Syncfusion.SfInput.WPF
 - Syncfusion.SfShared.WPF
 - Syncfusion.Shared.WPF
 - Syncfusion.Tools.Wpf
3. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> and declare the **PropertyGrid** control in XAML page.

XAML

```
<Window x:Class="PropertyGridSample.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="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="600" Width="500">
```

```
<Grid>
<syncfusion:PropertyGrid Name="propertyGrid1" Height="400" Width="300" >
</syncfusion:PropertyGrid>
</Grid>
</Window>
```

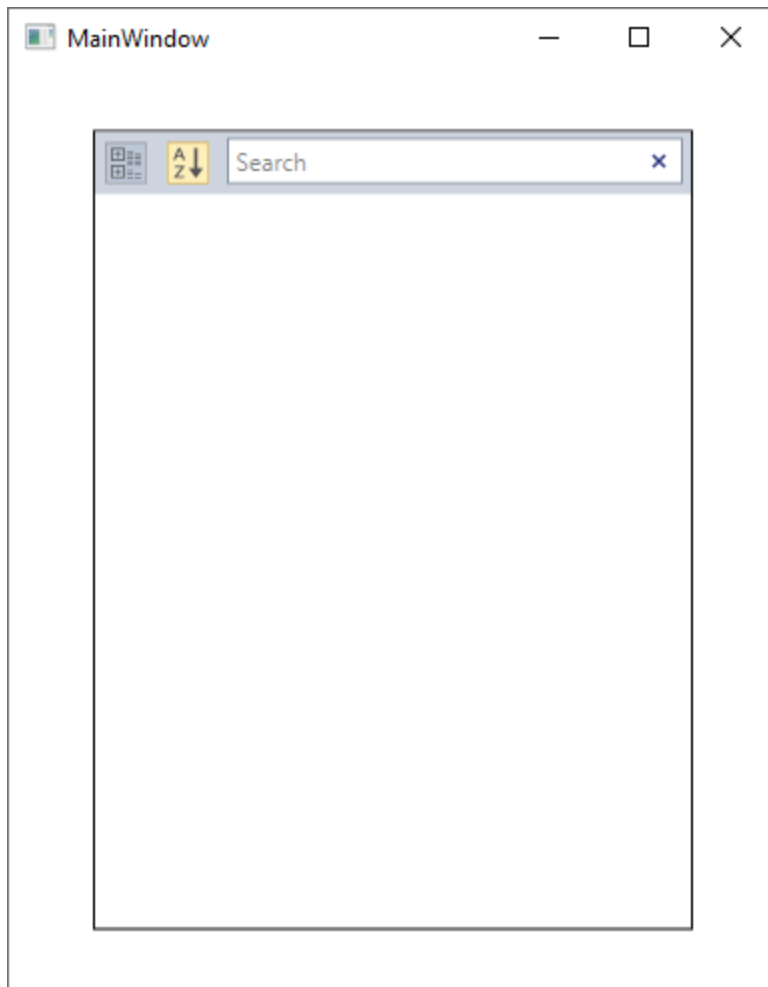
Adding WPF PropertyGrid via C#

To add the **PropertyGrid** 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.PropertyGrid.Wpf
 - Syncfusion.Shared.WPF
 - Syncfusion.Tools.Wpf
3. Include the required namespace and create an instance of **PropertyGrid** and add it to the window.

C#

```
using Syncfusion.Windows.PropertyGrid;
public partial class MainWindow : Window {
public MainWindow() {
InitializeComponent();
// Creating an instance of PropertyGrid control
PropertyGrid propertyGrid1 = new PropertyGrid();
// Setting height and width to PropertyGrid
propertyGrid1.Height = 300;
propertyGrid1.Width = 200;
//Adding PropertyGrid as window content
this.Content = propertyGrid1;
}
}
```



Populating the properties

We can display the properties of any object using the [SelectedObject](#) property. When the `SelectedObject` property is bound with an object, the properties of that object are parsed and displayed in the `PropertyGrid`.

C#

```
// Employee class to be explored in property grid.
public class Employee {
    public string EmployeeName { get; set; }
    public string ID { get; set; }
    public int Age { get; set; }
    public int Experiance { get; set; }
}

//Create ViewModel class with a property to be bounded with
PropertyGrid.SelectedObject
public class ViewModel {
    public object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            EmployeeName = "Johnson",
            Age = 25,
        }
    }
}
```

```
ID = "1234",
Experience = 3
};
}
```

We can populate the properties of the `SelectedObject` using XAML or C#.

XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
```

<div> <div> <div></div> <div>A Z</div> </div> <input type="text" value="Search"/> </div>	
Age	25
EmployeeName	Johnson
Experience	3
ID	1234

Here, the `SelectedEmployee` object is set as `SelectedObject` for the `PropertyGrid`. Thus, the `PropertyGrid` shows all the properties available in the `SelectedEmployee` object.

Custom Editor as value editors

The [PropertyGrid](#) control supports several built-in editors for edit the property values. We can assign own value editor(control) as a value editor for the properties instead of default value editors by using the [Editor](#) attribute or [CustomEditorCollection](#).

For example, if we create an `EmailID` property as a string type, `TextBox` will assigned as a value editor and all the text will be allowed. If we want to accept the input that is only in the mail id format, we can assign [MaskEdit](#) control with mail ID mask as the value editor for the `EmailID` property.

Creating the Custom Editor

To create `CustomEditor`, we need to implement `ITypeEditor` interface.

C#


```
//Custom Editor with email id mask.
public class EmailEditor : ITypeEditor {
    SfMaskedEdit maskededit;
    public void Attach(PropertyViewItem property, PropertyItem info) {
        if (info.CanWrite) {
            var binding = new Binding("Value")
            {
                Mode = BindingMode.TwoWay,
                Source = info,
                ValidatesOnExceptions = true,
                ValidatesOnDataErrors = true
            };
            BindingOperations.SetBinding(maskededit, SfMaskedEdit.ValueProperty,
            binding);
        }
        else {
            maskededit.IsEnabled = false;
            var binding = new Binding("Value")
            {
                Source = info,
                ValidatesOnExceptions = true,
                ValidatesOnDataErrors = true
            };
            BindingOperations.SetBinding(maskededit, SfMaskedEdit.ValueProperty,
            binding);
        }
    }
    public object Create(PropertyInfo propertyInfo) {
        // SfMaskedEdit assigned with EmailId mask
        maskededit = new SfMaskedEdit();
        maskededit.MaskType = MaskType.RegEx;
        maskededit.Mask = "[A-Za-z0-9._%~]+@[A-Za-z0-9]+.[A-Za-z]{2,3}";
        return maskededit;
    }
    public void Detach(PropertyViewItem property) {
    }
}
```

Assigning a Custom Editor for the Property

We will assign the **EmailEditor** as value editor for the **EmailID** property.

C#

```
//CustomEditor for the EmailID property
[Editor("EmailID", typeof(EmailEditor))]
public class Employee : NotificationObject {
    public string EmailID { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public int Experience { get; set; }
}
class ViewModel {
    public object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
    }
}
```

```
Age = 25,
Name = "mark",
Experience = 5,
EmailID = "mark@gt"
};
}
```

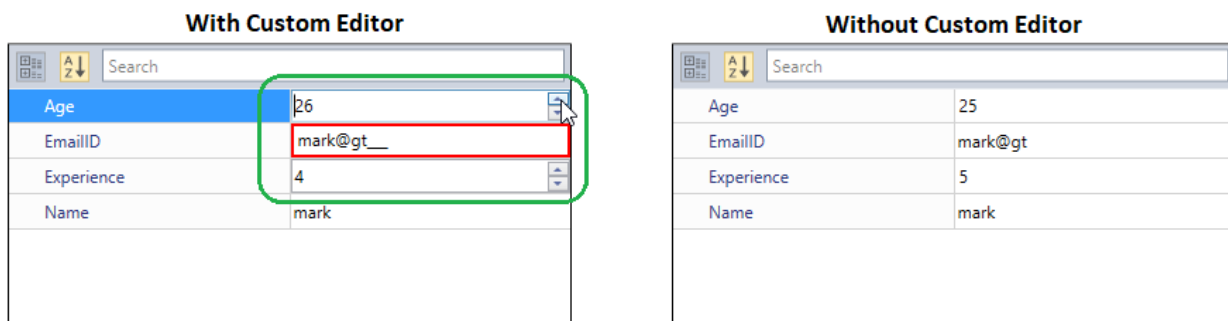
XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
```

Here, The EmailID property is accepts only the inputs which are in the email-id format by the EmailEditor.



Click [here](#) to download the sample that showcases the CustomEditor support.

Selected property item changed notification

The property item selection changed in PropertyGrid can be examined using [SelectedPropertyItemChanged](#) event. The SelectedPropertyItemChanged event contains the old and newly selected property item details in the OldValue and NewValue properties.

XML

```
<syncfusion:PropertyGrid
SelectedPropertyItemChanged="PropertyGrid_SelectedPropertyItemChanged"
SelectedObject="{Binding SelectedEmployee}"
Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
```

```
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.SelectedPropertyItemChanged +=
PropertyGrid_SelectedPropertyItemChanged;
```

You can handle this event as follows,

C#

```
private void PropertyGrid_SelectedPropertyItemChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
var oldPropertyItem = e.OldValue;
var newPropertyItem = e.NewValue;
}
```

Disable animation on loading selected object

You can load the selected's object property items without any animation into the **PropertyGrid** by using the [DisableAnimationOnObjectSelection](#) property value as **true**. The default value of **DisableAnimationOnObjectSelection** property is **false**.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
[Description("Name of the employee")]
public string Name { get; set; }
public string ID { get; set; }
[Description("Birth date of the employee")]
public DateTime DOB { get; set; }
}
public class ViewModel {
public Object SelectedEmployee { get; set; }
public ViewModel() {
SelectedEmployee = new Employee()
{
Name = "John",
ID = "381",
DOB = new DateTime(1995, 12, 24)
};
}
}
```

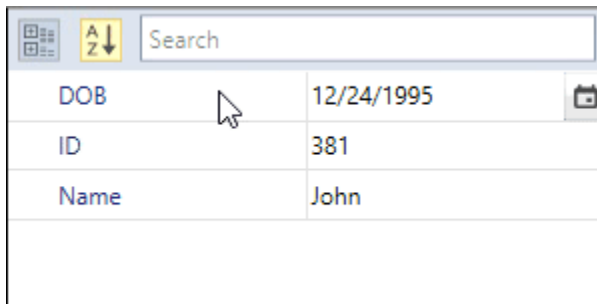
XML

```
<syncfusion:PropertyGrid DisableAnimationOnObjectSelection="True"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
</syncfusion:PropertyGrid.DataContext>
```

```
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DisableAnimationOnObjectSelection = true;
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
```



Note: [View Sample in GitHub](#)

Tooltip support

You can get the value and description about the property item through tooltip when hover the mouse on the respective property item and its value field. If the property item not contains any description, tooltip shows the property display name. You can restrict the tooltip support by setting the [EnableToolTip](#) property as `false`. The default value of [EnableToolTip](#) property is `true`.

C#

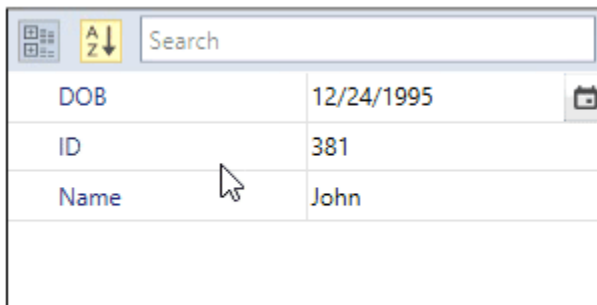
```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
    [Description("Name of the employee")]
    public string Name { get; set; }
    public string ID { get; set; }
    [Description("Birth date of the employee")]
    public DateTime DOB { get; set; }
}
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24)
        };
    }
}
```

XML

```
<syncfusion:PropertyGrid EnableToolTip="True"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.EnableToolTip = true;
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
```



Note: View [Sample](#) in GitHub

Override the property items

The **PropertyGrid** control notifies the users when a property item is created and is being added in the property collection of the control by using the [AutoGeneratingPropertyGridItem](#) event. The

AutoGeneratingPropertyGridItem event contains the following properties and allows us to change their value if required.

- **Cancel** - Allows users to skip adding the current property item in the **PropertyGrid**.
- **Category** - Gets or sets the name of the category for the property item.
- **Description** - Gets or sets a description of the property item.
- **DescriptionTemplate** - Gets or sets the template used to display the description of SelectedItem.
- **DisplayName** - Gets or sets a display name to the property item.
- **ExpandMode** - Gets or sets whether to populate nested properties of PropertyItem or not.
- **Order** - Gets or sets a value to arrange the property item into the property collection when the value of **SortDirection** property is **null**.
- **OriginalSource** - Gets the PropertyItem that is being added to the property collection of PropertyGrid.
- **ReadOnly** - Gets or sets a value indicating whether the property item is ready only or not.

XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
AutoGeneratingPropertyGridItem="propertyGrid1_AutoGeneratingPropertyGridItem"
Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.AutoGeneratingPropertyGridItem +=
propertyGrid1_AutoGeneratingPropertyGridItem;
```

You can handle this event as follows,

C#

```
private void propertyGrid1_AutoGeneratingPropertyGridItem(object sender,
Syncfusion.Windows.PropertyGrid.AutoGeneratingPropertyGridItemEventArgs e)
{
var categoryName = e.Category;
var description = e.Description;
var propertyItemDisplayName = e.DisplayName;
bool isReadOnly = e.ReadOnly;
bool isCancel = e.Cancel;
var itemOrder = e.Order;
var propertyItem = e.OriginalSource;
var descriptionTemplate = e.DescriptionTemplate;
}
```

Override editor

You can also apply a custom editor or change an existing custom editor of the property items in **PropertyGrid** in the [AutoGeneratingPropertyGridItem](#) event as shown below.

C#

```
private void propertyGrid1_AutoGeneratingPropertyGridItem(object sender,
Syncfusion.Windows.PropertyGrid.AutoGeneratingPropertyGridItemEventArgs e)
{
if(e.DisplayName == "Age")
{
PropertyItem propertyItem = e.OriginalSource as PropertyItem;
if(propertyItem.PropertyType == typeof(long))
{
propertyItem.Editor = new IntegerTextBoxEditor();
}
}
}
```

The screenshot shows a WPF PropertyGrid control. At the top, there is a toolbar with a grid icon, a sort icon (A-Z), and a search box containing the text "Search". Below the toolbar is a table with four rows of employee data:

Age	25
EmployeeName	Johnson
Experiance	3
ID	1234

Below the table is an empty row.

Property item value changed notification

The property item value changed in **PropertyGrid** can be examined using [ValueChanged](#) event. The **ValueChanged** event contains the old and newly changed property values by the **OldValue** and **NewValue** properties and **Property** contains the property item whose values is changed.

XML

```
<syncfusion:PropertyGrid ValueChanged="PropertyGrid_ValueChanged"
SelectedObject="{Binding SelectedEmployee}"
Name="propertyGrid1" >
  <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
  </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.ValueChanged += PropertyGrid_ValueChanged;
```

You can handle this event as follows,

C#

```
private void PropertyGrid_ValueChanged(object sender, ValueChangedEventArgs
args) {
  var valueChangedPropertyItem = args.Property;
  var newValue = args.NewValue;
  var oldValue = args.OldValue;
}
```

Click [here](#) to download the sample that showcases the **PropertyGrid** overall features.

Theme

PropertyGrid supports various built-in themes. Refer to the below links to apply themes for the PropertyGrid,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

AZ Search	
ID	0005A
First Name	Carl
Last Name	Johnson
Date of Birth	10/16/1987
Gender	Male ▼
Age	30
Favorite Color	Gray ▼
Permanent Employee	<input checked="" type="checkbox"/>
▶ Bank	Centura Banks
Email ID	carljohanson@gt.□□
Mobile Number	91,983,467,382
Country	USA ▼
Age Age of the actual person.	

Nested Properties in WPF PropertyGrid

The [PropertyGrid](#) control has support to expand instance properties of a class.

Explore the nested properties

You can choose whether the nested properties of the [SelectedObject](#) can be expanded or not by using the [PropertyExpandMode](#) property. By default, [PropertyExpandMode](#) value is [FlatMode](#), thus the nested properties are not shown. If you want to display the nested properties, you can set the [PropertyExpandMode](#) property as [NestedMode](#).

C#

```
// A Class that represents the nested properties
public class Address {
    public string State { get; set; }
    public string StreetName { get; set; }
    public string DoorNo { get; set; }
}
```



```

public override string ToString() {
    return DoorNo + ", " + StreetName + ", " + State;
}
}

public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public int Age { get; set; }
    // Property contains the nested properties
    public Address Address { get; set; }
}

public class ViewModel {
    public object SelectedEmployee { get; set; }
    public PropertyExpandModes PropertyExpandMode { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee() {
            Age = 23,
            ID = "1207",
            Name = "Mark",
            Address = new Address()
            {
                State = "New York",
                DoorNo = "10",
                StreetName = "Martin street"
            }
        };
        PropertyExpandMode = PropertyExpandModes.FlatMode;
    }
}

```

XML

```

<syncfusion:PropertyGrid PropertyExpandMode="NestedMode"
    SelectedObject="{Binding SelectedEmployee}"
    x:Name="propertyGrid1">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

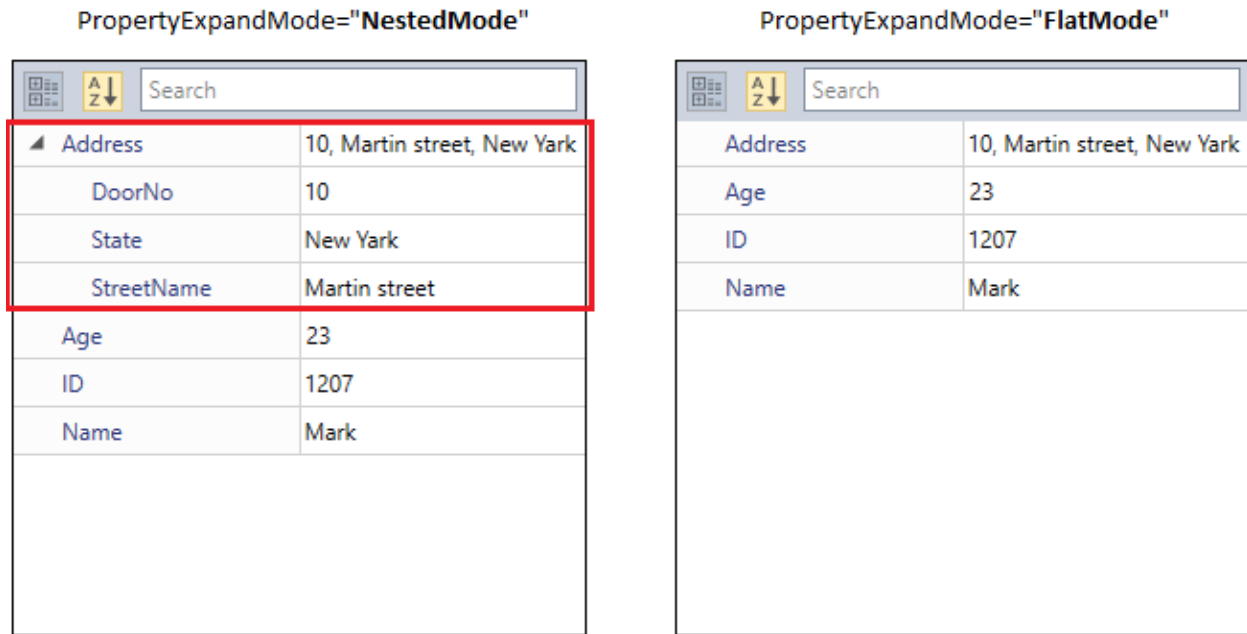
C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
    Binding("SelectedEmployee"));
propertyGrid1.PropertyExpandMode = PropertyExpandModes.NestedMode;

```

Here, `Address` is a class type property in the `Employee` class. It includes the `City`, `StreetName`, and `DoorNo` properties that are shown by setting `PropertyExpandMode` property as `NestedMode`.



Click [here](#) to download the sample that showcases the nested property expanding support.

Enable or disable nested properties, for a specific property

You can enable or disable nested properties, for one or more specific property item by using the attribute and event

Enable or disable nested properties using attribute

You can explore or hide the nested properties for any specific property item by setting the `PropertyGridAttribute.NestedPropertyDisplayMode` property value as `Show` or `None` for that specific property item. It will not be affected by the [PropertyExpandMode](#) property values.

C#

```
public class Address {
    public string State { get; set; }
    public string StreetName { get; set; }
    public string DoorNo { get; set; }
    public override string ToString() {
        return DoorNo + ", " + StreetName + ", " + State;
    }
}

public class Bank {
    public string BankName { get; set; }
    public string AccountNo { get; set; }
    public override string ToString() {
        return BankName + ", " + AccountNo;
    }
}

//Allow to explore nested property for the multiple property item
[PropertyGridAttribute(NestedPropertyDisplayMode =
    NestedPropertyDisplayMode.Show, PropertyName = "Bank, DOB")]
public class Employee {
    public string Name { get; set; }
    public DateTime DOB { get; set; }
    public DateTime JoiningDate { get; set; }
}
```

```

public Bank Bank { get; set; }
//Allow to explore nested property for the single property item
[PropertyGridAttribute(NestedPropertyDisplayMode =
NestedPropertyDisplayMode.Show)]
public Address Address { get; set; }
public Employee() {
    Name = "Mark";
    DOB = new DateTime(1997, 2, 25);
    JoiningDate= new DateTime(2020, 10, 5);
    Address = new Address()
    {
        State = "New Yark",
        DoorNo = "10",
        StreetName = "Martin street"
    };
    Bank = new Bank()
    {
        AccountNo = "84946448",
        BankName = "ABCBank"
    };
}
}

```

XML

```

<syncfusion:PropertyGrid PropertyExpandMode="FlatMode"
SelectedObject="{Binding Employee}"
SortDirection="{x:Null}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:Employee></local:Employee>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

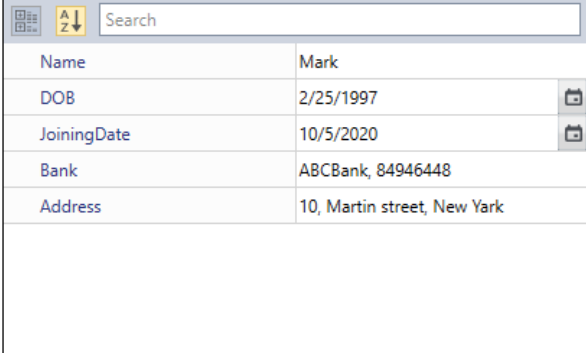
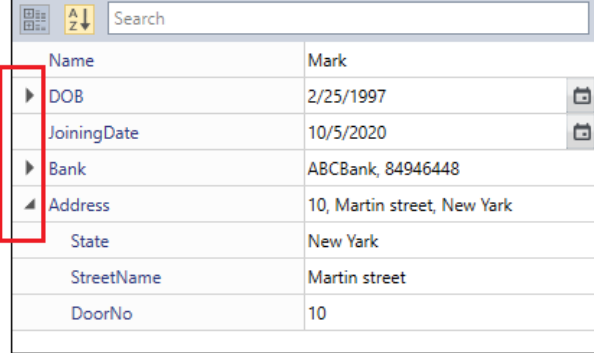
```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("Employee"));
propertyGrid1.PropertyExpandMode = PropertyExpandModes.FlatMode;
propertyGrid1.SortDirection = null;

```

Before using PropertyGridAttribute.NestedPropertyDisplayMode = Show		After using PropertyGridAttribute.NestedPropertyDisplayMode = Show	
			

Here, the `PropertyExpandMode` property value is `FlatMode`. But, Specific properties like `Bank`, `DOB` and `Address` property's nested properties are explored in the `PropertyGrid`.

Note: View [Sample](#) in GitHub

Enable or disable nested properties using event

You can explore or hide the nested properties for any specific property item without using the attributes at runtime by handling the [AutoGeneratingPropertyGridItem](#) event with

[AutoGeneratingPropertyGridItemEventArgs.ExpandMode](#) property value as `NestedMode` or `FlatMode` for that specific property item. It will not affected by the [PropertyExpandMode](#) property values.

C#

```
public class Address {
    public string State { get; set; }
    public string StreetName { get; set; }
    public string DoorNo { get; set; }
    public override string ToString() {
        return DoorNo + ", " + StreetName + ", " + State;
    }
}

public class Bank {
    public string BankName { get; set; }
    public string AccountNo { get; set; }
    public override string ToString() {
        return BankName + ", " + AccountNo;
    }
}

public class Employee {
    public string Name { get; set; }
    public DateTime DOB { get; set; }
    public DateTime JoiningDate { get; set; }
    public Bank Bank { get; set; }
    public Address Address { get; set; }
    public Employee() {
        Name = "Mark";
        DOB = new DateTime(1997, 2, 25);
        JoiningDate = new DateTime(2020, 10, 5);
        Address = new Address()
        {
            State = "New York",
```

```

DoorNo = "10",
StreetName = "Martin street"
};
Bank = new Bank()
{
AccountNo = "84946448",
BankName = "ABCBank"
};
}
}

```

XML

```

<syncfusion:PropertyGrid
AutoGeneratingPropertyGridItem="PropertyGrid_AutoGeneratingPropertyGridItem"
PropertyExpandMode="NestedMode"
SelectedObject="{Binding Employee}"
SortDirection="{x:Null}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:Employee></local:Employee>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.AutoGeneratingPropertyGridItem +=
PropertyGrid_AutoGeneratingPropertyGridItem
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("Employee"));
propertyGrid1.PropertyExpandMode = PropertyExpandModes.NestedMode;
propertyGrid1.SortDirection = null;

```

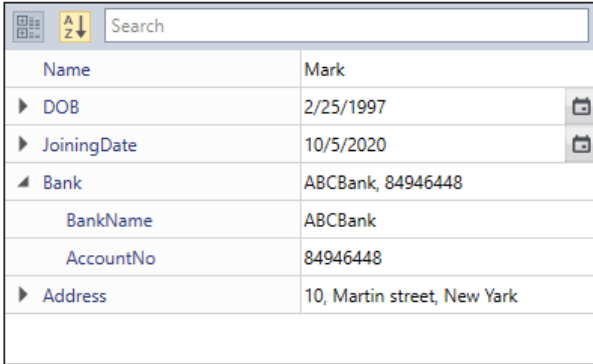
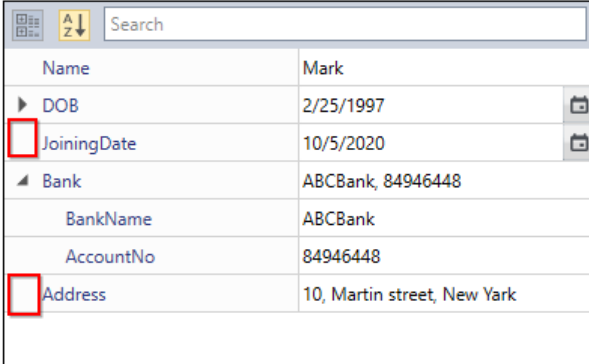
You can handle the event as follows

C#

```

private void PropertyGrid_AutoGeneratingPropertyGridItem(object sender,
AutoGeneratingPropertyGridItemEventArgs e) {
if(e.DisplayName=="JoiningDate" || e.DisplayName== "Address") {
e.ExpandMode = PropertyExpandModes.FlatMode;
}
}

```

Before handling AutoGeneratingPropertyGridItem event		After handling AutoGeneratingPropertyGridItem event	
			
Name	Mark	Name	Mark
▶ DOB	2/25/1997	▶ DOB	2/25/1997
▶ JoiningDate	10/5/2020	JoiningDate	10/5/2020
▲ Bank	ABCBank, 84946448	▲ Bank	ABCBank, 84946448
BankName	ABCBank	BankName	ABCBank
AccountNo	84946448	AccountNo	84946448
▶ Address	10, Martin street, New York	Address	10, Martin street, New York

Here, the `PropertyExpandMode` property value is `NestedMode`. But, specific properties like `JoiningDate` and `Address` property's nested properties are hidden in the `PropertyGrid`.

Note: View [Sample](#) in GitHub

ReadOnly Properties in WPF PropertyGrid

We can display the readonly properties with its value editor in the non editable state by default. If we want to change any property as readonly, it can be achieved by attributes and event in the [PropertyGrid](#).

ReadOnly properties using attributes

We can change the properties as read only by using the [ReadOnly](#) or [Editable](#) attributes. When the property is marked `ReadOnly` as `true` or `Editable` as `false`, the [PropertyGrid](#) will not allow the user to edit the property values.

C#

```
public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    [ReadOnly(true)]
    public DateTime DOB { get; set; }
    [Editable(false)]
    public int Experience { get; set; }
}

public class ViewModel {
    public object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24),
            Experience = 5;
        };
    }
}
```

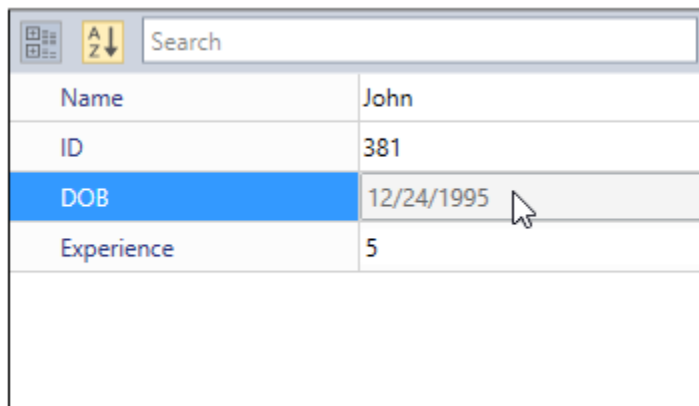
XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
SortDirection="{x:Null}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.SortDirection = null;
```

Here, the **DOB** and **Experience** are readonly properties by the attributes.



Name	John
ID	381
DOB	12/24/1995
Experience	5

Note: If you use both the **ReadOnly** attribute and **Editable** attribute, the **ReadOnly** attribute will have higher priority.

Click [here](#) to download the sample that showcases the **ReadOnly** support using attribute.

Change properties as readonly at runtime

We can change the properties as read only without using the attributes at runtime by handling the [AutoGeneratingPropertyGridItem](#) event with [AutoGeneratingPropertyGridItemEventArgs.ReadOnly](#) property.

When **AutoGeneratingPropertyGridItemEventArgs.ReadOnly** property value sets as **true**, the property will be classified as read only, then the PropertyGrid will not allow the user to edit the property values. The Default value of **AutoGeneratingPropertyGridItemEventArgs.ReadOnly** property is **false**.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
public string Name { get; set; }
public string ID { get; set; }
```

```

public DateTime DOB { get; set; }
public int Experience { get; set; }
}
public class ViewModel {
public object SelectedEmployee { get; set; }
public ViewModel() {
SelectedEmployee = new Employee()
{
Name = "John",
ID = "381",
DOB = new DateTime(1995, 12, 24),
Experience = 5;
};
}
}
}

```

XML

```

<syncfusion:PropertyGrid
AutoGeneratingPropertyGridItem="PropertyGrid1_AutoGeneratingPropertyGridItem"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.AutoGeneratingPropertyGridItem +=
PropertyGrid1_AutoGeneratingPropertyGridItem;

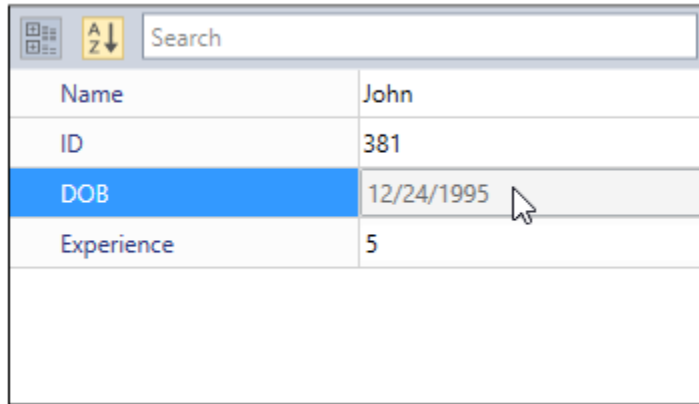
```

C#

```

private void PropertyGrid1_AutoGeneratingPropertyGridItem(object sender,
AutoGeneratingPropertyGridItemEventArgs e) {
// Change 'DOB' property as readonly
if (e.DisplayName == "DOB") {
e.ReadOnly = true;
}
}
}

```

Name	John
ID	381
DOB	12/24/1995
Experience	5

Here, the **DOB** property non-editable.

Click [here](#) to download the sample that showcases the **ReadOnly** support using **AutoGeneratingPropertyGridItem** event.

Attached Properties in WPF PropertyGrid

The **PropertyGrid** control has support to display the attached properties of **SelectedObject**.

Show or hide attached properties of **SelectedObject**

We can show or hide the attached properties of **SelectedObject** by handling the **AutoGeneratingPropertyGridItem** event with **AutoGeneratingPropertyGridItemEventArgs.Cancel** property. Below example shows, how to hide the attached properties using **AutoGeneratingPropertyGridItem** event.

XML

```
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="0.6*" />
<ColumnDefinition Width="0.4*" />
</Grid.ColumnDefinitions>
<syncfusion:PropertyGrid SelectedObject="{Binding ElementName=button}"
AutoGeneratingPropertyGridItem="PropertyGrid_AutoGeneratingPropertyGridItem"
/>
<syncfusion:ButtonAdv Name="button"
Label="SelectedObject"
Margin="10"
Height="25"
Width="200"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Grid.Column="1"
syncfusion:SkinStorageVisualStyle="Default" />
</Grid>
```

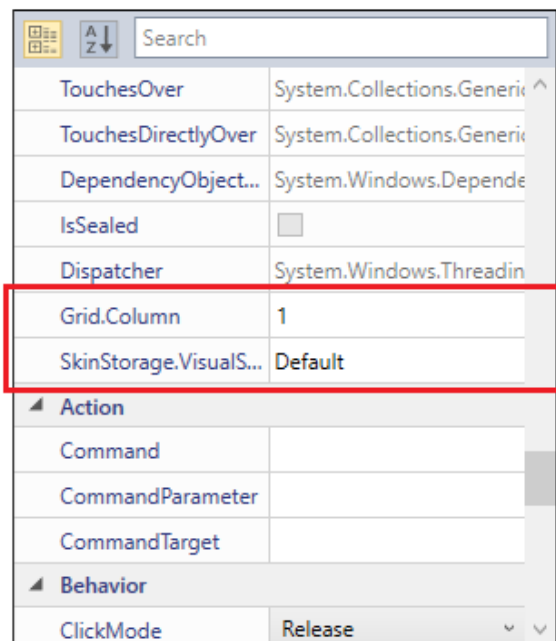
C#

```
private void PropertyGrid_AutoGeneratingPropertyGridItem(object sender,
AutoGeneratingPropertyGridItemEventArgs e)
{
if (e.DisplayName == "Grid.Column" || e.DisplayName ==
"SkinStorage.VisualStyle")
```

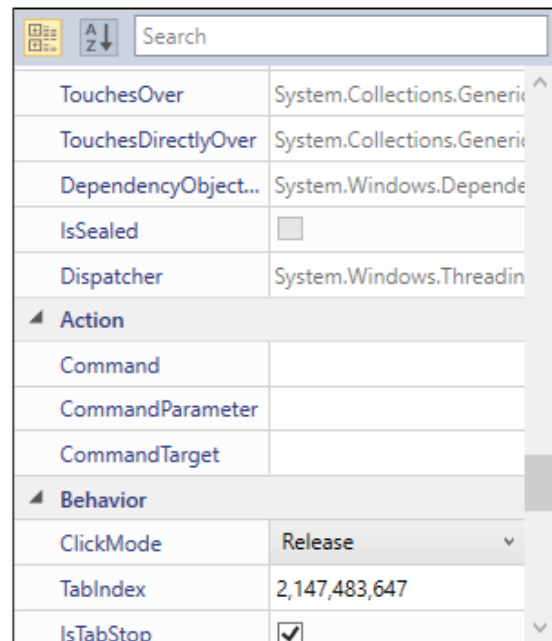
```
{
    e.Cancel = true;
}
```

Here, `Grid.Column` and `SkinStorageVisualStyle` are the attached properties of `SelectedObject`.

With Attached Properties



Without Attached Properties



Note: Download demo application from [GitHub](#).

Update value on lost focus in WPF PropertyGrid

We can update the value of property item either immediately when the editor property changes or during the editor's lost focus using `UpdateSourceMode` property of `PropertyGrid`. Values of `UpdateSourceMode` are `Immediately`, `ReturnOrLostFocus`. The default value is `Immediately`.

Binding modes

If the property `UpdateSourceMode` is `Immediately`, value of property item will be updated immediately when the editor property changes. Also, if the property `UpdateSourceMode` is `ReturnOrLostFocus`, value of property item will be updated only when editor lost its focus or when enter key is pressed.

XML

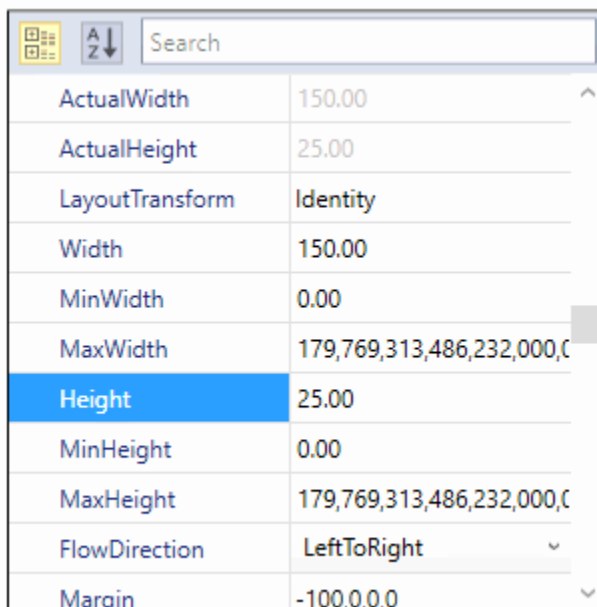
```
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="0.6*" />
<ColumnDefinition Width="0.4*" />
</Grid.ColumnDefinitions>
<syncfusion:PropertyGrid Margin="10"
x:Name="propertyGrid"
UpdateSourceMode="ReturnOrLostFocus"
SelectedObject="{Binding ElementName=button}" />
```

```
<syncfusion:ButtonAdv Name="button"
Margin="10"
Height="25"
Width="200"
Label="SelectedObject"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Grid.Column="1"
syncfusion:SkinStorage.VisualStudio="Default" />
</Grid>
```

C#

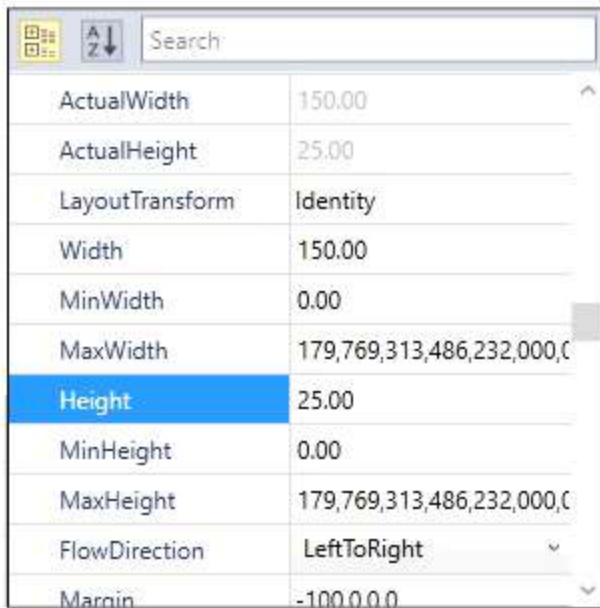
```
propertyGrid.UpdateSourceMode = UpdateSourceMode.ReturnOrLostFocus;
```

The SelectedObject value changes immediately when the target element property changes, as shown below:



ActualWidth	150.00
ActualHeight	25.00
LayoutTransform	Identity
Width	150.00
MinWidth	0.00
MaxWidth	179,769,313,486,232,000,0
Height	25.00
MinHeight	0.00
MaxHeight	179,769,313,486,232,000,0
FlowDirection	LeftToRight
Margin	-100.0.0.0

The SelectedObject value changes only when the target element lost its focus or when enter key is pressed, as shown below.

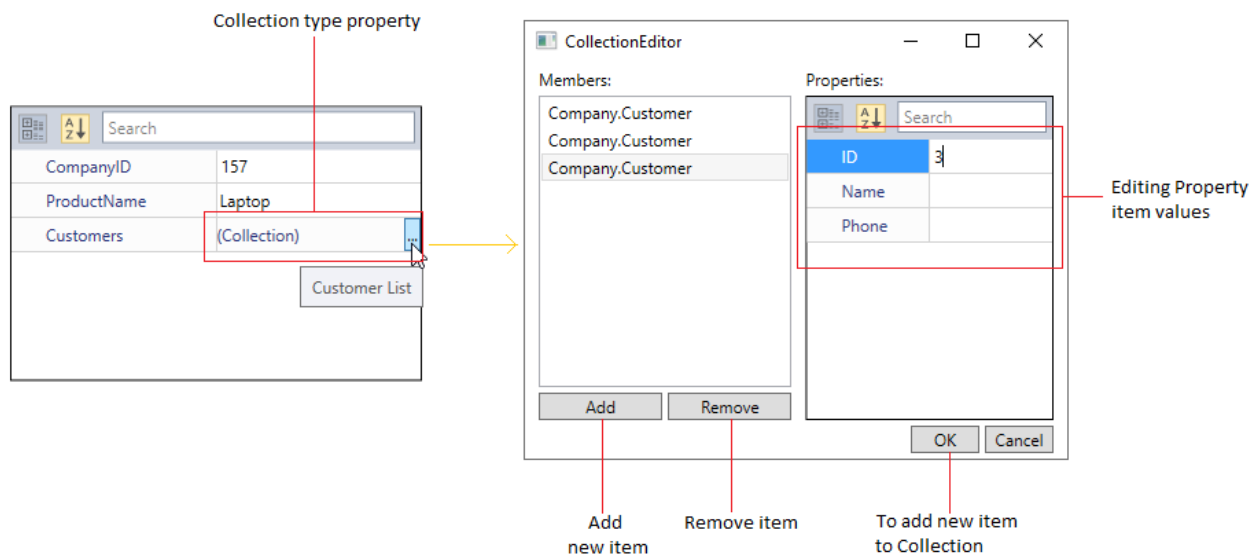


Collection Editor in WPF PropertyGrid

You can edit (add, remove) a collection type of property such as List, ObservableCollection using the Collection Editor in the [PropertyGrid](#).

How to add or remove items in collection using collection editor

You can add or remove the new items into the collection type properties by clicking the Add or Remove button. If you want to add the items into SelectedObject collection, you must click the Ok button to update the added items into the collection, otherwise it is not updated.



Note: Collection type property must be derived from the `ICollection` type to edit the collection properties in the `SelectedObject`. The `ICollection` collection property must contain the non-parameter constructor to edit the items in the collection, otherwise only the items in the collection property can be removed.

Note: You will not be able to edit a collection of primitive type, such as `List<int>` and `List<string>`.

Edit a selected object, which is of type collection

If you directly assign the collection instance as [SelectedObject](#), it generates value editor for the each collection property items.

C#

```
public class Persons {
    public string Name { get; set; }
    public string Address { get; set; }
    public override string ToString() {
        return Name;
    }
}

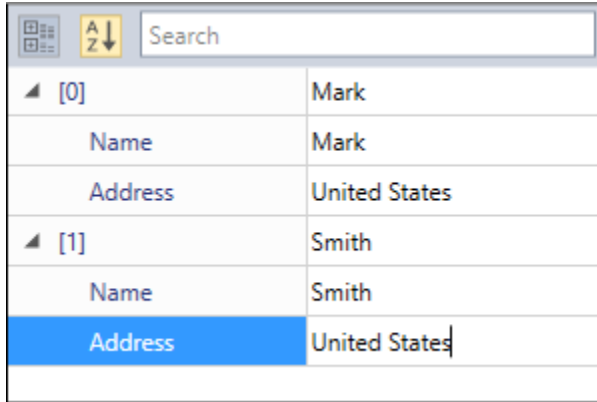
public class ViewModel {
    private ObservableCollection<Person> persons = new
        ObservableCollection<Person>();
    public ObservableCollection<Person> Persons {
        get {
            return persons;
        }
        set {
            persons = value;
        }
    }
    public ViewModel() {
        persons.Add(new Person() { Name = "Mark", Address = "United States" });
        persons.Add(new Person() { Name = "Smith", Address = "United States" });
    }
}
```

XML

```
<syncfusion:PropertyGrid PropertyExpandMode="NestedMode"
    SelectedObject="{Binding Persons}"
    x:Name="propertyGrid">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.DataContext = new ViewModel();
propertyGrid.SetBinding(PropertyGrid.SelectedObjectProperty, new
    Binding("Persons"));
propertyGrid1.PropertyExpandMode = PropertyExpandModes.NestedMode;
```



Note: View [Sample](#) on GitHub

Edit a selected object, which has a property of type collection.

You can add, remove or edit the collection type property item by using the **Collection Editor**.

C#

```
public class Customer {
    public int ID { get; set; }
    public string Name { get; set; }
    public string Phone { get; set; }
}

public class Employee {
    public int ID { get; set; }
    public string Name { get; set; }
    public string Phone { get; set; }
}

public class CustomerCollection : ObservableCollection<Customer> {
    public CustomerCollection() { }
    public override string ToString() {
        return "Customer List";
    }
}

public class EmployeeList : List<Employee> {
    public EmployeeList() { }
    public override string ToString() {
        return "Employee List";
    }
}

public class Product {
    public int CompanyID { get; set; }
    public string ProductName { get; set; }
    public CustomerCollection Customers { get; set; }
    public EmployeeList Employees { get; set; }
}

public class ViewModel {
    public Product DemoProduct { get; set; }
    public ViewModel() {
        DemoProduct = new Product()
        {
            CompanyID = 157,
            ProductName = "Laptop",
            Customers = new CustomerCollection

```

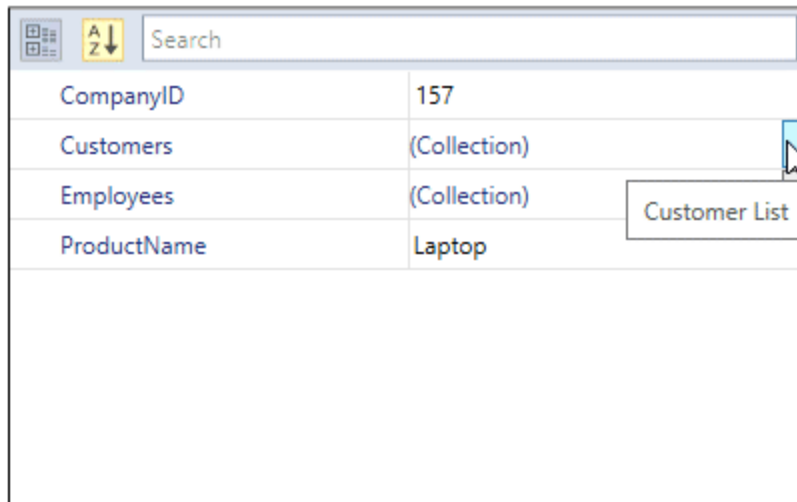
```
{
    new Customer() { ID = 0, Name = "John", Phone = "2065349857" },
    new Customer() { ID = 1, Name = "Peter", Phone = "2065981189" }
},
Employees = new EmployeeList
{
    new Employee() { ID = 0, Name = "Mark", Phone = "2065489864" },
    new Employee() { ID = 1, Name = "David", Phone = "2063481135" }
}
};
}
```

XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding DemoProduct}"
x:Name="propertyGrid">
    <syncfusion:PropertyGrid.DataContext>
        <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.DataContext = new ViewModel();
propertyGrid.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("DemoProduct"));
```



Note: View [Sample](#) on GitHub

Collection Editor for nested collection property

You can update the nested collection property item by using the **Collection Editor**.



C#

```
public class AddressDetails {
    public int DoorNo { get; set; }
    public string City { get; set; }
}
public class Supplier {
    public int ID { get; set; }
```



```

public string Name { get; set; }
public string Phone { get; set; }
public ObservableCollection<AddressDetails> Address { get; set; }
}
public class SupplierList : List<Supplier> {
public SupplierList() : base() { }
public override string ToString()
{
return "Supplier List";
}
}
public class Product {
public int ID { get; set; }
public string ProductName { get; set; }
public SupplierList Suppliers { get; set; }
}
public ViewModel() {
DemoProduct = new Product() {
ID = 0,
ProductName = "Office Table",
Suppliers = new SupplierList
{
new Supplier()
{
ID = 0,
Name = "Jack Plank",
Phone = "2065559857",
Address = new ObservableCollection<AddressDetails>()
{
new AddressDetails
{
City="Mexico",
DoorNo=456
}
}
}
};
}
}

```

XML

```

<syncfusion:PropertyGrid SelectedObject="{Binding DemoProduct}"
x:Name="propertyGrid">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

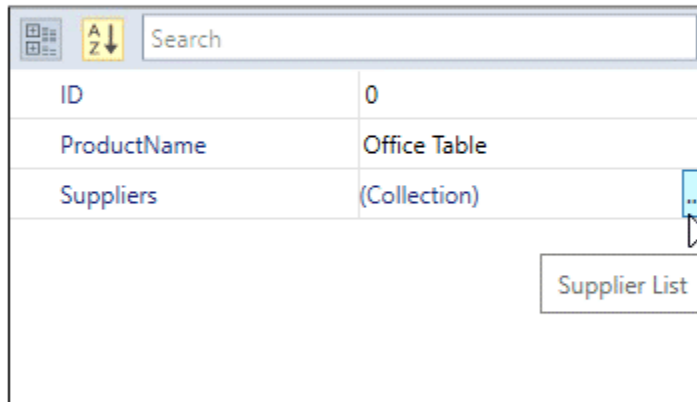
```

C#

```

PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.DataContext = new ViewModel();
propertyGrid.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("DemoProduct"));

```



Note: View [Sample](#) on GitHub

Readonly mode for collection type properties

If you want to restrict the user to add or remove the items in the collection type properties, handle the [CollectionEditorOpening](#) event and set the [CollectionEditorOpeningEventArgs.IsReadOnly](#) property value as `true`. The default value of [CollectionEditorOpeningEventArgs.IsReadOnly](#) property is `false`.

Note: You will not be able to add or remove the items into the collection type properties. But, you can edit and save the existing items that are available in the collection type properties.

C#

```
public class Customer {
    public int ID { get; set; }
    public string Name { get; set; }
    public string Phone { get; set; }
}

public class Employee {
    public int ID { get; set; }
    public string Name { get; set; }
    public string Phone { get; set; }
}

public class CustomerCollection : ObservableCollection<Customer> {
    public CustomerCollection() { }
}
```

```

public override string ToString() {
    return "Customer List";
}
}

public class EmployeeList : List<Employee> {
    public EmployeeList() { }
    public override string ToString() {
        return "Employee List";
    }
}

public class Product {
    public int CompanyID { get; set; }
    public string ProductName { get; set; }
    public CustomerCollection Customers { get; set; }
    public EmployeeList Employees { get; set; }
}

public class ViewModel {
    public Product DemoProduct { get; set; }
    public ViewModel() {
        DemoProduct = new Product()
        {
            CompanyID = 157,
            ProductName = "Laptop",
            Customers = new CustomerCollection
            {
                new Customer() { ID = 0, Name = "John", Phone = "2065349857" },
                new Customer() { ID = 1, Name = "Peter", Phone = "2065981189" }
            },
            Employees = new EmployeeList
            {
                new Employee() { ID = 0, Name = "Mark", Phone = "2065489864" },
                new Employee() { ID = 1, Name = "David", Phone = "2063481135" }
            }
        };
    }
}

```

XML

```

<syncfusion:PropertyGrid
    CollectionEditorOpening="propertyGrid_CollectionEditorOpening"
    SelectedObject="{Binding DemoProduct}"
    x:Name="propertyGrid">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

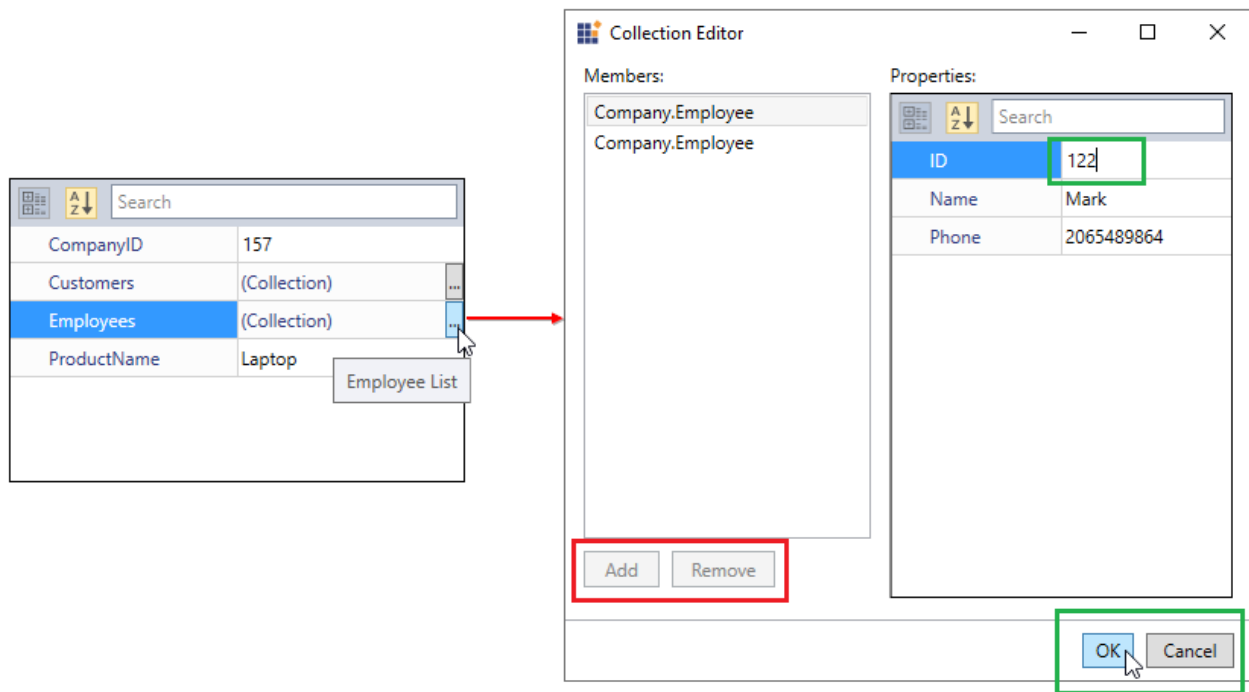
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.DataContext = new ViewModel();
propertyGrid.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("DemoProduct"));
propertyGrid.CollectionEditorOpening +=
propertyGrid_CollectionEditorOpening;

```

You can handle the `CollectionEditorOpening` event as follows,

C#

```
private void propertyGrid_CollectionEditorOpening(object sender,
CollectionEditorOpeningEventArgs e)
{
    //Enabling the readonly collection editor
    e.IsReadOnly = true;
}
```



Here, `Add` and `Remove` buttons are disabled. `Ok` and `Cancel` button are enabled to edit the existing item values.

Note: [View Sample in GitHub](#)

Readonly mode for specific property of collection type

If you want to restrict the user to add or remove the items in the specific property of collection type, create that collection property as the type of `ReadOnlyCollection`.

Note: You will not be able to add or remove the items into the readonly collection type properties. But, you can edit and save the existing items that are available in the readonly collection type properties.

C#

```
public class Customer
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Phone { get; set; }
}
```

```

public class Employee
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Phone { get; set; }
}

public class Product
{
    public int CompanyID { get; set; }
    public string ProductName { get; set; }
    public List<Employee> Employees { get; set; }
    internal List<Customer> customers;
    public ReadOnlyCollection<Customer> Customers
    {
        get
        {
            return customers.AsReadOnly();
        }
    }
}

public class ViewModel
{
    public Product DemoProduct { get; set; }
    public ViewModel()
    {
        DemoProduct = new Product()
        {
            CompanyID = 157,
            ProductName = "Laptop",
            customers = new List<Customer>
            {
                new Customer() { ID = 0, Name = "John", Phone = "2065349857" },
                new Customer() { ID = 1, Name = "Peter", Phone = "2065981189" }
            },
            Employees = new List<Employee>()
            {
                new Employee() { ID = 0, Name = "Mark", Phone = "2065489864" },
                new Employee() { ID = 1, Name = "David", Phone = "2063481135" }
            }
        };
    }
}

```

XML

```

<syncfusion:PropertyGrid SelectedObject="{Binding DemoProduct}"
x:Name="propertyGrid">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

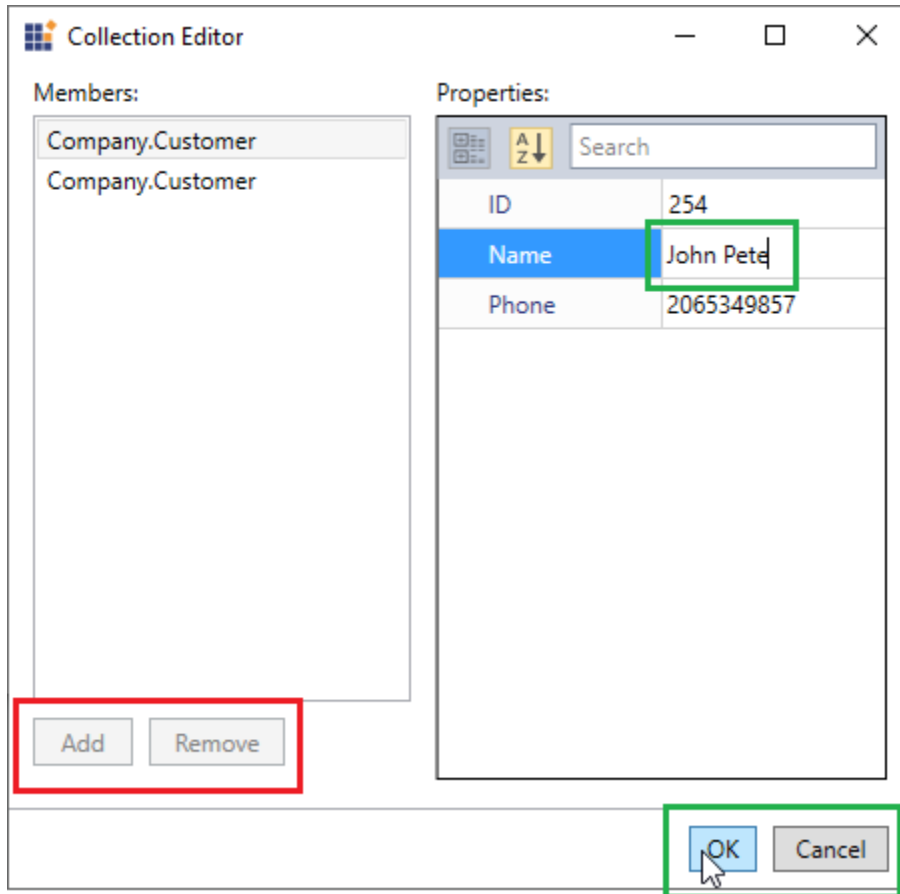
C#

```

PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.DataContext = new ViewModel();

```

```
propertyGrid.SetBinding(PropertyGrid.SelectedObjectProperty, new  
Binding("DemoProduct"));
```



Here, `Customers` property is a readonly collection type property. So, you will not be able to add or remove the items into the `Customers` property.

Note: [View Sample in GitHub](#)

Restrict the collection editor from opening

You can restrict the opening of collection editor window which used to edit the collection type properties in `PropertyGrid` by handling the [CollectionEditorOpening](#) event and set the [CollectionEditorOpeningEventArgs.Cancel](#) property value as `true`. The default value of `CollectionEditorOpeningEventArgs.Cancel` property is `false`.

C#

```
public class Customer {  
    public int ID { get; set; }  
    public string Name { get; set; }  
    public string Phone { get; set; }  
}  
public class Employee {  
    public int ID { get; set; }  
    public string Name { get; set; }  
    public string Phone { get; set; }  
}
```

```

}
public class CustomerCollection : ObservableCollection<Customer> {
public CustomerCollection() { }
public override string ToString() {
return "Customer List";
}
}
public class EmployeeList : List<Employee> {
public EmployeeList() { }
public override string ToString() {
return "Employee List";
}
}
public class Product {
public int CompanyID { get; set; }
public string ProductName { get; set; }
public CustomerCollection Customers { get; set; }
public EmployeeList Employees { get; set; }
}
public class ViewModel {
public Product DemoProduct { get; set; }
public ViewModel() {
DemoProduct = new Product()
{
CompanyID = 157,
ProductName = "Laptop",
Customers = new CustomerCollection
{
new Customer() { ID = 0, Name = "John", Phone = "2065349857" },
new Customer() { ID = 1, Name = "Peter", Phone = "2065981189" }
},
Employees = new EmployeeList
{
new Employee() { ID = 0, Name = "Mark", Phone = "2065489864" },
new Employee() { ID = 1, Name = "David", Phone = "2063481135" }
}
};
}
}
}

```

XML

```

<syncfusion:PropertyGrid
CollectionEditorOpening="propertyGrid_CollectionEditorOpening"
SelectedObject="{Binding DemoProduct}"
x:Name="propertyGrid">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.DataContext = new ViewModel();

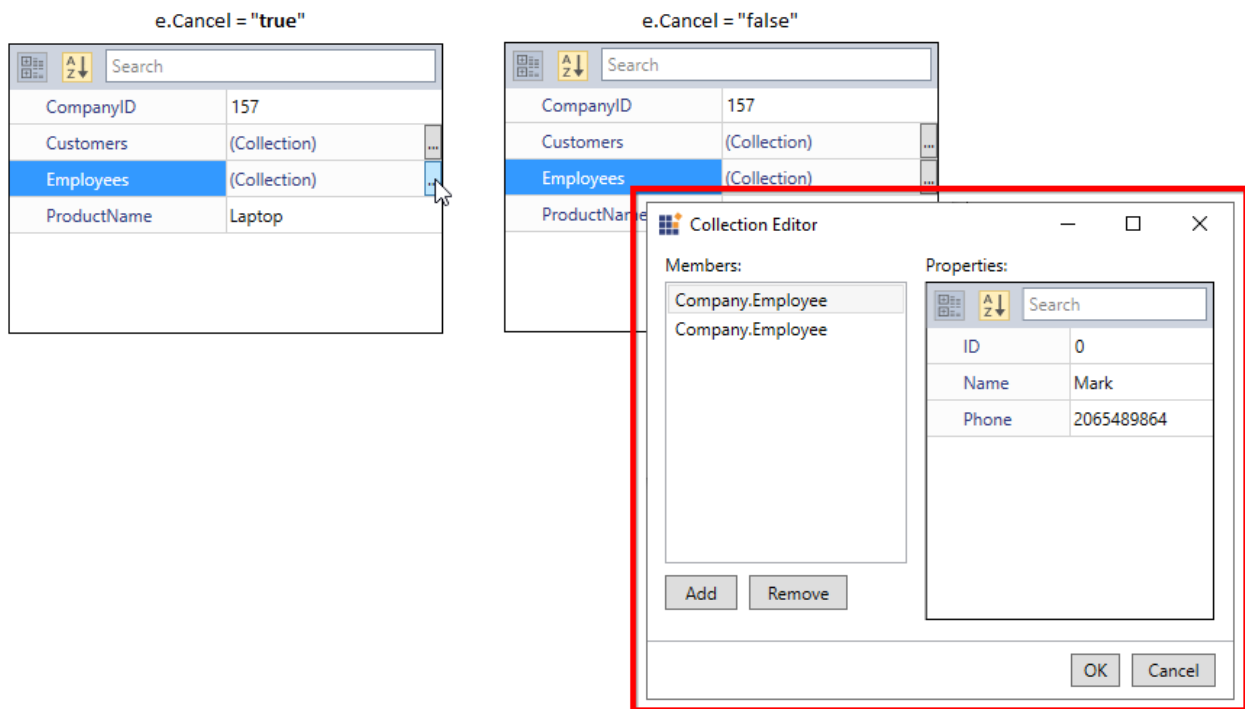
```

```
propertyGrid.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("DemoProduct"));
propertyGrid.CollectionEditorOpening +=
propertyGrid.CollectionEditorOpening;
```

You can handle the `CollectionEditorOpening` event as follows,

C#

```
private void propertyGrid_CollectionEditorOpening(object sender,
CollectionEditorOpeningEventArgs e)
{
    //Restrict collection editor window opening
    e.Cancel = false;
}
```



Note: [View Sample in GitHub](#)

Built-in Editor in WPF PropertyGrid

The [PropertyGrid](#) control supports several built-in editors. Based on the property type, the built-in editors automatically assigned as value editor for the properties and it allows only the valid inputs based on property type.

S.No	Property Type	Default Editor
1	int	IntegerTextBox
2	double	DoubleTextBox
3	string	TextBox

4	enum	ComboBox
5	DateTime	DateTimeEdit
6	bool	CheckBox
7	Brush	ColorPicker

C#

```
// Employee class to be explored in property grid.
public class Employee {
    public string EmployeeName { get; set; }
    public string ID { get; set; }
    public int Age { get; set; }
    public int Experiance { get; set; }
}

public class ViewModel {
    public object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            EmployeeName = "Johnson",
            Age = 25,
            ID = "1234",
            Experiance = 3
        };
    }
}
```

XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
    Name="propertyGrid1" >
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
```

Search	
Age	25
EmployeeName	Johnson
Experience	3
ID	1234

Here, Age, Experience and ID properties is a int type properties, they allow only the whole number inputs. EmployeeName is a string type property, so TextBox is assigned as a value editor and all the text will be allowed.

Built-in mask to restrict user input

You can restrict the user to enter valid value such as alphanumeric, binary, email-Id, IPv4, product Key, positive number, and etc by using the built-in [MaskAttribute](#).

Note: You can apply mask attribute to property of type Object or string only.

S_No | Mask Type | Example

1 Alphanumeric	0ac53gd6X7	2 Binary	01101011
3 CardNumber	9845-1837-8463-6238	4	
EmailId	johnabc@gt.a.co	5 Fraction	/347
6 HexaDecimal	\x12	7	
IPv4	627.813.646.358	8 IPv6	8825:8353:2520:2723:2926:9254:2824:7186
MobileNumber	4567893563	10 Number	6482975449492394684423
11 Octal	\0127	12	
PositiveNumber	246282474945452	13 ProductKey	HTY23-OPY67-8GR56-R01G5-64GFC

C#

```
using Syncfusion.Windows.PropertyGrid;
//EmailId mask for multiple properties
[Mask(MaskAttribute.EmailId,"EmailID_1, EmailID_2")]
[Mask(MaskAttribute.CardNumber, "CardNumberMask")]
public class Masks{
    [Mask(MaskAttribute.Binary)]
    public string BinaryMask { get; set; }
    public string CardNumberMask { get; set; }
    public string EmailID_1 { get; set; }
    public string EmailID_2 { get; set; }
    [Mask(MaskAttribute.PositiveNumber)]
    public string PositiveNumberMask { get; set; }
    [Mask(MaskAttribute.ProductKey)]
```

```

public string ProductKeyMask { get; set; }
}
//ViewModel.cs
public class ViewModel {
public object BuiltInMasks { get; set; }
public ViewModel() {
BuiltInMasks = new Masks()
{
BinaryMask = "0111",
CardNumberMask = "X984DH51837RD846362JH38",
EmailID_1 = "johnabc@",
EmailID_2 = "peter123@gta.co",
PositiveNumberMask = "-39479",
ProductKeyMask = "HTY23OPY678GR56R01G564GH"
};
}
}
}

```

XML

```

<syncfusion:PropertyGrid SelectedObject="{Binding BuiltInMasks}"
Name="propertyGrid" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel/>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

BinaryMask	0111
CardNumberMask	9845-1837-8463-6238
EmailID_1	johnabc@
EmailID_2	peter123@gta.co
PositiveNumberMask	39479
ProductKeyMask	HTY23-OPY67-8GR56-R01G5-64GH

Note: [View Sample in GitHub](#)

Mask attribute with custom mask to restrict user input

You can restrict the user from providing invalid input for the one or more specific property by setting your custom **Regex** based mask to the **MaskAttribute**. For example emails, phone numbers, zip codes, currency, etc.

C#

```

//Setting mask option for two specific properties
[MaskAttribute("[A-Za-z0-9._%~]+@[A-Za-z0-9]+.[A-Za-z]{2,3}"),
"Email_1,Email_2")]
public class Person {
public Person() {

```

```

Name = "Carl Johnson";
Age = 3;
Mobile = "2054449786";
Email_1 = "carljohnson@gta.com";
Email_2 = "johnson@gta.c";
}
public string Name { get; set; }
public string Email_1 { get; set; }
public object Email_2 { get; set; }
//Setting mask option for Mobile property
[MaskAttribute(@"\(\d{3}\) \d{3} - \d{4}")]
public string Mobile { get; set; }
//Setting mask option for Age properties
[MaskAttribute(@"\d{2}")]
public object Age { get; set; }
}

```

XML

```

<syncfusion:PropertyGrid x:Name="propertyGrid">
<syncfusion:PropertyGrid.SelectedObject>
<local:Person/>
</syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>

```

Property	Value
Age	3
Email_1	carljohnson@gta.com
Email_2	johnson@gta.c
Mobile	(205) 444 - 9786
Name	Carl Johnson

Here, the partial and wrong inputs are highlighted by the red color border.

Note: View [Sample](#) in GitHub

MaskEditor to restrict user input

You can restrict the user to enter particular character or value for the one or more specific property item by using the **Regex** based mask to the custom editor's **Mask** property which is derived from the **MaskEditor**.

Note: You must create the property as either **Object** or **string** type to use mask editor.

Note: Please refer the [Custom Editor](#) page to know more about how to set the custom editor for the property items.

C#

```

using System.ComponentModel;

```

```

using Syncfusion.Windows.PropertyGrid;
//Custom mask editor
public class EmailEditor : MaskEditor {
public EmailEditor() {
Mask = "[A-Za-z0-9._%~]+@[A-Za-z0-9]+.[A-Za-z]{2,3}"
}
}
//Custom mask editor
public class MobileNoEditor : MaskEditor {
public MobileNoEditor() {
Mask = @"\"(\d{3}\) \d{3} - \d{4}";
}
}
//Setting custom mask editors for the properties
[Editor("Email_1, Email_2", typeof(EmailEditor))]
[Editor("Mobile", typeof(MobileNoEditor))]
public class Person {
public Person() {
Name = "Carl Johnson";
Age = 3;
Mobile = "2054449";
Email_1 = "carljohnson@gta.c";
Email_2 = "johnson@gta.com";
}
public string Name { get; set; }
public string Email_1 { get; set; }
public object Email_2 { get; set; }
public string Mobile { get; set; }
public object Age { get; set; }
}



```

XML

```

<syncfusion:PropertyGrid x:Name="propertyGrid">
<syncfusion:PropertyGrid.SelectedObject>
<local:Person/>
</syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>

```

<div>   <input type="text" value="Search"/> </div>	
Age	3
Email_1	carljohnson@gta.c
Email_2	johnson@gta.com
Mobile	(205) 444 - 9
Name	Carl Johnson

Here, the partial and wrong inputs are highlighted by the red color border.

Note: View [Sample](#) in GitHub

Custom Editor in WPF PropertyGrid

The [PropertyGrid](#) control supports several built-in editors. Based on the property type, the built-in editors automatically assigned as value editor for the properties. You can assign own value editor(control) for the properties instead of default value editor by using the [Editor](#) attribute or [CustomEditorCollection](#).

For example, if you creates an `EmailID` property as a string type, `TextBox` is assigned as a value editor and all the text will be allowed. If you want to accept the input that is only in the mail id format, you can assign [SfMaskedEdit](#) control with email-id mask as the value editor for the `EmailID` property.

Best practice to follow

When using the [CustomEditorCollection](#) property to assign a custom editor to multiple properties of the same data type in the `PropertyGrid`, it is recommended to use the [EditorType](#), [PropertyType](#) and [HasPropertyType](#) properties of the [CustomEditor](#) class. Otherwise, for proper functionality, you must create a new `CustomEditor` class and assign it to each property using the `Properties` collection property.

Creating the Custom Editor

To create [CustomEditor](#), we need to implement [ITypeEditor](#) interface. Here, `SfMaskedEdit` control is assigned with mail id mask as `EmailEditor` and `UpDown` control is assigned with min, max value as `IntegerEditor`. `EmailEditor` and `IntegerEditor` are the custom editors.

C#

```
//Custom Editor for the EmailId properties.
public class EmailEditor : ITypeEditor {
    SfMaskedEdit maskededit;
    public void Attach(PropertyViewItem property, PropertyItem info) {
        if (info.CanWrite) {
            var binding = new Binding("Value")
            {
                Mode = BindingMode.TwoWay,
                Source = info,
                ValidatesOnExceptions = true,
                ValidatesOnDataErrors = true
            };
            BindingOperations.SetBinding(maskededit, SfMaskedEdit.ValueProperty,
            binding);
        }
        else {
            maskededit.IsEnabled = false;
            var binding = new Binding("Value")
            {
                Source = info,
                ValidatesOnExceptions = true,
                ValidatesOnDataErrors = true
            };
            BindingOperations.SetBinding(maskededit, SfMaskedEdit.ValueProperty,
            binding);
        }
    }
    public object Create(PropertyInfo propertyInfo) {
        maskededit = new SfMaskedEdit();
        maskededit.MaskType = MaskType.RegEx;
```

```

maskededit.Mask = "[A-Za-z0-9._%-]+@[A-Za-z0-9]+.[A-Za-z]{2,3}";
return maskededit;
}
public void Detach(PropertyViewItem property) {
}
}
//Custom Editor for the integer type properties.
public class IntegerEditor : ITypeEditor {
    UpDown upDown;
    public void Attach(PropertyViewItem property, PropertyItem info) {
        if (info.CanWrite) {
            var binding = new Binding("Value")
            {
                Mode = BindingMode.TwoWay,
                Source = info,
                ValidatesOnExceptions = true,
                ValidatesOnDataErrors = true
            };
            BindingOperations.SetBinding(upDown, UpDown.ValueProperty, binding);
        }
        else {
            upDown.IsEnabled = false;
            var binding = new Binding("Value")
            {
                Source = info,
                ValidatesOnExceptions = true,
                ValidatesOnDataErrors = true
            };
            BindingOperations.SetBinding(upDown, UpDown.ValueProperty, binding);
        }
    }
    public object Create(PropertyInfo propertyInfo) {
        upDown = new UpDown();
        upDown.ApplyZeroColor = false;
        upDown.MinValue = 0;
        upDown.MaxValue = 100;
        upDown.NumberDecimalDigits = 0;
        return upDown;
    }
    public void Detach(PropertyViewItem property) {
    }
}

```

Note: To assign a created custom editor to a properties, refer the [Assigning a Custom Editor](#) topic.

Creating Custom Editor for a dynamic property.

If the [SelectedObject](#) has a property of type `dynamic`, `ExpandoObject` or `ICustomTypeDescriptor`, we can create `CustomEditor` class by inheriting [BaseTypeEditor](#). You can initialize a new instance of the custom editor using the [BaseTypeEditor.Create\(PropertyDescriptor\)](#) function. Below example shows, how to get the value of dynamic properties using its `PropertyDescriptor` and apply the value in `ComboEditor` to `ComboBox` objects.

C#

```

//Custom Editor for the List<string> type properties.

```

```

public class ComboEditor : BaseTypeEditor
{
    ComboBox comboBox;
    public override void Attach(PropertyViewItem property, PropertyItem info)
    {
        var binding = new Binding("Value")
        {
            Mode = BindingMode.TwoWay,
            Source = info,
            ValidatesOnExceptions = true,
            ValidatesOnDataErrors = true
        };
        BindingOperations.SetBinding(comboBox, ComboBox.SelectedItemProperty,
            binding);
    }
    // Create a custom editor for a normal property
    public override object Create(PropertyInfo PropertyInfo)
    {
        throw new NotImplementedException();
    }
    // Create a custom editor for a dynamic property
    public override object Create(PropertyDescriptor PropertyDescriptor)
    {
        comboBox = new ComboBox();
        // Getting the values of dynamic property
        dynamic comboBoxItemsList =
            PropertyDescriptor.GetValue(PropertyDescriptor.Name);
        foreach (dynamic items in comboBoxItemsList)
        {
            comboBox.Items.Add(items);
        }
        comboBox.SelectedIndex = 0;
        return comboBox;
    }
    public override void Detach(PropertyViewItem property)
    {
        comboBox = null;
    }
}

```

Note: Download demo application from [GitHub](#).

Assigning a Custom Editor using Editor Attribute

We can assign the `CustomEditor` to any individual property by name of the property and to multiple properties based on the property type by using the `Editor` attribute.

C#

```

//CustomEditor for the specfic(EmailID) property
[Editor("EmailID", typeof(EmailEditor))]
//Custom Editor for the multiple(Tnteger type) properties
[Editor(typeof(int), typeof(IntegerEditor))]
public class Employee {
    public string EmailID { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}

```



```

public int Experience { get; set; }
}
class ViewModel {
public object SelectedEmployee { get; set; }
public ViewModel() {
SelectedEmployee = new Employee()
{
Age = 25,
Name = "mark",
Experience = 5,
EmailID = "mark@gt"
};
}
}
}

```

XML

```

<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

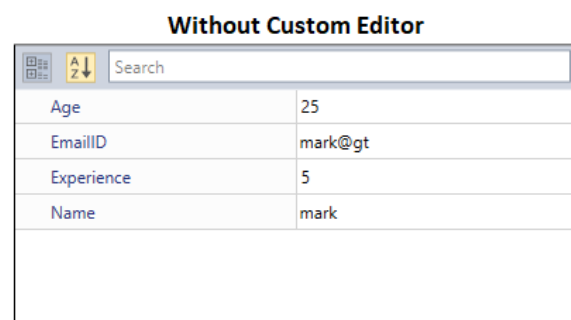
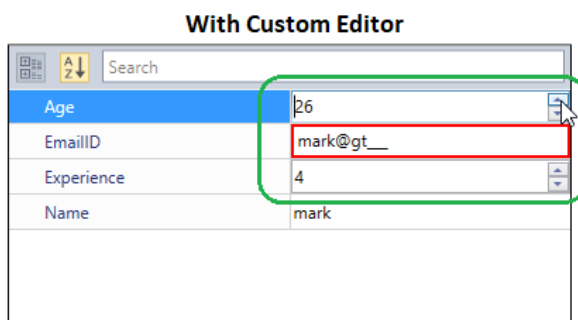
C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));

```

Here, EmailID property value editor changed from TextBox to MaskedEdit control with email id mask. Also, we assigned the IntegerEditor for the integer type properties, so it applied to the Experience and Age properties. Then, the value editors for the Experience and Age property is changed from NumericTextBox to Updown control.



Assigning a Custom Editor using Collection

We can assign the CustomEditor to any particular property and to multiple properties using the CustomEditorCollection.

Assigning a Custom Editor to the specific property

If we want to apply custom editor for any particular property, we need to create the `CustomEditor` instance, assign our own editor to the `CustomEditor.Editor` and add the property name to the `CustomEditor.Properties` collection. Then, add the `CustomEditor` instance to the `PropertyGrid.CustomEditorCollection`.

C#

```
public class Employee {
    public string EmailID { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public int Experience { get; set; }
}

class ViewModel {
    public object SelectedEmployee { get; set; }
    public CustomEditorCollection customEditorCollection = new
    CustomEditorCollection();
    public CustomEditorCollection CustomEditorCollection
    {
        get { return customEditorCollection; }
        set { customEditorCollection = value; }
    }
    public ViewModel() {
        SelectedEmployee = new Employee() { Age = 25, Name = "mark", Experience = 5,
        EmailID = "mark@gt" };
        // EmailEditor added to the collection and will applied to the "EmailID"
        property
        CustomEditor editor1 = new CustomEditor();
        editor1.Editor = new EmailEditor();
        editor1.Properties.Add("EmailID");
        CustomEditorCollection.Add(editor1);
    }
}
```

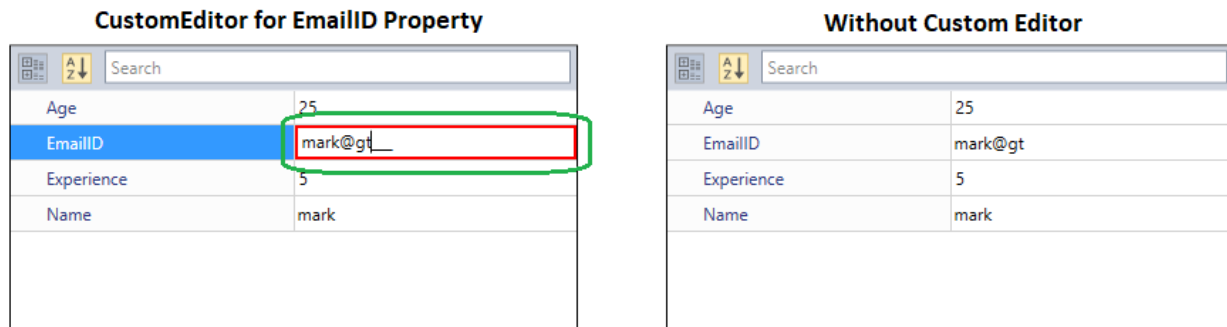
XML

```
<syncfusion:PropertyGrid CustomEditorCollection="{Binding
CustomEditorCollection}"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.CustomEditorCollectionProperty, new
Binding("CustomEditorCollection"));
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
```

Here, The EmailID is a string property, the TextBox is assigned as a default editor. We changed it as SfMaskedEdit textbox that accepts only inputs which are in the email id format.



Assigning a Custom Editor based on the property type

If we want to apply custom editor for multiple properties which are all contains same type, we need to create the CustomEditor instance, assign our own editor to the CustomEditor.Editor and sets the CustomEditor.HasPropertyType property to true. Then, mention the property type to the CustomEditor.PropertyType.

C#

```
public class Employee {
    public string EmailID { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public int Experience { get; set; }
}

class ViewModel {
    public object SelectedEmployee { get; set; }
    public CustomEditorCollection customEditorCollection = new
    CustomEditorCollection();
    public CustomEditorCollection CustomEditorCollection
    {
        get { return customEditorCollection; }
        set { customEditorCollection = value; }
    }
    public ViewModel() {
        SelectedEmployee = new Employee() { Age = 25, Name = "mark", Experience = 5,
        EmailID = "mark@gt" };
        // IntegerEditor added to the collection and will applied to the "int" type
        properties
        CustomEditor editor = new CustomEditor();
        editor.Editor = new IntegerEditor();
        editor.HasPropertyType = true;
        editor.PropertyType = typeof(int);
        CustomEditorCollection.Add(editor);
    }
}
```

XML

```
<syncfusion:PropertyGrid CustomEditorCollection="{Binding
CustomEditorCollection}"
```

```

SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.CustomEditorCollectionProperty, new
Binding("CustomEditorCollection"));
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));

```

With Customer Editor

Search	
Age	25
EmailID	mark@qta.com
Experience	4
Name	mark

Without Custom Editor

Search	
Age	25
EmailID	mark@gt
Experience	5
Name	mark

Here, we assigned the `IntegerEditor` custom editor for the integer type properties, so it applied to the `Experience` and `Age` properties. Then, the value editors for the `Experience` and `Age` property is changed from `NumericTextBox` to `Updown` control.

Click [here](#) to download the sample that showcases the `CustomEditor` support.

Assigning a Custom Editor by the editor type

By default, when we use the `CustomEditor.Editor` to target multiple properties in `PropertyGrid`, the same custom editor instance is used for all those properties. To maintain a dedicated or separate instance of the custom editor for each property, use `CustomEditor.EditorType`. The default value of the `EditorType` property is `null`.

You can set the value for `EditorType` property when custom editor is initialized in `ViewModel` class as shown below.

XML

```

<syncfusion:PropertyGrid CustomEditorCollection="{Binding
CustomEditorCollection}"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

public class Employee
{
    public string Country { get; set; }
    public double Experience { get; set; }
    public string Name { get; set; }
    public double Age { get; set; }
}

class ViewModel
{
    public object SelectedEmployee { get; set; }
    private CustomEditorCollection customEditorCollection = new
    CustomEditorCollection();
    public CustomEditorCollection CustomEditorCollection
    {
        get { return customEditorCollection; }
        set { customEditorCollection = value; }
    }
    public ViewModel()
    {
        SelectedEmployee = new Employee() { Age = 25, Name = "mark", Experience = 5,
        EmailID = "mark@gt.com" };
        CustomEditor editor1 = new CustomEditor()
        {
            EditorType = typeof(IntegerEditor),
            HasPropertyType = true,
            PropertyType = typeof(double)
        };
        CustomEditorCollection.Add(editor1);
    }
}

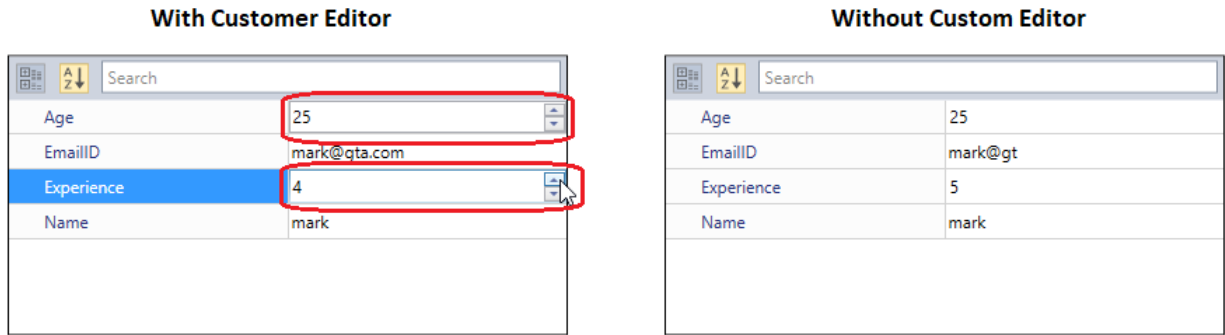
```

You can also set value for `CustomEditorType` property for `CustomEditor` class in the xaml file as shown below.

```

<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
<syncfusion:PropertyGrid.CustomEditorCollection>
<syncfusion:CustomEditorCollection>
<syncfusion:CustomEditor PropertyType="{x:Type a:Double}" HasPropertyType="True"
EditorType="{x:Type local:IntegerEditor}" />
</syncfusion:CustomEditorCollection>
</syncfusion:PropertyGrid.CustomEditorCollection>
</syncfusion:PropertyGrid>

```



Use constructor with parameters in custom editor

By default, [PropertyGrid](#) control only invokes the constructor without parameter in custom editor. You can invoke and pass arguments to the constructors with any number of parameters in custom editor using the [ConstructorParameter](#) property of [CustomEditor](#) class. The default value of [ConstructorParameter](#) is **null**. This can be achieved by following the below steps.

Note: [ConstructorParameter](#) property will work only if a value is assigned for the [EditorType](#) property in the [CustomEditor](#) class.

1. Create a custom editor for the desired property item in [PropertyGrid](#). Add constructor with parameters in the custom editor class.

C#

```
public class IntegerEditor : ITypeEditor
{
    IntegerTextBox integerTextBox;
    bool ShowUpDown = false;
    public IntegerEditor()
    {
    }
    // Constructor with parameter in custom editor
    public IntegerEditor(bool showUpDown)
    {
        ShowUpDown = showUpDown;
    }
    public void Attach(PropertyViewItem property, PropertyItem info)
    {
        if (info.CanWrite)
        {
            var binding = new Binding("Value")
            {
                Mode = BindingMode.TwoWay,
                Source = info,
                ValidatesOnExceptions = true,
                ValidatesOnDataErrors = true
            };
            BindingOperations.SetBinding(integerTextBox, IntegerTextBox.ValueProperty, binding);
        }
        else
        {
        }
    }
}
```

```

integerTextBox.IsEnabled = false;
var binding = new Binding("Value")
{
    Source = info,
    ValidatesOnExceptions = true,
    ValidatesOnDataErrors = true
};
BindingOperations.SetBinding(integerTextBox, IntegerTextBox.ValueProperty,
binding);
}
}
public object Create(PropertyInfo propertyInfo)
{
    integerTextBox = new IntegerTextBox()
    {
        ApplyZeroColor = false,
        MinValue = 0,
        MaxValue = 50,
        ShowSpinButton = ShowUpDown,
    };
    return integerTextBox;
}
public void Detach(PropertyViewItem property)
{
}
}

```

2. Create **Employee** and **ViewModel** classes with required properties. Pass the required objects in an object array and assign it to the **ConstructorParameter** property of **CustomEditor** as shown below.

C#

```

public class Employee
{
    public string EmailID { get; set; }
    public double Experience { get; set; }
    public string Name { get; set; }
    public long Age { get; set; }
}
class ViewModel
{
    public object SelectedEmployee { get; set; }
    private CustomEditorCollection customEditorCollection = new
    CustomEditorCollection();
    public CustomEditorCollection CustomEditorCollection
    {
        get { return customEditorCollection; }
        set { customEditorCollection = value; }
    }
    public int DecimalDigits { get; set; }
    public ViewModel()
    {

```

```

SelectedEmployee = new Employee() { Age = 25, Name = "Mark Anthony", EmailID
= "markanthony@syncfusion.com", Experience = 3 };
CustomEditor editor1 = new CustomEditor()
{
    EditorType = typeof(IntegerEditor),
    HasPropertyType = true,
    PropertyType = typeof(long),
    ConstructorParameters = new object [] {true}
};
CustomEditorCollection.Add(editor1);
}
}

```

XML

```

<syncfusion:PropertyGrid CustomEditorCollection="{Binding
CustomEditorCollection}" SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

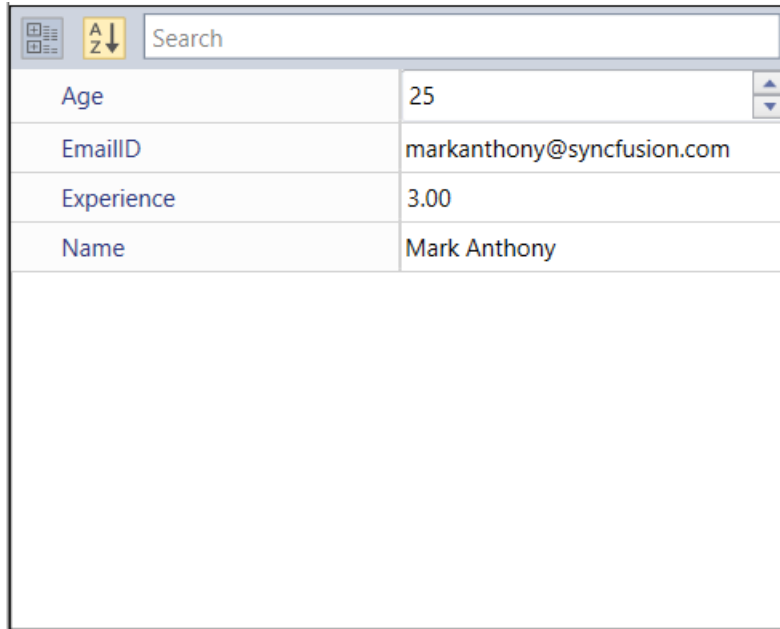
Since we have assigned the custom editor for property type **long(Int64)**, the custom editor will be applied for **Age** property item. You can also create custom editor and set value for **ConstructorParameter** property in xaml file as shown below.

XML

```

<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
<syncfusion:PropertyGrid.CustomEditorCollection>
<syncfusion:CustomEditorCollection>
<syncfusion:CustomEditor PropertyType="{x:Type a:Int64}"
HasPropertyType="True" EditorType="{x:Type local:IntegerEditor}" >
<syncfusion:CustomEditor.ConstructorParameters>
<x:Array Type="{x:Type a:Object}">
<a:Boolean>True</a:Boolean>
</x:Array>
</syncfusion:CustomEditor.ConstructorParameters>
</syncfusion:CustomEditor>
</syncfusion:CustomEditorCollection>
</syncfusion:PropertyGrid.CustomEditorCollection>
</syncfusion:PropertyGrid>

```

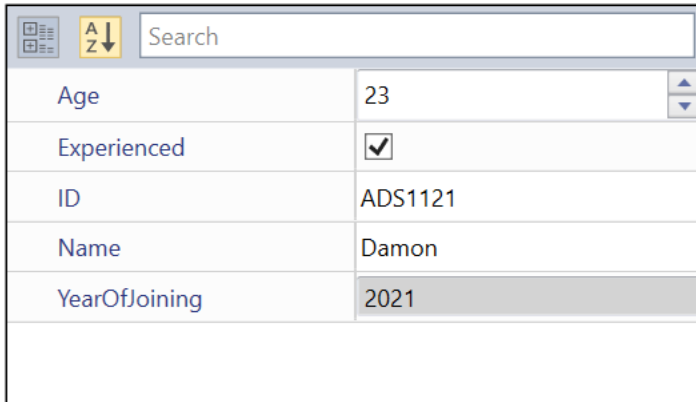
Create custom editor control for the property item

The [PropertyGrid](#) control allows users to create their editor controls using the [CustomEditor](#) class and [CustomEditorCollection](#) property for property items. The custom editors can be inherited from [ITypeEditor](#) and [BaseTypeEditor](#) interfaces.

The [Create](#) method in the custom editor class can be used to create and assign the required editor controls for specific property items. The Create method's [PropertyInfo](#) parameter allows users to customize the editor control based on the value of [PropertyInfo](#) class.

C#

```
public object Create(PropertyInfo propertyInfo)
{
    textBox = new IntegerTextBox();
    if (propertyInfo.Name == "Age")
    {
        textBox.MinValue = 20;
        textBox.MaxValue = 50;
    }
    if (propertyInfo.CanWrite)
        textBox.ShowSpinButton = true;
    return textBox;
}
```



Search	
Age	23
Experienced	<input checked="" type="checkbox"/>
ID	ADS1121
Name	Damon
YearOfJoining	2021

Note: If a [CustomEditor](#) class is inherited from [BaseTypeEditor](#) interface, the Create method will have [PropertyDescriptor](#) parameter value which can be used as per requirement.

Attach the custom editor with property item

The [PropertyGrid](#) control allows users to bind the essential properties of the custom editor control with the properties of the property items using the [Attach](#) method. You can also customize the property items using the [PropertyItem](#) and [PropertyViewItem](#) using the parameters in the [Attach](#) method.

C#

```
public void Attach(PropertyViewItem property, PropertyItem info)
{
    if (info.CanWrite)
    {
        property.FontFamily = new FontFamily("Comic Sans MS");
        var binding = new Binding("Value")
        {
            Mode = BindingMode.TwoWay,
            Source = info,
            ValidatesOnExceptions = true,
            ValidatesOnDataErrors = true
        };
        BindingOperations.SetBinding(textBox, IntegerTextBox.ValueProperty,
            binding);
    }
}
```

Search	
Age	23
Experienced	<input checked="" type="checkbox"/>
ID	ADS1121
Name	Damon
YearOfJoining	2015

Dispose the custom editor

The [PropertyGrid](#) control allows you to dispose the custom editor control and its dependent properties in CustomEditor class by using the [Detach](#) method. You can also dispose the [PropertyViewItem](#) which will be available from the parameter of [Detach](#) method.

C#

```
public void Detach(PropertyViewItem property)
{
    if (property != null && this.textBox != null)
    {
        this.textBox = null;
        property.Dispose();
        property = null;
    }
}
```

Category Editor in WPF PropertyGrid

The [PropertyGrid](#) control supports several built-in editors. [CategoryEditor](#) support enables us to set the related properties (one or more properties) under single or multiple category based on the need.

[CategoryEditor](#) can be applied in category view. While sorted view, default editors will be applied.

Adding Category Editor to PropertyGrid

If we want to display some related properties under the specific category, we can do it by the [CategoryEditor](#). We can add any number of [CategoryEditor](#). We must add property names in the [CategoryEditor.Properties](#) collection which are need to be categorized in the same category. Using the [CategoryEditor.EditorTemplate](#), we can create the own template for the categorized properties with its required value editors. The properties and its value editor can be placed in the [PropertyGrid](#) as our wish. To display the [CategoryEditor](#), we need the enable the [EnableGrouping](#) property as [true](#).

For example, the [Background](#), [BorderBrush](#) and [Foreground](#) are brush type properties. They will be categorized under 'A-Brushes' category. [ColorEdit](#) controls will be assigned as the value editor for the above properties and located in the same place. Based on the property selection, respective [ColorEdit](#) control will be in the view to pick the color for that property.

a) **ColorEdit** control visibility converter**C#**

```

using System;
using System.Windows;
using System.Windows.Data;
public class SelectedIndexToVisibility : IValueConverter {
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture) {
int index = (int)value;
Visibility vis;
if (index == 0 && parameter.ToString() == "Foreground") {
vis = Visibility.Visible;
}
else if (index == 1 && parameter.ToString() == "Background") {
vis = Visibility.Visible;
}
else if (index == 2 && parameter.ToString() == "BorderBrush") {
vis = Visibility.Visible;
}
else {
vis = Visibility.Collapsed;
}
return vis;
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture) {
return null;
}
}

```

b) Creating the CategoryEditor for the properties:

XML

```

<Window.Resources>
<local:SelectedIndexToVisibility x:Key="VisConv"/>
</Window.Resources>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition/>
<ColumnDefinition Width="300"/>
</Grid.ColumnDefinitions>
<syncfusion:PropertyGrid SelectedObject="{Binding ElementName=button}"
EnableGrouping="True" Name="propertyGrid1"
Margin="20" Grid.Column="0" >
<syncfusion:PropertyGrid.CategoryEditors>
<syncfusion:CategoryEditor Category="A-Brushes">
<syncfusion:CategoryEditor.Properties>
<syncfusion:CategoryEditorProperty Name="Background"/>
<syncfusion:CategoryEditorProperty Name="Foreground"/>
<syncfusion:CategoryEditorProperty Name="BorderBrush"/>
</syncfusion:CategoryEditor.Properties>
<syncfusion:CategoryEditor.EditorTemplate>
<DataTemplate>
<Grid>

```

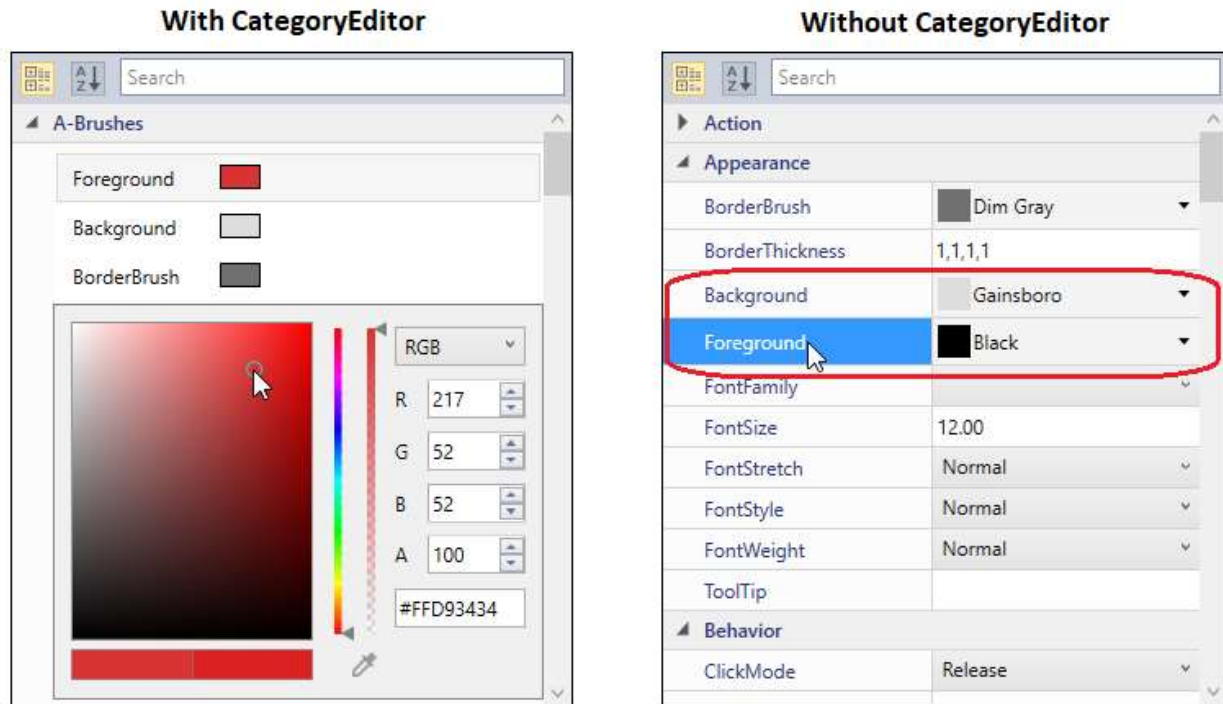
```

<Grid.RowDefinitions>
<RowDefinition />
<RowDefinition />
</Grid.RowDefinitions>
<ListBox BorderBrush="Transparent" x:Name="list" SelectedIndex="0">
<StackPanel HorizontalAlignment="Center" Orientation="Horizontal">
<TextBlock Text="Foreground" Margin="5" Width="80"/>
<Rectangle Fill="{Binding Path=CategoryValueProperties[Foreground].Value,
Mode=TwoWay}"
Stroke="Black" Height="15" Width="25" Margin="5"/>
</StackPanel>
<StackPanel HorizontalAlignment="Center" Orientation="Horizontal">
<TextBlock Text="Background" Margin="5" Width="80"/>
<Rectangle Fill="{Binding Path=CategoryValueProperties[Background].Value,
Mode=TwoWay}"
Stroke="Black" x:Name="background" Height="15" Width="25" Margin="5" />
</StackPanel>
<StackPanel HorizontalAlignment="Center" Orientation="Horizontal">
<TextBlock Text="BorderBrush" Margin="5" Width="80"/>
<Rectangle Fill="{Binding Path=CategoryValueProperties[BorderBrush].Value,
Mode=TwoWay}"
Stroke="Black" Height="15" Width="25" Margin="5"/>
</StackPanel>
</ListBox>
<syncfusion:ColorEdit Visibility="{Binding ElementName=list,
Path=SelectedIndex,
Converter={StaticResource VisConv}, ConverterParameter=Foreground}"
Brush="{Binding Path=CategoryValueProperties[Foreground].Value,
Mode=TwoWay}"
EnableToolTip="False" Grid.Row="1" IsGradientPropertyEnabled="False" />
<syncfusion:ColorEdit Visibility="{Binding ElementName=list,
Path=SelectedIndex,
Converter={StaticResource VisConv}, ConverterParameter=Background}"
Brush="{Binding Path=CategoryValueProperties[Background].Value,
Mode=TwoWay}"
EnableToolTip="False" Grid.Row="1" IsGradientPropertyEnabled="False"/>
<syncfusion:ColorEdit Visibility="{Binding ElementName=list,
Path=SelectedIndex,
Converter={StaticResource VisConv}, ConverterParameter=BorderBrush}"
Brush="{Binding Path=CategoryValueProperties[BorderBrush].Value,
Mode=TwoWay}"
EnableToolTip="False" Grid.Row="1" IsGradientPropertyEnabled="False"/>
</Grid>
</DataTemplate>
</syncfusion:CategoryEditor.EditorTemplate>
</syncfusion:CategoryEditor>
</syncfusion:PropertyGrid.CategoryEditors>
</syncfusion:PropertyGrid>
<StackPanel Grid.Column="1" VerticalAlignment="Center"
HorizontalAlignment="Center" >
<TextBlock Grid.Column="1" Margin="10" HorizontalAlignment="Center"
TextWrapping="Wrap" VerticalAlignment="Center" Width="168">
<Run Text="Selected Object: " /><Run FontWeight="Bold" Text="Button" />
</TextBlock>
<Button Name="button" Width="200" Height="50" VerticalAlignment="Center"
HorizontalAlignment="Center" Content="Click me"/>
</StackPanel>

```

```
</Grid>
```

By Default, the properties are arranged like the key-value pairs with its value editor. Using the **CategoryEditor**, the Background, Foreground and BorderBrush property value editors are arranged by our wish.



Click [here](#) to download the sample that showcases the **CategoryEditor** support.

DisplayName of Property in WPF PropertyGrid

By default, the property name is displayed in the **PropertyGrid**. We can change the display name of the properties instead of the property name by using the attributes and event.

Change property display name using attributes

We can give a meaningful name to the properties that are displayed in the **PropertyGrid** instead of the property name by using the **Name** field of the **Display** attribute and **DisplayName** attribute.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
    [Display(Name = "Employee Name")]
    public string Name { get; set; }
    [DisplayName("Employee ID")]
    public string ID { get; set; }
    public DateTime DOB { get; set; }
}
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
```

```

SelectedEmployee = new Employee()
{
    Name = "John",
    ID = "381",
    DOB = new DateTime(1995, 12, 24)
};
}
}

```

XML

```

<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));

```



Here, the **Name** and **ID** properties is displayed as **Employee Name** and **Employee ID** respectively.

Note: If you use both the **DisplayName** attribute and **Name** field of the **Display** attribute, the **Name** field of the **Display** attribute will have higher priority.

Click [here](#) to download the sample that showcases the property **Display Name** support using attributes.

Change property display name at runtime

We can set and change the property display name instead of the property name at runtime without using attributes by handling the [AutoGeneratingPropertyGridItem](#) event with [AutoGeneratingPropertyGridItemEventArgs.DisplayName](#) property.

C#

```

using System;
using System.ComponentModel;

```

```

using System.ComponentModel.DataAnnotations;
public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
}
public class ViewModel {
    public object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24)
        };
    }
}

```

XML

```

<syncfusion:PropertyGrid
    AutoGeneratingPropertyGridItem="PropertyGrid1_AutoGeneratingPropertyGridItem"
    SelectedObject="{Binding SelectedEmployee}"
    x:Name="propertyGrid1">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.AutoGeneratingPropertyGridItem +=
PropertyGrid1_AutoGeneratingPropertyGridItem;

```

C#

```

private void PropertyGrid1_AutoGeneratingPropertyGridItem(object sender,
AutoGeneratingPropertyGridItemEventArgs e) {
    //Name and DOB property names changed as 'Employee Name' and 'Date of
    Birth'.
    if (e.DisplayName == "Name") {
        e.DisplayName = "Employee Name";
    }
    else if (e.DisplayName == "DOB") {
        e.DisplayName = "Date of Birth";
    }
}

```




Here, the Name and DOB property display name is changed as Employee Name and Date of Birth by the `AutoGeneratingPropertyGridItemEventArgs.Name` property of the `AutoGeneratingPropertyGridItem` event, not by any attributes.

Click [here](#) to download the sample that showcases the property Display Name support using `AutoGeneratingPropertyGridItem` event.

Change width of property's name column

By default, you can change the width of property's name column by using grid splitter after loading the control. If you want to set or changes the width of property's name column when loading the `PropertyGrid`, use the `PropertyNameColumnDefinition` property.

You can use the `double` or `Star` values for changing the column definition. The default value of `PropertyNameColumnDefinition` property is `1`. *Property's value column always occupies 1*, which cannot be changed.

Note: `PropertyGrid.PropertyNameColumnWidth` property does not supports `Auto` size value.

For example, if you set the value for `PropertyNameColumnDefinition` property to `200`, then property's name column occupies `200` pixels. On the other hands, if `0.5*` is set to `PropertyNameColumnDefinition` property, then property's name column occupies, one third of available width. Remaining two third is occupied by property's value column.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
}
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24)
        }
    }
}
```

```
};
}
```

XML

```
<syncfusion:PropertyGrid PropertyNameColumnDefinition="0.5*"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
propertyGrid1.PropertyNameColumnDefinition = new GridLength(0.5,
GridUnitType.Star);
```

PropertyNameColumnDefinition = "0.5"

DOB	12/24/1995	
ID	381	
Name	John	

PropertyNameColumnDefinition = "200"

DOB	12/24/1995	
ID	381	
Name	John	

Note: [View Sample in GitHub](#)

Property Description in WPF PropertyGrid

You can display the description about the property using the description panel which is placed on the bottom of the [PropertyGrid](#) control. Description panel visibility can be managed by [DescriptionPanelVisibility](#) property. The default value of the [DescriptionPanelVisibility](#) is [Collapsed](#). To display the description panel, you should set [DescriptionPanelVisibility](#) property value as [Visible](#).

Property Description using attributes

We can give a meaningful description about the properties by using the [Description](#) attribute and [Display](#) attribute's [Description](#) field. This description will be displayed in [PropertyGrid](#) Description panel while focusing the property or its value editor.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
    [Display(Description = "Name of the employee")]
    public string Name { get; set; }
    public string ID { get; set; }
    [Description("Birth date of the employee")]
```

```

public DateTime DOB { get; set; }
}
public class ViewModel {
public Object SelectedEmployee { get; set; }
public ViewModel() {
SelectedEmployee = new Employee()
{
Name = "John",
ID = "381",
DOB = new DateTime(1995, 12, 24)
};
}
}
}

```

XML

```

<syncfusion:PropertyGrid DescriptionPanelVisibility="Visible"
DescriptionPanelHeight="50"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

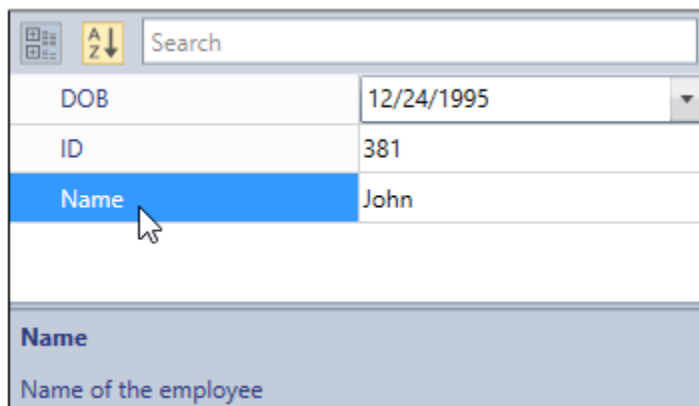
```

C#

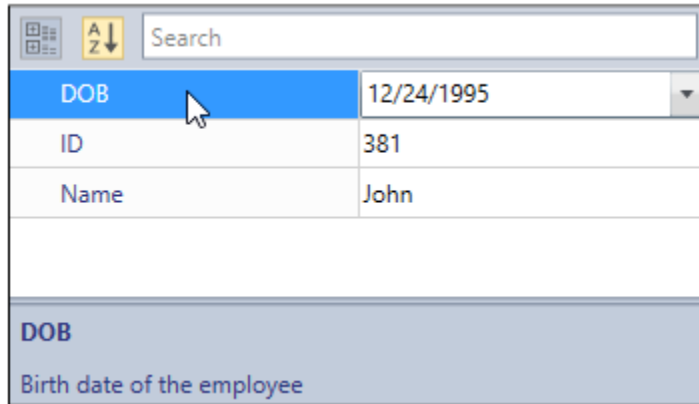
```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.DescriptionPanelVisibility = Visibility.Visible;
propertyGrid1.DescriptionPanelHeight = new GridLength(50);

```



Here, the description about the **Name** property is displayed by using the **Display** attribute's **Description** field.



Here, the description about the `DOB` property is displayed by using the `Description` attribute.

Note: If you use both the `Description` attribute and `Description` field of the `Display` attribute, the `Description` attribute will have higher priority.

Note: The `Display` attribute is contained in the [System.ComponentModel.Annotations.dll](#) assembly.

Note: [View Sample in GitHub](#)

Change Property Description at runtime

We can set the property description without using the attributes and can change the property description at runtime by handling the [AutoGeneratingPropertyGridItem](#) event with [AutoGeneratingPropertyGridItemEventArgs.Description](#) property.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    public int Experience { get; set; }
}
public class ViewModel {
    public object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24),
            Experience = 5;
        };
    }
}
```

XML

```
<syncfusion:PropertyGrid
    AutoGeneratingPropertyGridItem="PropertyGrid1_AutoGeneratingPropertyGridItem"
    >
```

```

DescriptionPanelVisibility="Visible"
DescriptionPanelHeight="50"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.AutoGeneratingPropertyGridItem +=
PropertyGrid1_AutoGeneratingPropertyGridItem;
propertyGrid1.DescriptionPanelVisibility = Visibility.Visible;
propertyGrid1.DescriptionPanelHeight = new GridLength(50);

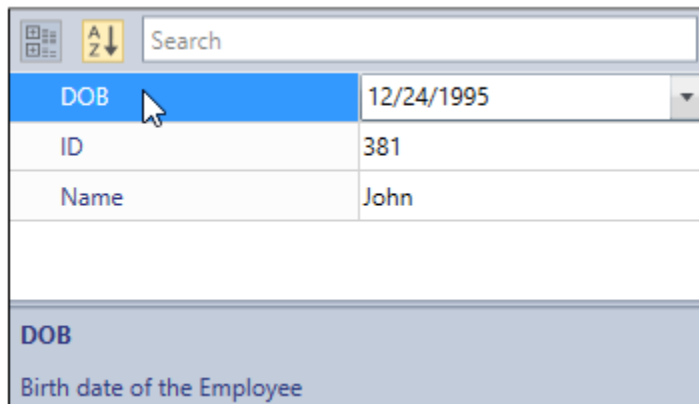
```

C#

```

private void PropertyGrid1_AutoGeneratingPropertyGridItem(object sender,
AutoGeneratingPropertyGridItemEventArgs e) {
    //Description about the DOB properties is added.
    if (e.DisplayName == "DOB") {
        e.Description = "Birth date of the Employee";
    }
}

```



Here, the DOB property description is added by the `AutoGeneratingPropertyGridItemEventArgs.Description` property of the `AutoGeneratingPropertyGridItem` event, not by any attributes.

Note: [View Sample in GitHub](#)

Setting description panel height

By default, the height of the description panel is automatically adjusted, based on the description text length of the property items. If you want to set description panel height manually, use the [DescriptionPanelHeight](#) property. The default value of `DescriptionPanelHeight` property is `Auto`.

Note: If length of the description text exceeds the `DescriptionPanelHeight` when manually setting the value to `DescriptionPanelHeight`, particular text is trimmed. you can see the full description text using the tooltip.

C#

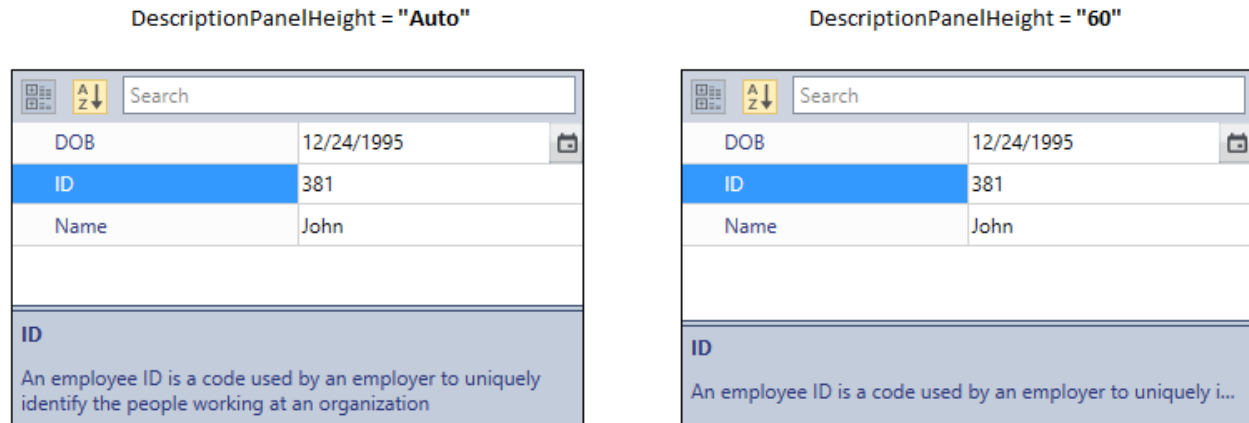
```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
    [Display(Description = "Name of the employee")]
    public string Name { get; set; }
    [Description("An employee ID is a code used by an employer to uniquely
    identify the people working at an organization")]
    public string ID { get; set; }
    public DateTime DOB { get; set; }
}
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24)
        };
    }
}
```

XML

```
<syncfusion:PropertyGrid DescriptionPanelVisibility="Visible"
DescriptionPanelHeight="60"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.DescriptionPanelVisibility = Visibility.Visible;
propertyGrid1.DescriptionPanelHeight = new GridLength(60);
```



Custom UI for description panel

You can customize the UI of description panel by using the [DescriptionTemplate](#) property. The `DataContext` of property is `PropertyItem`.

C#

```
public class Employee
{
    [Description("Name of the employee")]
    public string Name { get; set; }
    [Description("ID of the employee")]
    public string ID { get; set; }
    [Description("Birth date of the employee")]
    public DateTime DOB { get; set; }
    [Description("Age of the employee")]
    public int Age { get; set; }
    public Employee()
    {
        Name = "John";
        ID = "381";
        DOB = new DateTime(1995, 12, 24);
        Age = 26;
    }
}
```

XML

```
<syncfusion:PropertyGrid DescriptionPanelVisibility="Visible"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DescriptionTemplate>
<DataTemplate>
<StackPanel>
<TextBlock
Text="{Binding Name}"
FontSize="16"
Foreground="Red"
TextWrapping="Wrap"/>
<TextBlock
Text="{Binding Description}"
FontSize="14"
Foreground="Green"
TextWrapping="Wrap"/>
</TextBlock>
</StackPanel>
</DataTemplate>
</syncfusion:PropertyGrid.DescriptionTemplate>
</syncfusion:PropertyGrid>
```

```

</StackPanel>
</DataTemplate>
</syncfusion:PropertyGrid.DescriptionTemplate>
<syncfusion:PropertyGrid.SelectedObject>
<local:Employee></local:Employee>
</syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>

```

Age	26
DOB	12/24/1995
ID	381
Name	John

DOB
Birth date of the employee

Note: [View Sample in GitHub](#)

Different custom UI for specific property's description panel

You can customize the different UI for specific property's description panel by handling the [AutoGeneratingPropertyGridItem](#) event and using the [AutoGeneratingPropertyGridItemEventArgs.DescriptionTemplate](#) property. You can also use the [DescriptionTemplateSelector](#) property to customize the UI for specific property's description panel.

C#

```

public class Employee
{
    [Description("Name of the employee")]
    public string Name { get; set; }
    [Description("ID of the employee")]
    public string ID { get; set; }
    [Description("Birth date of the employee")]
    public DateTime DOB { get; set; }
    [Description("Age of the employee")]
    public int Age { get; set; }
    public Employee()
    {
        Name = "John";
        ID = "381";
        DOB = new DateTime(1995, 12, 24);
        Age = 26;
    }
}

```

XML

```

<Grid x:Name="grid">
<Grid.Resources>

```



```

<DataTemplate x:Key="template1">
  <StackPanel>
    <TextBlock
      Text="{Binding Name}"
      FontSize="16"
      Foreground="Red"
      TextWrapping="Wrap"/>
    <TextBlock
      Text="{Binding Description}"
      FontSize="14"
      Foreground="Green"
      TextWrapping="Wrap"/>
    </StackPanel>
  </DataTemplate>
<DataTemplate x:Key="template2">
  <StackPanel>
    <TextBlock
      Text="{Binding Name}"
      FontSize="16"
      Foreground="BlueViolet"
      TextWrapping="Wrap"/>
    <TextBlock
      Text="{Binding Description}"
      FontSize="14"
      Foreground="DarkCyan"
      TextWrapping="Wrap"/>
    </StackPanel>
  </DataTemplate>
</Grid.Resources>
<syncfusion:PropertyGrid
  AutoGeneratingPropertyGridItem="propertyGrid1_AutoGeneratingPropertyGridItem"
  DescriptionPanelVisibility="Visible"
  x:Name="propertyGrid1">
  <syncfusion:PropertyGrid.SelectedObject>
    <local:Employee></local:Employee>
  </syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>
</Grid>

```

You can handle the event as follows,

C#

```

private void propertyGrid1_AutoGeneratingPropertyGridItem(object sender,
  AutoGeneratingPropertyGridItemEventArgs e)
{
  if (e.DisplayName == "Name" || e.DisplayName == "DOB")
  {
    e.DescriptionTemplate = grid.TryFindResource("template1") as DataTemplate;
  }
  else if (e.DisplayName == "ID" || e.DisplayName == "Age")
  {
    e.DescriptionTemplate = grid.TryFindResource("template2") as DataTemplate;
  }
}

```

Search	
Age	26
DOB	12/24/1995
ID	381
Name	John
DOB Birth date of the employee	

Search	
Age	26
DOB	12/24/1995
ID	381
Name	John
ID ID of the employee	

Note: [View Sample in GitHub](#)

Note: If you use `DescriptionTemplate` and `AutoGeneratingPropertyGridItemEventArgs.DescriptionTemplate` properties to customize the description panel. Then, `AutoGeneratingPropertyGridItemEventArgs.DescriptionTemplate` have higher priority.

Custom Property Definition in WPF PropertyGrid

In this section, let us see how to configure properties manually in (xaml or C#) instead of event or attributes.

Define PropertyItem manually

By default, property items of `PropertyGrid.SelectedObject` are automatically generated in the `PropertyGrid` control by using the `AutoGeneratingPropertyGridItem` event. Now, you can restrict the auto generated items and manually define a property items through the XAML by using the `PropertyGridItem`.

Adding defined PropertyGridItems into PropertyGrid

If you want to load the manually defined property items into the `PropertyGrid`, add that `Items` collection property. You can enable it only by setting the `AutoGenerateItems` property value as `false`. The default value of `AutoGenerateItems` property is `true`.

Note: When `AutoGenerateItems` is `false`, `AutoGeneratingPropertyGridItem` event will not be triggered.

In the following example, `AutoGeneratingPropertyGridItem` event not triggered by disabling the `AutoGenerateItems` and items which are manually added in the `Items` collection only loaded in the `PropertyGrid`.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public enum Gender {
    Male,
    Female
}
public class Employee {
    public string Name { get; set; }
}
```

```

public string ID { get; set; }
public DateTime DOB { get; set; }
public Gender Gender { get; set; }
}
public class ViewModel {
public Object SelectedEmployee { get; set; }
public ViewModel() {
SelectedEmployee = new Employee()
{
Name = "John",
ID = "381",
DOB = new DateTime(1995, 12, 24),
Gender = Gender.Male
};
}
}
}

```

XML

```

<syncfusion:PropertyGrid AutoGenerateItems="False"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
<syncfusion:PropertyGrid.Items>
<syncfusion:PropertyGridItem PropertyName="Name"/>
<syncfusion:PropertyGridItem PropertyName="DOB"/>
</syncfusion:PropertyGrid.Items>
</syncfusion:PropertyGrid>

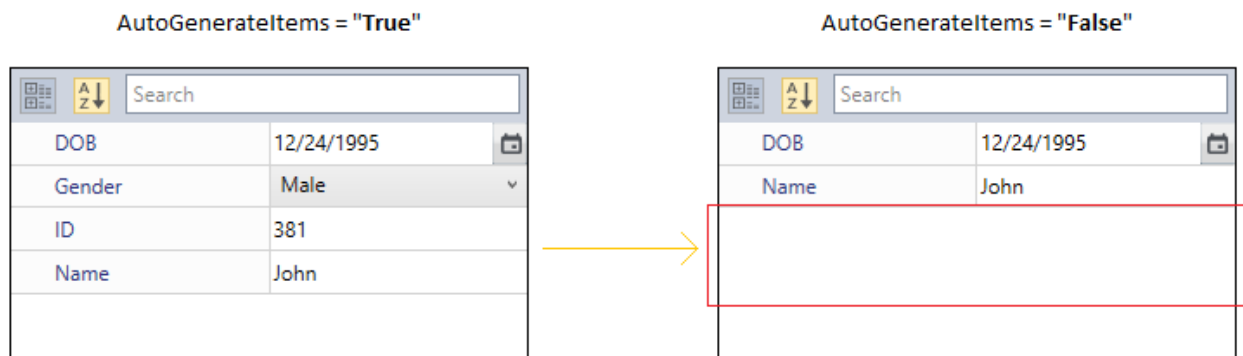
```

C#

```

propertyGrid1.AutoGenerateItems = false;
propertyGrid1.Items.Add(new PropertyGridItem() { PropertyName = "Name" });
propertyGrid1.Items.Add(new PropertyGridItem() { PropertyName = "DOB" });

```



Note: [View Sample in GitHub](#)

Add or remove PropertyItem at runtime

You can manually add or remove the property item at runtime by adding or removing that item from the [Items](#) collection property. You can also clear all the property items from the `PropertyGrid`.

C#

```

using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public enum Gender {
    Male,
    Female
}
public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    public Gender Gender { get; set; }
}
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24),
            Gender = Gender.Male
        };
    }
}

```

XML

```

<syncfusion:PropertyGrid AutoGenerateItems="False"
    SelectedObject="{Binding SelectedEmployee}"
    x:Name="propertyGrid1">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
    <syncfusion:PropertyGrid.Items>
    <syncfusion:PropertyGridItem PropertyName="Name"/>
    <syncfusion:PropertyGridItem PropertyName="DOB"/>
    </syncfusion:PropertyGrid.Items>
    </syncfusion:PropertyGrid>
    <StackPanel>
    <Button Name="addItem"
        Click="AddItems_Click"
        Content="Add Items"></Button>
    <Button x:Name="removeItems"
        Click="RemoveItems_Click"
        Content="Remove Items"></Button>
    <Button Name="clearItems"
        Click="ClearItems_Click"
        Content="Clear Items"></Button>
    </StackPanel>

```

C#

```

propertyGrid1.AutoGenerateItems = false;

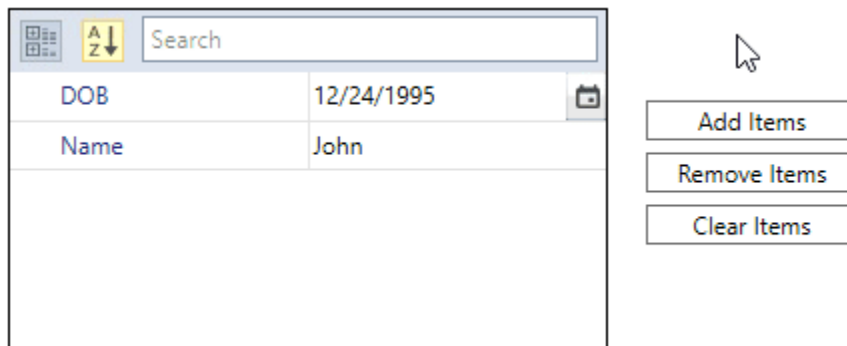
```

```
propertyGrid1.Items.Add(new PropertyGridItem()
{
    PropertyName = "Name"
});
propertyGrid1.Items.Add(new PropertyGridItem()
{
    PropertyName = "DOB"
});
addItem.Click += AddItems_Click;
removeItems.Click += RemoveItems_Click;
clearItems.Click += ClearItems_Click;
```

You can dynamically add the property item as follows,

C#

```
private void AddItem_Click(object sender, RoutedEventArgs e) {
    propertyGrid1.Items.Add(new PropertyGridItem()
    {
        PropertyName = "Gender"
    });
}
private void RemoveItems_Click(object sender, RoutedEventArgs e) {
    propertyGrid1.Items.RemoveAt(1);
}
private void ClearItems_Click(object sender, RoutedEventArgs e) {
    propertyGrid1.Items.Clear();
}
```



Custom definition of PropertyItem

You can customize the display name, description, category, nested mode, readonly state and value editor for any property items.

- [PropertyGridItem.DisplayName](#) - To customize the display name for the property items.
- [PropertyGridItem.Description](#) - To customize the description for the property items.
- [PropertyGridItem.CategoryName](#) - To customize the category for the property items.
- [PropertyGridItem.IsReadOnly](#) - To customize the readonly state for the property items.
- [PropertyGridItem.Editor](#) - To customize the own value editor for the property items.
- [PropertyGridItem.NestedPropertyDisplayMode](#) - To customize the nested mode of the property items.
- [PropertyGridItem.Visibility](#) - To customize the visibility for the property items.

Note: [View Sample in GitHub](#)

Manually add own value editor and categorize the properties

If you want to add own value editor and categorize the property items manually, use the `PropertyGridItem.Editor` and `PropertyGridItem.CategoryName` properties to the required property items.

Note: You can refer [Create Custom Value Editor](#) page to know more about how to create a own value editors.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class EmailEditor : MaskEditor {
    public EmailEditor() {
        Mask = @"[A-Za-z0-9._%~]+@[A-Za-z0-9]+\.[A-Za-z]{2,3}";
    }
}
public enum Gender {
    Male,
    Female
}
public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    public Gender Gender { get; set; }
    public string EmailID { get; set; }
}
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24),
            Gender = Gender.Male,
            EmailID = "john@123.c"
        };
    }
}
```

XML

```
<syncfusion:PropertyGrid AutoGenerateItems="False"
    EnableGrouping="True"
    SelectedObject="{Binding SelectedEmployee}"
    x:Name="propertyGrid1">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
    <syncfusion:PropertyGrid.Items>
    <syncfusion:PropertyGridItem PropertyName="Name"
```

```

CategoryName="Basic Info"/>
<syncfusion:PropertyGridItem PropertyName="ID"
CategoryName="Basic Info"/>
<syncfusion:PropertyGridItem PropertyName="DOB"
CategoryName="Basic Info"/>
<syncfusion:PropertyGridItem PropertyName="Gender"
CategoryName="Additional Info"/>
<syncfusion:PropertyGridItem PropertyName="EmailID"
CategoryName="Additional Info">
<!--Adding own value editor for the EmailID property-->
<syncfusion:PropertyGridItem.Editor>
<local:EmailEditor/>
</syncfusion:PropertyGridItem.Editor>
</syncfusion:PropertyGridItem>
</syncfusion:PropertyGrid.Items>
</syncfusion:PropertyGrid>

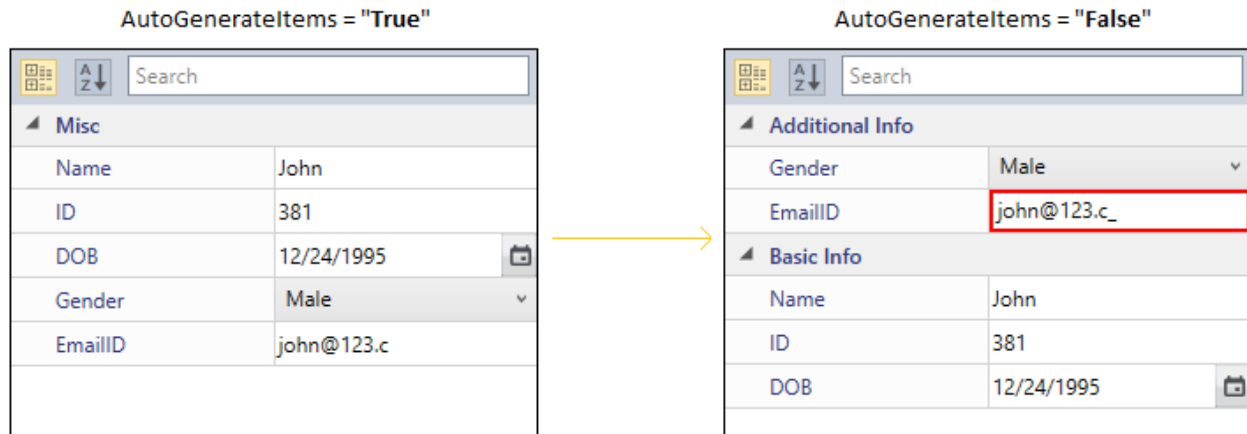
```

C#

```

propertyGrid1.AutoGenerateItems = false;
propertyGrid1.EnableGrouping = true;
propertyGrid1.Items.Add(new PropertyGridItem()
{
    PropertyName = "Name",
    CategoryName = "Basic Info"
});
propertyGrid1.Items.Add(new PropertyGridItem()
{
    PropertyName = "ID",
    CategoryName = "Birth date of the employee"
});
propertyGrid1.Items.Add(new PropertyGridItem()
{
    PropertyName = "DOB",
    CategoryName = "Basic Info"
});
propertyGrid1.Items.Add(new PropertyGridItem()
{
    PropertyName = "Gender",
    CategoryName = "Additional Info"
});
propertyGrid1.Items.Add(new PropertyGridItem()
{
    PropertyName = "EmailID",
    CategoryName = "Additional Info"
});

```



Here, the masked textbox with email-id mask is assigned as a value editor for the `EmailID` property and properties are categorized under `Basic Info` and `Additional Info`.

Note: [View Sample in GitHub](#)

Manually define nested properties for `PropertyItem`

If you want to manually explore only particular nested properties of any property item, add that particular nested properties into the respective property item's `PropertyGridItem.Items` collection. You must enable nested property mode by using `PropertyGridItem.PropertyExpandMode` property value as `NestedMode` to explore the nested properties. The default value of `PropertyGridItem.PropertyExpandMode` property is `null`.

C#

```
public class Bank {
    public string BankName { get; set; }
    public int CustomerID { get; set; }
    public long AccountNumber { get; set; }
    public override string ToString() {
        return BankName;
    }
}

public class Employee {
    public string ID { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public Bank Bank { get; set; }
    public DateTime DOB { get; set; }
}

public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            DOB = new DateTime(1995, 01, 31),
            Name = "Johnson",
            ID = "895",
            Age = 25,
            Bank = new Bank()
            {
                AccountNumber = 123456789,
            }
        }
    }
}
```



```

CustomerID = 356,
BankName = "ABC Bank"
},
};
}
}

```

XML

```

<syncfusion:PropertyGrid AutoGenerateItems="False"
PropertyExpandMode="NestedMode"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
<syncfusion:PropertyGrid.Items>
<syncfusion:PropertyGridItem PropertyName="Name"/>
<syncfusion:PropertyGridItem PropertyName="ID"/>
<syncfusion:PropertyGridItem PropertyName="DOB"
NestedPropertyDisplayMode="Show">
<syncfusion:PropertyGridItem.Items>
<syncfusion:PropertyGridItem PropertyName="Day"/>
<syncfusion:PropertyGridItem PropertyName="Month"/>
<syncfusion:PropertyGridItem PropertyName="Year"/>
</syncfusion:PropertyGridItem.Items>
</syncfusion:PropertyGridItem>
</syncfusion:PropertyGrid.Items>
</syncfusion:PropertyGrid>

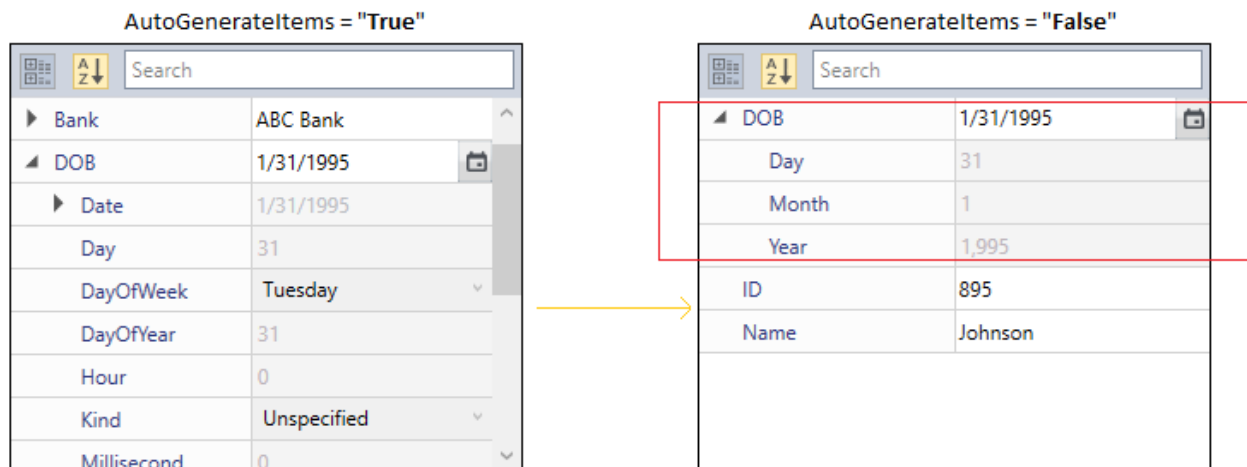
```

C#

```

propertyGrid1.AutoGenerateItems = false;
propertyGrid1.PropertyExpandMode = PropertyExpandModes.NestedMode;
propertyGrid1.Items.Add(new PropertyGridItem()
{
PropertyName = "Name"
});
propertyGrid1.Items.Add(new PropertyGridItem()
{
PropertyName = "ID"
});
ObservableCollection<PropertyGridItem> DOB_Items = new
ObservableCollection<PropertyGridItem>();
DOB_Items.Add(new PropertyGridItem() { PropertyName = "Day" });
DOB_Items.Add(new PropertyGridItem() { PropertyName = "Month" });
DOB_Items.Add(new PropertyGridItem() { PropertyName = "Year" });
propertyGrid1.Items.Add(new PropertyGridItem()
{
PropertyName = "DOB",
Items = DOB_Items
});

```



Note: [View Sample in GitHub](#)

Manually customize the UI of description panel

You can customize the different UI for specific property's description panel by using the [PropertyGridItem.DescriptionTemplate](#) property. The `DataContext` of [PropertyGridItem.DescriptionTemplate](#) property is `PropertyGridItem`.

C#

```
using System;
using System.ComponentModel;
public enum Gender {
    Male,
    Female
}
public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    public Gender Gender { get; set; }
    public string EmailID { get; set; }
}
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24),
            Gender = Gender.Male,
            EmailID = "john@123.c"
        };
    }
}
```

XML

```
<Grid x:Name="grid">
<Grid.Resources>
```

```

<DataTemplate x:Key="template1">
  <StackPanel>
    <TextBlock
      Text="{Binding Name}"
      FontSize="16"
      Foreground="Red"
      TextWrapping="Wrap"/>
    <TextBlock
      Text="{Binding Description}"
      FontSize="14"
      Foreground="Green"
      TextWrapping="Wrap"/>
    </StackPanel>
  </DataTemplate>
<DataTemplate x:Key="template2">
  <StackPanel>
    <TextBlock
      Text="{Binding Name}"
      FontSize="16"
      Foreground="BlueViolet"
      TextWrapping="Wrap"/>
    <TextBlock
      Text="{Binding Description}"
      FontSize="14"
      Foreground="DarkCyan"
      TextWrapping="Wrap"/>
    </StackPanel>
  </DataTemplate>
</Grid.Resources>
<syncfusion:PropertyGrid
  DescriptionPanelVisibility="Visible"
  AutoGenerateItems="False"
  x:Name="propertyGrid1">
  <syncfusion:PropertyGrid.SelectedObject>
    <local:Employee></local:Employee>
  </syncfusion:PropertyGrid.SelectedObject>
  <syncfusion:PropertyGrid.Items>
    <syncfusion:PropertyGridItem PropertyName="Name"
      Description="Name of the Employee"
      DescriptionTemplate="{StaticResource template1}" >
    </syncfusion:PropertyGridItem>
    <syncfusion:PropertyGridItem PropertyName="ID"
      Description="ID of the Employee"
      DescriptionTemplate="{StaticResource template2}"/>
    <syncfusion:PropertyGridItem PropertyName="DOB"
      Description="Date o Birth of the Employee"/>
  </syncfusion:PropertyGrid.Items>
</syncfusion:PropertyGrid>
</Grid>

```

DOB	12/24/1995
ID	381
Name	John
Name Name of the Employee	

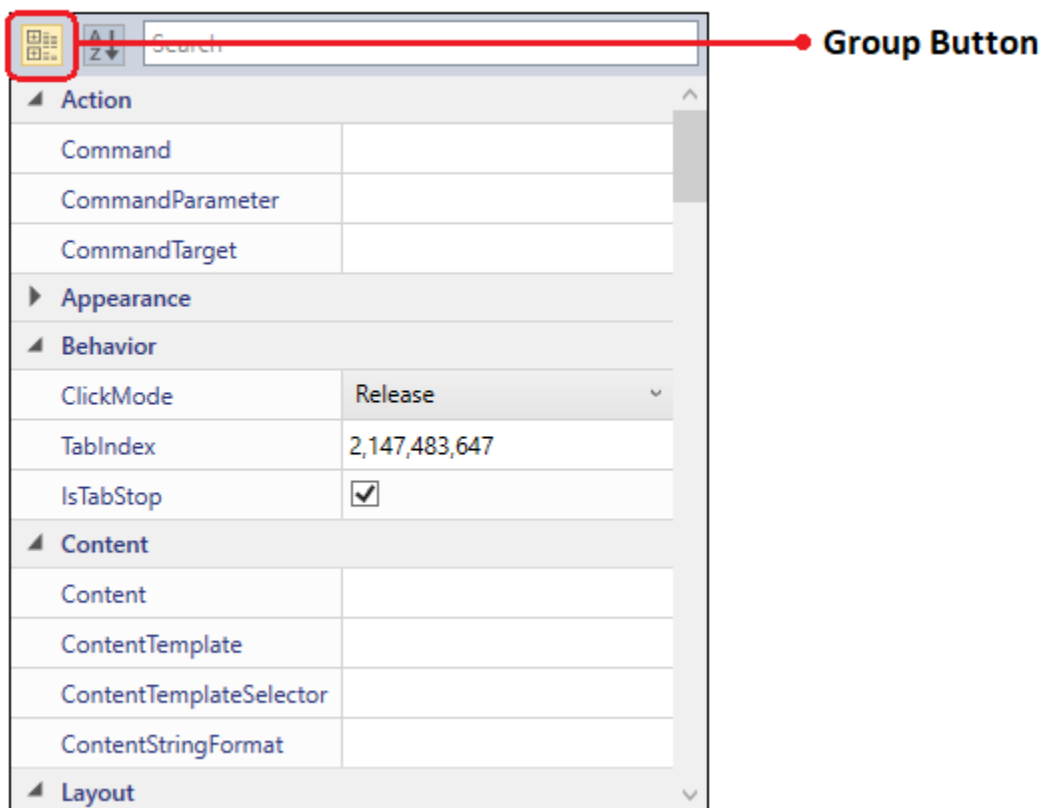
DOB	12/24/1995
ID	381
Name	John
ID ID of the Employee	

Note: [View Sample in GitHub](#)

Grouping in WPF PropertyGrid

We can combine the properties and club them into expandable groups according to our needs by **Category** attribute and **Display.GroupName** field. By default, the grouped properties are displayed in sorted view. If we want to display the property in grouped view, we can set the [EnableGrouping](#) property to **true**.

Properties in Group Mode



Grouping using attributes

Properties in the [PropertyGrid](#) will be grouped based on the name specified in the [Category](#) attribute and **GroupName** field of the **Display** attribute. If the property item doesn't have any category name, that property will be grouped under **Misc** category.

C#

```

using System;
using System.ComponentModel;
public class Employee {
    [Category("Basic Info")]
    public string ID { get; set; }
    [Display(GroupName = "Basic Info")]
    public string Name { get; set; }
    [Display(GroupName = "Additional Info")]
    public int Experience { get; set; }
    [Category()]
    public int Age { get; set; }
    public DateTime DOB { get; set; }
}

public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            DOB = new DateTime(1995, 01, 31),
            Name = "Johnson",
            Experience = 5,
            ID = "895",
            Age = 25,
        };
    }
}

```

XML

```

<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
    EnableGrouping="True"
    x:Name="propertyGrid1">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.EnableGrouping = true;

```

Additional Info	
Experience	5
Basic Info	
ID	895
Name	Johnson
Misc	
Age	25
DOB	1/31/1995

Here, ID, Name properties are grouped under the 'Basic Info' category and Experience property is grouped under 'Additional Info' category by using Category attribute or Display.GroupName field.

The Age and DOB properties do not contain any group name, so they are grouped under the Misc category.

Note: If we use both the Category attribute and GroupName field of the Display attribute, the Category attribute will have higher priority.

Grouping the Properties at runtime

We can group the properties in the PropertyGrid without using the attributes at runtime by handling the [AutoGeneratingPropertyGridItem](#) event with [AutoGeneratingPropertyGridItemEventArgs.Category](#) property. Based on the value of Category property, the properties are grouped.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    public int Experience { get; set; }
}
public class ViewModel {
    public object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24),
            Experience = 5;
        };
    }
}
```

```
}
}
```

XML

```
<syncfusion:PropertyGrid
AutoGeneratingPropertyGridItem="PropertyGrid1_AutoGeneratingPropertyGridItem"
EnableGrouping = "True"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.AutoGeneratingPropertyGridItem +=
PropertyGrid1_AutoGeneratingPropertyGridItem;
propertyGrid1.EnableGrouping= true;
```

C#

```
private void PropertyGrid1_AutoGeneratingPropertyGridItem(object sender,
AutoGeneratingPropertyGridItemEventArgs e) {
//Experience and DOB properties grouped under 'Additional Info' category.
if (e.DisplayName == "Experience") {
e.Category = "Additional Info";
}
else if (e.DisplayName == "DOB") {
e.Category = "Additional Info";
}
//Name and ID properties grouped under 'Basic Info' category.
else if (e.DisplayName == "Name") {
e.Category = "Basic Info";
}
else if (e.DisplayName == "ID") {
e.Category = "Basic Info";
}
}
```

Additional Info	
DOB	12/24/1995
Experience	0

Basic Info	
Name	John
ID	381

Here, the **DOB** and **Experience** properties are grouped under the **Additional Info** category and **Name** and **ID** properties are grouped under the **Basic Info** category

Click [here](#) to download the sample that showcases the property grouping support using `AutoGeneratingPropertyGridItem` event.

Expand or Collapse Category group

We can expand or collapse the grouped properties programmatically by using [ExpandCategory](#) and [CollapseCategory](#) methods in the `PropertyGrid`. These methods will accept group name as argument.

Expand Category group

`ExpandCategory` method will expand the specified category if it is in collapsed view.

C#

```
//Expand the Identity category group
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid1.EnableGrouping= true;
propertyGrid1.ExpandCategory("Identity");
```

Expand Category

Address	
Location	America

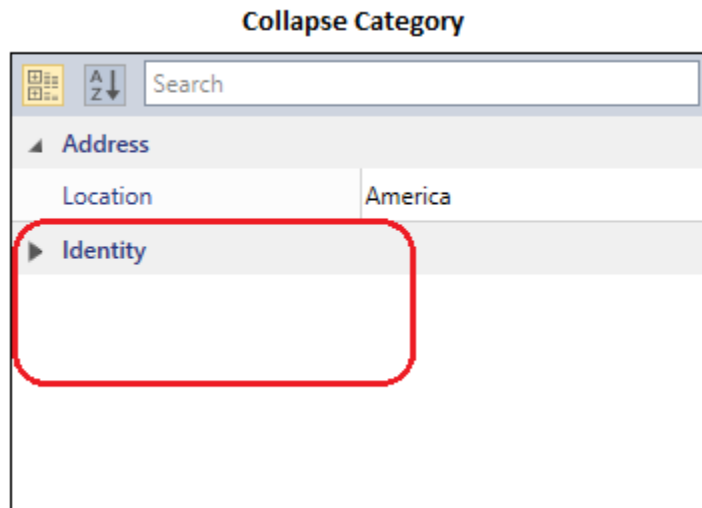
Identity	
Name	John
ID	SF001

Collapse Category group

`CollapseCategory` method will collapse the specified category if it is in expand view.

C#

```
//Collapse the Identity category group
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid1.EnableGrouping= true;
propertyGrid1.CollapseCategory("Identity");
```

**Show or Hide the Group Button**

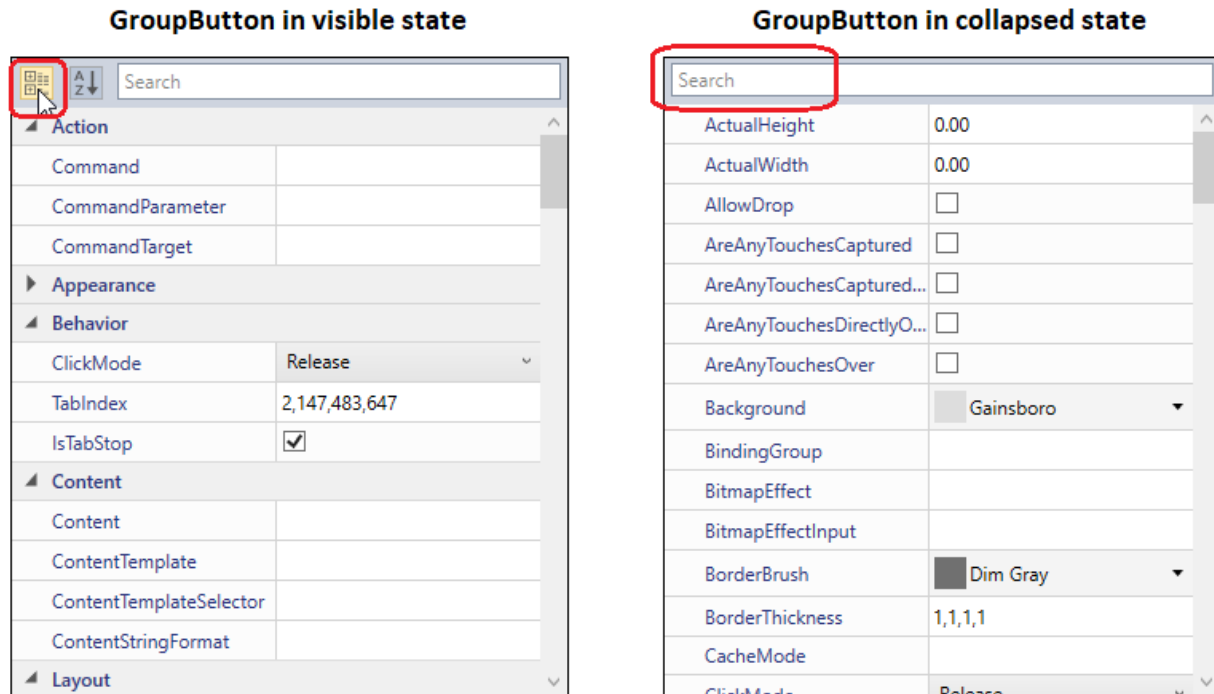
We can change the view of the properties from sorted view to grouped view by the **GroupButton**. We can show or hide the group button by using the **ButtonPanelVisibility** property. If we want to hide the **GroupButton**, set the **ButtonPanelVisibility** property as **Collapsed**. The Default value of the **ButtonPanelVisibility** property is **Visible**.

XML

```
<syncfusion:PropertyGrid x:Name="propertyGrid1" Width="350" Height="400"
ButtonPanelVisibility="Collapsed">
<syncfusion:PropertyGrid.SelectedObject>
<Button></Button>
</syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.Width = 350;
propertyGrid.Height = 400;propertyGrid.SelectedObject = new Button();
propertyGrid1.ButtonPanelVisibility = Visibility.Collapsed;
```

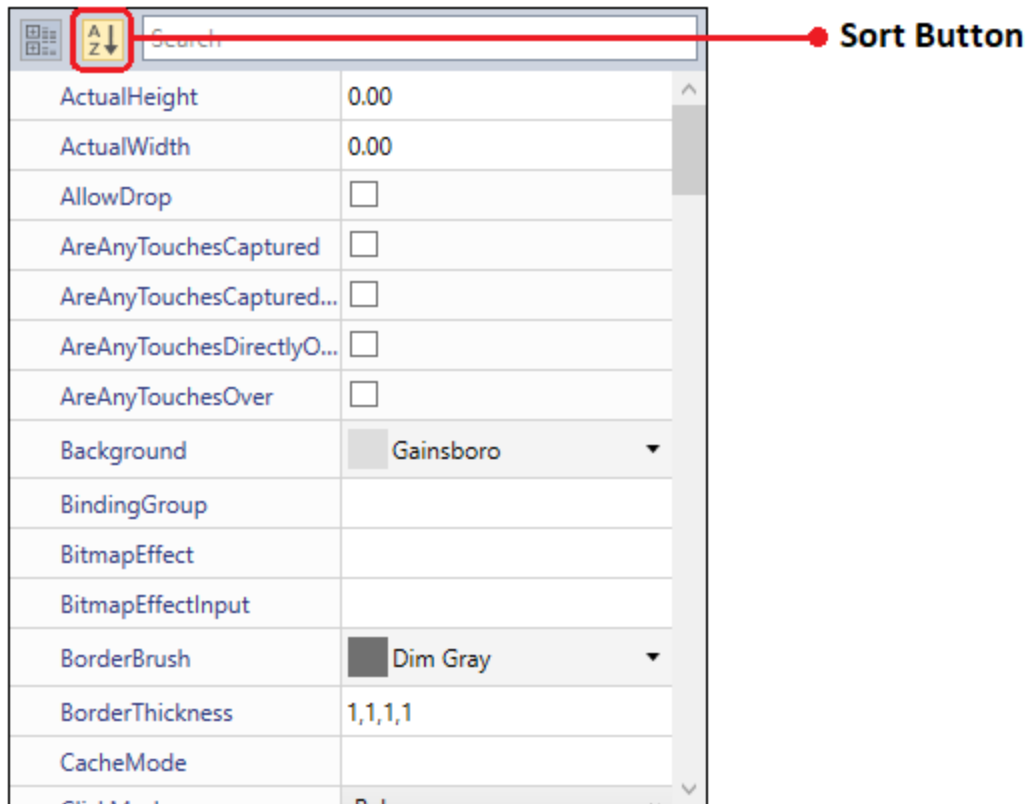


Click [here](#) to download the sample that showcases the property grouping support.

Sorting in WPF PropertyGrid

You can sort the properties according to your needs. You can change the sorting order of the properties by [SortDirection](#) property. Values of [SortDirection](#) property is [Ascending](#), [Descending](#) and [Null](#). Based on the [SortDirection](#) property, the property item's nested properties also sorted. The Default value of the [SortDirection](#) property is [Ascending](#) order.

Properties in Sorted order



Sorting the Properties

Properties in the [PropertyGrid](#) are sorted by using the [SortDirection](#) property. They sorted based only on the name of the property, not by the display name of the property. If the properties are in the grouped view, then the groups are sorted based on the group name either [Ascending](#) or [Descending](#) order.

C#

```
using System;
using System.ComponentModel;
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee();
    }
}
public class Employee
{
    [Category("Identity")]
    public string Name { get; set; }
    [Category("Contact Details")]
    public string Email { get; set; }
    [Category("Account Details")]
    public string Bank { get; set; }
    [Category("Identity")]
```

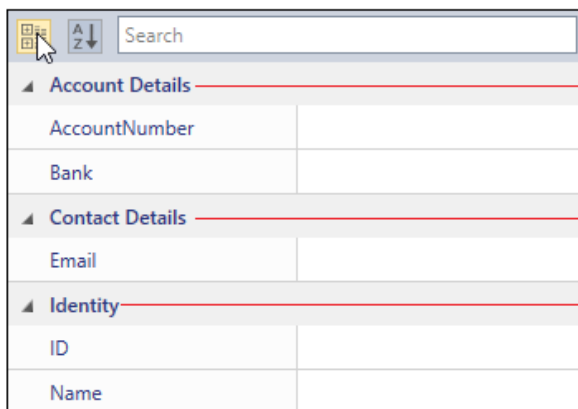
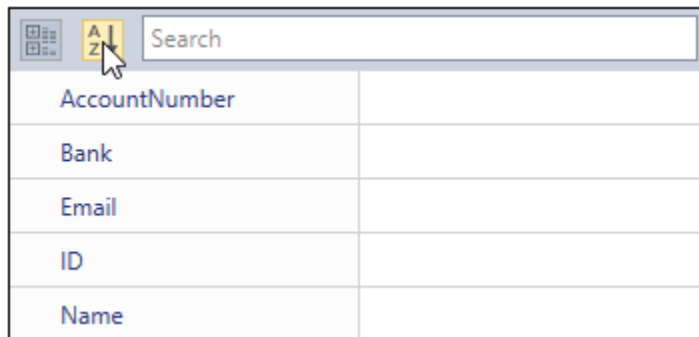
```
public string ID { get; set; }
[Category("Account Details")]
public string AccountNumber { get; set; }
}
```

XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
SortDirection="Ascending"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

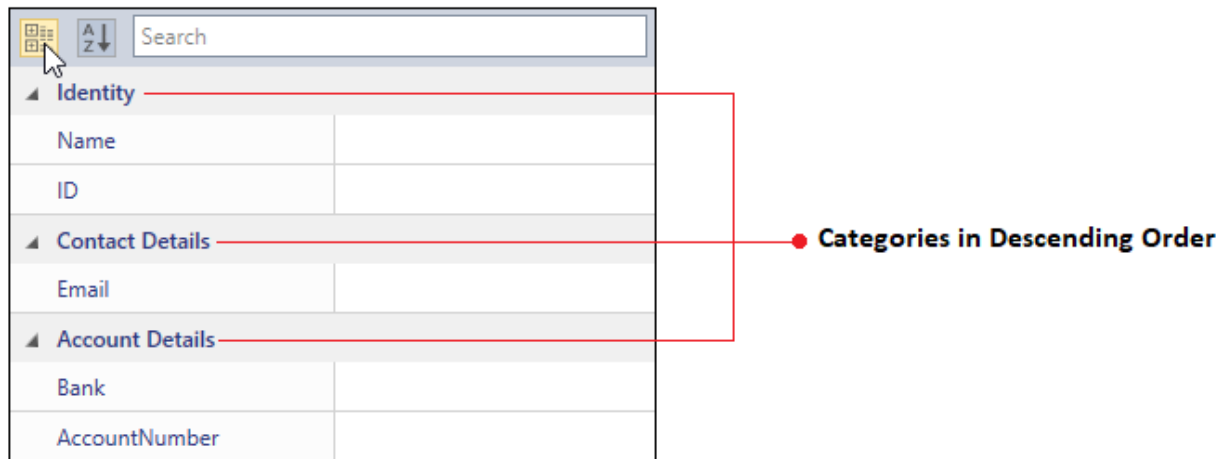
C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.SortDirection = ListSortDirection.Ascending;
```

SortDirection= Ascending

● Categories in Ascending order

SortDirection= Descending



Disable the Sorting

We can disable the sorting by setting the `SortDirection` property as `null`. When sorting is disabled, the properties are arranged based on the value of the `Order` attributes or on the order they added into the [SelectedObject](#).

C#

```
using System;
using System.ComponentModel;
public class Employee {
    [Category("Identity")]
    public String Gender { get; set; }
    [Category("Address")]
    public String Country { get; set; }
    [Category("Contact Details")]
    public string Email { get; set; }
    [Category("Identity")]
    public string FirstName { get; set; }
    public string Designation { get; set; }
    [Category("Identity")]
    public string LastName { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    [Category("Contact Details")]
    public string Mobile { get; set; }
    public int Age { get; set; }
}
```

```

public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            FirstName = "Carl",
            LastName = "johnson",
            Age = 30,
            Country = "United States",
            Designation = "Team Lead",
            DOB = new DateTime(2000, 12, 01),
            Email = "carljanson@gta.com",
            Gender = "Male",
            ID = "SF001",
            Mobile = "1234567890"
        };
    }
}

```

XML

```

<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
SortDirection="{x:Null}" x:Name="propertyGrid1" >
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

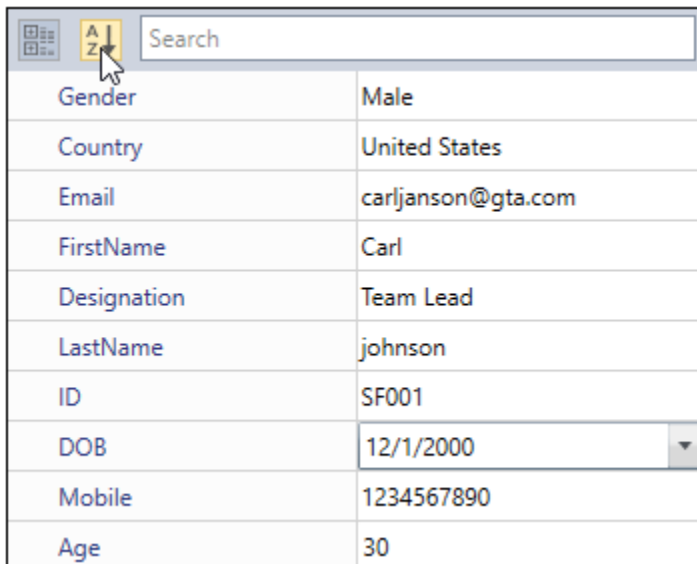
```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.SortDirection = null;

```



Property	Value
Gender	Male
Country	United States
Email	carljanson@gta.com
FirstName	Carl
Designation	Team Lead
LastName	johnson
ID	SF001
DOB	12/1/2000
Mobile	1234567890
Age	30

Here, the properties are arranged from the **Gender** property and end with **Age** property by the order in which they were added to the class.

Show or Hide the Sort Button

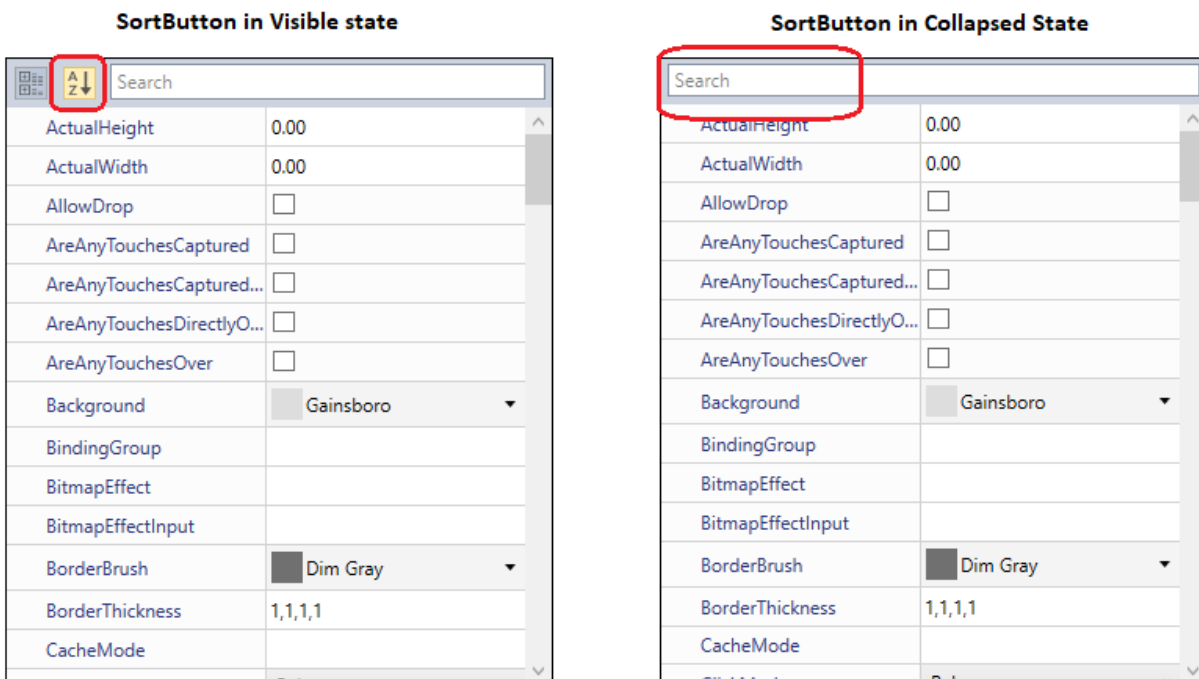
We can change the view of the properties from grouped view to sort view by the **SortButton**. We can show or hide the sort button by using the **ButtonPanelVisibility** property. If we want to hide the **SortButton**, set the **ButtonPanelVisibility** property as **Collapsed**. The Default value of the **ButtonPanelVisibility** property is **Visible**.

XML

```
<syncfusion:PropertyGrid x:Name="propertyGrid1" Width="350" Height="400"
ButtonPanelVisibility="Collapsed">
<syncfusion:PropertyGrid.SelectedObject>
<Button></Button>
</syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.Width = 350;
propertyGrid.Height = 400;
propertyGrid.SelectedObject = new Button();
propertyGrid1.ButtonPanelVisibility = Visibility.Collapsed;
```



Click [here](#) to download the sample that showcases the property sorting support using the attributes.

Ordering in WPF PropertyGrid

We can order the properties according to our needs. We can change the order of the properties by attributes and event.

Ordering using Attribute

The properties in [PropertyGrid](#) will be ordered based on the value specified in the [Order](#) field of [Display](#) attribute. If we need to change the order of the properties, we should set the [SortDirection](#) property to [null](#).

C#

```
public class Employee {
    [Display(Order = 4)]
    public String Gender { get; set; }
    [Display(Order = 7)]
    [Category("Address")]
    public String Country { get; set; }
    [Display(Order = 6)]
    [Category("Contact Details")]
    public string Email { get; set; }
    [Display(Order = 0)]
    [Category("Identity")]
    public string FirstName { get; set; }
    public string Designation { get; set; }
    [Display(Order = 1)]
    [Category("Identity")]
    public string LastName { get; set; }
    public string ID { get; set; }
    [Display(Order = 4)]
    public DateTime DOB { get; set; }
    [Display(Order = 5)]
    [Category("Contact Details")]
    public string Mobile { get; set; }
    [Display(Order = 2)]
    public int Age { get; set; }
}

public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee() {
            FirstName = "Carl",
            LastName = "johnson",
            Age = 30,
            Country = "United States",
            Designation = "Team Lead",
            DOB = new DateTime(2000, 12, 01),
            Email = "carljanson@gta.com",
            Gender = "Male",
            ID = "SF001",
            Mobile = "1234567890"
        };
    }
}
```

XML

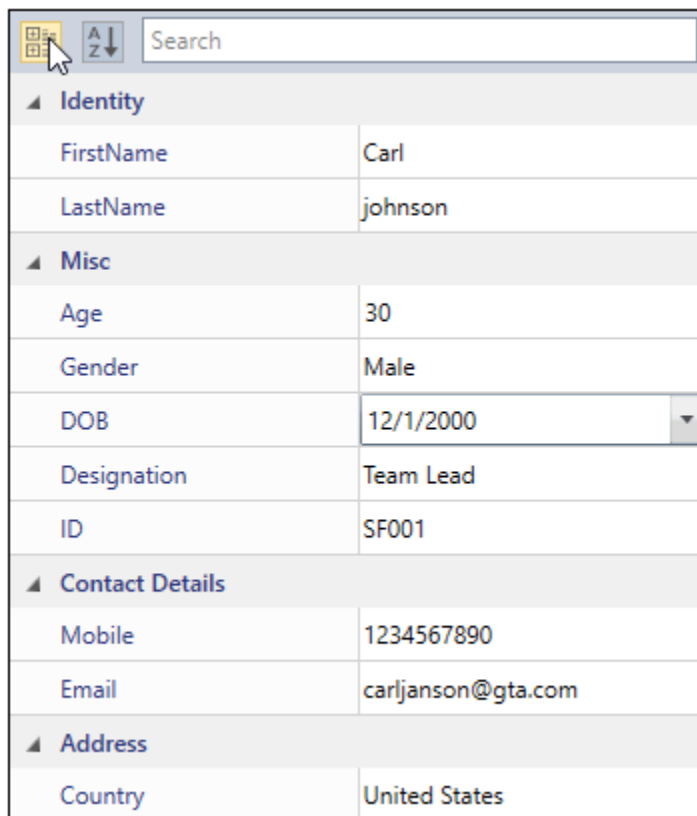

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
EnableGrouping="True"
SortDirection="{x:Null}" x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.EnableGrouping = true;
propertyGrid1.SortDirection = null;
```

Ordering in Group view

Based on the value specified in the **Order** field of **Display** attribute, the property items will be ordered and grouped. Groups are ordered according to group which contains the lower ordered property.

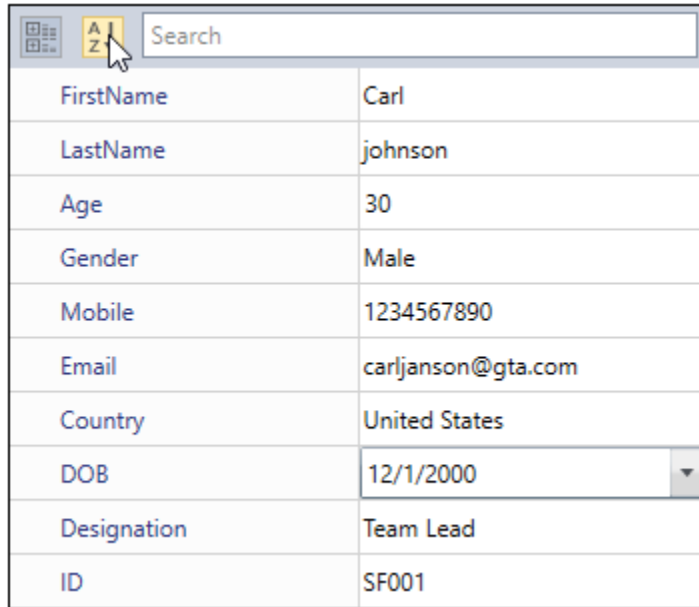


Identity	
FirstName	Carl
LastName	johnson
Misc	
Age	30
Gender	Male
DOB	12/1/2000
Designation	Team Lead
ID	SF001
Contact Details	
Mobile	1234567890
Email	carljanson@gta.com
Address	
Country	United States

Here, the **FirstName** property contains the lower order as **0** and categorized under the **Identity** category. Then the **Identity** category ordered as first. After that, **Age** property contains the lower order as **2** from another category, So **Misc** category ordered as **second** and vice versa.

Ordering with Sort view

If more than one property has same **Order** value, then the properties with distinct numbers will be added first, then the duplicate order properties will be added. Also, if any of the properties doesn't have order, that properties will be arranged at last based on the order in which they were added in the class.



FirstName	Carl
LastName	johnson
Age	30
Gender	Male
Mobile	1234567890
Email	carljanson@gta.com
Country	United States
DOB	12/1/2000
Designation	Team Lead
ID	SF001

Here, the property **DOB** and **Gender** has same order but the **Gender** property is added prior than **DOB**. So the **Gender** property will be arranged in 4th place, and the **DOB** is arranged after Country property. And the other non-specified order properties are arranged based on the order in which they were added to the Class.

Ordering based on properties defined in class

The Ordering can be performed without using any Attributes. If the **SortDirection** property is **null** and the properties have no custom order, then the properties will be ordered according to the order in which they were added to the class.

C#

```
using System;
using System.ComponentModel;
public class Employee {
    [Category("Identity")]
    public String Gender { get; set; }
    [Category("Address")]
    public String Country { get; set; }
    [Category("Contact Details")]
    public string Email { get; set; }
    [Category("Identity")]
    public string FirstName { get; set; }
    public string Designation { get; set; }
    [Category("Identity")]
    public string LastName { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    [Category("Contact Details")]
    public string Mobile { get; set; }
```

```
public int Age { get; set; }
}
public class ViewModel {
public Object SelectedEmployee { get; set; }
public ViewModel() {
SelectedEmployee = new Employee()
{
FirstName = "Carl",
LastName = "johnson",
Age = 30,
Country = "United States",
Designation = "Team Lead",
DOB = new DateTime(2000, 12, 01),
Email = "carljanson@gta.com",
Gender = "Male",
ID = "SF001",
Mobile = "1234567890"
};
}
}
```

XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
SortDirection="{x:Null}" x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.SortDirection = null;
```


Change Property order at runtime

We can set the property order without using the attributes and can change the property order at runtime by handling the [AutoGeneratingPropertyGridItem](#) event with [AutoGeneratingPropertyGridItemEventArgs.Order](#) property.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    public int Experience { get; set; }
}
public class ViewModel {
    public object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24),
            Experience = 5;
        };
    }
}
```

XML

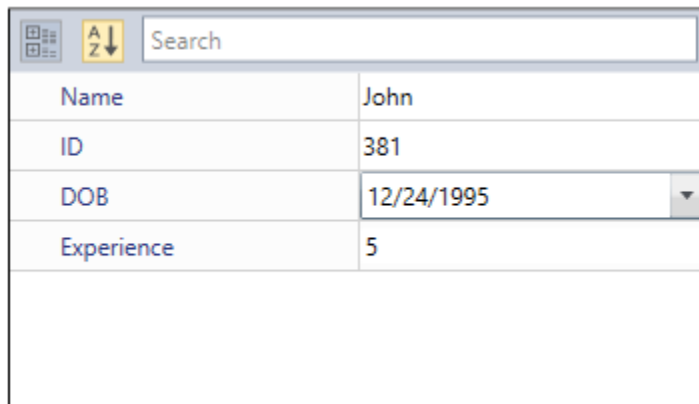
```
<syncfusion:PropertyGrid
AutoGeneratingPropertyGridItem="PropertyGrid1_AutoGeneratingPropertyGridItem"
SortDirection="{x:Null}"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.SortDirection = null;
propertyGrid1.AutoGeneratingPropertyGridItem +=
PropertyGrid1_AutoGeneratingPropertyGridItem;
```

C#

```
private void PropertyGrid1_AutoGeneratingPropertyGridItem(object sender,
AutoGeneratingPropertyGridItemEventArgs e) {
    //Experience, DOB, Name and ID properties ordered by the Order property.
    if (e.DisplayName == "Experience") {
        e.Order = 3;
    }
    else if (e.DisplayName == "DOB") {
        e.Order = 2;
    }
    else if (e.DisplayName == "Name") {
        e.Order = 0;
    }
    else if (e.DisplayName == "ID") {
        e.Order = 1;
    }
}
```



Here, the **Name** property is arranged at first and **Experience** property arranged at fourth position based on the value specified in the **AutoGeneratingPropertyGridItemArgs.Order** property.

Click [here](#) to download the sample that showcases the property **Description** support using **AutoGeneratingPropertyGridItem** event.

Virtualization in WPF PropertyGrid

By loading only items that are within viewport, UI virtualization allows **PropertyGrid** to load faster. Virtualization is enabled by default.

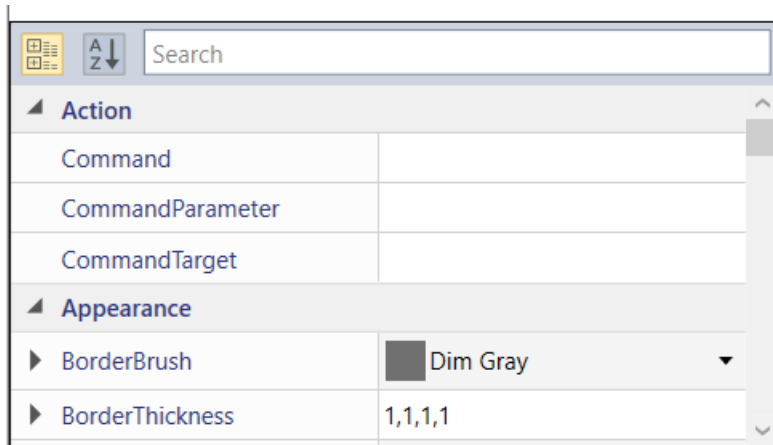
C#

```
PropertyGrid propertyGrid = new PropertyGrid();
propertyGrid.IsVirtualizing = true;
propertyGrid.EnableGrouping = true;
propertyGrid.PropertyExpandMode = PropertyExpandModes.NestedMode;
propertyGrid.SelectedObject = new Button();
```

XML

```
<syncfusion:PropertyGrid x:Name="propertyGrid" IsVirtualizing="True"
PropertyExpandMode="NestedMode" EnableGrouping="True">
<syncfusion:PropertyGrid.SelectedObject>
<Button />
```

```
</syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>
```



Note: When Virtualization is enabled, only properties that are in viewport will be in loaded state.

Filtering and Searching in WPF PropertyGrid

We can decide the properties which are need to be displayed in [PropertyGrid](#) by hiding the unwanted properties using collection and attributes. We can navigate to the particular property by using the default SearchBox.

Hide the Properties using Collection

We can hide the properties using the [HidePropertiesCollection](#) property. It is used to hide the mentioned properties which are already present in [SelectedObject](#). Properties can also be hidden at runtime using [HidePropertiesCollection](#).

C#

```
public class Employee {
    public string Name { get; set; }
    public string Email { get; set; }
    public string Bank { get; set; }
    public string ID { get; set; }
    public string AccountNumber { get; set; }
    public int Age { get; set; }
    public int Experience { get; set; }
}

public class ViewModel {
    private ObservableCollection<string> hidePropertyItems = new
    ObservableCollection<string>();
    public Object SelectedEmployee { get; set; }
    public ObservableCollection<string> HidePropertyItems
    {
        get { return hidePropertyItems; }
        set { hidePropertyItems = value; }
    }
    public ViewModel() {
        var employee = new Employee()
        {
            Email = "john@gta.com",
            AccountNumber="23456784",

```

```

Bank="ABC bank",
Name = "Johnson",
Experience=5,
ID = "895",
Age = 35,
};
HidePropertyItems.Add(nameof(employee.AccountNumber));
HidePropertyItems.Add(nameof(employee.Email));
HidePropertyItems.Add(nameof(employee.Bank));
SelectedEmployee = employee;
}
}

```

XML

```

<syncfusion:PropertyGrid HidePropertiesCollection="{Binding
HidePropertyItems}"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" Width="350" Height="200" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

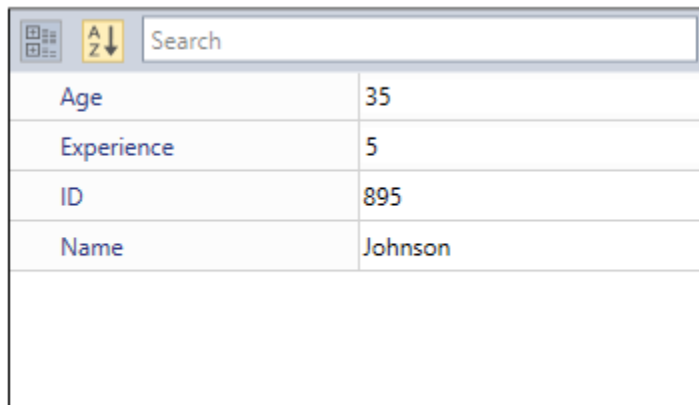
```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));

```



Age	35
Experience	5
ID	895
Name	Johnson

Here, the PropertyGrid hides the properties AccountNumber, Bank and Email properties which are specified in the HidePropertiesCollection.

Note: HidePropertiesCollection cannot hide the properties which are added using DynamicDescriptor.

Click [here](#) to download the sample that showcases the filtering support using the HidePropertiesCollection.

Hide the Properties using Attributes

We can hide properties by setting the [Browsable](#) value as false or [Bindable](#) as false, which properties will not be displayed in [PropertyGrid](#). Functionalities of [Browsable](#) and [Bindable](#) attributes are same.

C#

```
using System;
using System.ComponentModel;
public class Employee {
    [Browsable(false)]
    public string Bank { get; set; }
    [Bindable(false)]
    public string AccountNumber { get; set; }
    [Browsable(false)]
    public string Email { get; set; }
    public string Name { get; set; }
    public string ID { get; set; }
    public int Age { get; set; }
    public int Experience { get; set; }
}
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Email = "john@gta.com",
            AccountNumber = "23456784",
            Bank = "ABC bank",
            Name = "Johnson",
            Experience = 5,
            ID = "895",
            Age = 35,
        };
    }
}
```

XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
    <syncfusion:PropertyGrid.DataContext>
        <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
```

<div> <div> <div></div> <div></div> <div></div> </div> <div> <div>A</div> <div>Z</div> <div>↓</div> </div> <div>Search</div> </div>	
Age	35
Experience	5
ID	895
Name	Johnson

Here, the PropertyGrid not displayed the AccountNumber, Bank and Email properties which are specified Browsable and Bindable attribute value as false.

Note: If we use both the Browsable and Bindable attributes, the Browsable attribute have a higher priority.

Hide the Properties using Display.AutoGeneratedField

If a property has the value of Browsable attribute as true and Bindable attribute as true, then properties are not hidden. We can hide the property by using the [AutoGeneratedField](#) of Display as false. Value of AutoGeneratedField has no effect when the property is marked Browsable or Bindable as false.

C#

```
public class Employee
{
    [Browsable(true)]
    [Bindable(true)]
    [Display(AutoGenerateField = false)]
    public string Bank { get; set; }
    [Display(AutoGenerateField = false)]
    [Browsable(true)]
    [Bindable(true)]
    public string AccountNumber { get; set; }
    [Browsable(false)]
    [Bindable(true)]
    [Display(AutoGenerateField = false)]
    public string ID { get; set; }
    public string Email { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public int Experience { get; set; }
}

public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Email = "john@gtta.com",
            AccountNumber = "23456784",
            Bank = "ABC bank",
            Name = "Johnson",
        }
    }
}
```

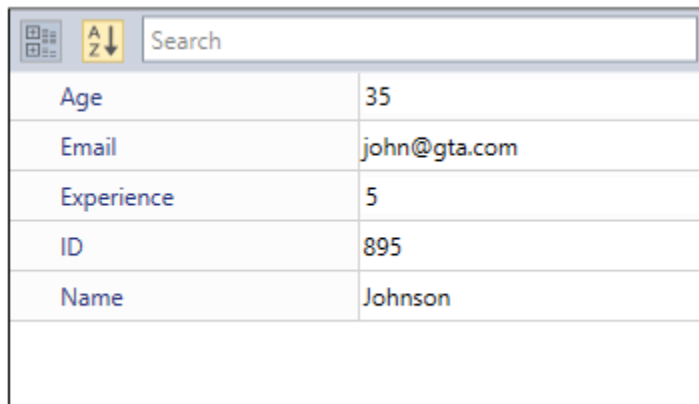
```
Experience = 5,
ID = "895",
Age = 35,
};
}
```

XML

```
<syncfusion:PropertyGrid SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
```



Age	35
Email	john@gt.com
Experience	5
ID	895
Name	Johnson

Here, the PropertyGrid not displayed the Bank and AccountNumber properties. The Bank and AccountNumber property are not displayed by value of Browsable attribute as true and Bindable attribute as true and the AutoGeneratedField of DisplayAttribute is false. In ID property, the Browsable is false, then the AutoGeneratedField property have no effect.

Click [here](#) to download the sample that showcases the filtering support using the attributes.

Hide the Properties at runtime

We can hide the properties in the PropertyGrid without using the attributes at runtime by handling the [AutoGeneratingPropertyGridItem](#) event with [AutoGeneratingPropertyGridItemEventArgs.Cancel](#) property as true.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
```

```

public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    public int Experience { get; set; }
}

public class ViewModel {
    public object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24),
            Experience = 5;
        };
    }
}

```

XML

```

<syncfusion:PropertyGrid
    AutoGeneratingPropertyGridItem="PropertyGrid1_AutoGeneratingPropertyGridItem"
    SelectedObject="{Binding SelectedEmployee}"
    x:Name="propertyGrid1">
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.AutoGeneratingPropertyGridItem +=
PropertyGrid1_AutoGeneratingPropertyGridItem;

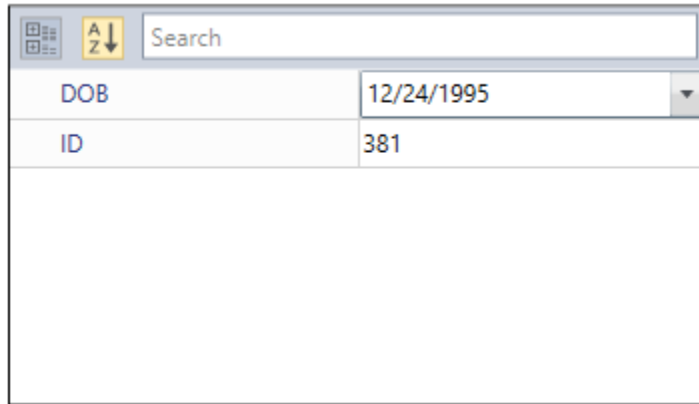
```

C#

```

private void PropertyGrid1_AutoGeneratingPropertyGridItem(object sender,
AutoGeneratingPropertyGridItemEventArgs e) {
    //Name and Experience properties hided in the PropertyGrid control.
    if (e.DisplayName == "Experience") {
        e.Cancel = true;
    }
    else if (e.DisplayName == "Name") {
        e.Cancel = true;
    }
}

```



Here, the `Experience` and `Name` properties are not displayed in the `PropertyGrid`, since the properties were restricted to be added in `PropertyGrid` by the `AutoGeneratingPropertyGridItem` event.

Click [here](#) to download the sample that showcases the property filtering support using `AutoGeneratingPropertyGridItem` event.

Searching the Properties

If the `PropertyGrid.SelectedObject` contains more properties, it is difficult to find the individual property and nested properties. Now, you can easily get the required properties by searching the property name in the `SearchBox`. `SearchBox` will filter and display the properties which contain the searched text. `SearchBox` is shown by default; you can hide it by setting `SearchBoxVisibility` property as `Collapsed`.

C#

```
// A Class that represents the nested properties
public class Address {
    public string State { get; set; }
    public string StreetName { get; set; }
    public string DoorNo { get; set; }
    public override string ToString() {
        return DoorNo + ", " + StreetName + ", " + State;
    }
}

public class Employee {
    public string Name { get; set; }
    public string ID { get; set; }
    public int Age { get; set; }
    // Property contains the nested properties
    public Address Address { get; set; }
}

public class ViewModel {
    public object SelectedEmployee { get; set; }
    public PropertyExpandModes PropertyExpandMode { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee() {
            Age = 23,
            ID = "1207",
            Name = "Mark",
            Address = new Address()
        {
            State = "New York",
```

```

DoorNo = "10",
StreetName = "Martin street"
};
PropertyExpandMode = PropertyExpandModes.FlatMode;
}
}

```

XML

```

<syncfusion:PropertyGrid PropertyExpandMode="NestedMode"
SelectedObject="{Binding SelectedEmployee}"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

```

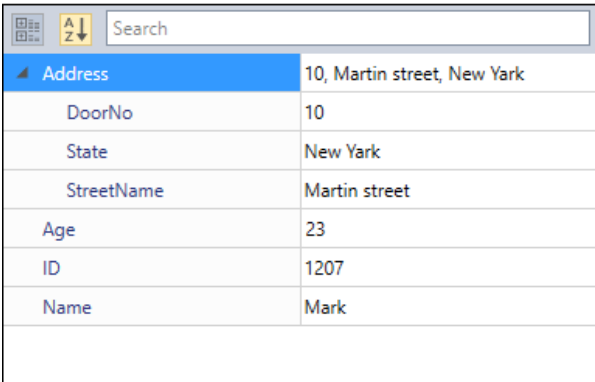
C#

```

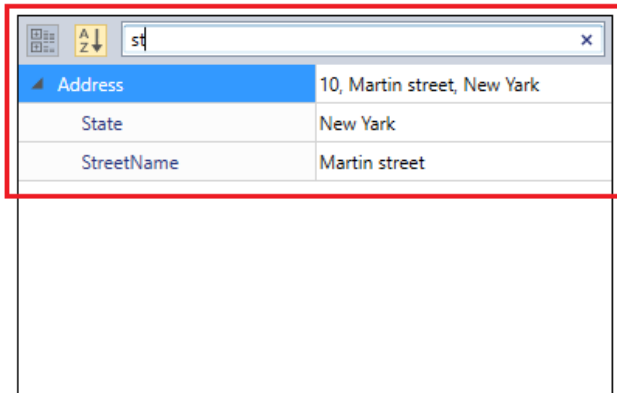
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.PropertyExpandMode = PropertyExpandModes.NestedMode;

```

Before searching



After searching



Here, The **Age** property is searched in the SearchBox.

SearchBoxVisibility = Collapsed**XML**

```

<syncfusion:PropertyGrid Name="propertyGrid1" SelectedObject="{Binding
SelectedEmployee }" SearchBoxVisibility="Collapsed" />

```

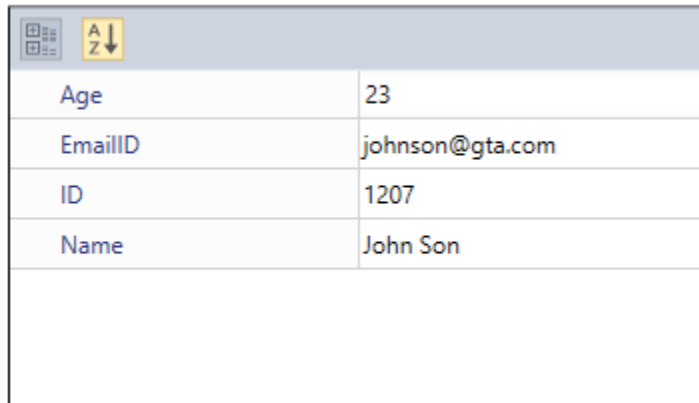
C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();

```

```
propertyGrid1.SelectedObject = (propertyGrid1.DataContext as
ViewModel).SelectedEmployee;
propertyGrid1.SearchBoxVisibility = Visibility.Collapsed;
```



Age	23
EmailID	johnson@gta.com
ID	1207
Name	John Son

Here, The SearchBox is hidden in the **PropertyGrid**.

Click [here](#) to download the sample that showcases the property searching in the SearchBox support.

#


Keyboard Navigation in WPF PropertyGrid

In this section, we will see available keyboard shortcuts and how to override the default navigation.

Keyboard Navigation between property items

The following table explains how the navigation performed between properties,

S.No	Key	Description
1	Up	Selection will be moved from current property to previous property.
2	Down	Selection will be moved from current property to next property.
3	Home	Selection will be moved from current property to first property of the PropertyGrid.
4	End	Selection will be moved from current property to last property of the PropertyGrid.
5	Left	Selection will be moved from current property to previous property. When the property EnableGrouping is 'true' and the Header of the Category group is selected, and the group is expanded, then the Category group will be collapsed, and collapsed category group header remains selected.
6	Right	Selection will be moved from current property to next property. When the property EnableGrouping is true and the Header of the Category group is selected and the group is not expanded, then the Category group will be expanded, and expanded category group header remains selected.
7	Tab	When the PropertyName field is focused then the focus will be moved to value field and for next Tab key press, focus will move to next property Name field from current property Value field.

		
8	Esc	If the property's value field is focused, then the focus has been moved to property's name field.

Handling focus of the editors

By default, **PropertyGrid** will handle the keyboard navigation, so pressing **keydown**(Up and Down) will move the focus to next/previous editor from current editor. For all built-in editors, moving focus to next editor will be handled by **PropertyGrid**. For custom editors, property navigation (focus) will not happen if custom editor handles up or down key. To override keyboard navigation for custom editors, override [ShouldPropertyGridTryToHandleKeyDown](#) method from **BaseTypeEditor**.

For example, if you use **ComboBox** as custom editor, up and down key will be handled by it. So, property navigation will not happen. You can override **ShouldPropertyGridTryToHandleKeyDown** and return **true**, to allow property grid control to handle the key down events. When it returns **false**, the editor will handles the key down event.

C#

```
//Custom combobox editor
public class ComboBoxEditor : BaseTypeEditor {
    ComboBox enumCombo;
    public override void Attach(PropertyViewItem property, PropertyItem info) {
        var binding = base.CreatePropertyInfoBinding(info, enumCombo);
        BindingOperations.SetBinding(enumCombo, ComboBox.SelectedItemProperty,
            binding);
    }
    public override object Create(PropertyInfo PropertyInfo) {
        return this.CreateEditor(PropertyInfo.PropertyType);
    }
    public override object Create(PropertyDescriptor PropertyDescriptor) {
        return this.CreateEditor(PropertyDescriptor.PropertyType);
    }
    public override void Detach(PropertyViewItem property) {
        if (enumCombo != null) {
            BindingOperations.ClearAllBindings(enumCombo);
            BindingOperations.ClearBinding(enumCombo, ComboBox.SelectedItemProperty);
        }
        enumCombo.ItemsSource = null;
        enumCombo.Items.Clear();
        enumCombo = null;
    }
    public override bool ShouldPropertyGridTryToHandleKeyDown(Key key) {
        if (key == Key.Up || key == Key.Down) {
            return false;
        }
        return true;
    }
}
/// <summary>
/// Creates and initializes a new instance of the ComboBox editor.
/// </summary>
```



```

/// <param name="propertyType">The property type</param>
/// <returns>The EnumComboEditor</returns>
private ComboBox CreateEditor(Type propertyType)
{
    enumCombo = new ComboBox() {
        ItemsSource = EnumHelper.GetValues(propertyType),
        BorderThickness = new Thickness(0)
    };
    return enumCombo;
}
}

```

C#

```

//Person.cs
[Editor("Gender", typeof(ComboBoxEditor))]
public class Person {
    public Person() {
        FirstName = "Carl";
        LastName = "Johnson";
        Age = 30;
        Mobile = 91983467382;
        Email = "carljohnson@gta.com";
        ID = "0005A";
        DOB = new DateTime(1987, 10, 16);
        Gender = Gender.Male;
    }
    public Gender Gender { get; set; }
    public string Email { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string ID { get; set; }
    public DateTime DOB { get; set; }
    public long Mobile { get; set; }
    public int Age { get; set; }
}
public enum Gender {
    Male,
    Female
}

```

XML

```

<syncfusion:PropertyGrid DefaultPropertyPath="Age"
    SelectedPropertyItem="{Binding SelectedPropertyItem, Mode=TwoWay}">
    <syncfusion:PropertyGrid.SelectedObject>
    <local:Person />
    </syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>

```

Note: View [Sample](#) in GitHub.

Appearance in WPF PropertyGrid

This section explains different UI customization, styling, theming options available in [PropertyGrid](#) control.

Setting the Foreground

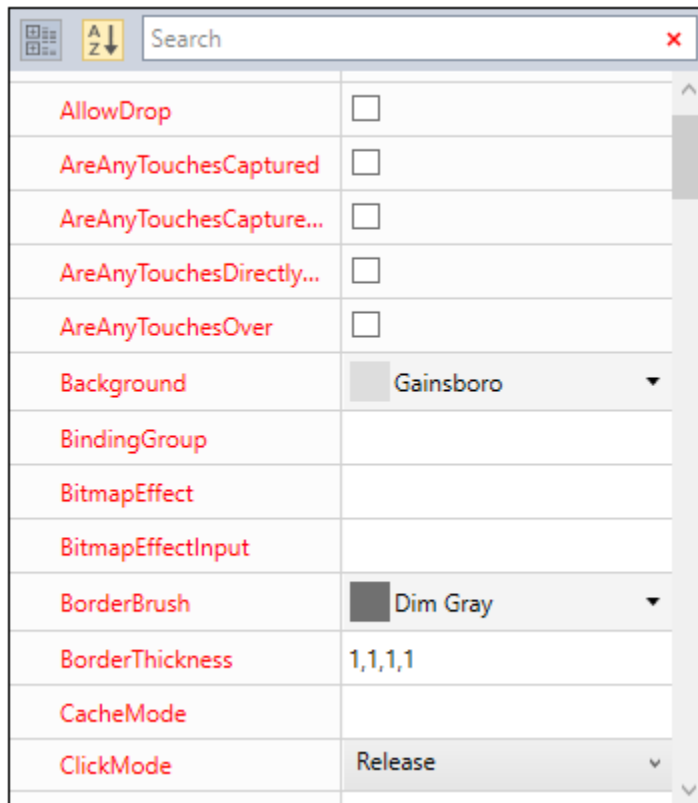
We can change the foreground color for properties of [SelectedObject](#) by setting the **Foreground** property. The default color value of **Foreground** property is **Blue**.

XML

```
<syncfusion:PropertyGrid Foreground="Red" x:Name="propertyGrid1">
  <syncfusion:PropertyGrid.SelectedObject>
    <Button></Button>
  </syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.SelectedObject = new Button();
propertyGrid1.Foreground = Brushes.Red;
```



Setting the Background and FontWeight

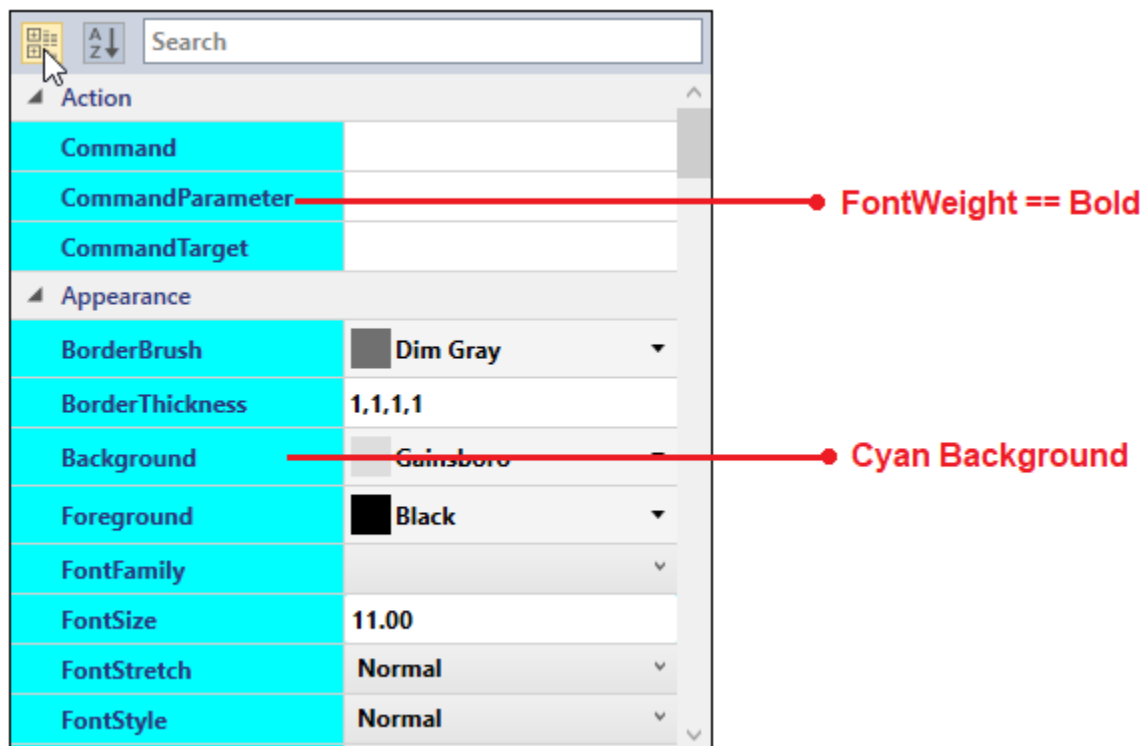
We can change the background and font weight for all the properties by using the [ViewBackgroundColor](#) and **FontWeight** properties.

XML

```
<syncfusion:PropertyGrid ViewBackgroundColor="Cyan" FontWeight="Bold"
x:Name="propertyGrid1">
<syncfusion:PropertyGrid.SelectedObject>
<Button></Button>
</syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.SelectedObject = new Button();
propertyGrid1.ViewBackgroundColor = Brushes.Cyan;
propertyGrid1.FontWeight = FontWeights.Bold;
```

*Background and FontWeight for the Editable and Readonly Properties*

If we want to differentiate between editable and readonly properties, we can do this by using [EditableBackground](#) and [EditableFontWeight](#) properties to highlights the editable properties and use the [ReadOnlyBackground](#) and [ReadOnlyFontWeight](#) properties to highlights the readonly properties.

C#

```
class Employee {
public string EmployeeName { get; set; }
[Editable(false)]
public int EmployeeID { get; set; }
public int Age { get; set; }
[ReadOnly(true)]
public DateTime DOB { get; set; }
}
```

```

public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            EmployeeName = "John",
            DOB = new DateTime(1995, 01, 08),
            Age=25,
            EmployeeID = 036
        };
    }
}

```

XML

```

<syncfusion:PropertyGrid EditableBackground="LightGreen"
EditableFontWeight="Bold"
ReadOnlyBackground="LightPink" ReadOnlyFontWeight="UltraLight"
SelectedObject="{Binding SelectedEmployee}" x:Name="propertyGrid1" >
    <syncfusion:PropertyGrid.DataContext>
    <local:ViewModel></local:ViewModel>
</syncfusion:PropertyGrid.DataContext>
</syncfusion:PropertyGrid>

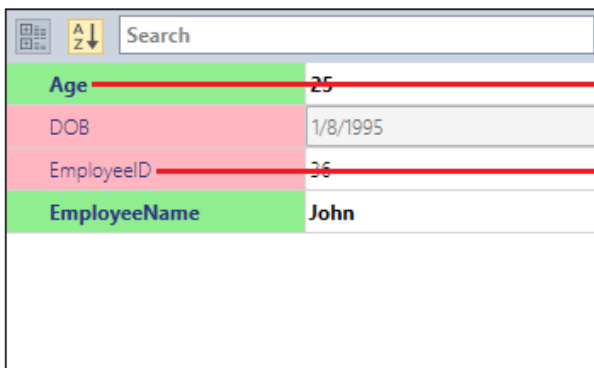
```

C#

```

PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.DataContext = new ViewModel();
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new
Binding("SelectedEmployee"));
propertyGrid1.EditableBackground = Brushes.LightGreen;
propertyGrid1.EditableFontWeight = FontWeights.Bold;
propertyGrid1.ReadOnlyBackground = Brushes.LightPink;
propertyGrid1.ReadOnlyFontWeight = FontWeights.UltraLight;

```



**EditableBackground = "LightGreen",
EditableFontWeight = "Bold"**

**ReadOnlyBackground = "LightPink"
ReadOnlyFontWeight = "UltraLight"**

Note: If you use `EditableBackground` or `ReadOnlyBackground` properties with `ViewBackgroundColor` property, `EditableBackground` and `ReadOnlyBackground` properties have higher priority.

Note: If you use `EditableFontWeight` or `ReadOnlyFontWeight` properties with `FontWeight` property, `EditableFontWeight` and `ReadOnlyFontWeight` properties have higher priority.

Category Header's foreground and background

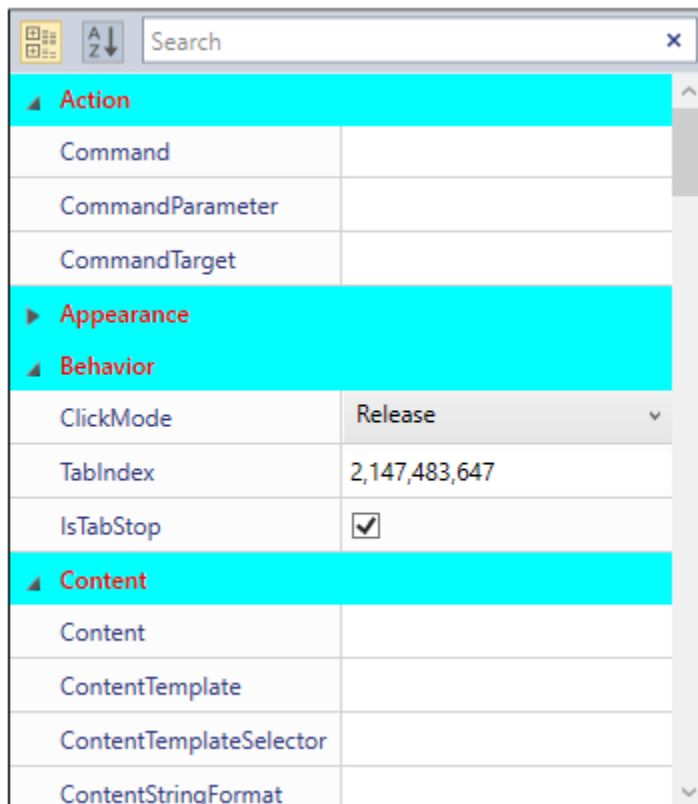
We can change the background and foreground of the category header by setting the `LineColor` property and `CategoryForeground` property. The `LineColor` property value applied to the background and `CategoryForeground` property value applied to the foreground of the category header only on category view. To enable category view, use the `EnableGrouping` property as `true`.

XML

```
<syncfusion:PropertyGrid LineColor="Cyan" CategoryForeground="Red"
x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.SelectedObject>
<Button></Button>
</syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.SelectedObject = new Button();
propertyGrid1.LineColor = Brushes.Cyan;
propertyGrid1.CategoryForeground = Brushes.Red;
```



Customize the height of [PropertyViewItem](#) and [PropertyCategoryViewItem](#)

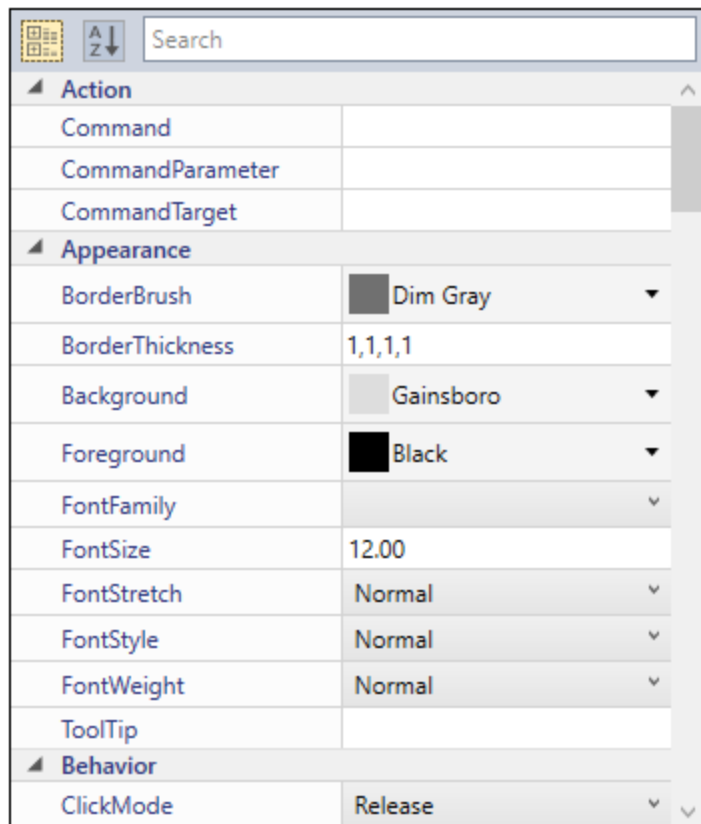
We can customize the height of [PropertyViewItem](#) and [PropertyCategoryViewItem](#) using its **Padding** property by overriding style in PropertyGrid.

XML

```
<Window x:Class="PropertyGrid_CustomEditor.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:PropertyGrid_CustomEditor"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d" WindowStartupLocation="CenterScreen"
Title="MainWindow" Height="450" Width="600">
<Window.Resources>
<Style TargetType="syncfusion:PropertyViewItem" >
<Setter Property="Padding" Value="0"/>
</Style>
<Style TargetType="syncfusion:PropertyCategoryViewItem">
<Setter Property="Padding" Value="0"/>
</Style>
</Window.Resources>
<Grid>
<syncfusion:PropertyGrid Margin="10" x:Name="propertyGrid1" >
<syncfusion:PropertyGrid.SelectedObject>
<Button></Button>
</syncfusion:PropertyGrid.SelectedObject>
</syncfusion:PropertyGrid>
</Grid>
</Window>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();
propertyGrid1.SelectedObject = new Button();
```



Tooltip support

You can get the value and description about the property item through tooltip when hover the mouse on the respective property item and its value field. If the property item not contains any description, tooltip shows the property display name. You can restrict the tooltip support by setting the [EnableToolTip](#) property as `false`. The default value of [EnableToolTip](#) property is `true`.

C#

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
public class Employee {
    [Description("Name of the employee")]
    public string Name { get; set; }
    public string ID { get; set; }
    [Description("Birth date of the employee")]
    public DateTime DOB { get; set; }
}
public class ViewModel {
    public Object SelectedEmployee { get; set; }
    public ViewModel() {
        SelectedEmployee = new Employee()
        {
            Name = "John",
            ID = "381",
            DOB = new DateTime(1995, 12, 24)
        };
    }
}
```

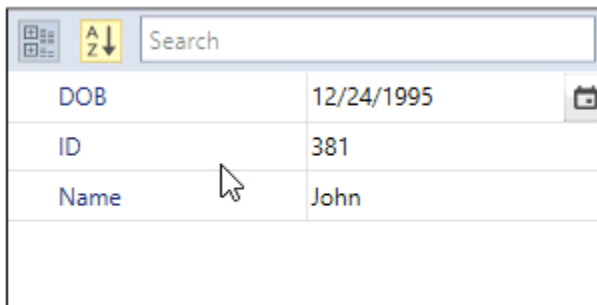
```
}
```

XML

```
<syncfusion:PropertyGrid EnableToolTip="True"  
SelectedObject="{Binding SelectedEmployee}"  
x:Name="propertyGrid1">  
  <syncfusion:PropertyGrid.DataContext>  
    <local:ViewModel></local:ViewModel>  
  </syncfusion:PropertyGrid.DataContext>  
</syncfusion:PropertyGrid>
```

C#

```
PropertyGrid propertyGrid1 = new PropertyGrid();  
propertyGrid1.EnableToolTip = true;  
propertyGrid1.DataContext = new ViewModel();  
propertyGrid1.SetBinding(PropertyGrid.SelectedObjectProperty, new  
Binding("SelectedEmployee"));
```








Note: View [Sample](#) in GitHub

Theme

PropertyGrid supports various built-in themes. Refer to the below links to apply themes for the PropertyGrid,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

  Search	
ID	0005A
First Name	Carl
Last Name	Johnson
Date of Birth	10/16/1987 
Gender	Male ▼
Age	30
Favorite Color	 Gray ▼
Permanent Employee	
▶ Bank	Centura Banks
Email ID	carljohnson@gt.□□
Mobile Number	91,983,467,382
Country	USA ▼
Age Age of the actual person.	

Localization in WPF PropertyGrid

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the [PropertyGrid](#) by adding resource file. Application culture can be changed by setting `CurrentUICulture` after `InitializeComponent` method.

Below application culture changed to `French`.

C#

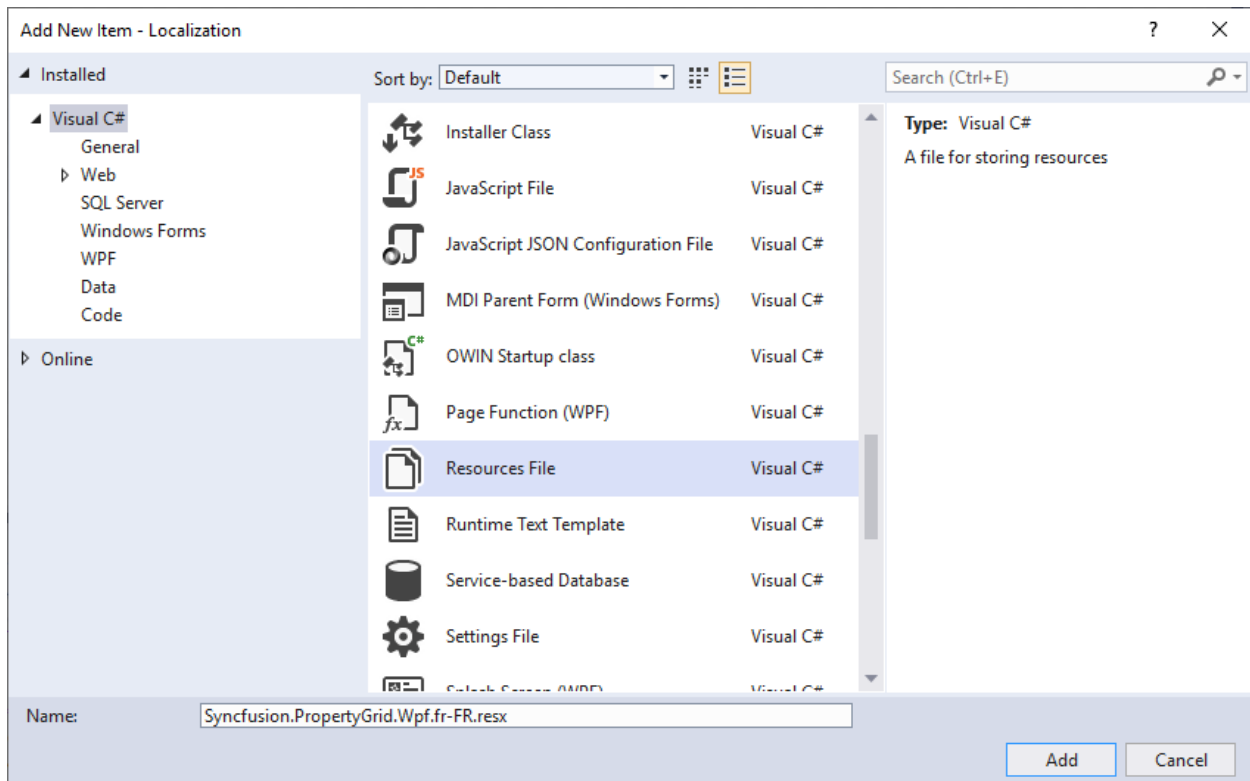
```
public MainWindow()
{
    InitializeComponent();
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("fr-FR");
}
```

To localize the `PropertyGrid` based on `CurrentUICulture` using resource files, follow the below steps.

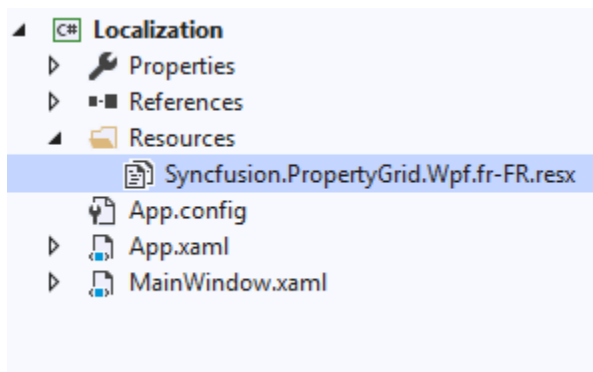
Step 1: Create new folder and named as `Resources` in your application.

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.PropertyGrid.Wpf.<culture name>.resx`. For example, you have to give name as `Syncfusion.PropertyGrid.Wpf.fr-FR.resx` for `French` culture.



Step 4: Now, select `Add` option to add the resource file in `Resources` folder.



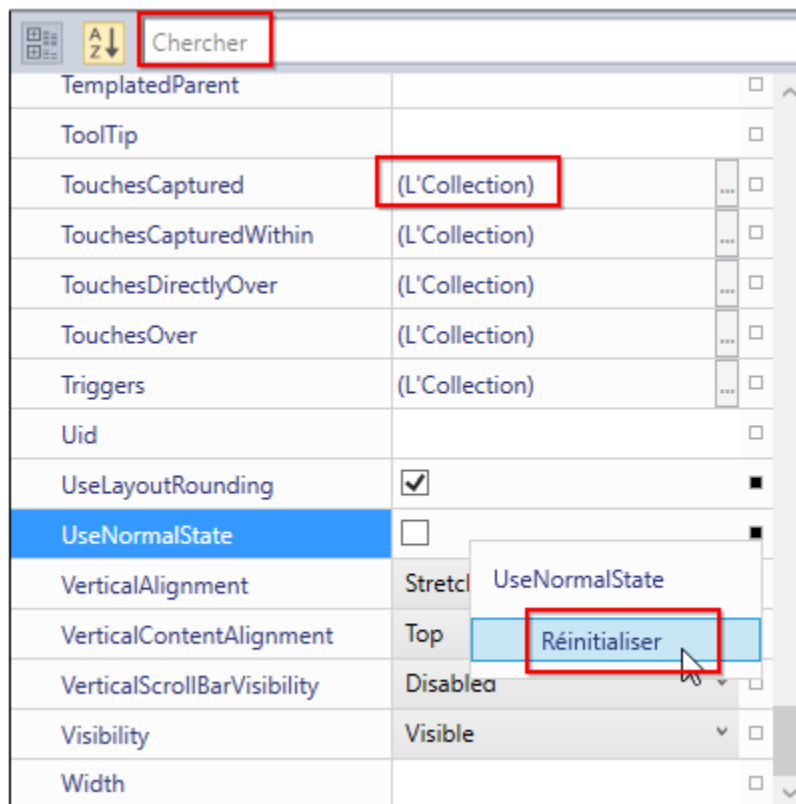
Step 5: Add the Name/Value pair in Resource Designer of `Syncfusion.PropertyGrid.Wpf.fr-FR.resx` file and change its corresponding value to corresponding culture.

Syncfusion.PropertyGrid.Wpf.fr-FR.resx

Strings Add Resource Remove Resource Access Modifier: No code gen

	Name	Value	Comment
	Add	Ajouter	Add button text in CollectionEditor dialog.
	Alphabetical	Alphabétique	Tooltip text for button, to sort properties alphabetically in PropertyGrid.
	Cancel	Annuler	Cancel button text in CollectionEditor dialog.
	Categorized	Catégorisé	Tooltip text for button, to categorize properties in PropertyGrid.
	Clear	Claire	Tooltip text for clear button of search box in PropertyGrid.
	CollectionEditorDialogTitle	L'CollectionEditor	Title text of CollectionEditor dialog.
	CollectionEditorWatermark	(L'Collection)	Watermark text of collection editor in PropertyGrid.
	OK	D'accord	OK button text in CollectionEditor dialog.
	Remove	Retirer	Remove button text in CollectionEditor dialog.
	Reset	Réinitialiser	Text for menu item, to reset value of property in PropertyGrid.
	Search	Chercher	Watermark text of search textbox in PropertyGrid.
*			

The following screenshot shows the localized PropertyGrid control.



Note: View [Sample](#) in GitHub

SfRadialMenu

WPF Radial Menu (SfRadialMenu) Overview

The Radial Menu displays a hierarchical menu in a circular layout. Typically used as a context menu, it can expose more menu items in the same space than traditional menus.

Key Features

- Items Source – Any business object collection can be bound to control.
- Commanding – Each item can be bound to a command that could perform an action.
- Color Palette – Easy to form radial color palette layout.



Getting Started with WPF Radial Menu (SfRadialMenu)

Namespace : Syncfusion.Windows.Controls.Navigation

Assembly : Syncfusion.SfRadialMenu.WPF (in Syncfusion.SfRadialMenu.WPF.dll)

The following code sample shows how to create the RadialMenu from code-behind and XAML.

XML

```
<Page xmlns:navigation="clr-
namespace:Syncfusion.Windows.Controls.Navigation;assembly=Syncfusion.SfRadialMenu.Wpf">
<Grid>
<navigation:SfRadialMenu>
<navigation:SfRadialMenuItem Header="Bold"/>
<navigation:SfRadialMenuItem Header="Cut"/>
<navigation:SfRadialMenuItem Header="Copy"/>
<navigation:SfRadialMenuItem Header="Paste"/>
</navigation:SfRadialMenu>
</Grid>
</Page>
```

C#

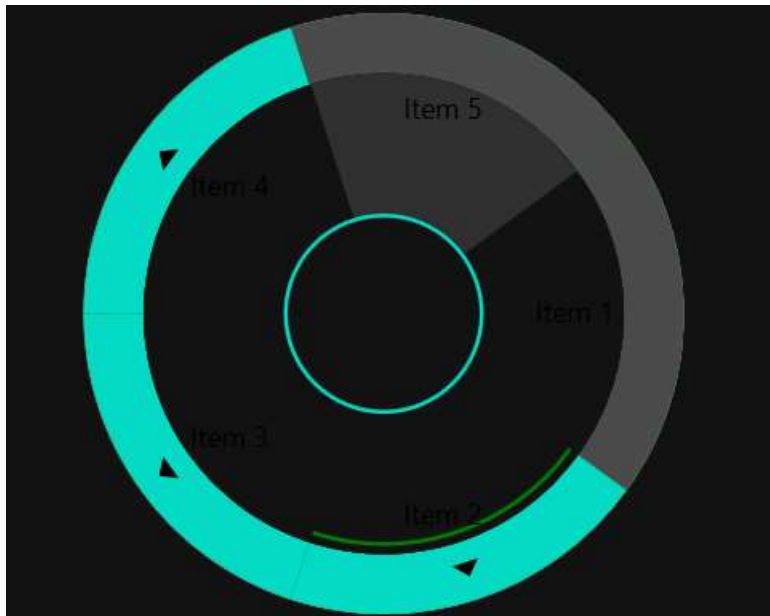
```
SfRadialMenu radialMenu = new SfRadialMenu();
SfRadialMenuItem bold = new SfRadialMenuItem() { Header = "Bold" };
SfRadialMenuItem cut = new SfRadialMenuItem() { Header = "Cut" };
SfRadialMenuItem copy = new SfRadialMenuItem() { Header = "Copy" };
```

```
SfRadialMenuItem paste = new SfRadialMenuItem() { Header = "Paste" };
radialMenu.Items.Add(bold);
radialMenu.Items.Add(cut);
radialMenu.Items.Add(copy);
radialMenu.Items.Add(paste);
```

Theme

Radial Menu supports various built-in themes. Refer to the below links to apply themes for the Radial Menu,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



Populating Items in WPF Radial Menu (SfRadialMenu)

Items Source

Radial menu items can be populated with the business object collection. Let us create a RadialMenu which will show the list of Application commands.

The Application command model look likes below.

C#

```
public class ApplicationCommand
{
    public string Name { get; set; }
    public string ImagePath { get; set; }
    public ICommand Command { get; set; }
}
```

Create the Application command collection as follows.

C#

```
private List<ApplicationCommand> options;
public List<ApplicationCommand> Options
{
    get { return options; }
    set { options = value; }
}
```

Populate the Application command collection as follows.

C#

```
Options = new List<ApplicationCommand>();
Options.Add(new ApplicationCommand() { Name="Bold" , ImagePath="bold.png"
});
Options.Add(new ApplicationCommand() { Name = "Cut"
, ImagePath="cut.png"});
Options.Add(new ApplicationCommand() { Name = "Copy"
, ImagePath="copy.png"});
Options.Add(new ApplicationCommand() { Name = "Paste"
, ImagePath="paste.png"});
```

Bind the Application command collection to the ItemsSource property of the RadialMenu control.

XML

```
<navigation:SfRadialMenu IsOpen="True" ItemsSource="{Binding Options}"/>
```

This will populate the RadialMenu as shown in the image below.



Display Member Path

DisplayMemberPath property of the Radial Menu used to define which business model property needs to be displayed inside the header of the Radial Menu items.

XML

```
<navigation:SfRadialMenu IsOpen="True" ItemsSource="{Binding Options}"
DisplayMemberPath="Name"
/>
```



Displaying member path

Command Path

CommandPath property of the Radial Menu can be used to bind the command in the business object to the radial menu item when items are populated using data binding.

XML

```
<navigation:SfRadialMenu IsOpen="True" DisplayMemberPath="Name"
  CommandPath="Command"
  ItemsSource="{Binding Options}" />
```

Item Template

ItemTemplate property of the RadialMenu can be used to customize the header part of the radial menu items.

XML

```
<navigation:SfRadialMenu IsOpen="True" ItemsSource="{Binding Options}">
  <navigation:SfRadialMenu.ItemTemplate>
    <DataTemplate>
      <StackPanel >
        <Image Height="15" Width="15" Source="{Binding ImagePath}" />
        <TextBlock Margin="0,5,0,0" Text="{Binding Name}" />
      </StackPanel>
    </DataTemplate>
  </navigation:SfRadialMenu.ItemTemplate>
</navigation:SfRadialMenu>
```



Icon in WPF Radial Menu (SfRadialMenu)

The Icon property of the RadialMenu is used to customize the icon displayed in the center of RadialMenu circle.

XML

```
<navigation:SfRadialMenu IsOpen="True" >
  <navigation:SfRadialMenu.Icon>
    <Grid Background="White">
      <Image Source="ms-appx:///Assets/text.png" Width="20"
        Stretch="Uniform"/>
    </Grid>
  </navigation:SfRadialMenu.Icon>
</navigation:SfRadialMenu>
```



Populating Color Palette in WPF Radial Menu (SfRadialMenu)

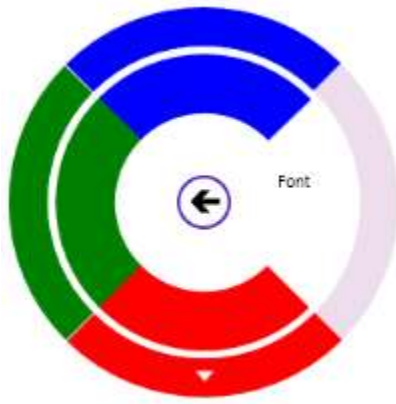
Color Palette can be formed in Radial Menu with the Radial Color Items.

XML

```
<navigation:SfRadialMenu IsOpen="True" >
  <navigation:SfRadialMenuItem Header="Font Color">
    <navigation:SfRadialMenuItem Header="Font"/>
    <navigation:SfRadialColorItem Color="Red">
      <navigation:SfRadialColorItem Color="DarkRed"/>
      <navigation:SfRadialColorItem Color="IndianRed"/>
      <navigation:SfRadialColorItem Color="OrangeRed"/>
    </navigation:SfRadialColorItem>
  </navigation:SfRadialMenuItem>
</navigation:SfRadialMenu>
```



```
<navigation:SfRadialColorItem Color="MediumVioletRed"/>
</navigation:SfRadialColorItem>
<navigation:SfRadialColorItem Color="Green"/>
<navigation:SfRadialColorItem Color="Blue"/>
</navigation:SfRadialMenuItem>
</navigation:SfRadialMenu>
```



ToolTip in WPF Radial Menu (SfRadialMenu)

ToolTip support available for the radial menu items. This will show when mouse over the corresponding item.

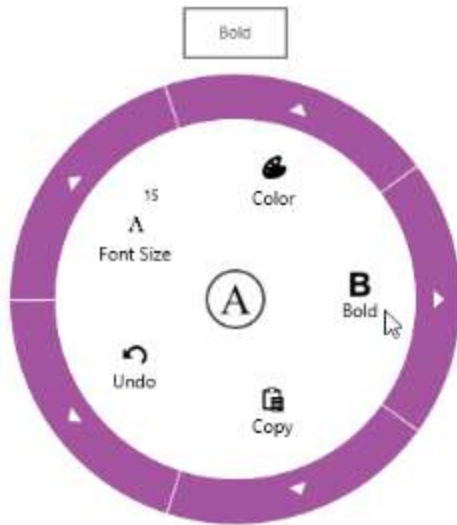
ToolTip Placement

Position of the tooltip displayed relative to the Radial Menu can be customized using ToolTipPlacement property. This have the following options.

- None: Tooltip is hidden from the display.
- Left: Tooltip is displayed left of the Radial Menu.
- Top: Tooltip is displayed on top of the Radial Menu.
- Right: Tooltip is displayed right of the Radial Menu.
- Bottom: Tooltip is displayed at the bottom of the Radial Menu.

XML

```
<navigation:SfRadialMenuItem Tooltip="Bold" TooltipPlacement="Top" />
```



Layout Types in WPF Radial Menu (SfRadialMenu)

There are two different layout types available for SfRadialMenu.

- Default
- Custom

Both the layout types divide the available space equally among all the children in the circular panel.

Default

Number of segments in the panel is determined by children count in the level. Hence segment count in each hierarchical level differs. RadialMenuItem is arranged in the sequential order as added in the RadialMenu.

Custom

Number of segments in the panel is determined by VisibleSegmentsCount property. Hence segment count in all the hierarchical levels are same. RadialMenuItem is arranged in any order based on the SegmentIndex property.

VisibleSegmentsCount

VisibleSegmentsCount property is used to specify the number of segments available in circular panel. When children count is greater than the value given in VisibleSegmentsCount property then overflowing children are not arranged in the panel. When children count is lesser than VisibleSegmentsCount property then remaining segments are left free.

XML

```
<navigation:SfRadialMenu LayoutType="Custom" VisibleSegmentsCount="7" />
```

C#

```
SfRadialMenu radialMenu = new SfRadialMenu();
radialMenu.LayoutType = LayoutType.Custom;
radialMenu.VisibleSegmentsCount = 7;
```

SegmentIndex

SegmentIndex property is used to specify the index of the SfRadialMenuItem in circular panel. Based on the index, the RadialMenuItems are inserted in the segment. When SegmentIndex is not specified for a RadialMenuItem (or) two or more RadialMenuItems having the same SegmentIndex, then the menu item is arranged in the next available free segment.

XML

```
<navigation:SfRadialMenu LayoutType="Custom" VisibleSegmentsCount="7" />
<navigation:SfRadialMenuItem Header="Item 2" SegmentIndex="1" />
<navigation:SfRadialMenuItem Header="Item 5" SegmentIndex="4" />
<navigation:SfRadialMenuItem Header="Item 1" SegmentIndex="0" />
<navigation:SfRadialMenuItem Header="Item 6" SegmentIndex="5" />
<navigation:SfRadialMenuItem Header="Item 3" SegmentIndex="2" />
</navigation:SfRadialMenu>
```

C#

```
SfRadialMenu radialMenu = new SfRadialMenu();
radialMenu.LayoutType = LayoutType.Custom;
radialMenu.VisibleSegmentsCount = 7;
SfRadialMenuItem item2 = new SfRadialMenuItem() { Header = "Item 2",
SegmentIndex = 1 };
SfRadialMenuItem item5 = new SfRadialMenuItem() { Header = "Item 5",
SegmentIndex = 4 };
SfRadialMenuItem item1 = new SfRadialMenuItem() { Header = "Item 1",
SegmentIndex = 0 };
SfRadialMenuItem item6 = new SfRadialMenuItem() { Header = "Item 6",
SegmentIndex = 5 };
SfRadialMenuItem item3 = new SfRadialMenuItem() { Header = "Item
3", SegmentIndex = 2 };
radialMenu.Items.Add(item2);radialMenu.Items.Add(item5);radialMenu.Items.Add
(item1);
radialMenu.Items.Add(item6); radialMenu.Items.Add(item3); </td></tr>
```



Appearance and Styling

Radius

RadiusX and RadiusY properties in the Radial Menu can be used to define the X and Y axis radius to render the control.

XML

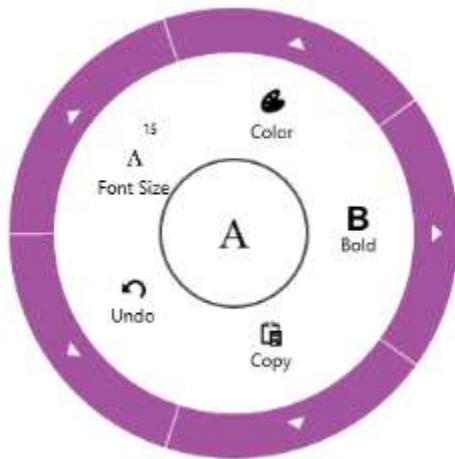
```
<navigation:SfRadialMenu RadiusX="150" RadiusY="150" />
```

CenterRimRadiusFactor

CenterRimRadiusFactor property can be used to define the radius of the center rim (inner circle).

XML

```
<navigation:SfRadialMenu CenterRimRadiusFactor="0.3" IsOpen="True" />
```

**RimBackground**

RimBackground property used to fill the outer rim (outer circle).

XML

```
<navigation:SfRadialMenu IsOpen="True" RimBackground="Green" />
```



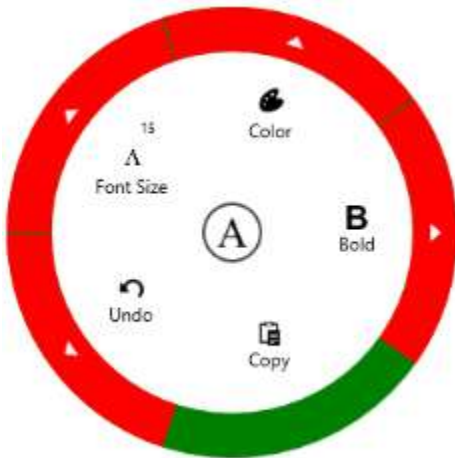
RadialMenu outer ring filled with color

RimActiveBrush

RimActiveBrush property used to fill the expander rim and this expander rim only visible when the items have sub items.

XML

```
<navigation:SfRadialMenu RimActiveBrush="Red" RimBackground="Green"
IsOpen="True" />
```



RimInactiveBrush

RimInactiveBrush property used to fill the expander rim item background when the corresponding menu item doesn't have sub items.

XML

```
<syncfusion:SfRadialMenu >
<syncfusion:SfRadialMenuItem Header="Item 1" RimActiveBrush="SlateBlue"
RimInactiveBrush="Red"/>
<syncfusion:SfRadialMenuItem Header="Item 2" RimActiveBrush="Green">
<syncfusion:SfRadialMenuItem Header="Item 21" />
</syncfusion:SfRadialMenuItem>
<syncfusion:SfRadialMenuItem Header="Item 3" RimActiveBrush="Yellow">
<syncfusion:SfRadialMenuItem Header="Item 31"/>
</syncfusion:SfRadialMenuItem>
<syncfusion:SfRadialMenuItem Header="Item 4" RimActiveBrush="Orange">
<syncfusion:SfRadialMenuItem Header="Item 41"/>
</syncfusion:SfRadialMenuItem >
<syncfusion:SfRadialMenuItem Header="Item 5" IsExpanderVisible="False"
RimActiveBrush="Black" RimInactiveBrush="GreenYellow" />
</syncfusion:SfRadialMenu>
```

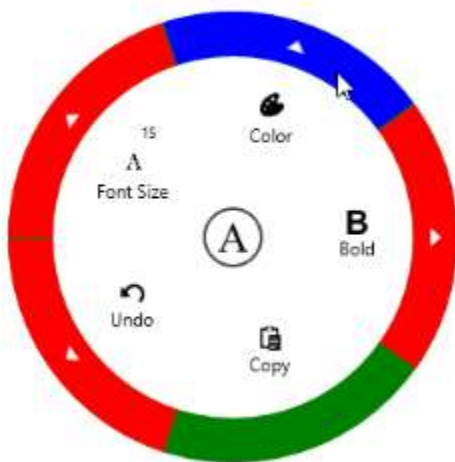


RimHoverBrush

The RimHoverBrush property can be used to fill the expander rim while the pointer is hovering over it.

XML

```
<navigation:SfRadialMenu RimActiveBrush="Red" RimBackground="Green"
RimHoverBrush="Blue" IsOpen="True" />
```



IsExpanderVisible

Expander arrow in the OuterRim of SfRadialMenu visibility can be changed by IsExpanderVisible property of SfRadialMenuItem. By default, IsExpanderVisible value is True.

XML

```
<syncfusion:SfRadialMenu >
<syncfusion:SfRadialMenuItem Header="Item 1" RimActiveBrush="SlateBlue">
<syncfusion:SfRadialMenuItem Header="Item 21"/>
</syncfusion:SfRadialMenuItem>
<syncfusion:SfRadialMenuItem Header="Item 2" RimActiveBrush="Green" >
<syncfusion:SfRadialMenuItem Header="Item 21" />
</syncfusion:SfRadialMenuItem>
```

```

<syncfusion:SfRadialMenuItem Header="Item 3" RimActiveBrush="Yellow">
<syncfusion:SfRadialMenuItem Header="Item 31"/>
</syncfusion:SfRadialMenuItem>
<syncfusion:SfRadialMenuItem Header="Item 4" RimActiveBrush="Orange">
<syncfusion:SfRadialMenuItem Header="Item 41"/>
</syncfusion:SfRadialMenuItem >
<syncfusion:SfRadialMenuItem Header="Item 5" IsExpanderVisible="False"
RimActiveBrush="Black" >
<syncfusion:SfRadialMenuItem Header="Item 41"/>
</syncfusion:SfRadialMenuItem>
</syncfusion:SfRadialMenu>

```



RimRadiusFactor

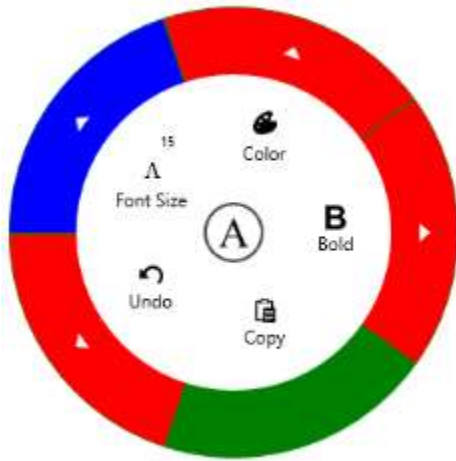
RimRadiusFactor property of Radial Menu can be used to set the radius of the items panel. Lowest values to this factor increases the thickness of the outer rim. Highest values to this factor decreases the thickness of the outer rim.

XML

```

<navigation:SfRadialMenu RimActiveBrush="Red" RimRadiusFactor="0.7"
RimBackground="Green" RimHoverBrush="Blue" IsOpen="True" />

```



Navigation Button Style

The navigation button displayed in the center of radial menu can be styled using `NavigationButtonStyle` property.

XML

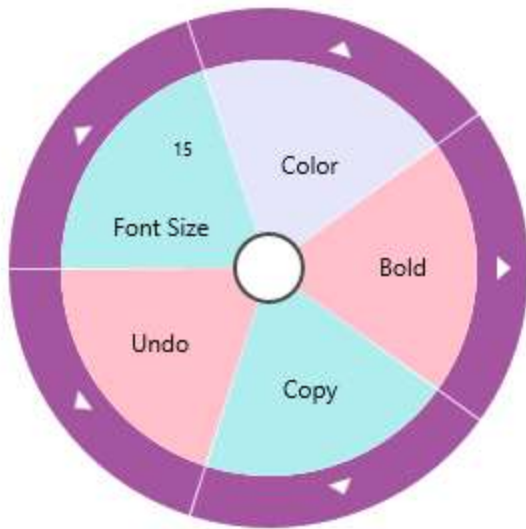
```
<syncfusion:SfRadialMenu NavigationButtonStyle="{StaticResource
NavigationButtonStyle}"
IsOpen="True" />
```

MenuBackgroundColor

Each `SfRadialMenuItem` can be set a different background color using `MenuBackgroundColor` property.

XML

```
<navigation:SfRadialMenu IsOpen="True">
<navigation:SfRadialMenuItem Header="Bold" MenuBackgroundColor="Pink">
<navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
<navigation:SfRadialMenuItem Header="Copy"
MenuBackgroundColor="PaleTurquoise">
<navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
<navigation:SfRadialMenuItem Header="Undo" MenuBackgroundColor="Pink">
<navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
<navigation:SfRadialMenuItem Header="Font Size"
MenuBackgroundColor="PaleTurquoise">
<navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
<navigation:SfRadialMenuItem Header="Color" MenuBackgroundColor="Lavender">
<navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
</navigation:SfRadialMenu>
```

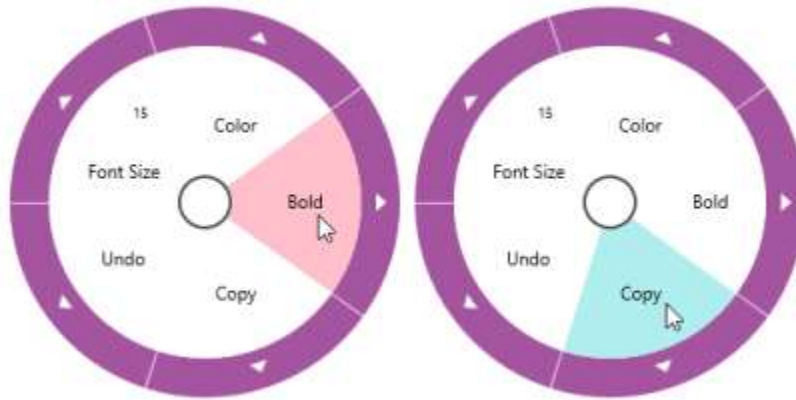



MenuMouseOverBackgroundColor

Each SfRadialMenuItem can be set with a different background color on mouse over by using the MenuMouseOverBackgroundColor property. Before that, the ShowMouseOverStyle property should be set to true.

XML

```
<navigation:SfRadialMenu IsOpen="True">
  <navigation:SfRadialMenuItem Header="Bold"
    MenuMouseOverBackgroundColor="Pink" ShowMouseOverStyle="true">
  </navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
  <navigation:SfRadialMenuItem Header="Copy"
    MenuMouseOverBackgroundColor="PaleTurquoise" ShowMouseOverStyle="true">
  </navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
  <navigation:SfRadialMenuItem Header="Undo"
    MenuMouseOverBackgroundColor="Pink" ShowMouseOverStyle="true">
  </navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
  <navigation:SfRadialMenuItem Header="Font Size"
    MenuMouseOverBackgroundColor="PaleTurquoise" ShowMouseOverStyle="true">
  </navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
  <navigation:SfRadialMenuItem Header="Color"
    MenuMouseOverBackgroundColor="Lavender" ShowMouseOverStyle="true">
  </navigation:SfRadialMenuItem/>
</navigation:SfRadialMenuItem>
</navigation:SfRadialMenu>
```



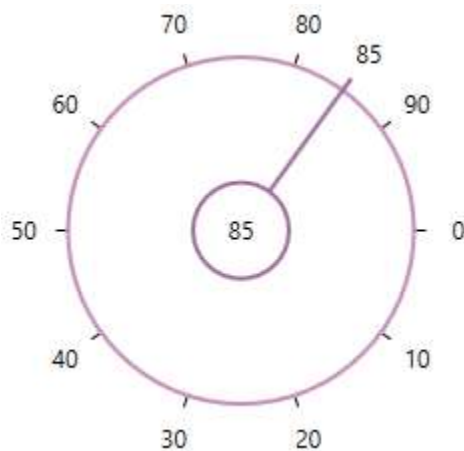
SfRadialSlider

WPF Radial Slider (SfRadialSlider) Overview

The [SfRadialSlider](#) provides an optimized interface for selecting a numeric value within the minimum and maximum range in the circular track.

Key features

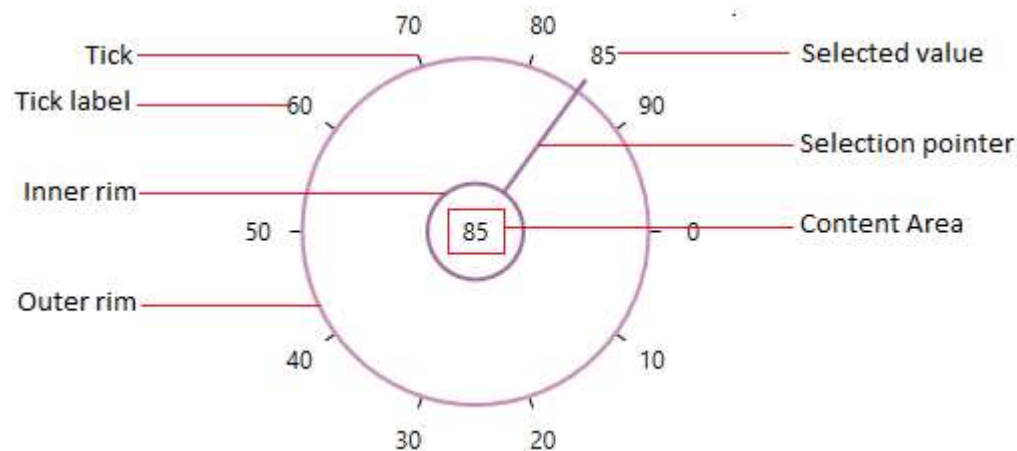
- Customization – Can customize outer rim, inner rim, labels, ticks, and pointer.
- Content – Can place content inside the inner rim.
- Step interval support
- Min-Max value support
- Sweep direction support
- Start-End angle support



Getting Started with WPF Radial Slider (SfRadialSlider)

This section explains how to create a WPF [SfRadialSlider](#) and explains about its structure.

Structure of SfRadialSlider



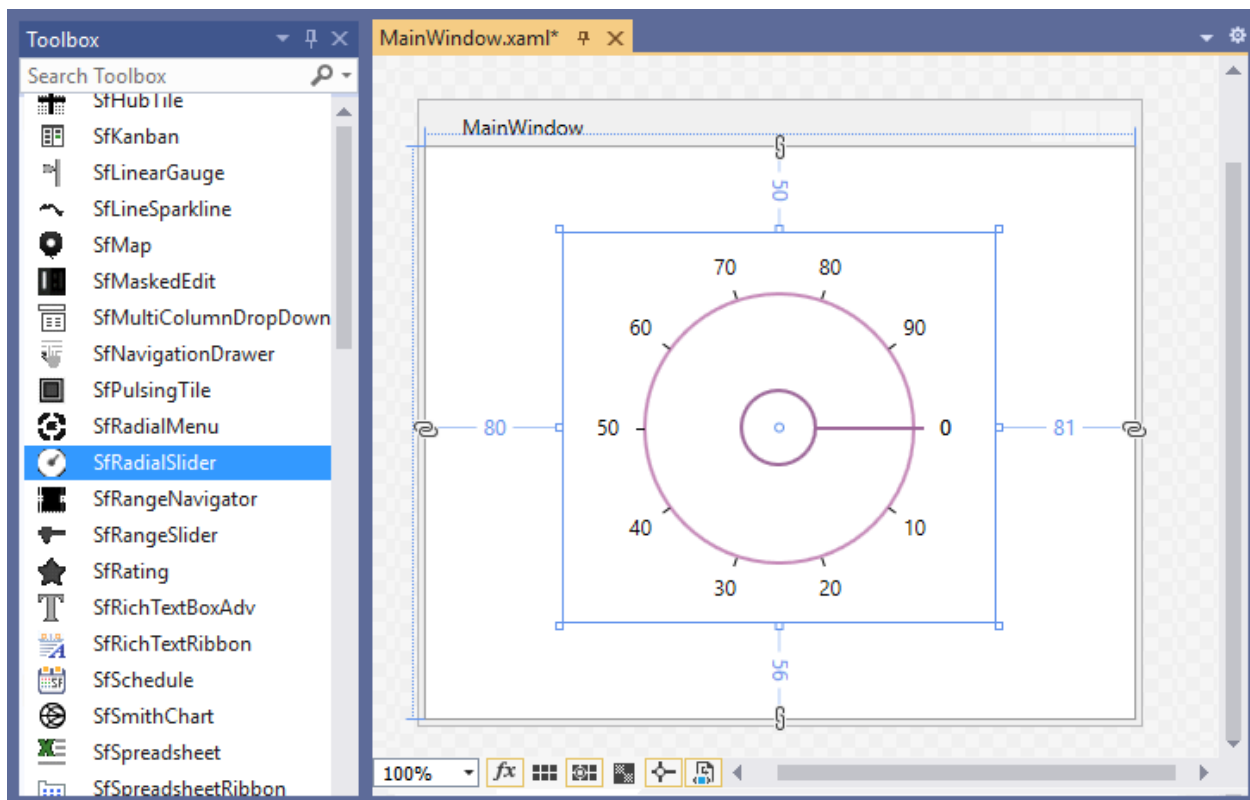
Assembly deployment

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 SfRadialSlider via designer

1) The SfRadialSlider 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.SfRadialMenu.WPF*
Syncfusion.SfShared.WPF



2) Set the properties for SfRadialSlider in design mode using the SmartTag feature.

Adding WPF SfRadialSlider via XAML

To add the **SfRadialSlider** 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.SfRadialMenu.WPF* *Syncfusion.SfShared.WPF* 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the SfRadialSlider in XAML page.

XML

```
<Window x:Class="SfRadialSlider_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:SfRadialSlider_sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<syncfusion:SfRadialSlider Name="radialSlider"
Height="200"
Width="200"/>
</Grid>
</Window>
```

Adding WPF SfRadialSlider via C#

To add the **SfRadialSlider** 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.

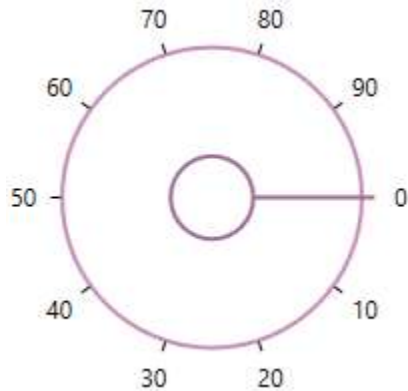
C#

```
using Syncfusion.Windows.Shared;
```

4) Create an instance of **SfRadialSlider**, and add it to the window.

C#

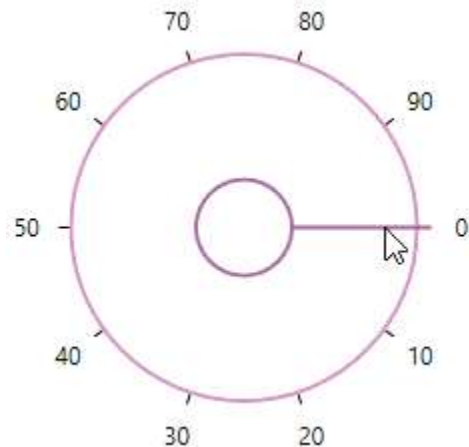
```
// Creating an instance of the SfRadialSlider
SfRadialSlider radialSlider = new SfRadialSlider();
// Setting height and width to SfRadialSlider
radialSlider.Height = 200;
radialSlider.Width = 200;
```



Note: [View Sample in GitHub](#)

Select tick value

You can select any tick value by dragging the pointer along the circular track or clicking on the corresponding track value. You can get the selected value by using the `Value` property. The default value of `Value` property is 0.



Select tick value programmatically

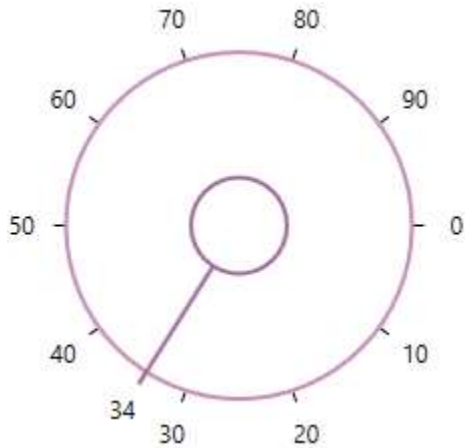
You can select a tick value programmatically by assigning the tick value to the `Value` property.

XML

```
<syncfusion:SfRadialSlider Value="34"
Name="radialSlider" />
```

C#

```
radialSlider.Value = 34;
```



Note: [View Sample in GitHub](#)

Display selected value

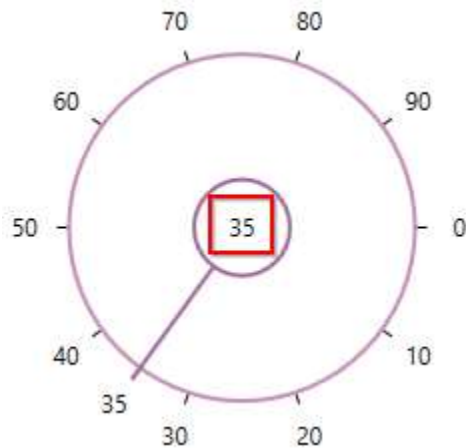
You can display the selected value in the content area of the `SfRadialSlider` by setting the selected value to the `Content` property. The default value of `Content` property is null.

C#

```
public class ViewModel
{
    private double selectedValue;
    public double SelectedValue {
        get {
            return selectedValue;
        }
        set {
            selectedValue= value;
        }
    }
}
```

XML

```
<syncfusion:SfRadialSlider Content="{Binding SelectedValue,Mode=TwoWay}"
Value="{Binding SelectedValue,Mode=TwoWay}"
Name="radialSlider">
    <syncfusion:SfRadialSlider.DataContext>
        <local:ViewModel/>
    </syncfusion:SfRadialSlider.DataContext>
</syncfusion:SfRadialSlider>
```



Note: [View Sample in GitHub](#)

Change min-max tick value

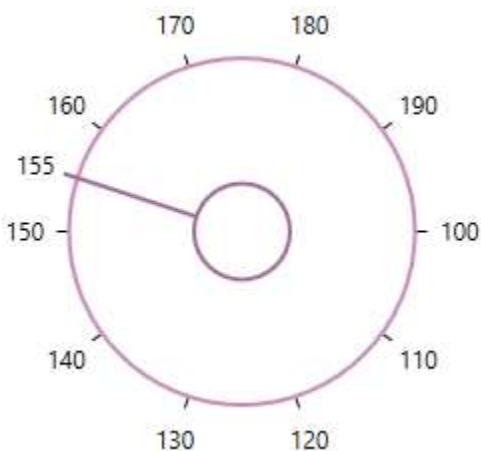
You can change the minimum and maximum ticks value of the `SfRadialSlider` by using the `Minimum` and `Maximum` properties. The default value of `Minimum` property is 0 and `Maximum` property is 100.

XML

```
<syncfusion:SfRadialSlider Minimum="100"
Maximum="200"
Name="radialSlider" />
```

C#

```
radialSlider.Minimum = 100;
radialSlider.Maximum = 200;
```



Note: [View Sample in GitHub](#)

Change start and end position

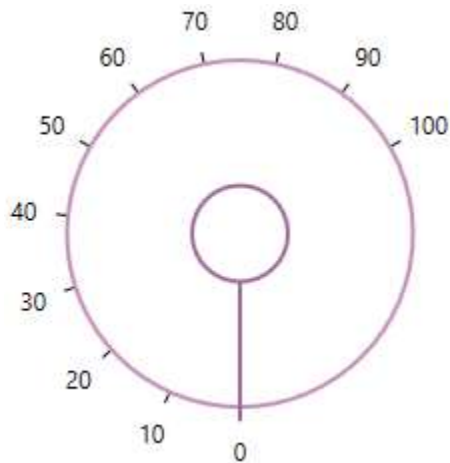
You can change starting and ending position for generating the ticks in the circular track by using the `StartAngle` and `EndAngle` properties. The default value of `EndAngle` property is 0 and `StartAngle` property is 360.

XML

```
<syncfusion:SfRadialSlider StartAngle="90"  
EndAngle="330"  
Name="radialSlider" />
```

C#

```
radialSlider.StartAngle = 90;  
radialSlider.EndAngle = 300;
```



Note: [View Sample in GitHub](#)

Change tick display interval

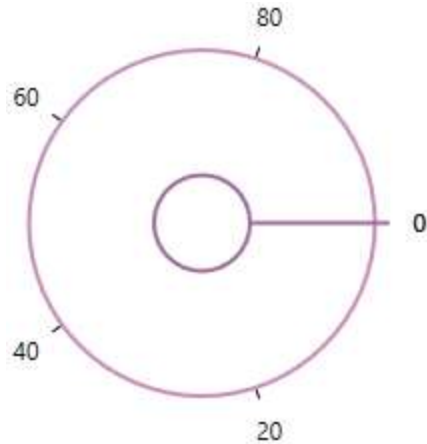
You can change the tick display interval from the Minimum to Maximum values by using the [TickFrequency](#) property. Based on [TickFrequency](#)'s multiples, the display interval is set from Minimum to Maximum value.

XML

```
<syncfusion:SfRadialSlider TickFrequency="20"  
Name="radialSlider" />
```

C#

```
radialSlider.TickFrequency = 20;
```

Note: [View Sample in GitHub](#)

Step interval

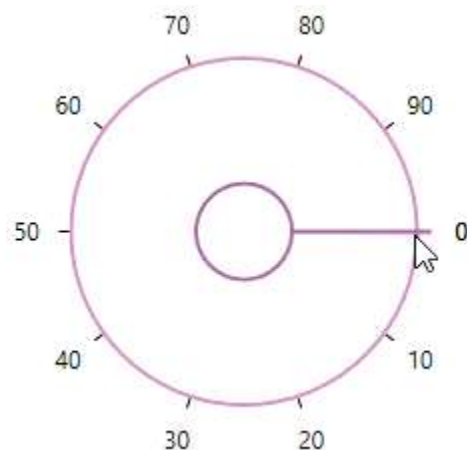
If you want to control the smallest possible range of value to be selected in SfRadialSlider, use the `SmallChange` property. The default value of `SmallChange` property is 0.1. For example, if `SmallChange` is set to 5, then it is only possible to select values that are multiples of 5.

XML

```
<syncfusion:SfRadialSlider SmallChange="5"
Name="radialSlider" />
```

C#

```
radialSlider.SmallChange = 5;
```



Note: [View Sample in GitHub](#)

Change inner and outer rim radius

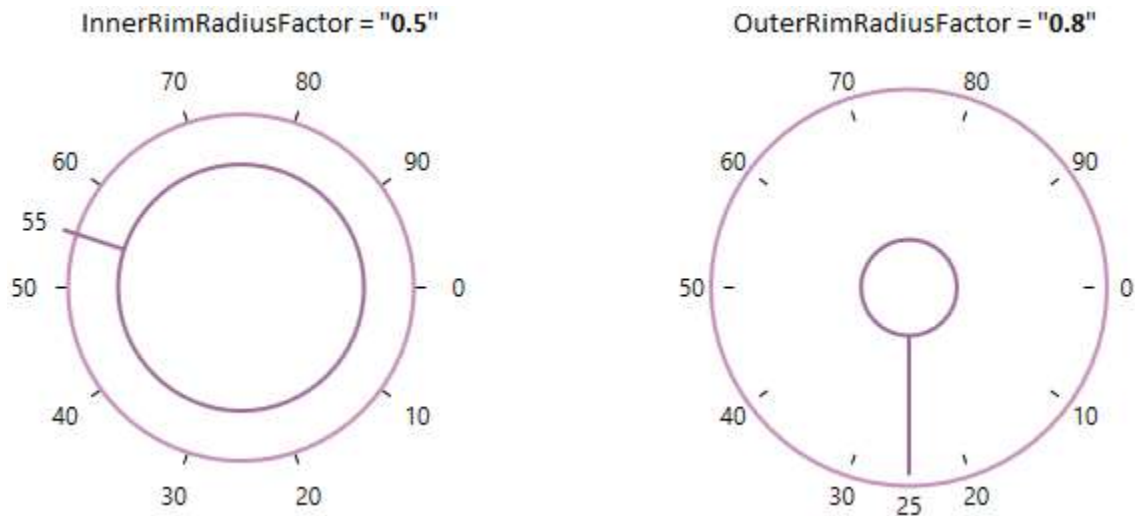
You can change inner and outer rim radius of the SfRadialSlider by using the `InnerRimRadiusFactor` and `OuterRimRadiusFactor` properties. The default value of `InnerRimRadiusFactor` property is 0.2 and `OuterRimRadiusFactor` property is 0.7.

XML

```
<syncfusion:SfRadialSlider InnerRimRadiusFactor="0.5"  
OuterRimRadiusFactor="0.8"  
Name="radialSlider" />
```

C#

```
radialSlider.InnerRimRadiusFactor = 0.5;  
radialSlider.OuterRimRadiusFactor = 0.8;
```



Note: [View Sample in GitHub](#)

Change tick direction

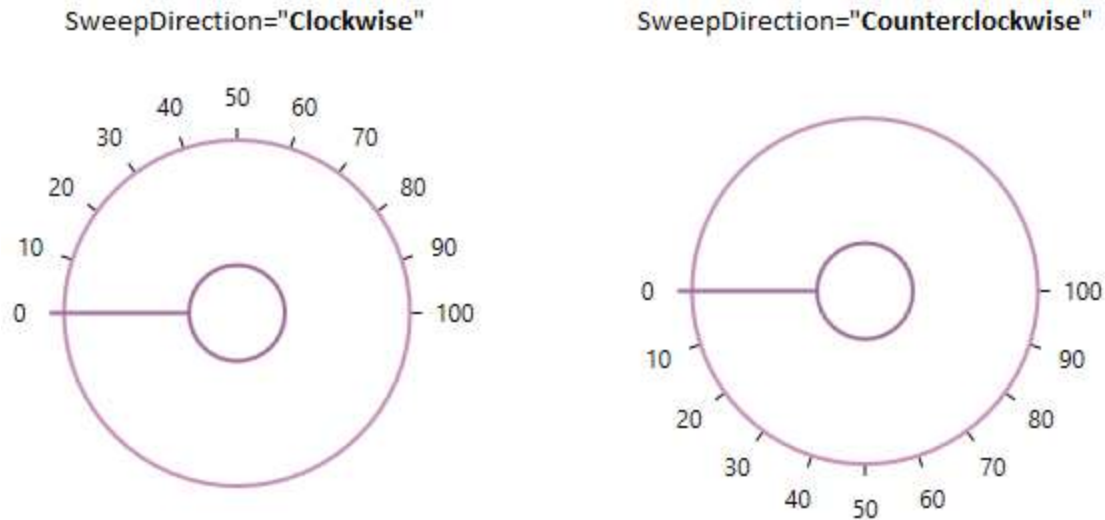
You can change the ticks direction either clockwise or counter-clockwise direction by using the [SweepDirection](#) property. The default value of `SweepDirection` property is `Clockwise`.

XML

```
<syncfusion:SfRadialSlider SweepDirection="Counterclockwise"  
StartAngle="180"  
EndAngle="360"  
Name="radialSlider" />
```

C#

```
radialSlider.SweepDirection = SweepDirection.Counterclockwise;
```



Note: [View Sample in GitHub](#)

Text formatting

You can customize the text format for the specific or all tick labels by handling the [DrawLabel](#) event and setting the [DrawLabelEventArgs.Handled](#) property value as `true`. You can change the content and foreground of the tick labels by using the [DrawLabelEventArgs.Text](#) and [DrawLabelEventArgs.Foreground](#) properties. You can also change the font family and font size of the tick labels by using the [DrawLabelEventArgs.FontFamily](#) and [DrawLabelEventArgs.FontSize](#) properties.

XML

```
<syncfusion:SfRadialSlider DrawLabel="sfRadialSlider_DrawLabel"
Name="sfRadialSlider">
  <TextBlock Text="{Binding ElementName=sfRadialSlider, Path=Value}"
FontSize="15"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</syncfusion:SfRadialSlider>
```

C#

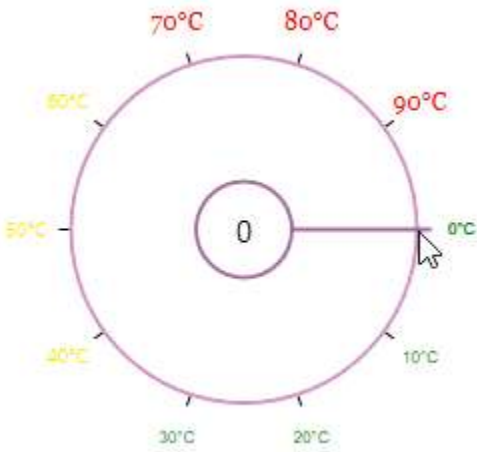
```
sfRadialSlider.DrawLabel += sfRadialSlider_DrawLabel;
```

You can handle the event as follows,

C#

```
private void sfRadialSlider_DrawLabel(object sender, DrawLabelEventArgs e) {
    e.Handled = true;
    e.Text += "°C";
    if (e.Value <= 33) {
        e.FontSize = 8;
        e.FontFamily = new FontFamily("Arial");
        e.Foreground = Brushes.Green;
    }
    else if (e.Value > 33 && e.Value <= 66) {
        e.FontSize = 10;
    }
}
```

```
e.FontFamily = new FontFamily("Courier");
e.Foreground = Brushes.Gold;
}
else {
e.FontSize = 12;
e.FontFamily = new FontFamily("Georgia");
e.Foreground = Brushes.Red;
}
}
```



Note: [View Sample in GitHub](#)

Value changed notification

The selected value changed in `SfRadialSlider` can be examined using `ValueChanged` event. The `ValueChanged` event contains the old and newly selected tick value in the `OldValue` and `NewValue` properties.

XML

```
<syncfusion:SfRadialSlider ValueChanged="RadialSlider_ValueChanged"
Name="radialSlider"/>
```

C#

```
SfRadialSlider radialSlider = new SfRadialSlider();
radialSlider.ValueChanged += RadialSlider_ValueChanged;
```

You can handle the event as follows,

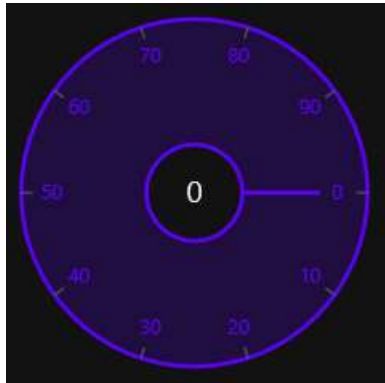
C#

```
private void RadialSlider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e) {
//Get old and new selected tick value
var oldValue = e.OldValue;
var newValue = e.NewValue;
}
```

Theme

SfRadialSlider supports various built-in themes. Refer to the below links to apply themes for the SfRadialSlider,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

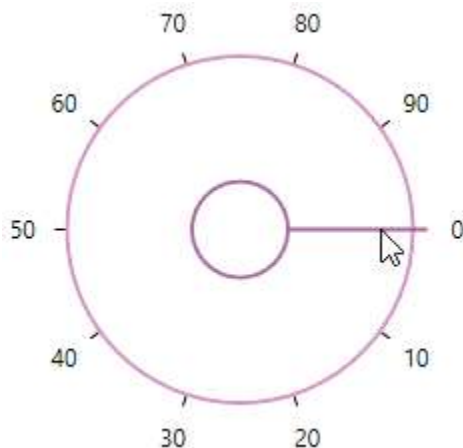


Working With RadialSlider in WPF Radial Slider (SfRadialSlider)

This section explains different UI customization and common features available in [SfRadialSlider](#) control.

Select tick value

You can select any tick value by dragging the pointer along the circular track or clicking on the corresponding track value. You can get the selected value by using the **Value** property. The default value of **Value** property is **0**.



Select tick value programmatically

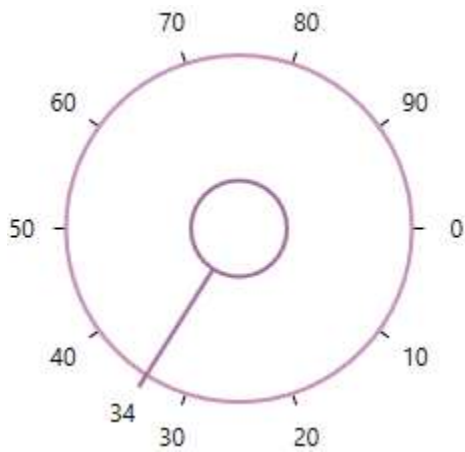
You can select a tick value programmatically by assigning the tick value to the **Value** property.

XML

```
<syncfusion:SfRadialSlider Value="34"
Name="radialSlider" />
```

C#

```
radialSlider.Value = 34;
```



Note: [View Sample in GitHub](#)

Display selected value

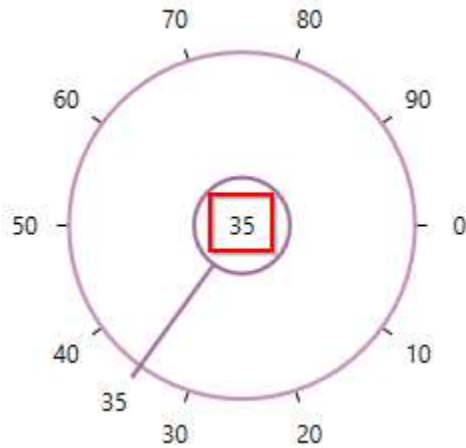
You can display the selected value in the content area of the SfRadialSlider by setting the selected value to the [Content](#) property. The default value of Content property is null.

C#

```
public class ViewModel
{
    private double selectedValue;
    public double SelectedValue {
        get {
            return selectedValue;
        }
        set {
            selectedValue = value;
        }
    }
}
```

C#

```
<syncfusion:SfRadialSlider Content="{Binding SelectedValue,Mode=TwoWay}"
Value="{Binding SelectedValue,Mode=TwoWay}"
Name="radialSlider">
    <syncfusion:SfRadialSlider.DataContext>
    <local:ViewModel/>
    </syncfusion:SfRadialSlider.DataContext>
</syncfusion:SfRadialSlider>
```



Note: [View Sample in GitHub](#)

Custom UI for display content

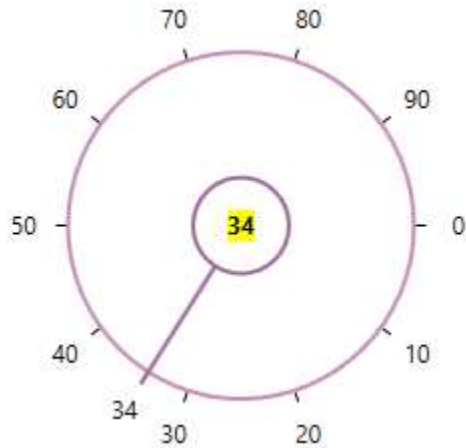
You can customize the appearance of display content area by using the `ContentTemplate` property. The default value of `ContentTemplate` property is `null`. The `DataContext` of the `ContentTemplate` property is `SfRadialSlider`.

C#

```
public class ViewModel
{
    private double selectedValue;
    public double SelectedValue {
        get {
            return selectedValue;
        }
        set {
            selectedValue= value;
        }
    }
}
```

XML

```
<syncfusion:SfRadialSlider Value="{Binding SelectedValue,Mode=TwoWay}"
Name="radialSlider">
  <syncfusion:SfRadialSlider.ContentTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding SelectedValue}"
        FontWeight="Bold"
        Background="Yellow"/>
    </DataTemplate>
  </syncfusion:SfRadialSlider.ContentTemplate>
  <syncfusion:SfRadialSlider.DataContext>
    <local:ViewModel/>
  </syncfusion:SfRadialSlider.DataContext>
</syncfusion:SfRadialSlider>
```



Change min-max tick value

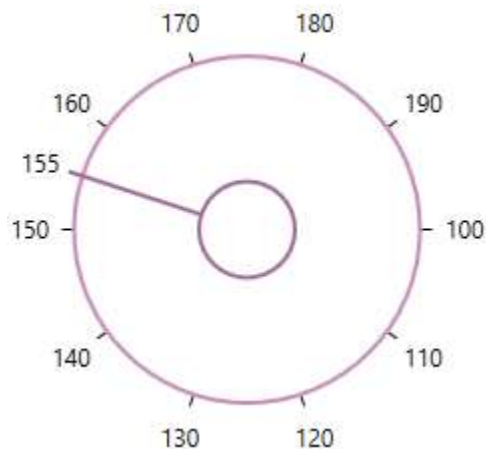
You can change the minimum and maximum ticks value of the `SfRadialSlider` by using the `Minimum` and `Maximum` properties. The default value of `Minimum` property is 0 and `Maximum` property is 100.

XML

```
<syncfusion:SfRadialSlider Minimum="100"
Maximum="200"
Name="radialSlider" />
```

C#

```
radialSlider.Minimum = 100;
radialSlider.Maximum = 200;
```



Note: [View Sample in GitHub](#)

Change start and end position

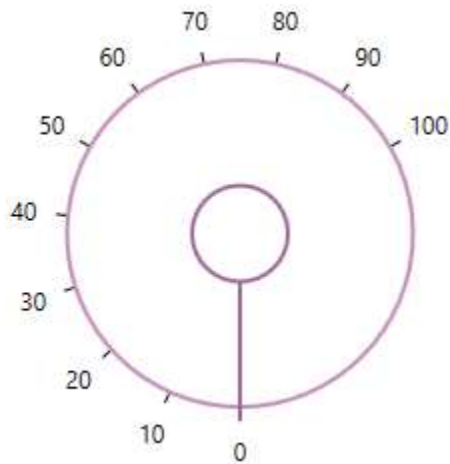
You can change starting and ending position for generating the ticks in the circular track by using the `StartAngle` and `EndAngle` properties. The default value of `StartAngle` property is 0 and `EndAngle` property is 360.

XML


```
<syncfusion:SfRadialSlider StartAngle="90"  
EndAngle="330"  
Name="radialSlider" />
```

C#

```
radialSlider.StartAngle = 90;  
radialSlider.EndAngle = 300;
```



Note: [View Sample in GitHub](#)

Change tick display interval

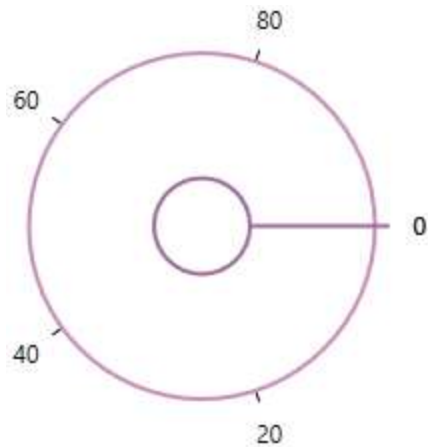
You can change the tick display interval from the **Minimum** to **Maximum** values by using the [TickFrequency](#) property. Based on **TickFrequency**'s multiples, the display interval is set from **Minimum** to **Maximum** value.

XML

```
<syncfusion:SfRadialSlider TickFrequency="20"  
Name="radialSlider" />
```

C#

```
radialSlider.TickFrequency = 20;
```



Note: [View Sample in GitHub](#)

Step interval

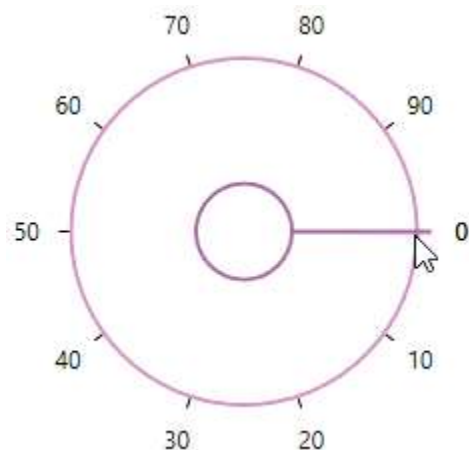
If you want to control the smallest possible range of value to be selected in `SfRadialSlider`, use the `SmallChange` property. The default value of `SmallChange` property is `0.1`. For example, if `SmallChange` is set to `5`, then it is only possible to select values that are multiples of `5`.

XML

```
<syncfusion:SfRadialSlider SmallChange="5"
Name="radialSlider" />
```

C#

```
radialSlider.SmallChange = 5;
```



Note: [View Sample in GitHub](#)

Change tick direction

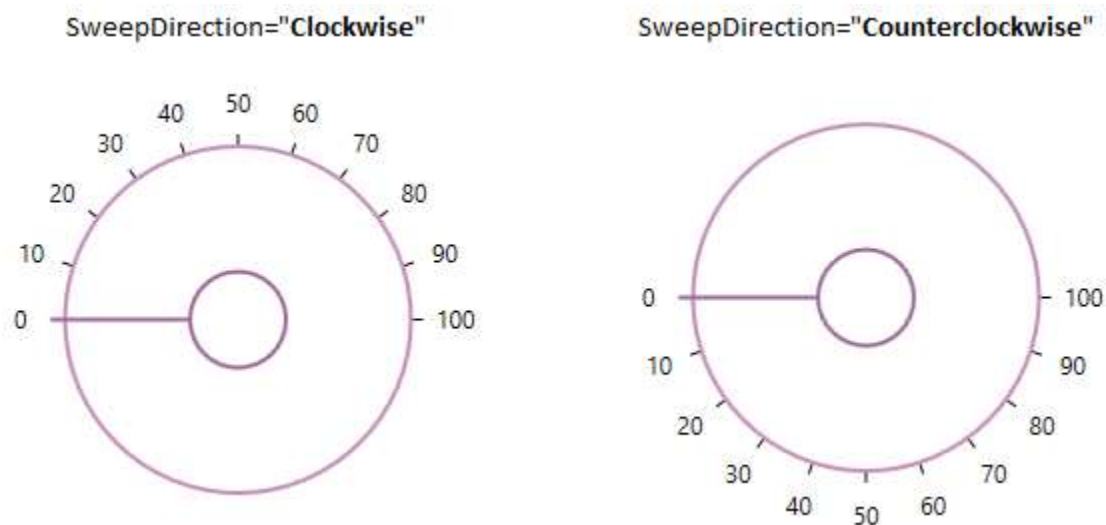
You can change the ticks direction either clockwise or counter-clockwise direction by using the [SweepDirection](#) property. The default value of `SweepDirection` property is `Clockwise`.

XML

```
<syncfusion:SfRadialSlider SweepDirection="Counterclockwise"
StartAngle="180"
EndAngle="360"
Name="radialSlider" />
```

C#

```
radialSlider.SweepDirection = SweepDirection.Counterclockwise;
```



Note: [View Sample in GitHub](#)

Display maximum value

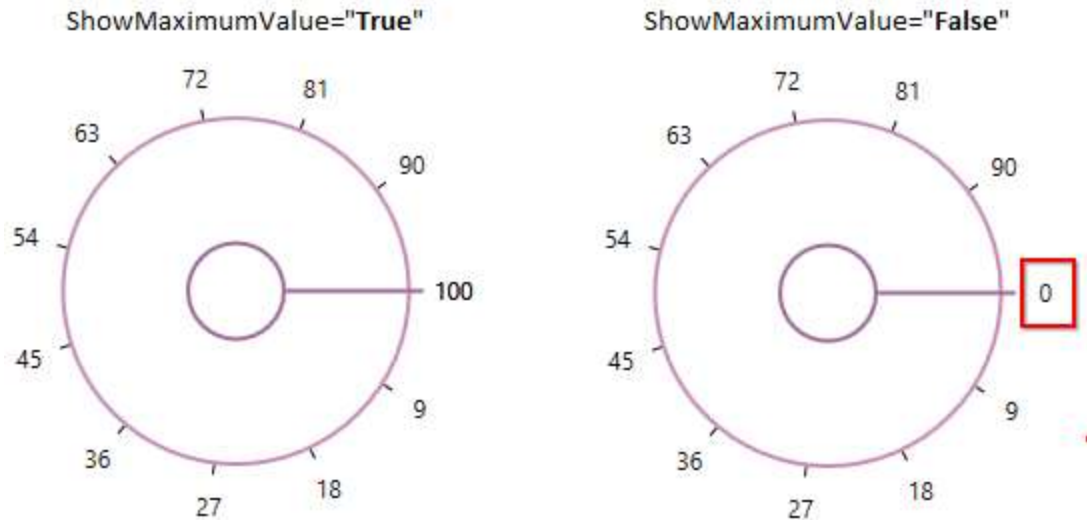
By default, the maximum value is not shown in the **SfRadialSlider**. If you want to show the maximum value when difference of the minimum and maximum value is not a **TickFrequency** multiples, use the [ShowMaximumValue](#) property value as **true**. The default value of **ShowMaximumValue** property is **false**.

XML

```
<syncfusion:SfRadialSlider TickFrequency="9"
ShowMaximumValue="True"
Name="radialSlider" />
```

C#

```
radialSlider.TickFrequency = 9;
radialSlider.ShowMaximumValue = true;
```



Note: [View Sample in GitHub](#)

Change tick radius

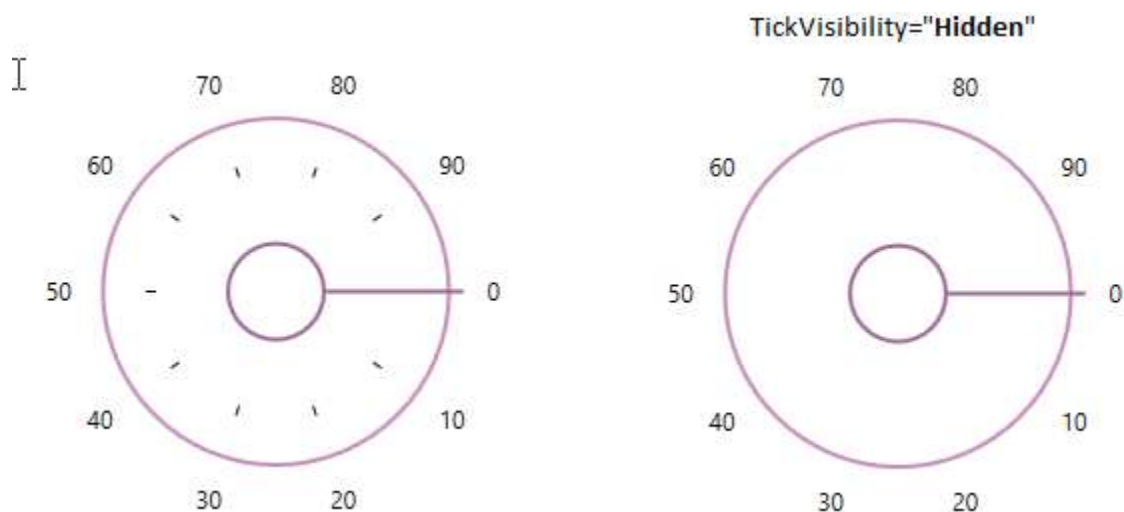
You can change the each tick radius by using the [TickRadiusFactor](#) property. The default value of [TickRadiusFactor](#) property is 0.72. You can hide the ticks by using the [TickVisibility](#) property value as Hidden.

XML

```
<syncfusion:SfRadialSlider TickRadiusFactor="0.5"
TickVisibility="Visible"
Name="radialSlider" />
```

C#

```
radialSlider.TickRadiusFactor = 0.5;
radialSlider.TickVisibility = Visibility.Visible;
```



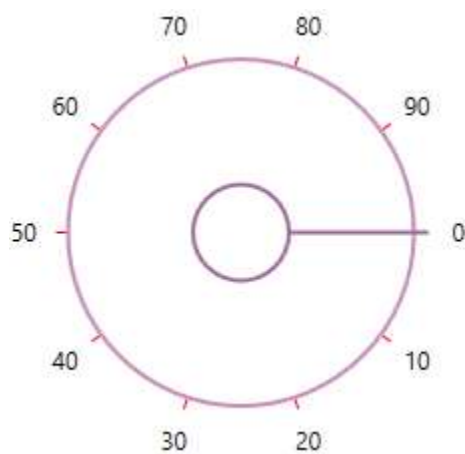
Note: [View Sample in GitHub](#)

Custom UI of ticks

You can customize the appearance of ticks by using the `TickTemplate` property. The default value of `TickTemplate` property is `null`. The `DataContext` of the `TickTemplate` property is `SfRadialSlider` tick value count.

XML

```
<syncfusion:SfRadialSlider Name="radialSlider">
  <syncfusion:SfRadialSlider.TickTemplate>
    <DataTemplate>
      <Border Background="Red"/>
    </DataTemplate>
  </syncfusion:SfRadialSlider.TickTemplate>
</syncfusion:SfRadialSlider>
```



Note: [View Sample in GitHub](#)

Change tick label radius

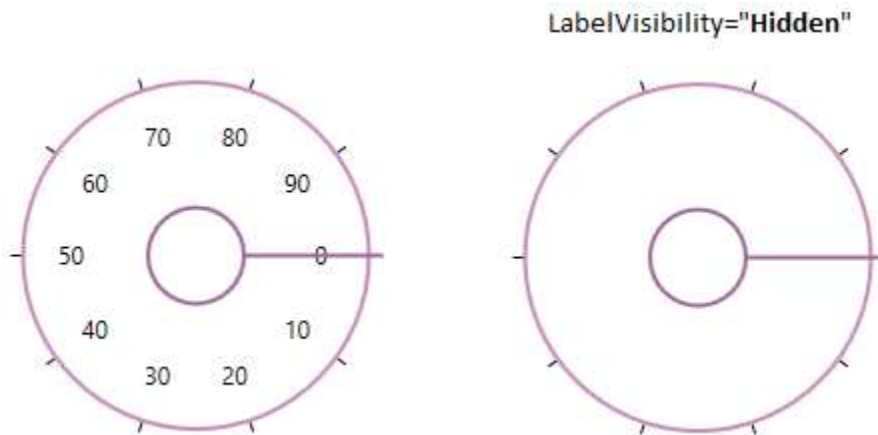
You can change the each tick label radius by using the `LabelRadiusFactor` property. The default value of `LabelRadiusFactor` property is `0.87`. You can hide the tick labels by using the `LabelVisibility` property value as `Hidden`.

XML

```
<syncfusion:SfRadialSlider LabelRadiusFactor="0.5"
  LabelVisibility="Visible"
  Name="radialSlider" />
```

C#

```
radialSlider.LabelRadiusFactor = 0.5;
radialSlider.LabelVisibility = Visibility.Visible;
```



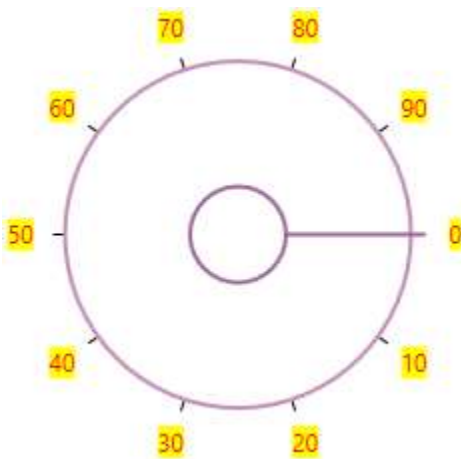
Note: [View Sample in GitHub](#)

Custom UI of tick labels

You can customize the appearance of tick labels by using the `LabelTemplate` property. The default value of `LabelTemplate` property is `null`. The `DataContext` of the `LabelTemplate` property is `SfRadialSlider` tick values.

XML

```
<syncfusion:SfRadialSlider Name="radialSlider">
  <syncfusion:SfRadialSlider.LabelTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding}"
        Foreground="Red"
        Background="Yellow"/>
    </DataTemplate>
  </syncfusion:SfRadialSlider.LabelTemplate>
</syncfusion:SfRadialSlider>
```



Note: [View Sample in GitHub](#)

Change inner rim radius

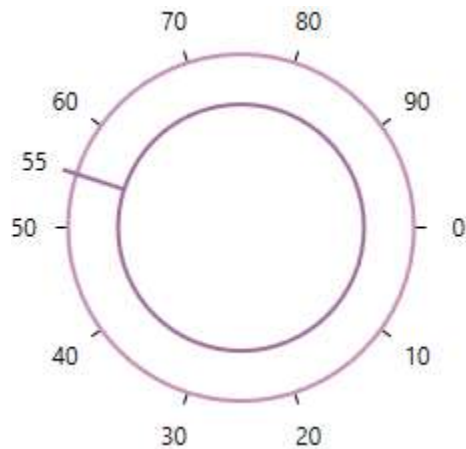
You can change inner rim(circle in the center of radial slider) radius of the `SfRadialSlider` by using the `InnerRimRadiusFactor` property. The default value of `InnerRimRadiusFactor` property is `0.2`.

XML

```
<syncfusion:SfRadialSlider InnerRimRadiusFactor="0.5"  
Name="radialSlider" />
```

C#

```
radialSlider.InnerRimRadiusFactor = 0.5;
```



Note: [View Sample in GitHub](#)

Custom appearance of inner rim

You can change the background of the inner rim by using the [InnerRimFill](#)

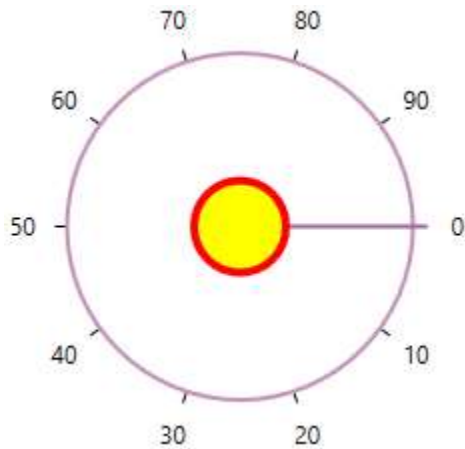
property. You can also change the border color and border thickness of the inner rim by using the [InnerRimStroke](#) and [InnerRimStrokeThickness](#) properties. The default value of [InnerRimStroke](#) property is `LightSlateGray` and [InnerRimStrokeThickness](#) property is `2`.

XML

```
<syncfusion:SfRadialSlider InnerRimFill="Yellow"  
InnerRimStroke="Red"  
InnerRimStrokeThickness="4"  
Name="radialSlider" />
```

C#

```
radialSlider.InnerRimFill = Brushes.Yellow;  
radialSlider.InnerRimStroke = Brushes.Red;  
radialSlider.InnerRimStrokeThickness = 4;
```



Note: [View Sample in GitHub](#)

Change outer rim radius

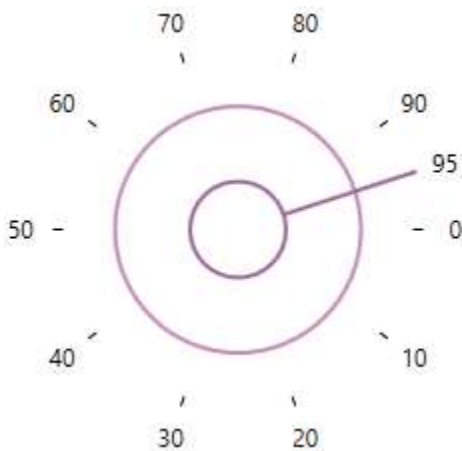
You can change outer rim radius of the SfRadialSlider by using the [OuterRimRadiusFactor](#) properties. The default value of [OuterRimRadiusFactor](#) property is 0.7.

XML

```
<syncfusion:SfRadialSlider OuterRimRadiusFactor="0.5"
Name="radialSlider" />
```

C#

```
radialSlider.OuterRimRadiusFactor = 0.5;
```



Note: [View Sample in GitHub](#)

Custom appearance of outer rim

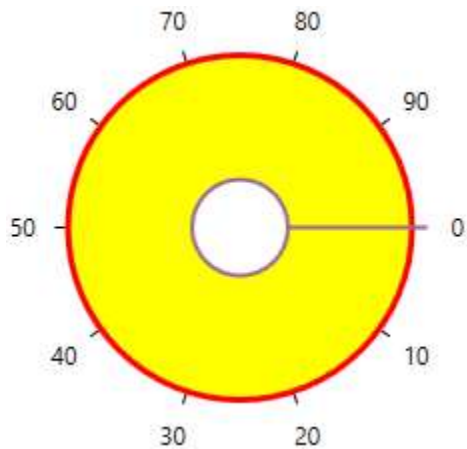
You can change the fill color of the outer rim by using the [Background](#) property. You can also change the border color and border thickness of the outer rim by using the [OuterRimStroke](#) and [OuterRimStrokeThickness](#) properties. The default value of [OuterRimStroke](#) property is Rosy Brown and [OuterRimStrokeThickness](#) property is 2.

XML


```
<syncfusion:SfRadialSlider Background="Yellow"  
OuterRimStroke="Red"  
Name="radialSlider" />
```

C#

```
radialSlider.Background = Brushes.Yellow;  
radialSlider.OuterRimStroke = Brushes.Red;
```



Note: [View Sample in GitHub](#)

Change selection pointer radius

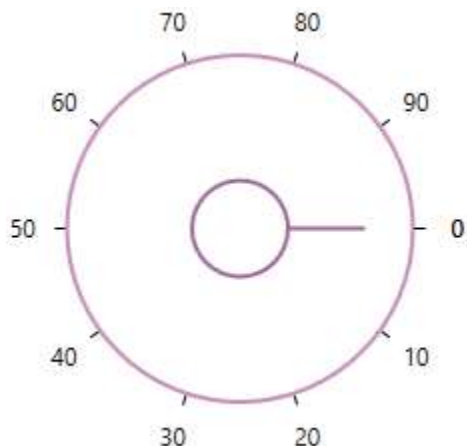
You can change the selection pointer radius by using the [PointerRadiusFactor](#) property. The default value of `PointerRadiusFactor` property is 0.75.

XML

```
<syncfusion:SfRadialSlider PointerRadiusFactor="0.5"  
Name="radialSlider" />
```

C#

```
radialSlider.PointerRadiusFactor = 0.5;
```



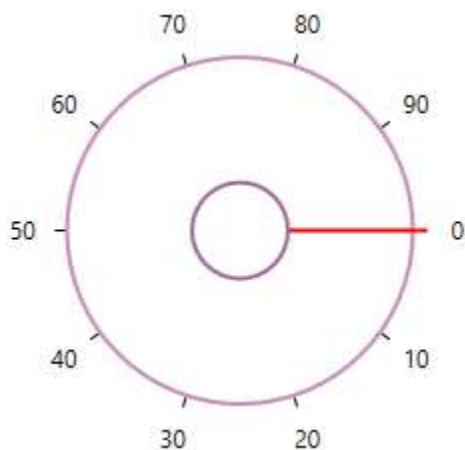
Note: [View Sample in GitHub](#)

Custom UI of selection pointer

You can customize the appearance of selection pointer by using the [PointerStyle](#) property. The `DataContext` of the `PointerStyle` property is `syncfusion:RadialPointer`.

XML

```
<syncfusion:SfRadialSlider Name="radialSlider">
  <syncfusion:SfRadialSlider.PointerStyle>
    <Style TargetType="syncfusion:RadialPointer">
      <Setter Property="Height" Value="2"/>
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="syncfusion:RadialPointer">
            <Border Background="Red"/>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </syncfusion:SfRadialSlider.PointerStyle>
</syncfusion:SfRadialSlider>
```



Note: [View Sample in GitHub](#)

Custom UI of preview selection pointer

You can customize the appearance of preview selection pointer by using the [PreviewPointerStyle](#) property. The `DataContext` of the `PreviewPointerStyle` property is `syncfusion:RadialPreviewPointer`.

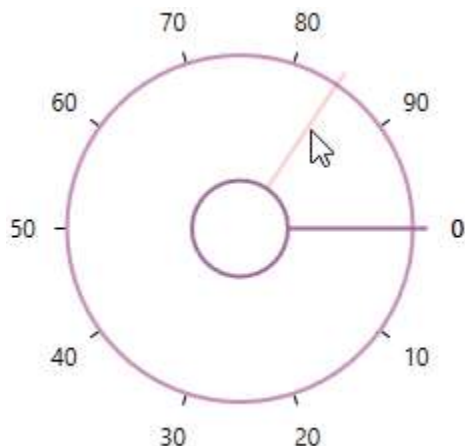
XML

```
<syncfusion:SfRadialSlider Name="radialSlider">
  <syncfusion:SfRadialSlider.PreviewPointerStyle>
    <Style TargetType="syncfusion:RadialPreviewPointer">
      <Setter Property="Height" Value="2"/>
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="syncfusion:RadialPreviewPointer">
            <Border Opacity="0.2" Background="Red"/>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </syncfusion:SfRadialSlider.PreviewPointerStyle>
</syncfusion:SfRadialSlider>
```

```

</Setter.Value>
</Setter>
</Style>
</syncfusion:SfRadialSlider.PreviewPointerStyle>
</syncfusion:SfRadialSlider>

```



Note: [View Sample in GitHub](#)

Text formatting

You can customize the text format for the specific or all tick labels by handling the [DrawLabel](#) event and setting the [DrawLabelEventArgs.Handled](#) property value as `true`. You can change the content and foreground of the tick labels by using the [DrawLabelEventArgs.Text](#) and [DrawLabelEventArgs.Foreground](#) properties. You can also change the font family and font size of the tick labels by using the [DrawLabelEventArgs.FontFamily](#) and [DrawLabelEventArgs.FontSize](#) properties.

XML

```

<syncfusion:SfRadialSlider DrawLabel="sfRadialSlider_DrawLabel"
Name="sfRadialSlider">
<TextBlock Text="{Binding ElementName=sfRadialSlider, Path=Value}"
FontSize="15"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</syncfusion:SfRadialSlider>

```

C#

```
sfRadialSlider.DrawLabel += sfRadialSlider_DrawLabel;
```

You can handle the event as follows,

C#

```

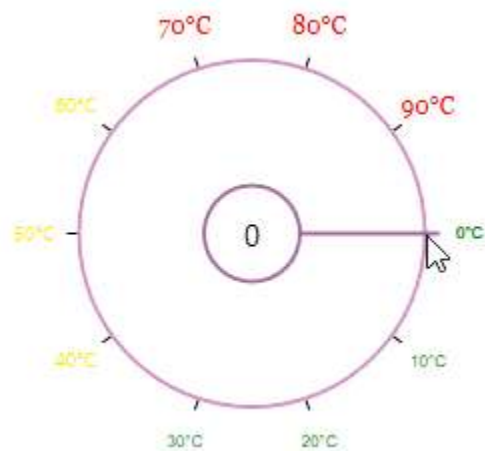
private void sfRadialSlider_DrawLabel(object sender, DrawLabelEventArgs e) {
e.Handled = true;
e.Text += "°C";
if (e.Value <= 33) {
e.FontSize = 8;
e.FontFamily = new FontFamily("Arial");
}
}

```

```

e.Foreground = Brushes.Green;
}
else if (e.Value > 33 && e.Value <= 66) {
e.FontSize = 10;
e.FontFamily = new FontFamily("Courier");
e.Foreground = Brushes.Gold;
}
else {
e.FontSize = 12;
e.FontFamily = new FontFamily("Georgia");
e.Foreground = Brushes.Red;
}
}
}

```



Note: [View Sample in GitHub](#)

Value changed notification

The selected value changed in **SfRadialSlider** can be examined using **ValueChanged** event. The **ValueChanged** event contains the old and newly selected tick value in the **OldValue** and **newValue** properties.

XML

```

<syncfusion:SfRadialSlider ValueChanged="RadialSlider_ValueChanged"
Name="radialSlider"/>

```

C#

```

SfRadialSlider radialSlider = new SfRadialSlider();
radialSlider.ValueChanged += RadialSlider_ValueChanged;

```

You can handle the event as follows,

C#

```

private void RadialSlider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e) {
//Get old and new selected tick value
var oldValue = e.OldValue;

```

```
var newValue = e.NewValue;  
}
```

Appearance in WPF Radial Slider (SfRadialSlider)

This section explains different styling, theming options available in [SfRadialSlider](#) control.

Setting the foreground

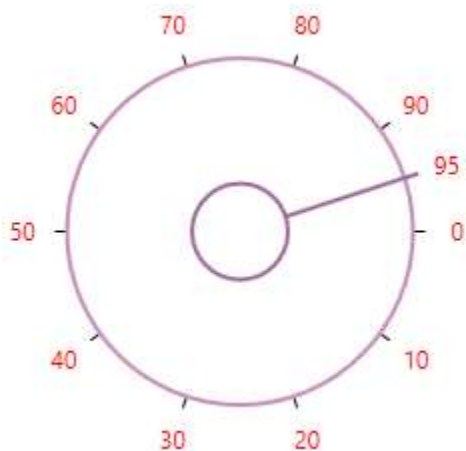
You can change the foreground color of the **SfRadialSlider** by using the **Foreground** property. The default value of **Foreground** property is **Black**.

XML

```
<syncfusion:SfRadialSlider Foreground="Red"  
Name="radialSlider" />
```

C#

```
radialSlider.Foreground = Brushes.Red;
```



Note: [View Sample in GitHub](#)

Setting the background

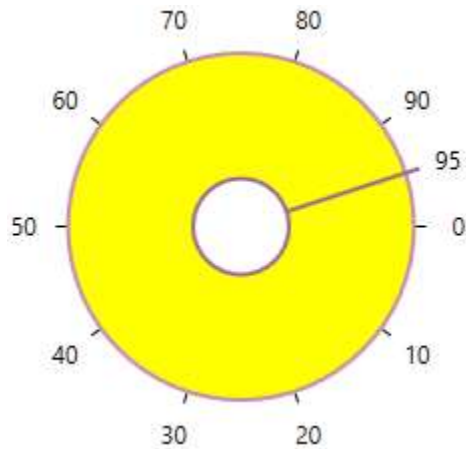
You can change the background color of the **SfRadialSlider** by using the **Background** property. The default value of **Background** property is **White**.

XML

```
<syncfusion:SfRadialSlider Background="Yellow"  
Name="radialSlider" />
```

C#

```
radialSlider.Background = Brushes.Yellow;
```



Note: [View Sample in GitHub](#)

Change flow direction

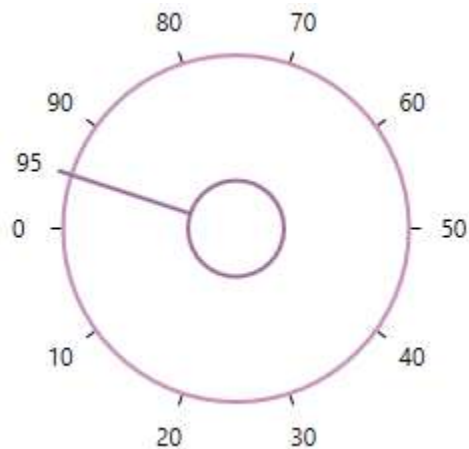
You can change the flow direction of the `SfRadialSlider` layout from right to left by setting the `FlowDirection` property value as `RightToLeft`. The default value of `FlowDirection` property is `LeftToRight`.

XML

```
<syncfusion:SfRadialSlider FlowDirection="RightToLeft"
Name="radialSlider" />
```

C#

```
radialSlider.FlowDirection = FlowDirection.RightToLeft;
```



Note: [View Sample in GitHub](#)

Theme

`SfRadialSlider` supports various built-in themes. Refer to the below links to apply themes for the `SfRadialSlider`,

- [Apply theme using SfSkinManager](#)

- [Create a custom theme using ThemeStudio](#)

