

USER GUIDE

Essential Studio for WPF

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

SfRangeSlider	67
WPF Range Slider (SfRangeSlider) Overview	67
Getting Started with WPF Range Slider (SfRangeSlider)	67
Assembly deployment.....	67
Creating a simple application with SfRangeSlider	67
Theme	69
Visual Structure in WPF Range Slider (SfRangeSlider)	69
Minimum and Maximum in WPF Range Slider (SfRangeSlider)	70
Ticks in WPF Range Slider (SfRangeSlider)	70
Tick Frequency	70
MinorTickFrequency	71
Step Frequency	71
Snaps To	72
Tick Placement	72
Range in WPF Range Slider (SfRangeSlider)	75
ShowRange.....	76
RangeStart.....	76
RangeEnd	77
Drag Selected Range	78
Intermediate Values in WPF Range Slider (SfRangeSlider)	78
Thumb ToolTip in WPF Range Slider (SfRangeSlider)	79
Thumb ToolTip Precision.....	79
Thumb ToolTipFormat	79
Thumb ToolTip Position	80
ThumbInterval.....	82
Orientation in WPF Range Slider (SfRangeSlider)	83
Direction Reversed in WPF Range Slider (SfRangeSlider)	84
Gestures in WPF Range Slider (SfRangeSlider)	85
Label Support in WPF Range Slider (SfRangeSlider)	85
CustomLabels	86
ShowCustomLabels	87
LabelPlacement.....	88
ShowValueLabels	89
ValuePlacement	90
LabelOrientation	92

Touch Support in WPF Range Slider (SfRangeSlider)	93
Styling and Appearance in WPF Range Slider (SfRangeSlider).....	97
InactiveTrackStyle	97
ActiveTrackStyle.....	97
ThumbStyle	99
Tick Customization	100
Events in WPF RangeSlider(SfRangeSlider)	109
How to trigger LabelLoaded event?	109
How to trigger RangeChangedEvent?	109
How to trigger RangeStartChanged event?	110
How to trigger RangeEndChanged event?	111
SfRating	112
WPF Rating (SfRating) Overview	112
Key features	113
Getting Started with WPF Rating (SfRating)	113
Assembly deployment.....	113
Creating Application with SfRating control.....	113
Creating project	113
Adding control via designer	113
Adding control manually in XAML.....	113
Add control manually in C#.....	114
Customize number of rating items	115
Set value.....	115
Precision of selection	115
Theme	116
Precision in WPF Rating (SfRating).....	116
ToolTip in WPF Rating (SfRating)	118
Restrict User Selection in WPF Rating (SfRating).....	119
Appearance and Styling in WPF Rating (SfRating)	119
Set fill color	119
Set stroke color	120
Set stroke thickness	121
ReportViewer	123
WPF ReportViewer Overview	123
Getting Started with WPF Report Viewer	124

Creating ReportViewer through Visual Studio.....	124
Show RDLC Reports.....	127
Load SSRS Reports.....	130
ReportViewer API of Essential Studio WPF.....	132
Properties.....	132
Methods.....	134
Events.....	135
ReportViewer Theme support	135
Limitations in WPF ReportViewer control	145
RDL Specification support	145
Layout Process	145
Unsupported expression.....	145
HTML Formatted Data with Report	145
Supported HTML Tags.....	145
Limitations of Cascading Style Sheet Attributes	146
How to.....	146
Create RDLC Report In VS2010 and Show It in Report Viewer	146
SSRS shared/embedded datasource credential information in ReportViewer	163
Provide the Report Parameters in code behind	164
Provide the Data Source credential information in code behind	164
Can you use SharePoint Integrated Mode in ReportViewer?	164
Can you use Azure SSRS reports in ReportViewer?	165
ReportWriter.....	170
WPF ReportWriter Overview	170
Getting Started with WPF ReportWriter.....	170
Adding ReportWriter to an application	170
Export RDL Reports	173
Export RDLC Reports	175
Saving reports in WPF ReportWriter Control.....	178
Saving report as PDF	178
Saving report as Excel	179
Saving report as Word	180
Saving report as an HTML	181
ReportWriter API of Essential Studio WPF.....	182
Constructors.....	182

Properties.....	182
Methods.....	183
Events.....	183
How to.....	183
Save a report as stream	183
Ribbon	184
WPF Ribbon control Overview	184
Components of Ribbon controls	184
Key features	184
Getting Started with WPF Ribbon.....	185
Add ribbon	185
Set icon for RibbonWindow	186
Set visual styles	187
Add RibbonTab.....	188
Add RibbonBar	189
Add RibbonButton	190
Add DropDownButton	191
Add SplitButton.....	193
Add RibbonGallery	194
Add RibbonComboBox.....	196
Add QAT	197
Add BackStage.....	200
Add application menu	202
Adding custom controls to the title bar	206
Set simplified layout.....	213
Theme	213
BackStage in WPF Ribbon	214
BackStage settings in Ribbon	214
Add BackStageCommandButton.....	217
Setting image to BackStageCommandButton.....	219
Add BackStageTabItem	227
Add BackStage separator	229
BackStage items position	232
Different types of Animation	234
Animation duration.....	234

Placement Customization	236
Customize the BackStageButton visibility.....	266
Application Menu in WPF Ribbon	268
Adding MenuItems in ApplicationMenu.....	268
Adding ApplicationItems in ApplicationMenu	283
Ribbon Items	298
RibbonButton in WPF Ribbon	298
RibbonDropDownButton in WPF Ribbon.....	319
RibbonSplitButton in WPF Ribbon	337
RibbonComboBox in WPF Ribbon.....	355
RibbonGallery in WPF Ribbon	359
RibbonTextBox in WPF Ribbon	385
RibbonCheckBox in WPF Ribbon.....	387
RibbonRadioButton in WPF Ribbon	390
RibbonListBox in WPF Ribbon.....	393
RibbonStatusBar in WPF Ribbon	396
SimpleMenuButton in WPF Ribbon	398
RibbonMenuItem in WPF Ribbon	406
RibbonTabPanellItem in WPF Ribbon	412
RibbonSeparator in WPF Ribbon	413
RibbonItemHost in WPF Ribbon	416
Quick Access ToolBar in WPF Ribbon.....	422
Adding custom item to the ContextMenu	424
How to disable the RibbonContextMenu	425
Add items to QuickAccessToolBar (QAT)	427
Add default QAT items.....	427
Add items to QAT Menu items.....	428
Add items to QAT customize window	430
Add custom QAT items	433
Add items from Ribbon context menu.....	436
Add custom items using RibbonItemHost	437
Add custom RibbonTab and RibbonBar	440
Add items to the customized RibbonTab.....	450
How to disable the customization in Ribbon	454
Simplified Layout in WPF Ribbon	455

Switching between simplified and normal layouts	456
Visibility of the Ribbon items between normal and simplified layout.....	456
Setting image for Ribbon items	458
Customizing the Ribbon during runtime through the QAT window	459
Resizing Ribbon in simplified layout	460
Dealing with Ribbon in WPF Ribbon	460
Three types of RibbonState	460
How to change the RibbonState in run time	463
Resize Ribbon Window	464
RibbonBar Positioning.....	464
Resize based on collapse order.....	476
Setting collapse image for RibbonBar	500
Grouping RibbonTabs using ContextTabGroups.....	506
Creating ContextTabGroup	506
Add ContextTabGroup to the simplified layout.....	506
Multiple ContextTabs.....	508
ContextTabGroup heading.....	509
Changing the visibility at run time	510
Creating ContextTabGroup in MVVM	511
Detecting selection changes in RibbonTab	523
Dealing with Ribbon Items in WPF Ribbon	524
Adding items to ButtonPanel in XAML.....	524
Adding items to ButtonPanel in code behind	525
Changing size of ribbon items.....	526
List of controls which support SizeForm	526
Add command to RibbonLauncherButton	527
Set RibbonBar LauncherButton Command	527
Show HelpButton in RibbonWindow	528
Serialization and Deserialization in WPF Ribbon	530
Use case scenarios	530
Persist ribbon states any time while running the application	531
Saving ribbon states.....	531
Save and load many ribbon states	533
Persisting ribbon states by XML writer	535
Reset Ribbon States	537

Delete ribbon states.....	537
Ribbon Merge in WPF Ribbon.....	538
Creating MDI window and enabling menu merging	538
Merge Type	544
Merge Order	545
Merging and Unmerging in code	547
Ribbon ModelTab in WPF Ribbon	548
Use case scenarios	548
Adding modal tabs to an application	549
Add Modal Tab to the simplified layout	549
How to handle modal tabs in ribbon	551
ToolTip in WPF Ribbon.....	552
Adding tooltip for ribbon items	552
Setting the ToolTip for the upper and lower half of the split button	554
Options for inserting help text in ScreenTip	555
KeyBoard Support in WPF Ribbon	556
Adding KeyTip to the ribbon items	556
KeyTip for QAT items	557
How to access particular item in ribbon using KeyTip.....	558
How does BackStage can be accessed using KeyTip.....	560
Touch Support in WPF Ribbon	561
Styling and Templates in WPF Ribbon	562
Theme	562
Patterns and Practices in WPF Ribbon.....	563
Ribbon with MVVM.....	563
Practice with PRISM	570
How to.....	576
Handle Pop-Up Opening and Closing Event in ApplicationMenu	576
SfRichTextBoxAdv	577
WPF RichTextBox (SfRichTextBoxAdv) Overview.....	577
Features	577
Getting Started with WPF RichTextBox (SfRichTextBoxAdv)	578
Assembly Reference.....	578
Adding SfRichTextBoxAdv to an application	578
Using SfRichTextBoxAdv as a standard RichTextBox	579

Creating Document editor with Ribbon.....	581
Theme	584
Document Structure in WPF RichTextBox (SfRichTextBoxAdv)	585
Document Properties in WPF RichTextBox (SfRichTextBoxAdv)	587
Word Count.....	587
Paragraph Count	587
Page Count	587
Current Page number.....	588
Layout Types in WPF RichTextBox (SfRichTextBoxAdv).....	589
Pages	589
Continuous	590
Block.....	590
Commands in WPF RichTextBox (SfRichTextBoxAdv).....	591
UI Command to access character formatting	592
List of available Commands	592
Selection in WPF RichTextBox (SfRichTextBoxAdv)	603
Multi Selection	607
Apply Formatting for selection	607
Binding Selection format properties.....	608
Keyboard shortcuts to perform selection	609
How to show blinking cursor and selection highlight even when the control lost focus	611
How to determine the editing context type	611
How to delete the selected content	612
Find and Replace in WPF RichTextBox (SfRichTextBoxAdv)	612
Replacing existing text	614
Options Pane.....	615
Spell Check in WPF RichTextBox (SfRichTextBoxAdv).....	615
Adding Custom Dictionaries.....	616
Multilingual Spell Check Support	617
Clipboard in WPF RichTextBox (SfRichTextBoxAdv)	617
UI Command to access clipboard operations	617
Undo Redo in WPF RichTextBox (SfRichTextBoxAdv).....	618
UI Command to perform undo/redo operations.....	618
Enable/Disable Undo Redo	618
List in WPF RichTextBox (SfRichTextBoxAdv).....	619

Single Level List	620
Multilevel List.....	620
Adding List.....	621
Level overrides	624
Editing list.....	626
Hyperlink in WPF RichTextBox (SfRichTextBoxAdv).....	626
Hyperlink ScreenTip	628
Hyperlink Navigation.....	630
Image in WPF RichTextBox (SfRichTextBoxAdv)	631
Image Resizer	631
Text wrapping style	631
Positioning the image	632
Shapes in WPF RichTextBox (SfRichTextBoxAdv).....	632
Supported shapes	632
Text box Shape	632
Shape Resizer	632
Text wrapping style	633
Positioning the shape.....	633
Table in WPF RichTextBox (SfRichTextBoxAdv)	633
UI Commands for accessing table	635
Comment in WPF RichTextBox (SfRichTextBoxAdv)	637
UI Commands for accessing comment.....	638
Customizing comment visual style.....	639
Document Styles in WPF RichTextBox (SfRichTextBoxAdv).....	640
Default style	641
Style hierarchy	641
Create new style	642
Modify an existing style	643
Apply style.....	644
Clear formatting.....	645
Import and Export in WPF RichTextBox (SfRichTextBoxAdv)	645
UI Commands for importing/exporting documents	647
Asynchronous import settings	647
Events to notify document starts and completes loading and saving	648
Printing Contents in WPF RichTextBox (SfRichTextBoxAdv)	648

UI Command for printing	649
Mini Toolbar in WPF RichTextBox (SfRichTextBoxAdv)	649
Enable/Disable Mini Toolbar.....	650
Automatic Suggestion in WPF RichTextBox (SfRichTextBoxAdv).....	650
Dialogs in WPF RichTextBox (SfRichTextBoxAdv)	661
UI Commands for accessing dialogs.....	661
Customizing dialogs	662
Localization in WPF RichTextBox (SfRichTextBoxAdv)	666
Setting Current UI Culture	666
Adding Resource file	667
MVVM in WPF RichTextBox (SfRichTextBoxAdv).....	668
Creating a View Model.....	668
Implementing extension class for SfRichTextBoxAdv	672
Creating XAML View.....	676
Virtualization in WPF RichTextBox (SfRichTextBoxAdv)	677
Styles and Templates in WPF RichTextBox (SfRichTextBoxAdv).....	677
Styling the SfRichTextBoxAdv	695
Setting Background for WPF RichTextBox	697
Setting Background for Document Pages	700
FAQ Section.....	702
Opening large size documents in WPF SfRichTextBoxAdv control	702
SfScheduler	702
WPF Scheduler (SfScheduler) Overview	702
Key features	702
Getting Started with WPF Scheduler (SfScheduler).....	705
Assembly deployment.....	705
Create simple application with SfScheduler	705
Change different SfScheduler Views.....	706
Appointments	707
Change first day of week.....	712
Show busy indicator	713
Theme	713
Day and Week Views in WPF Scheduler (SfScheduler)	714
Change time interval.....	714
Change time interval height.....	715

Flexible working days and working hours	715
Special time regions	717
Full screen scheduler	722
Change time ruler size	723
Minimum appointment duration	724
Minimum display appointments count in all day panel	724
Time text formatting	725
View header	726
Timeline Views in WPF Scheduler (SfScheduler)	728
Change time interval	728
Change time interval width	729
Flexible working days and working hours	730
Change days count	731
Blackout dates	732
Special time regions	733
Full screen scheduler	736
Change time ruler size	737
Minimum appointment duration	738
Time text formatting	738
View header	739
Appointment height	742
Month View in WPF Scheduler (SfScheduler)	742
Month agenda view	742
Appointment display mode	744
Appointment display count	745
Month navigation direction	746
Date format	747
View header	748
Leading and Trailing days visibility	751
Blackout dates	752
Show week number	754
Customize week number template	755
Customize month cell appearance	756
Customize month view appointments	758
Customize more appointments indicator in month cell	761

Resource Grouping in WPF Scheduler (SfScheduler)	762
Grouping by Resources	762
Resource Grouping types	763
Assigning resources to appointments.....	764
Scheduler Resource Mapping	766
Create business object for Resource	766
Resource header size	769
Resource auto height	770
Resource minimum height.....	771
Visible resource count.....	771
Assign resources to special time regions	773
Appearance customization	775
Appointments in WPF Scheduler (SfScheduler).....	779
Scheduler item source and Mapping	781
Creating business objects	782
Spanned appointments.....	784
All day appointments	785
Recurrence Appointment.....	786
Recurrence pattern exceptions.....	791
Add an exception appointment to the recurrence pattern	797
Appearance customization	799
Load On Demand in WPF Scheduler (SfScheduler).....	800
QueryAppointments event	801
Load On Demand command	802
Load On Demand for recurring appointment	804
Appointment Editing in WPF Scheduler (SfScheduler)	804
Adding appointments	804
Editing appointment	805
Visible/Collapse the built-in editors in appointment editor window	806
Disable appointment editing.....	808
Delete appointments	808
Appointment Resizing	809
Reminder in WPF Scheduler (SfScheduler)	810
Enable reminder.....	810
Adding reminders.....	811

Creating business object for reminder.....	812
ReminderAlertOpening event.....	814
ReminderAlertActionChanged events	814
Appointment drag and drop in WPF Scheduler (SfScheduler).....	815
Disable drag and drop	815
Show/Hide the time indicator on appointment dragging.....	816
Appointment dragging time indicator text formatting	816
AppointmentDragOver event	817
AppointmentDragStarting event.....	817
AppointmentDropping event.....	818
Time Zone in WPF Scheduler (SfScheduler).....	818
Create appointments in different time zones.....	822
Display Appointments based on client's time zone	823
Display appointments based on Scheduler time zone.....	823
Display appointments at same time everywhere regardless of client's time zone	823
Updating StartTime and EndTime after drag and drop appointment based on time zone.....	823
Date Navigations in WPF Scheduler (SfScheduler)	824
Range for visible dates	824
Programmatic date navigation	824
Programmatic date selection	825
Programmatically change to adjacent dates	825
Allow view navigation	826
Show date picker.....	827
Allowed views	827
Header in WPF Scheduler (SfScheduler)	828
Header height	828
Header date format	828
Appearance customization	829
Events in WPF Scheduler (SfScheduler)	831
CellTapped	831
CellDoubleTapped.....	832
CellLongPressed	832
SelectionChanged	833
SelectionChanging.....	833
ViewHeaderCellTapped.....	834

HeaderTapped.....	834
WeekNumberTapped.....	835
AppointmentTapped.....	835
ContextMenu and Commands in WPF Scheduler (SfScheduler)	836
Cell context menu	836
Appointment context menu.....	837
SchedulerContextMenuOpening event	838
Calendar Types in WPF Scheduler (SfScheduler)	838
Types of Calendar.....	838
DateTime values in Calendar types.....	839
Localization in WPF Scheduler (SfScheduler).....	840
Set Current UI Culture to the Application.....	840
Localization using Resource file	841
Accessibility Support in WPF Scheduler (SfScheduler)	843
Screen reader support	843
Keyboard navigation	847
Migrating from SfSchedule to SfScheduler in WPF Scheduler.....	848
Adding Reference.....	848
Initialization.....	849
Major improvements of SfScheduler	849
SfSpellChecker.....	852
WPF SpellChecker (SfSpellChecker) Overview	852
Control structure.....	853
Features	853
Getting Started with WPF SpellChecker (SfSpellChecker)	853
Control Structure	853
Assembly deployment.....	853
Adding WPF SfSpellChecker to an application.....	854
Fix spelling mistakes using spell check dialog.....	855
Fix spelling mistakes using context menu.....	856
Disable spell checking	857
Get suggestions for misspelled word.....	858
Ignore SpellCheck for particular types of text	859
SpellCheck for any language(culture)	860
Add custom words to dictionary.....	861

Event to notify when spell check is completed.....	861
Theme	862
Custom Dictionary in WPF SpellChecker (SfSpellChecker)	863
Default SpellCheck Dictionary.....	863
Load your own dictionaries for any language.....	863
SpellCheck using Hunspell dictionary	863
SpellCheck using Ispell dictionary	866
SpellCheck using OpenOffice dictionary	868
Add custom words to dictionary.....	871
Switch language(Culture) at runtime.....	874
Appearance in WPF SpellChecker (SfSpellChecker).....	876
Theme	876
SplitButton	877
WPF Split Button Overview.....	877
Key features	877
Getting Started with WPF Split Button	877
Control structure.....	878
Assembly deployment.....	878
Creating simple application with SplitButton	878
Setting label	880
Setting size mode	881
Setting icon template.....	882
Setting icon template selector	885
Setting image	887
Setting icon width and height.....	888
IsDefault mode.....	889
Adding items to Split Button.....	889
Theme	891
Data Binding in WPF Split Button.....	892
Creating model.....	892
Creating view model	892
Bind data from view model.....	893
Bind command from view model.....	894
Command Binding in WPF Split Button	897
Dropdown Menu Items in WPF Split Button.....	901

Setting icon for dropdown menu items	901
Setting icon bar visibility	901
Setting scrollbar visibility	902
Checkable dropdown menu items	904
Resizing dropdown menu	905
Adding custom dropdown menu items	906
Setting icon bar visibility for custom dropdown menu items	907
Dropdown Direction in WPF Split Button	909
Multiline Text in WPF Split Button (SplitButtonAdv)	911
Events in WPF Split Button	912
DropDownOpening	912
DropDownOpened	912
DropDownClosing	913
DropDownClosed	913
Click	913
Events for dropdown menu items	914
Styles and Templates in WPF Split Button	914
Edit appearance in Expression Blend	914
Edit appearance in Visual Studio	916
Themes in WPF Split Button	919
SfSpreadsheet	919
WPF Spreadsheet (SfSpreadsheet) Overview	919
Key Features	919
Choose between SfSpreadsheet and Spreadsheet control	920
Getting Started with WPF Spreadsheet (SfSpreadsheet)	922
Assemblies Deployment	922
Create a Simple Application with Spreadsheet	923
Creating a new Excel Workbook	926
Opening an existing Excel Workbook	926
Saving the Excel Workbook	927
Displaying Charts and Sparklines	928
Working With Spreadsheet in WPF Spreadsheet (SfSpreadsheet)	928
Accessing the Worksheet	928
Accessing the Grid	929
Setting the ActiveSheet programmatically	930

Accessing the cell or range of cells	930
Accessing the value of a cell	931
Setting the value or formula to a cell.....	931
Clearing the value or formatting from a cell.....	931
Refreshing the view	931
Scrolling the Grid programmatically	932
Formula Bar.....	932
Identify whether the workbook is modified or not	932
Suppress message boxes in Spreadsheet	932
Suspend and resume formula calculation	933
Close the popup programmatically.....	933
Identify when the active sheet is changed	933
Selection in WPF Spreadsheet (SfSpreadsheet).....	933
Accessing the Current cell.....	934
Accessing the Selected ranges	934
Adding or Clearing the Selection.....	934
Move Current Cell	935
Converting GridRangeInfo into IRange	935
Properties, Methods and Events.....	935
Key Navigation	936
Editing in WPF Spreadsheet (SfSpreadsheet)	937
Editing	937
Data Validation.....	939
Hyperlink	940
Formatting in WPF Spreadsheet (SfSpreadsheet)	941
Cell Background	942
Font	942
Cell Borders	943
Cell Alignment.....	943
Wrap Text.....	943
Merge Cells	944
Number Format.....	944
Built-in Styles	945
Format as Table.....	945
Clear formatting	946

Conditional Formatting in WPF Spreadsheet (SfSpreadsheet)	946
Highlight Cell Rules.....	946
Data Bars	948
Color Scales	949
Icon Sets	950
Themes in WPF Spreadsheet (SfSpreadsheet).....	951
Sorting and Filtering in WPF Spreadsheet (SfSpreadsheet).....	952
Filtering	952
Programmatic Sorting and Filtering	952
Unsupported Features	953
Limitations.....	953
Formulas in WPF Spreadsheet (SfSpreadsheet)	954
Adding Formula into cell	954
Named Ranges	954
Supported functions	955
Rows and Columns in WPF Spreadsheet (SfSpreadsheet).....	974
Insert Rows and Columns.....	974
Delete Rows and Columns	975
Hide Rows and Columns	975
Unhide Rows and Columns	976
Row Height and Column Width	976
Freeze Rows and Columns	976
Unfreeze Rows and Columns	976
Auto Fit Rows and Columns	977
Worksheet Management in WPF Spreadsheet (SfSpreadsheet)	977
Insert and Delete worksheet.....	977
Hide and Unhide worksheets.....	978
Hide or unhide sheet tabs.....	978
Rename a worksheet	978
Worksheet Protection.....	978
Gridlines	979
Headings.....	979
Zooming	980
Events.....	980
Data Management in WPF Spreadsheet (SfSpreadsheet)	981

Import from DataTable	981
Export to DataTable	981
Interactive Features in WPF Spreadsheet (SfSpreadsheet)	982
Clipboard Operations	982
Undo/Redo	983
Context menu	983
Cell Comments	985
Find and Replace in WPF Spreadsheet (SfSpreadsheet)	985
Find	985
Replace All	988
Replace	988
Outline in WPF Spreadsheet (SfSpreadsheet)	989
Group rows and columns	989
Ungroup rows and columns	989
Collapse or Expand Group	989
Change Outline Settings	990
Clear Outlines	990
Shapes in WPF Spreadsheet (SfSpreadsheet)	990
Charts	990
Sparklines	991
Pictures	991
TextBoxes	992
Accessing the selected Shapes	992
Select a Shape Programmatically	992
Clear a Selection	993
Conversion in WPF Spreadsheet (SfSpreadsheet)	993
Convert to Image	993
Convert to PDF	993
Convert to HTML	994
Localization in WPF Spreadsheet (SfSpreadsheet)	994
Set Current UI Culture to the Application	994
Localization using Resource file	994
Modifying the localized strings in Resource file	996
Printing in WPF Spreadsheet (SfSpreadsheet)	997
Ribbon Customization in WPF Spreadsheet (SfSpreadsheet)	997

Using Control Template	997
Using Event	998
Custom Formula in WPF Spreadsheet (SfSpreadsheet)	1000
Cell Customization in WPF Spreadsheet (SfSpreadsheet)	1001
Limitations in WPF Spreadsheet (SfSpreadsheet)	1004
Release memory held by AutomationPeer	1004
TabControlExt	1004
WPF TabControl (TabControlExt) Overview.....	1004
Key features	1005
Getting Started with WPF TabControl (TabControlExt)	1005
Structure of TabControl	1005
Assembly deployment.....	1006
Adding WPF TabControl via designer.....	1006
Adding WPF TabControl via XAML	1006
Adding WPF TabControl via C#	1007
Adding TabItem.....	1007
Placement of TabItem.....	1008
Selecting TabItem.....	1009
Closing the tab item	1010
Add new TabItem using new button.....	1012
Tab list menu for switching tabs	1013
Enable or disable tab navigation bar	1015
Show or hide built-in context menu	1016
Localization support.....	1017
Theme	1017
DataBinding in WPF TabControl (TabControlExt)	1018
Adding tab items using data binding	1018
TabItem Header	1020
TabItem content	1021
Editing tab header.....	1021
Select tab in WPF TabControl (TabControlExt)	1022
Select tab item using mouse or keyboard	1022
Tab selection changed notification.....	1023
Load the previously selected tab item content	1023
Display mode of the selected tab item	1024

Customize selected tab item header	1025
Close tabs in WPF TabControl (TabControlExt)	1026
Closing tab item	1026
Show or hide close button	1027
Restrict or allow closing the tab item	1029
Hide or delete item when closing a tab	1030
Tab item closed notification	1031
TabControl Header in WPF TabControl (TabControlExt)	1031
Setting tab item header	1031
Edit tab item header at runtime	1032
Restrict header editing for specific tab item.....	1033
Custom UI for the edit tab item header.....	1033
Setting size and alignment of tab header	1034
Setting image for tab item header.....	1035
Setting tooltip	1036
Hide tab header when there is single tab item.....	1037
Customize tab item header.....	1037
Change tab item background.....	1037
Change tab item foreground.....	1038
Arrange tabs in WPF TabControl (TabControlExt)	1039
Rearrange position of tab items	1039
Tab item order changed notification	1041
Restrict tab item reordering	1042
Tab items alignment	1043
Arrange tab item on single or multiple lines.....	1044
Restrict size of a TabItem.....	1047
Navigate using mouse or keyboard	1047
Navigate using scroll button	1048
Navigate using tab list menu.....	1049
Scroll items using mouse wheel.....	1050
Pin and Unpin TabItems in WPF TabControl (TabControlExt)	1050
Enabling pin and unpin behaviors.....	1050
Functionality of PinButton	1051
Pin and Unpin tab items using PinButton	1051
Pin and Unpin the tab items programmatically	1051

Pin and Unpin tab items using ContextMenu	1052
Re-order pinned tabs	1053
NewButton Feature in WPF TabControl (TabControlExt)	1053
Adding New tab button and new tab item	1053
Select a new tab item while creating it by new button	1054
Auto hide new button when no child tab item.....	1055
Custom template for the new button.....	1056
Change background and border thickness of new button	1056
Context menu in WPF TabControl (TabControlExt)	1057
Default tab item context menu.....	1057
Custom context menu item for the tab items	1059
Hide default context menu items	1062
Custom template for default tab item context menu	1063
Tab list context menu for switching tabs.....	1063
Show hidden tab items in tab list context menu	1064
Show specific type tab items in tab list context menu	1065
Show multi-type tab items in tab list context menu	1066
Custom tab list context menu item	1067
CustomMenuItem as Separator.....	1070
Custom template for the tab list context menu	1072
Custom template for the tab list menu item	1072
Appearance in WPF TabControl (TabControlExt)	1073
Change flow direction	1073
Theme	1074
TabNavigation	1074
WPF Tab Navigation Overview.....	1074
Appearance and structure of the control	1075
Getting Started with WPF Tab Navigation	1076
Assembly deployment.....	1076
Creating simple application with TabNavigation	1076
Theme	1080
Data binding in WPF Tab Navigation	1080
Binding IEnumerable.....	1080
Binding data from XML	1081
Appearance in WPF Tab Navigation.....	1085

Show/hide the Header	1085
Show/hide the NavigationButton	1086
Show/hide the TabStrip	1088
Animation in WPF Tab Navigation	1089
TabSplitter	1091
WPF Tab Splitter Overview	1091
Features	1091
Getting Started with WPF Tab Splitter	1091
Assembly deployment	1091
Adding the TabSplitter control via XAML	1091
Adding the TabSplitter control via C#	1092
Adding splitter item	1092
Tab orientation	1094
Collapsing bottom panel	1095
Theme	1096
Interactive Features in WPF Tab Splitter	1097
Adding TabSplitterItem to the TabSplitter Control	1097
Panel Items	1098
Splitter Page	1099
Collapsing Bottom Panel	1101
Setting BottomPanelHeight of TabSplitter	1102
Orientation in WPF Tab Splitter	1103
Selected Page in WPF Tab Splitter	1104
Layout Related Features in WPF Tab Splitter	1105
Customizing the Appearance of TabSplitter	1105
Hide TabSplitterItem header tab in TabSplitter	1106
TaskBar	1107
WPF TaskBar Overview	1107
Features	1107
Getting Started with WPF TaskBar	1107
Overview of Taskbar control	1108
Why to use our TaskBar control	1108
Creating the Taskbar and its children element using Docking Manager	1108
Theme	1110
Adding a TaskBar Item in WPF TaskBar control	1110

Adding Content to TaskBar Item	1111
Setting Header Image in WPF TaskBar	1112
Setting Group Padding in WPF TaskBar	1113
Changing the size of Expander Button in WPF TaskBar	1114
Collapsing the TaskBar in WPF TaskBar	1115
Setting Group Margin in WPF TaskBar	1116
Setting Group Width in WPF TaskBar	1117
Changing Orientation of Taskbar in WPF TaskBar	1119
GroupOrientationChanged Event	1120
Changing the flow directions in WPF TaskBar	1120
Setting the Animation Speed in WPF TaskBar	1121
Methods handled with Speed Property	1122
Appearance in WPF TaskBar	1122
Theme	1122
SfTextBoxExt	1123
WPF Autocomplete (SfTextBoxExt) Overview	1123
Getting Started with WPF Autocomplete (SfTextBoxExt)	1124
Assembly deployment	1124
Creating a simple application	1124
Populating AutoComplete with Data	1126
AutoComplete modes	1128
Selection	1129
Theme	1129
Autocomplete and filtering in WPF Autocomplete (SfTextBoxExt)	1130
AutoComplete source	1130
Custom data	1130
Customize using the ItemTemplate	1131
Filtering options	1132
Prefix characters constraint	1135
Working with case sensitivity	1136
Showing image in token and drop-down	1136
Display a message when suggestions are empty	1137
Restricting the maximum items filtered	1138
Single and multiple selection in WPF Autocomplete (SfTextBoxExt)	1138
Single selection	1138

Multi selection	1138
Multiple selection using tokens	1138
Multiple selection using delimiter	1141
Setting and retrieving SelectedItem	1142
Retrieving SelectedValue	1145
Displaying images.....	1146
Diacritic Sensitivity in WPF Autocomplete (SfTextBoxExt)	1146
Highlighting Matched Text in WPF Autocomplete (SfTextBoxExt)	1147
First occurrence	1147
Multiple occurrence.....	1148
Unmatched.....	1149
Textbox customization in WPF Autocomplete (SfTextBoxExt)	1149
Water mark	1150
Customizing the TextBox.....	1151
Setting the Dropdown Icon	1151
Setting the ClearButton	1152
Dropdown customization in WPF Autocomplete (SfTextBoxExt)	1153
Customize the background	1153
Drop-down placement.....	1153
Setting the maximum height	1155
Open the drop-down on focus.....	1155
Open drop-down with a delay	1156
Customizing the SelectedItem background	1156
How to.....	1157
Perform multi-path search in WPF Autocomplete	1157
SfTextInputLayout.....	1160
WPF TextInputLayout (SfTextInputLayout) Overview	1160
Key features	1161
Getting Started with WPF TextInputLayout (SfTextInputLayout).....	1161
Adding TextInputLayout reference	1161
Initialize TextInputLayout	1161
Adding hint.....	1162
Theme	1162
Supported Input Views in WPF TextInputLayout (SfTextInputLayout).....	1162
TextBox	1163

PasswordBox	1163
ComboBox	1163
ComboBoxAdv	1164
Autocomplete (SfTextBoxExt)	1165
Container Type in WPF TextInputLayout (SfTextInputLayout)	1165
Outlined	1165
Filled	1166
None	1166
Hint Position in WPF TextInputLayout (SfTextInputLayout)	1167
Float	1167
AlwaysFloat	1167
None	1168
Assistive Labels in WPF TextInputLayout (SfTextInputLayout)	1168
Helper text	1168
Error message	1169
Character counter	1170
Custom Icons in WPF TextInputLayout (SfTextInputLayout)	1170
Leading view	1170
Trailing view	1171
Customization in WPF TextInputLayout (SfTextInputLayout)	1172
Focused color	1172
Unfocused color	1172
Error color	1173
Container color	1173
Outline corner radius	1174
Right to Left in WPF TextInputLayout (SfTextInputLayout)	1174
TileView	1175
WPF Tile View Overview	1175
Structure of the TileViewControl	1175
Key features	1176
Getting Started with WPF Tile View	1176
Structure of TileViewControl	1176
Assembly deployment	1176
Adding WPF TileViewControl via designer	1176
Adding WPF TileViewControl via XAML	1177

Adding WPF TileViewControl via C#	1177
Populating items using TileViewItem.....	1178
Populating items using collection binding	1179
Select a TileViewItem.....	1180
Selected item changed notification	1182
Arrange TileViewItem in rows and columns	1182
Minimize or maximize the TileViewItem	1183
Closing TileViewItem.....	1185
Custom UI of TileViewItem	1185
Theme	1186
Data Binding in WPF Tile View	1187
Data binding to Objects	1187
Data binding with XML.....	1188
Virtualization support	1190
TileViewItem header	1190
TileViewItem content.....	1192
Different UI for TileViewItem content	1193
Working with TileView in WPF Tile View	1194
Populating items using TileViewItem.....	1194
Populating items using binding.....	1195
Select a TileViewItem.....	1195
Selected item changed notification	1197
Display TileViewItem splitter	1197
Custom UI of TileViewItem header	1198
Custom UI of TileViewItem content.....	1200
Arrange TileViewItem in WPF Tile View.....	1201
Rearrange position of TileViewItem	1201
Restrict rearranging of TileViewItem	1202
Arrange TileViewItem in rows and columns	1202
Arrange TileViewItem in custom order.....	1203
Change row and column size	1205
Navigate to TileViewItem.....	1206
Navigate to hidden items using scroll bar.....	1206
Change built-in animation duration.....	1207
Disable built-in navigation animation	1208

Close TileViewItem in WPF Tile View	1209
Show close button.....	1209
Closing TileViewItem.....	1209
Closing TileViewItem programmatically	1210
Hide or delete TileViewItem when closing a item	1211
Custom UI of close button	1212
TileViewItem closing notification.....	1213
TileViewItem Header in WPF Tile View.....	1213
Setting TileViewItem header.....	1214
Change minimized and maximized header	1214
Hide the TileViewItem header	1215
Change TileViewItem header height.....	1216
Change TileViewItem header cursor.....	1217
Custom appearance of TileViewItem header	1217
Custom UI of TileViewItem header.....	1220
Custom UI of minimized TileViewItem header	1222
Custom UI of maximized TileViewItem header	1223
Minimize TileViewItem in WPF Tile View	1224
Minimize the TileViewItem	1224
Direction for minimized items	1225
Allocate size for minimized TileViewItem	1225
Change minimized TileViewItem content.....	1226
Custom UI of minimized TileViewItem content.....	1227
Change minimized TileViewItem header	1228
Minimized state changed notification	1228
Maximize TileViewItem in WPF Tile View	1228
Maximize the TileViewItem	1228
Maximize on click the header	1229
Show maximize button only on mouse hover	1230
Hide maximize button.....	1231
Custom UI of the maximize button.....	1231
Change maximized TileViewItem content	1232
Custom UI of maximized TileViewItem content	1233
Change maximized TileViewItem header	1234
Maximized state changed notification.....	1234

Appearance in WPF Tile View	1234
Setting the foreground.....	1235
Setting the background.....	1235
Setting the border	1236
Change flow direction	1237
Theme	1239
SfTimePicker.....	1239
WPF TimePicker (SfTimePicker) Overview.....	1239
Getting Started with WPF TimePicker (SfTimePicker)	1240
Structure of SfTimePicker	1240
Assembly deployment.....	1240
Add control through designer.....	1241
Adding control manually in XAML.....	1241
Add control manually in C\#	1241
Setting the time	1242
Time changed notification	1242
Display the time using the FormatString	1243
Specifying format for the TimeSelector	1243
Set selected value on lost focus.....	1244
Localization support.....	1245
Theme	1245
Setting Time in WPF TimePicker (SfTimePicker).....	1246
Setting Time using property.....	1246
Setting Null Value.....	1246
Setting WaterMark text	1247
Set selected value on lost focus.....	1248
Setting the time using editing.....	1248
Setting the Input Scope for the On-Screen Keyboard.....	1249
Restrict selecting time limit	1249
Time changed notification	1250
Time Formatting in WPF TimePicker (SfTimePicker)	1250
Display the time using the FormatString	1251
Specifying format for the TimeSelector	1251
Customizing DropDown in WPF TimePicker (SfTimePicker)	1252
Change DropDown height.....	1252

Show or hide DropDown button	1252
Time Selector in WPF TimePicker (SfTimePicker)	1253
Change the Cell templates	1253
Change the HourCell Template	1253
Change the MinuteCell Template	1255
Change the MeridiemCell Template	1256
Change size of cells	1257
TimeSelector item spacing	1258
Appearance in WPF TimePicker (SfTimePicker)	1259
Setting the Foreground	1259
Setting the Background	1259
Change flow direction	1260
Theme	1261
TimeSpanEdit	1262
WPF TimeSpan Editor (TimeSpanEdit) Overview	1262
Key features	1262
Getting Started with WPF TimeSpan Editor (TimeSpanEdit)	1262
Control Structure	1262
Assembly deployment	1262
Adding WPF TimeSpanEdit via designer	1263
Adding WPF TimeSpanEdit via XAML	1263
Adding WPF TimeSpanEdit via C\#	1263
Set or change time span value	1264
Change display format of time span	1264
Value Changed Notification	1265
Restrict the time within minimum and maximum time span	1265
Localization support	1266
Theme	1266
Working with TimeSpanEdit in WPF TimeSpan Editor (TimeSpanEdit)	1266
Date, hour and minute field navigation	1267
Increase or decrease the time fields with specific interval	1267
Change the time value	1267
Setting null value	1269
Show watermark when value is null	1270
Change display format of time span	1270

Value Changed Notification	1271
ReadOnly support	1272
Restrict the time within minimum and maximum time span	1272
Appearance in WPF TimeSpan Editor (TimeSpanEdit).....	1272
Setting the background.....	1272
Setting the foreground.....	1273
Change flow direction	1273
Theme	1274
ToolBarAdv.....	1274
WPF ToolBar (ToolBarAdv) Overview	1274
Features	1274
Getting Started with WPF ToolBar (ToolBarAdv).....	1274
Adding ToolBarAdv to a WPF Application.....	1274
Appearance and Structure of the Control	1275
Setting icon template.....	1275
Properties.....	1278
Theme	1281
Dealing with ToolBarAdv control in WPF ToolBar (ToolBarAdv)	1281
Specifying the Position of ToolBarAdv's in a ToolBarTrayAdv	1281
ToolBarAdv with Overflow items	1282
Show or hide Gripper	1283
Orientation of ToolBarTrayAdv	1283
Add or Remove buttons	1284
Hiding the ToolBarItem	1285
ToolBarAdv State in WPF ToolBar (ToolBarAdv).....	1286
Specifying location for floating ToolBarAdv.....	1287
Restrict Docking of ToolBarAdv for a specific position	1287
ToolBarManager in WPF ToolBar (ToolBarAdv).....	1288
Customization in WPF ToolBar (ToolBarAdv)	1289
Customizing Floating ToolBarAdv	1289
Customize FrameworkElement's Style	1289
Theme	1290
SfTreeMap.....	1290
WPF TreeMap (SfTreeMap) Overview	1290
Getting Started with WPF TreeMap (SfTreeMap).....	1290

Configuring the SfTreeMap Control	1290
Customizing the TreeMap Control	1292
Theme	1294
see also.....	1295
WeightValuePath in WPF TreeMap (SfTreeMap)	1295
ColorValuePath in WPF TreeMap (SfTreeMap)	1296
LeafItemSettings in WPF TreeMap (SfTreeMap)	1296
LabelPath.....	1297
LabelTemplate.....	1297
Gap	1298
BorderBrush	1298
TreeMap Levels in WPF TreeMap (SfTreeMap)	1299
TreeMapFlatLevel	1299
TreeMapHierarchicalLevel:	1302
TreeMap Layout in WPF TreeMap (SfTreeMap)	1306
Squarified Layout	1306
SliceAndDiceAuto Layout:	1307
SliceAndDiceHorizontal Layout:	1308
SliceAndDiceVertical Layout:	1309
ColorMapping in WPF TreeMap (SfTreeMap)	1310
TreeMap ColorMapping:	1310
TreeMapLevel ColorMapping:	1310
UniColorMapping	1311
RangeBrushColorMapping	1312
DesaturationColorMapping	1313
PaletteColorMapping	1314
GroupColorMapping	1315
see also.....	1316
TreeMap Legend in WPF TreeMap (SfTreeMap)	1316
Headers and Labels in WPF TreeMap (SfTreeMap)	1318
Headers	1318
Labels	1319
ToolTip Support in WPF TreeMap (SfTreeMap).....	1321
Selection Support in WPF TreeMap (SfTreeMap).....	1322
see also.....	1324

Customizing Leaf Nodes in WPF TreeMap (SfTreeMap)	1324
Drill Down Support in WPF TreeMap (SfTreeMap)	1326
Enabling Drill Down	1326
see also	1328
SfTreeGrid	1328
Getting Started with WPF TreeGrid (SfTreeGrid)	1328
Assembly Deployment	1328
Theme	1341
Data Binding in WPF TreeGrid (SfTreeGrid)	1342
Binding with IEnumerable	1343
Binding with dynamic data object	1343
Binding Complex properties	1343
Binding Indexer properties	1343
AutoExpandMode	1344
Expanding a tree node	1344
Collapsing a tree node	1346
Expand/Collapse a node based on mapping property	1348
LiveNodeUpdateMode	1348
Events	1349
View	1349
Columns in WPF TreeGrid (SfTreeGrid)	1351
Defining Columns	1351
Column manipulation	1358
Resizing Columns	1359
Column drag and drop	1361
Freezing Columns	1362
Stacked Headers	1363
Binding column properties with ViewModel	1365
Column Types in WPF TreeGrid (SfTreeGrid)	1366
TreeGridColumn	1366
TreeGridTextColumnBase	1378
TreeGridTextColumn	1379
TreeGridNumericColumn	1380
TreeGridCurrencyColumn	1381
TreeGridPercentColumn	1383

TreeGridDateTimeColumn	1384
TreeGridCheckBoxColumn	1387
TreeGridTemplateColumn	1388
TreeGridComboBoxColumn	1393
TreeGridHyperlinkColumn	1397
TreeGridMaskColumn	1400
Custom column support.....	1402
How To	1416
Column Sizing in WPF TreeGrid (SfTreeGrid)	1419
Sorting in WPF TreeGrid (SfTreeGrid)	1425
Sort column in double click.....	1426
Sorting order	1426
Multi column sorting.....	1427
Programmatic Sorting	1428
Custom sorting.....	1429
Handling events	1431
Filtering in WPF TreeGrid (SfTreeGrid)	1431
FilterLevel.....	1431
Programmatic filtering.....	1432
UI filtering	1435
Changing filter UI for grid.....	1437
Changing filter UI for a column	1437
Setting default filter popup style for a specific column.....	1438
Check box filtering	1438
Advanced filtering.....	1438
Instant filtering.....	1439
Filtering null values	1441
Changing AdvancedFilter type when loading dynamic ItemsSource.....	1443
Customization using events	1443
Appearance	1445
Editing in WPF TreeGrid (SfTreeGrid)	1450
Support for IEditableObject	1451
Events.....	1454
Programmatically edit the cell	1456
ReadOnly	1457

Mouse and Keyboard operations for UIElement inside Template	1457
Providing keyboard control to UIElement inside CellTemplate.....	1458
Providing mouse control to UIElement inside template.....	1459
Selection in WPF TreeGrid (SfTreeGrid).....	1460
Current cell navigation.....	1460
Selection modes.....	1460
Disable selection for rows and columns	1460
Multiple row selection	1461
Get selected rows	1461
Programmatic selection	1462
Scrolling rows and columns	1464
Mouse and keyboard behaviors	1464
Events.....	1467
Appearance	1470
Customize selection behaviors	1473
Clipboard Operations in WPF TreeGrid (SfTreeGrid)	1478
Copy	1478
Paste.....	1479
Cut	1480
Events.....	1481
Handle the clipboard operations programmatically.....	1485
Node CheckBox in WPF TreeGrid (SfTreeGrid)	1491
Indeterminate State Support	1492
Recursive Checking	1492
Saving and loading Node CheckBox state from the property in data object.....	1493
Disabling CheckBox for certain nodes.....	1494
Collapsing CheckBox for certain nodes.....	1495
Handling Selection based on CheckBox State.....	1498
Events.....	1501
Programmatically Processing Node CheckBox.....	1501
Getting Checked nodes	1502
Data Validation in WPF TreeGrid (SfTreeGrid).....	1502
Built-in validations	1502
Built-in validation using IDataErrorInfo / INotifyDataErrorInfo	1502
Built-in validation using Data Annotation.....	1505

Custom validation through events.....	1506
Error icon and tip customization.....	1507
Validation with CheckBox column	1518
Limitations.....	1519
Interactive Features in WPF TreeGrid (SfTreeGrid)	1519
Context menu.....	1519
Drag and drop row	1526
ToolTip in WPF TreeGrid (SfTreeGrid)	1541
Record cell tooltip	1541
Header tooltip.....	1542
Tooltip customization	1543
Events.....	1548
Rows in WPF TreeGrid (SfTreeGrid).....	1549
Rows in WPF TreeGrid (SfTreeGrid).....	1549
Row indicators and its description.....	1549
Show row index in row header	1549
Rows in WPF TreeGrid (SfTreeGrid).....	1550
Hiding header row.....	1550
Change the orientation of column header text to vertical	1551
Change the position of sort icon in header cell	1553
Customize style of header row	1556
Merge Cells in WPF TreeGrid (SfTreeGrid)	1558
Column wise merging cells by fixed range.....	1559
Merge all cells in an entire parent node	1559
Localization in WPF TreeGrid (SfTreeGrid)	1562
Localize the drag and drop window text in treegrid.....	1562
Localize when the resource file is present in different assembly or different namespace	1565
Edit default culture resource	1565
Styles and Templates in WPF TreeGrid (SfTreeGrid)	1567
Styling Column Header.....	1567
Conditional Styling in WPF TreeGrid (SfTreeGrid)	1570
Conditional Styling in WPF TreeGrid (SfTreeGrid)	1570
Style cells using converter.....	1570
Style cells based on record using converter	1572
Style cells using triggers	1573

Style cells using style selector	1574
Add image to cell	1575
Conditional Styling in WPF TreeGrid (SfTreeGrid)	1576
Style rows using converter	1576
Style rows using style selector	1577
Conditional Styling in WPF TreeGrid (SfTreeGrid)	1578
Grid Lines customization in WPF TreeGrid (SfTreeGrid)	1580
Record rows	1580
Header rows	1583
Limitations	1583
Themes in WPF TreeGrid (SfTreeGrid)	1583
Built-in Themes	1583
Printing in WPF TreeGrid (SfTreeGrid)	1584
Print parent and expanded child nodes	1585
Print customization	1586
Export To Excel in WPF TreeGrid (SfTreeGrid)	1587
Export options	1587
Save options	1591
Export to HTML	1593
Export to mail	1593
Export to XML	1594
Export to CSV	1594
Customize row height and column width	1595
Customize Cell appearance when exporting	1595
Customize exported workbook and worksheet	1599
Performance	1602
How to	1604
Export To PDF in WPF TreeGrid (SfTreeGrid)	1605
Export options	1606
Define header and footer for PDF page	1607
PDF page orientation	1608
Save options	1609
Open exported PDF without saving in disk	1610
Cell appearance customization when exporting	1611
Embed fonts in PDF file	1616

Export parent and expanded child nodes	1616
Export Middle Eastern languages (Arabic and Hebrew) content to PDF	1617
MVVM in WPF TreeGrid (SfTreeGrid)	1618
Bind the SelectedItem property of treegrid	1618
Bind button command to view model	1619
Bind combobox column ItemsSource from view model	1620
Bind view model ItemsSource to ComboBox inside template	1621
Bind columns from view model	1622
Load On Demand in WPF TreeGrid (SfTreeGrid)	1623
Load on demand using command	1623
Load on demand using event	1629
UI Automation in WPF TreeGrid (SfTreeGrid)	1633
Helpers in WPF TreeGrid (SfTreeGrid)	1639
IndexResolver	1639
Prototype table	1640
Dispose	1640
SfTreeNavigator	1640
WPF Tree Navigator (SfTreeNavigator) Overview	1640
Getting Started with WPF Tree Navigator (SfTreeNavigator)	1641
Theme	1642
Populating Items in WPF Tree Navigator (SfTreeNavigator)	1642
Items source	1642
Item template	1644
See Also	1645
Header Template in WPF Tree Navigator (SfTreeNavigator)	1645
Navigation Mode in WPF Tree Navigator (SfTreeNavigator)	1646
Default	1646
Extended	1646
Selected Item in WPF Tree Navigator (SfTreeNavigator)	1647
SfTreeView	1649
WPF TreeView (SfTreeView) Overview	1649
Key features	1649
Getting Started with WPF TreeView (SfTreeView)	1650
Assembly Deployment	1650
Creating simple application with SfTreeView	1650

Theme	1663
Data Population in WPF TreeView (SfTreeView)	1664
Populating Nodes by data binding - Bound Mode	1664
Populating Nodes without data source - Unbound Mode.....	1670
Appearance in WPF TreeView (SfTreeView).....	1672
ItemTemplate.....	1672
BindingContext for ItemTemplate	1673
ItemTemplate Selector.....	1674
Indentation.....	1676
ExpanderWidth	1676
ExpanderPosition	1677
Level based styling	1677
Animation.....	1678
Expand and Collapse in WPF TreeView (SfTreeView)	1679
Expand Action Trigger	1679
Auto Expand Mode	1680
Expand or collapse the nodes based on property of underlying data object.....	1680
Programmatic Expand and Collapse	1682
Expand and Collapse using Keyboard	1683
Events.....	1683
Interactivity in WPF TreeView (SfTreeView)	1683
Interacting with TreeView items.....	1683
Scrolling in WPF TreeView (SfTreeView).....	1684
Bring Into View.....	1684
Horizontal scrolling	1686
Editing in WPF TreeView (SfTreeView)	1687
Edit mode	1688
Programmatic Editing	1689
Revert the edited changes while pressing Escape key.....	1689
Events.....	1691
CRUD Operations in WPF TreeView (SfTreeView)	1692
Add nodes	1692
Delete nodes	1692
Modify nodes	1693
Selection in WPF TreeView (SfTreeView)	1693

UI Selection	1694
Programmatic Selection.....	1695
Select the nodes based on property of underlying data object	1696
Selected items.....	1698
Selected item style.....	1698
Events.....	1698
Key Navigation	1699
FocusBorderColor	1699
FocusBorderThickness	1699
How to add selection on right click.....	1699
Limitation	1700
Checkbox in WPF TreeView (SfTreeView).....	1700
Working with Checkbox in BoundMode	1700
Working with Checkbox in UnboundMode.....	1703
CheckBox State	1705
Get or Set Checked Items.....	1706
Events.....	1706
Drag and drop in WPF TreeView (SfTreeView)	1706
Dragging multiple items.....	1707
Drag and drop events.....	1708
Customizing the drag and drop operation.....	1710
Drag and drop between two TreeView's	1712
Tree lines in WPF TreeView (SfTreeView).....	1714
Enable tree line for root nodes.....	1715
Customizing the tree lines	1716
Context menu in WPF TreeView (SfTreeView)	1718
ContextMenu for Nodes	1718
Built-in Commands.....	1718
Custom Commands.....	1719
Events.....	1722
MVVM in WPF TreeView (SfTreeView)	1723
Binding properties in MVVM pattern	1723
Event to command.....	1726
Load On Demand in WPF TreeView (SfTreeView)	1728
Handling expander visibility.....	1733

On-demand loading of child items.....	1734
Item Height Customization in WPF TreeView (SfTreeView)	1735
Customize Item Height.....	1735
Customize Item height using `QueryNodeSize` event	1736
Autofit item height based on content.....	1736
Limitations.....	1738
Right to left(RTL) in WPF TreeView (SfTreeView)	1738
UpDown	1739
WPF NumericUpdown (UpDown) Overview	1739
Features	1739
Getting Started with WPF NumericUpdown (UpDown)	1740
Structure of UpDown	1740
Assembly deployment.....	1740
Adding WPF UpDown control via designer	1740
Adding WPF UpDown control via XAML	1741
Adding WPF UpDown control via C#.....	1741
Value	1742
Step Value	1742
Number formatting.....	1742
Theme	1743
Restriction in WPF NumericUpdown (UpDown).....	1744
Value	1744
Null value	1744
Watermark	1745
Minimum and Maximum value.....	1745
AllowEdit	1747
Interaction in WPF NumericUpdown (UpDown)	1747
Keyboard and Mouse support	1747
Step	1747
Animation speed	1748
Range Adorner	1748
Number Formatting in WPF NumericUpdown (UpDown)	1749
Decimal digit	1749
Group separator.....	1749
NumberFormatInfo	1749

Culture.....	1750
Text alignment	1750
Styles and Templates in WPF NumericUpdown (UpDown)	1751
Positive color.....	1751
Negative color	1751
Zero color	1752
Focused color	1752
Theme	1752
Wizard Control	1753
WPF Wizard Control Overview	1753
Features	1753
Getting Started with WPF Wizard Control.....	1753
Highlighting features.....	1753
Assembly deployment.....	1753
Creating Application with WizardControl	1753
Creating project	1753
Adding control via designer	1754
Adding control manually in XAML.....	1754
Adding control manually in C#	1755
Adding multiple pages	1755
Theme	1756
Interactive Features in WPF Wizard Control	1757
Populating pages in Wizard Control.....	1757
Populating by Wizard Pages.....	1757
Populating by Data Binding.....	1758
Wizard Page Type.....	1762
Navigation Buttons of Wizard Page	1763
Next Page and Previous Page Navigation	1765
Closing the Wizard Window	1766
Event for Next Button in Wizard Control	1766
Layout Related Features in WPF Wizard Control.....	1767
Setting the Minimum Height for the Interior Wizard Page Header.....	1767
Setting the Banner Background Color.....	1767
Setting the Banner Image	1768
Setting Minimum Width for the Banner Image on the Exterior Wizard Page	1768

Theme	1769
AutoComplete (Classic)	1770
WPF AutoComplete (Classic) Overview	1770
Feature summary	1770
Getting Started with WPF AutoComplete (Classic)	1771
Structure of the AutoComplete control	1771
Add AutoComplete to an application	1771
Theme	1777
Basic Core Features in WPF AutoComplete (Classic)	1777
Using basic core features in an application	1778
Sample link	1778
FilePath Registry Custom Data Source Support in WPF AutoComplete	1778
Adding data source support to an application	1779
Tables for properties, methods, and events	1780
Sample link	1780
Data Binding in WPF AutoComplete (Classic)	1780
Adding data binding to an application	1780
Tables for properties and events	1782
Selection Mode Support in WPF AutoComplete (Classic)	1782
Tables for properties and events	1783
Sample link	1783
Auto Append Support in WPF AutoComplete (Classic)	1784
Adding auto append support to an application	1784
Tables for properties and events	1784
Sample link	1784
Filter Support in WPF AutoComplete (Classic)	1784
Adding filter support to an application	1785
Tables for properties, methods, and events	1785
Sample link	1785
Custom Filtration Support in WPF AutoComplete (Classic)	1785
Using custom filtration support in an application	1786
Sample Link	1786
History Support in WPF AutoComplete (Classic)	1786
Using history support in an application	1786
Tables for methods, and events	1787

Sample link	1787
Popup Resize Support in WPF AutoComplete (Classic)	1787
Adding pop-up resizing support to an application	1787
Sample link	1787
Appearance in WPF AutoComplete (Classic)	1788
Theme	1788
Show the shadow effect	1788
How to	1789
Bind the Business objects in WPF AutoComplete (Classic)	1789
Chart (Classic)	1790
WPF Chart (Classic) Overview	1790
Real World Scenarios	1790
Key Features	1790
User Guide Organization	1791
Document Conventions	1791
Getting Started with WPF Chart (Classic)	1792
Feature Summary	1792
Elaborate Structure of Chart Control	1793
Creating a Simple Chart using Designer	1793
Creating a Data-Binding Application for Chart Control	1801
Class Diagram	1815
Data Binding in WPF Chart (Classic)	1817
IList Data Source	1817
XML Data Source	1818
ObservableCollection Data Source	1819
CollectionViewSource Data Source	1821
LINQ Data Source	1823
Data-Binding in WPF Chart (Classic) for Child Level Properties	1825
Data Binding Support	1826
IChartDataPoint interface	1827
Chart-Area in WPF Chart (Classic)	1829
Adding Chart Area	1829
Multiple Areas	1830
Chart-Area in WPF Chart (Classic) Header	1830
Chart Header	1831

Chart-Area in WPF Chart (Classic) Context Menu.....	1833
Background	1834
Chart Watermark Support	1836
Chart-Area in WPF Chart (Classic) Layout Customization	1839
Synchronization of Chart Axis	1842
IDictionary	1844
Lazy Loading Support for Chart WPF	1845
Revamping 3D charts in Chart WPF Feature	1845
Interactive Cursors	1846
Splitter for SyncChartArea should be implemented	1848
Bind Array kind of Objects in Chart WPF	1849
SmallChange and LargeChange Properties for Chart Area Scrolling Bar	1850
Adding SmallChange and LargeChange Properties for Chart Area Scrolling Bar to an Application	1850
Additional Zooming Functionality for SyncChartAreas	1850
Chart-Series in WPF Chart (Classic).....	1852
Populating Chart Series	1852
Series Customization.....	1854
Chart-Series in WPF Chart (Classic) Types	1857
Chart-Series in WPF Chart (Classic) Look and Feel.....	1858
Chart-Series in WPF Chart (Classic) Template.....	1859
Chart-Series in WPF Chart (Classic) Adornments.....	1860
Chart Segment Labels	1867
Chart-Series in WPF Chart (Classic) Empty Points.....	1870
Applying Different Colors to Chart Series Segments	1874
Highlighting Series.....	1877
Highlighting Data Points.....	1878
Selecting Points	1880
Side-By-Side Series	1882
AutoDiscard Property.....	1883
Controlling the Visibility of Chart Legend Items	1884
Creating Predefined Shapes for Annotation Objects.....	1885
Empty point support for FastLine Chart type	1886
Customization support for FastChart types	1887
Smart Labels Support.....	1888
Technical Indicators	1891

Chart-Types in WPF Chart (Classic)	1900
Line Charts	1900
Bar Charts.....	1910
Column Charts.....	1918
Area Charts.....	1928
Accumulation Charts.....	1938
XY Charts (Bubble and Scatter)	1941
Financial Charts.....	1946
Pie Charts	1961
Polar Chart	1964
Step Charts.....	1973
Break Lines	1973
Heat Map Control	1974
FastChart.....	1979
Custom Charts.....	1981
Chart-Axis in WPF Chart (Classic).....	1983
Indexed X Values	1983
ChartAxis Range	1984
ChartAxis GridLines	1995
ChartAxis Lines	1998
Chart Striplines.....	1999
ChartAxis OriginLines	2004
Chart-Axis in WPF Chart (Classic) Ticks	2005
Chart-Axis in WPF Chart (Classic) Orientation	2007
Inverted Axis	2008
Opposed Axis.....	2009
Multiple Axes	2010
Axis Range Selection	2011
Logarithmic Axes.....	2012
Chart-Axis in WPF Chart (Classic) Header	2015
Chart Striplines.....	2016
Scale Break Support	2017
Retaining Axis Position.....	2020
Chart-Axis in WPF Chart (Classic) Improvements	2022
Smart Axis Labels	2025

Placing axis labels and series segment in between Ticklines.....	2027
Chart-Labels in WPF Chart (Classic)	2029
Chart Font Settings.....	2029
Chart Axis Label.....	2030
Customizing Label Text	2031
Chart Axis Label Rotate	2033
Intersecting Labels	2034
Legend Panel Customization.....	2036
Chart-Appearance in WPF Chart (Classic)	2037
Chart Styles	2037
Chart Skins.....	2038
Chart Animation	2043
Properties.....	2044
Chart Series Effects	2046
Design Time Support in WPF Chart (Classic)	2047
Chart Control Smart Tag Support.....	2047
Chart and ChartArea Legends in WPF Chart (Classic)	2049
Legend Label	2049
User-Interaction in WPF Chart (Classic)	2050
Zooming	2050
Highlighting And Selection	2057
ToolTip	2057
Toolbar	2060
Property Settings Dialog	2062
Customizing Context Menu.....	2068
Built-in Drag-and-Drop Support for Chart Series	2071
ChartAreaBounds.....	2072
Serialization.....	2076
Property Dialog for Chart.....	2079
Adding Scroll Bar to a Chart	2083
Chart-Events in WPF Chart (Classic).....	2085
Chart Axis Events.....	2085
Chart Series Mouse Events	2085
Chart MouseEventArgs	2086
3D-Charts in WPF Chart (Classic)	2087

Enabling 3D Mode.....	2087
Annotations in WPF Chart (Classic).....	2087
Annotations in WPF Chart (Classic) at X-Y Coordinates.....	2087
Annotations in WPF Chart (Classic) At Control Coordinates.....	2089
Annotation Shapes.....	2090
Localization Support in WPF Chart (Classic)	2091
Use Case Scenario	2091
Creating an Application.....	2091
Export Chart To PDF in WPF Chart (Classic).....	2097
Methods.....	2097
Print in WPF Chart (Classic).....	2098
Use Case Scenarios.....	2099
Methods.....	2099
Sample Link	2100
Print in WPF Chart (Classic)ing a Chart	2100
Statistical Formula and Utility Functions in WPF Chart (Classic)	2100
Use Case Scenarios.....	2100
Methods.....	2100
Sample Link	2104
Statistical Formulas	2105
Perform ANOVA Test	2106
Perfrom F-Test	2107
Perform T-Test	2108
Perform Z-Test	2109
Normal Distribution	2110
F-Cumulative Distribution	2111
T-Cumulative Distribution.....	2112
Sparkline in WPF Chart (Classic).....	2113
Use Case Scenarios.....	2113
Properties.....	2114
Types of Sparklines	2115
Drawing Sparkline in an Application	2115
Marker Support.....	2117
Range Band for Sparkline Chart	2119
Timeline in WPF Chart (Classic).....	2120

Use Case Scenario	2120
Appearance and Structure	2121
Feature Summary.....	2121
Adding TimeLine Control.....	2121
Properties.....	2122
Setting the Starting and Ending Date.....	2123
Setting the Starting and Ending Value	2123
How to.....	2124
Diagram (classic)	2132
WPF Diagram (classic) Overview.....	2132
Real World Scenarios	2132
Key Features.....	2136
User Guide Organization.....	2138
Document Conventions	2138
Getting Started with WPF Diagram (classic)	2139
Diagram Architecture.....	2139
Class Diagram	2140
Creating a WPF application.....	2141
Nodes in WPF Diagram (classic).....	2183
Create Node	2193
Node Shapes	2195
Node Content.....	2198
Node Position.....	2201
Node Rotate	2203
Node Resize.....	2205
Node Label	2207
Gripper	2217
Node Selection	2219
Select and Move Nodes	2222
Deleting Node Without its Edges	2222
Customize the Label, Context Menu for Nodes	2223
Customization of Node Movement.....	2223
Resize Handler Customization	2226
Edges, Degree and Neighbors.....	2229
Node Layout.....	2231

Line Connectors in WPF Diagram (classic)	2233
FirstSegmentLength:	2240
LastSegmentLength:.....	2240
AutoAdjustPoints:	2240
Create Line Connector	2242
Setting Constraints for EnableConnection Property	2245
Setting Constraints for EnableConnection Property	2245
Connector Type	2247
Polyline.....	2251
Decorator Shapes	2259
Customize Line Connectors	2270
Line Connector Label.....	2276
Line Bridging.....	2290
Line Routing	2294
Select, Move, Delete LineConnector.....	2296
Customize the Label, Context Menu for LineConnector	2297
Connection Port in WPF Diagram (classic)	2297
Create Connection Port on Node	2299
Adding Connection Port at Runtime	2300
Create Connection Port on Line Connector	2301
PortShape.....	2302
Port Visibility	2303
Customize PortStyle	2303
CustomPathStyle	2305
AllowPortDrag.....	2305
Connections to Ports.....	2306
Groups in WPF Diagram (classic)	2309
Create Group.....	2310
Select Group.....	2314
Edit Group	2315
Connecting Groups.....	2316
Ungroup	2318
Layers	2321
Hidden or Active Layer	2324
Diagram Model in WPF Diagram (classic)	2326

Layout.....	2327
Pictorial Representation of Spacing Properties	2329
Tree Orientation.....	2331
Clear Nodes and Connections	2334
Bind Data to Diagram Control.....	2335
Cyclic path in Hierarchical-Tree Layout.....	2337
Table Expand Mode	2339
Row Count and Column Count.....	2340
Enable Table Layout with Varied Node Sizes	2342
Diagram View in WPF Diagram (classic).....	2344
Create Rulers.....	2351
Specify Bounds	2355
Panning	2356
Creating Page	2357
Page editing option	2358
Clear the Selection List on Right-Click.....	2359
Fit-to-Page Support	2360
Table Layout for Selected Nodes	2362
PageMargin	2365
Virtualization for DiagramControl.....	2366
Measurement Units	2368
Grid Lines	2375
Snapping.....	2380
Zoom Commands	2385
Nudge Commands.....	2387
Clipboard Commands.....	2388
ZOrder Commands	2391
ZOrder Mode.....	2392
Alignment Commands.....	2394
Bottom Alignment.....	2397
Spacing Commands	2397
Delete Command	2398
Sizing Commands	2399
Undo and Redo Command.....	2400
Printing Enhancements for Diagram Page	2402

Drawing Tools	2405
Export Diagram	2414
Export to Clipboard	2418
Touch Support.....	2422
Overview Control	2424
BringIntoCenter.....	2426
Item Selection Mode.....	2427
Context View in WPF Diagram (classic)	2429
Tables for Properties and Methods	2429
Adding ContextViewManager to an Application	2430
Creating Different Views in the ContextViewManager in WPF Diagram.....	2431
SymbolPalette in WPF Diagram (classic).....	2432
Methods for SymbolGroups in SymbolPalette	2432
Methods for SymbolFilters in SymbolPalette	2433
Adding Through SymbolPalette	2452
Customize the SymbolPalette.....	2452
SymbolPalette in WPF Diagram (classic) Serialization	2460
Bind to ItemSource	2461
Serialization in WPF Diagram (classic)	2463
Save Diagram Page.....	2464
Localization	2467
Event Mechanism in WPF Diagram (classic)	2470
Events for Nodes and Connections	2470
General in WPF Diagram (classic)	2475
Select Nodes and Connectors	2476
Move Nodes and Connectors.....	2476
Customize the Label of Nodes and Line Connectors	2476
Behavior Changes in WPF Diagram (classic)	2482
How to.....	2482
Common in WPF Diagram (classic)	2482
Advanced support in WPF Diagram (Classic)	2492
GridDataControl (Classic).....	2498
WPF GridDataControl (Classic) Overview	2498
Key Features.....	2501
User Guide Organization.....	2501

Document Conventions	2501
Feature Summary.....	2502
Data Presentation	2503
Interactive Features	2504
Visual Styles and Expression Blend	2504
Getting Started with WPF GridDataControl (Classic).....	2505
Appearance and Structure of the Grid.....	2505
Class Diagram	2508
Add Essential Grid to an Application.....	2509
GridDataControl in WPF GridDataControl (Classic)	2528
Major Control Classes	2528
Data Model in WPF GridDataControl (Classic).....	2529
ICollectionViewAdv	2529
Grouping in ICollectionViewAdv	2530
Summaries in Grouping.....	2533
Sorting in ICollectionViewAdv.....	2533
Data Binding in WPF GridDataControl (Classic)	2534
Data Binding mechanisms.....	2534
Important Data Binding Properties.....	2535
Data Providers.....	2535
Database Data.....	2536
Business Objects	2538
XAML Binding	2541
Complex Property Binding	2543
Notify Property changes	2545
Data Error Validation	2546
Custom Data Error Validation	2547
Synchronize Current Selection.....	2552
Unbound Columns	2555
Operations on Unbound Columns	2556
UnboundRows.....	2564
Data-Presentation in WPF GridDataControl (Classic)	2566
Grouping	2566
GroupCollapsed Event.....	2570
Sorting	2570

Filters.....	2575
Advanced Filtering	2577
Grid FilterBar.....	2580
Paging Support for GridDataControl.....	2584
Details View.....	2587
Column Drag and Drop	2592
Column Options	2594
ColumnBasedSizing	2596
Summaries	2598
Caption Summaries	2603
Hierarchy.....	2608
Stacked Headers.....	2610
ToolTips	2614
Cell Comments	2616
Column Auto Sizing	2619
Look and Feel	2623
Row Styles	2623
Conditional Formatting	2624
Custom Skin.....	2635
Enhancement of GridDataCommandManager	2654
QueryCellInfoCommand	2655
SortColumnChangingCommand.....	2655
GridDataControl Column Chooser	2656
Adding Column Chooser to an Application	2657
Exporting and Persistence in WPF GridDataControl (Classic).....	2659
Exporting GDC to Excel.....	2659
Export to PDF	2664
Serialization in GridDataControl	2678
Events in WPF GridDataControl (Classic).....	2679
Subscribing to Events	2679
Unsubscribe the events	2680
Performance in WPF GridDataControl (Classic).....	2682
High Frequency Updates.....	2682
PLINQ Support in GridDataControl	2684
Features-that-work-in-PLINQ in WPF GridDataControl (Classic)	2684

Real-Time-Application in WPF GridDataControl (Classic)	2684
Portfolio Grid.....	2684
VS2010-Designer-support in WPF GridDataControl (Classic)	2685
Activating Designer	2685
Basic Properties.....	2687
Column Properties	2687
Row Properties.....	2688
Cell Properties.....	2688
Visible Columns.....	2689
Generate Columns	2689
Generate Columns	2689
Special Cell Types	2690
Clear Columns	2691
Freezable Support in WPF GridDataControl (Classic)	2691
Grid-Localization-Support in WPF GridDataControl (Classic)	2691
Adding Localization to an Application.....	2692
MVVM-Enhancements in WPF GridDataControl (Classic)	2695
View – View Model Communication.....	2695
Adding Commands to a GridData Control	2696
By using a Command with a Custom Parameter	2697
GridTreeControl (Classic)	2700
WPF GridTreeControl (Classic) Overview.....	2700
Key Features.....	2703
User Guide Organization.....	2703
Document Conventions	2703
Feature Summary.....	2704
Overview in WPF GridTreeControl (Classic)Control.....	2704
Interactive Features	2705
Serialization.....	2705
Getting Started with WPF GridTreeControl (Classic)	2705
Appearance and Structure of the Grid.....	2706
Class Diagram.....	2708
Adding Essential Grid to an Application.....	2709
Adding the GridTree Control to a WPF Application	2709
Architecture in WPF GridTreeControl (Classic).....	2719

Accessing the Underlying Grid control.....	2720
GridTree Control Properties in WPF GridTreeControl (Classic)	2721
GridTreeNode Objects in WPF GridTreeControl (Classic)	2724
GridTree Control Events in WPF GridTreeControl (Classic)	2725
Data Population in WPF GridTreeControl (Classic)	2726
Order and Visibility of Columns in the GridTree Control	2726
Columns in WPF GridTreeControl (Classic)	2727
Bound Column.....	2727
Unbound Column	2727
Bound Columns.....	2728
Tables for Properties, Methods, and Events.....	2728
Properties.....	2728
Cell Types	2730
Static	2731
RichText.....	2733
ImageCell.....	2753
Unbound Columns	2756
Interactive Features in WPF GridTreeControl (Classic).....	2760
Selection Support.....	2760
Sorting	2761
Column Sizing.....	2764
Update Mode	2765
Appearance	2766
Blendability	2771
Adding Styles to an Application	2772
Serialization.....	2782
TreeViewAdv (Classic)	2784
WPF TreeViewAdv (Classic) Overview	2784
Features	2784
Getting Started with WPF TreeViewAdv (Classic).....	2785
Assembly deployment.....	2785
Add TreeViewAdv to Project.....	2785
Adding WPF TreeViewAdv via designer	2785
Adding WPF TreeViewAdv via XAML	2786
Adding WPF TreeViewAdv via C#.....	2787

Adding TreeView item to TreeViewAdv control	2787
Set VisualStyle	2788
Setting ItemsSource for TreeviewAdv	2789
Multiple Selection in TreeViewAdv	2790
Drag and Drop in TreeViewAdv	2791
MultiColumn TreeView in TreeViewAdv	2792
Theme	2793
Populating with Data in WPF TreeViewAdv (Classic)	2794
Through XAML	2794
Through programmatically	2795
Through DataBinding	2796
Visual Structure in WPF TreeViewAdv (Classic)	2800
Images for items in WPF TreeViewAdv (Classic)	2801
Node image	2801
Customize image size in TreeViewItemAdv	2802
Expand the item in WPF TreeViewAdv (Classic)	2803
Animation type	2805
Animation speed	2807
Expand animation	2808
Expand/Collapse Images in WPF TreeViewAdv (Classic)	2809
Node editing in WPF TreeViewAdv (Classic)	2810
Setting node in EditMode	2812
Node editing event	2813
Root lines for Items in WPF TreeViewAdv (Classic)	2814
Selecting a Node in WPF TreeViewAdv (Classic)	2815
Selecting an item through programmatically	2815
Enable to allow multiple selection in TreeViewAdv	2816
Selecting an item through AddNodeToSelectedItems collection	2817
Sorting TreeViewItemAdv in WPF TreeViewAdv (Classic)	2818
Sorting field	2819
Dragging TreeView items in WPF TreeViewAdv (Classic)	2820
Transparent dragging image	2821
Fake drag indicator	2822
TreeViewVirtualization in WPF TreeViewAdv (Classic)	2823
VirtualizationMode is Normal	2823

VirtualizationMode is extended.....	2824
LoadOnDemand in WPF TreeViewAdv (Classic).....	2827
Creating a MultiColumnTreeView in WPF TreeViewAdv (Classic)	2829
Header for MultiColumn	2830
Auto-Resize of columns in Multicolumn TreeView.....	2831
Allowing reordering columns	2833
Binding SelectedItem in MVVM Pattern in WPF TreeViewAdv (Classic)	2833
Use case scenario	2833
Customizing Data Templates in WPF TreeViewAdv (Classic).....	2834
ItemTemplate.....	2834
Item template selector	2839
Edit template.....	2841
Edit template selector	2842
Header template	2844
Cell template.....	2845
Appearance in WPF TreeViewAdv (Classic)	2846
Customizing the appearance of the TreeViewAdv.....	2846
Theme	2848
Customizing root lines.....	2849
Styles in WPF TreeViewAdv (Classic)	2851
Setting drag indicator style	2851
Setting expand style	2853
Setting item container style	2855
Schedule (Classic).....	2856
WPF Schedule (Classic) Overview	2856
Introduction	2856
Key Features.....	2857
Getting Started with WPF Schedule (Classic).....	2859
Assembly deployment.....	2859
Create a project	2859
Add control manually in XAML	2859
Add control manually in C#;	2859
Scheduler Views-(Day, Week, WorkWeek, TimeLine and Month)	2860
Day View	2861
Week View	2861

Work Week View	2862
Month View.....	2863
TimeLine View.....	2864
Appointments	2865
Views in WPF Schedule (Classic)	2869
Header date format	2870
Time formatting	2870
Enable auto formatting.....	2871
Change time interval.....	2872
Change time interval height.....	2873
Change between 12-hour and 24-hour format	2874
Change first day of week.....	2876
Non-accessible timeslots	2876
Change non-working days.....	2878
Collapsed hours.....	2879
Change working hours	2880
Current time indicator	2883
Change hours or minutes time label visibility.....	2885
Appearance	2886
Change schedule view settings based on the views at run time	2892
Month View in WPF Schedule (Classic).....	2893
Change header date format.....	2893
Change header background	2894
Change first day of week.....	2895
Change active and inactive month dates background.....	2896
Change the border color	2897
Change the selection background.....	2898
Current day highlighting	2899
Timeline View in WPF Schedule (Classic).....	2900
Header date format	2900
Time formatting	2901
Change time interval.....	2902
Change time interval height.....	2903
Change between 12-hour and 24-hour format	2904
Non-accessible timeslots	2905

Collapsed hours.....	2907
Change working hours	2908
Current time indicator	2911
Change hours or minutes time label visibility.....	2913
Appearance	2914
Appointments in WPF Schedule (Classic).....	2918
Adding appointment	2918
AppointmentCollectionChanged.....	2919
Data Binding.....	2920
Add new appointment	2923
Appointment template customization.....	2925
Types of appointments	2926
Appointment generating behavior	2928
All day appointment panel.....	2930
Appointment tooltip	2931
Change the appointment selection color	2933
Customize the appointment status collection	2934
Appointment Editing in WPF Schedule (Classic)	2935
Editing Appointment.....	2935
Edit recurring appointment	2937
Disable appointment editing.....	2939
Create read only appointment.....	2939
Appointment deleting.....	2940
Appointment resizing.....	2941
Drag and drop	2943
Appointment-Navigation in WPF Schedule (Classic)	2945
PreviousNavigationButtonTemplate.....	2945
NextNavigationButtonTemplate	2946
Reminder in WPF Schedule (Classic).....	2947
Setting reminder for an Appointment	2947
Configuring Reminder Duration	2948
Create a custom binding for ReminderTime.....	2948
Handling Reminder events.....	2949
Time-Zone in WPF Schedule (Classic)	2950
Recurrence Appointment.....	2950

Recurrence Rule	2950
Recurrence Pattern	2951
Recurrence Rule Generator	2953
Creating Custom Recurrence Appointment using Recurrence Builder	2954
RecursiveExceptionDates.....	2957
Import-and-Export in WPF Schedule (Classic)	2959
Exporting in the schedule control.....	2959
Importing in the schedule control	2960
Resources in WPF Schedule (Classic)	2961
Creating Resource for Schedule.....	2962
Adding ResourceType to a resource collection.....	2963
Adding a Resource to a ResourceType.....	2963
Creating appointments by specifying the resource	2963
Adding Resources in code behind.....	2964
SubResource Support.....	2964
N Number of Resources in Day View	2966
Localization in WPF Schedule (Classic).....	2968
Set Current UI Culture to the Application.....	2968
Localization using Resource file	2968
Visible-Dates-customization in WPF Schedule (Classic)	2971
ContextMenuType in WPF Schedule (Classic).....	2972
Commands in WPF Schedule (Classic)	2973
AddNewCommand	2974
EditCommand	2974
DeleteCommand	2974
CopyCommand.....	2974
PasteCommand.....	2974
DragAndDropCommand.....	2974
Event in WPF Schedule (Classic)	2975
SkinStorage (Classic)	2976
WPF SkinStorage (Classic) Overview.....	2976
Feature summary	2977
Built-in skins	2977
Setting VisualStyle in XAML	2977
Setting VisualStyle in C#.....	2977

Active color scheme	2978
Setting ActiveColorScheme property in XAML	2978
Setting ActiveColorScheme property in C#	2979
Metro theme customization	2980
Setting MetroBackgroundBrush property in XAML	2980
Setting MetroBackgroundBrush property in C#	2980
Performance	2981
ResourceDictionary path for Syncfusion themes.....	2981
How to.....	2983
Override Syncfusion Themes in WPF SkinStorage (Classic)	2983
Switch between Skins at Run time in WPF SkinStorage (Classic).....	2984
Switch between Overridden Styles in WPF SkinStorage (Classic).....	2986
Set and Override Visual Style in SkinStorage	2988
Set VisualStyle for Derived in SkinStorage.....	2989
Resource ID for Syncfusion Themes in WPF SkinStorage (Classic)	2990
Touch UI in WPF SkinStorage (Classic).....	2991
SpreadsheetControl (Classic)	2993
WPF SpreadsheetControl (Classic) Overview.....	2993
Create, Open and Save Excel Document.....	2994
Formatting.....	2994
Formulas	2995
Review Options	2995
View Options.....	2995
SpreadsheetRibbon Support	2996
Getting Started with WPF SpreadsheetControl (Classic)	2996
Creating a WPF Application	2996
Adding Spreadsheet Control to WPF Application	2997
Loading Excel Files in Spreadsheet Control.....	2998
Appearance and Structure of the Controls	2998
Architecture	2999
Creating an Excel Document in WPF SpreadsheetControl (Classic)	3000
Open and Save Excel Document in WPF SpreadsheetControl (Classic).....	3000
Open Excel Document.....	3001
Save Excel Documents	3002
Formatting in WPF SpreadsheetControl (Classic)	3003

Conditional Formatting	3003
Number formatting	3005
Center Across Selection	3005
Appearance in WPF SpreadsheetControl (Classic).....	3006
Fonts.....	3006
Merge cells.....	3008
Borders.....	3008
Freeze Panes	3009
Wrap Text Support.....	3009
Overview	3009
Use Case Scenario	3009
Data Management in WPF SpreadsheetControl (Classic).....	3010
Formulas	3010
Data Validation.....	3010
Comments.....	3013
Import and Export from Data Table.....	3014
Clipboard Support	3016
Relative reference Copy and paste	3017
Documents Settings in WPF SpreadsheetControl (Classic).....	3021
Protect and unprotect workbook	3021
Protect and unprotect worksheet.....	3022
Protect/Unprotect worksheet using Command	3023
Encrypt workbook.....	3023
Encrypt workbook using the Commands	3023
Hide and Unhide Worksheet.....	3024
Add or Remove Worksheet.....	3025
Document Settings Options	3026
Links in WPF SpreadsheetControl (Classic)	3028
Bookmarks and Hyperlinks	3028
Command Support in WPF SpreadsheetControl (Classic)	3028
Localization in WPF SpreadsheetControl (Classic).....	3032
SpreadsheetRibbon Support in WPF SpreadsheetControl (Classic)	3034
RichTextBoxAdv (Classic)	3035
WPF RichTextBoxAdv (Classic) Overview.....	3035
The features of the RichTextBoxAdv control include:	3036

Use Case Scenarios.....	3036
Sample Link	3036
Getting Started with WPF RichTextBoxAdv (Classic)	3036
Adding RichTextBoxAdv to an Application.....	3036
Creating the RichTextBoxAdv Control in Visual Studio	3036
Creating the RichTextBoxAdv Control in Expression Blend	3038
Appearance	3039
Paragraph Alignment and Indentation in WPF RichTextBoxAdv (Classic)	3040
Adding Paragraph Alignment to an Application	3040
Text Formatting Using SpanAdv in WPF RichTextBoxAdv (Classic)	3041
Adding SpanAdv to an Application.....	3042
Text Formatting Using HyperlinkAdv in WPF RichTextBoxAdv (Classic)	3042
Add HyperlinkAdv to an Application	3043
Inserting-Images in WPF RichTextBoxAdv (Classic)	3043
Adding Images to an Application	3044
Limitations.....	3044
Inserting-a-UI-Element in WPF RichTextBoxAdv (Classic)	3044
Adding UIElement to an Application.....	3045
Limitations.....	3045
Insert-Tables in WPF RichTextBoxAdv (Classic)	3045
Editing Support	3046
Selection Support.....	3046
Insert n*n Tables.....	3046
Insert n*n Columns	3046
Insert n*n Rows	3046
Delete Table	3046
Delete n*n Rows	3046
Delete n*n Columns.....	3046
Merging.....	3046
Layout-Modes in WPF RichTextBoxAdv (Classic)	3046
Import and Export in WPF RichTextBoxAdv.....	3047
HTML Import/Export.....	3047
DOC Import/Export and DOCX Import/Export.....	3048
XAML Import/Export.....	3050
Text Import/Export	3050

Clipboard-Support in WPF RichTextBoxAdv (Classic)	3051
Commands-Support in WPF RichTextBoxAdv (Classic)	3051
Code-Snippet in WPF RichTextBoxAdv (Classic)	3053
RichTextBoxAdv Methods	3054
Keyboard-Support in WPF RichTextBoxAdv (Classic)	3055
Context-Menu-Support in WPF RichTextBoxAdv (Classic)	3057
Zooming-and-Printing in WPF RichTextBoxAdv (Classic)	3057
Disable-Editing in WPF RichTextBoxAdv (Classic)	3058
RichTextRibbon-Support in WPF RichTextBoxAdv (Classic)	3058
Touch-Events in WPF RichTextBoxAdv (Classic)	3058
How to	3059
Bind the content of the WPF RichTextBoxAdv (Classic) By Using MVVM	3059

SfRangeSlider

WPF Range Slider (SfRangeSlider) Overview

[SfRangeSlider](#) control allows the value range to be selected within the defined minimum and maximum limit. The range can be selected by moving the Thumb control along a track. In terms of appearance, this control is highly customizable and offers many options such as orientation, selection range, custom label support, touch support, snap to tick, tick placement, tooltip support etc.



Key Features:

- **Orientation** — Set the orientation of the RangeSlider either horizontally or vertically.
- **SelectionRange** — Select the range of values using two thumbs.
- **Move the Thumb to the Tapped Position** — Allow the thumb of RangeSlider to tapped position of the track.
- **Styling and Appearance** — The control is completely customizable in terms of the UI. Change the appearance of the control as required.
- **ToolTip Support** — Shows the value of the RangeSlider. It is fully customizable.
- **Gestures** — Provides both KeyGesture and MouseGesture.

Getting Started with WPF Range Slider (SfRangeSlider)

This section describes how to design a [SfRangeSlider](#) control in a WPF application and overview of its basic functionalities.

Assembly deployment

Namespace: Syncfusion.Windows.Controls.Input

Assembly: Syncfusion.SfInput.WPF (in Syncfusion.SfInput.WPF.dll)

Dependent assembly: Syncfusion.SfShared.WPF.dll

Creating a simple application with SfRangeSlider

The [SfRangeSlider](#) control can be added to an application using Visual Studio.

Create the WPF application with [SfRangeSlider](#) control as follows:

1. [Creating project](#)
2. [Adding control via designer](#)
3. [Adding control manually in code](#)

Creating the project

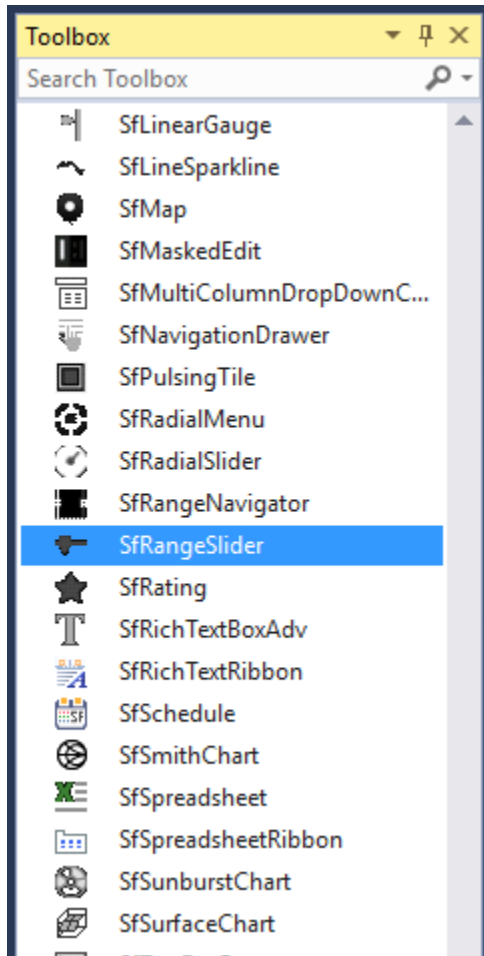
The steps to create a [SfRangeSlider](#) control by using Visual Studio in C# are as follows:

1. Open Visual Studio. 2. On the File menu, select New -> Project. This opens the New Project Dialog box.

Adding a control via designer

[SfRangeSlider](#) control can be added to the application by dragging it from the toolbox and dropping it in a designer view. The following required assembly references will be added automatically:

- Syncfusion.SfInput.WPF
- Syncfusion.SfShared.WPF



XML

```
<editors:SfRangeSlider
Width="500"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Maximum="100"
Minimum="0"
Value="50" />
```

Adding a control manually in code

The following code sample shows how to create the [SfRangeSlider](#) from code-behind.

C#

```
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 500,
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center,
    Minimum = 0,
```

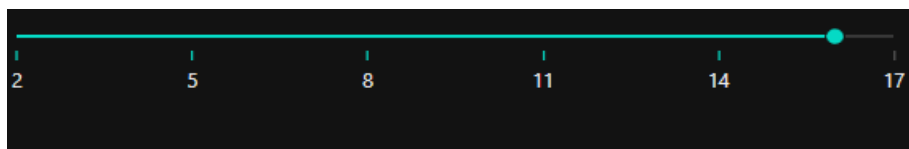
```
Maximum = 100,
Value = 60
};
```



Theme

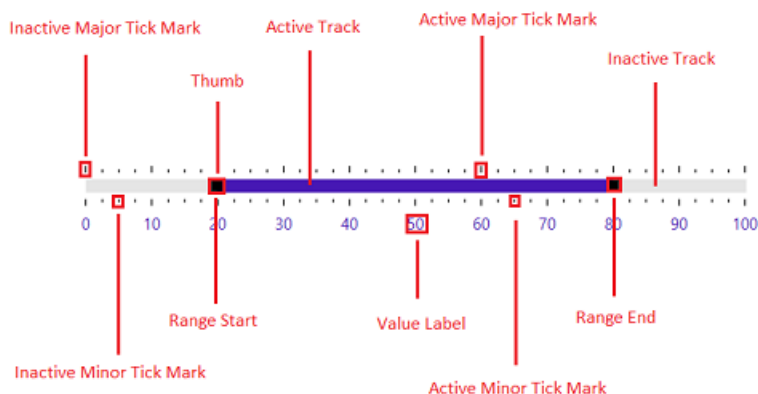
SfRangeSlider supports various built-in themes. Refer to the below links to apply themes for the SfRangeSlider,

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



Visual Structure in WPF Range Slider (SfRangeSlider)

This section describes the visual elements of the RangeSlider control and defines terms and concepts used in the RangeSlider.



- **Inactive Major Tick Mark** — Inactive major ticks to give the better indication between the Minimum to RangeStart and RangeEnd to Maximum.
- **Inactive Minor Tick Mark** — Inactive minor ticks to give the better indication between Major ticks in unselected part.
- **Active Minor Tick Mark** — Active minor ticks to give the better indication between Major ticks in selection part.
- **Active Major Tick Mark** — Active major ticks to give the better indication between the RangeStart and RangeEnd thumbs.
- **Thumb** — Thumb can be dragged along the track.
- **Active Track** — Selection range marked with RangeStart and RangeEnd thumbs.
- **Inactive Track** — The horizontal or vertical line is used to move the thumbs along it.
- **Range Start** — Thumb indicates the start of the selection range.
- **Range End** — Thumb indicates the end of the selection range.

- **Value Label** — indicates the value of the RangeSlider.

Minimum and Maximum in WPF Range Slider (SfRangeSlider)

Gets or sets the minimum and maximum possible value of the range.

XML

```
<editors:SfRangeSlider  
Width="500"  
Maximum="100"  
Minimum="0" />
```

C#

```
Grid parentGrid = new Grid();  
SfRangeSlider rangeSlider = new SfRangeSlider()  
{  
    Width = 500,  
    Maximum = 100,  
    Minimum = 0  
};  
parentGrid.Children.Add(rangeSlider);  
this.Content = parentGrid;
```

Ticks in WPF Range Slider (SfRangeSlider)

RangeSlider makes it possible to place tick marks along the track in a uniform manner and also to customize the position of the tick marks.

Tick Frequency

The [TickFrequency](#) property is used to define the number of ticks along the track, based on Minimum and Maximum values.

XML

```
<editors:SfRangeSlider  
Width="200"  
Maximum="100"  
Minimum="0"  
TickFrequency="20"  
Value="40" />
```

C#

```
Grid parentGrid = new Grid();  
SfRangeSlider rangeSlider = new SfRangeSlider()  
{  
    Width = 200,  
    Maximum = 100,  
    Minimum = 0,  
    TickFrequency = 20,  
    Value = 40  
};  
parentGrid.Children.Add(rangeSlider);  
this.Content = parentGrid;
```



Note: When the `SnapTo` property is set to `Ticks`, the `TickFrequency` is used to specify the interval between snap points.

MinorTickFrequency

The `MinorTickFrequency` property, determines the number of minor ticks on the track between the major ticks.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
MinorTickFrequency="3"
TickFrequency="10"
TickPlacement="BottomRight"
Value="40" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    MinorTickFrequency=3,
    TickFrequency = 10,
    TickPlacement=TickPlacement.BottomRight,
    Value = 40
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Step Frequency

When the `SnapTo` property is set to `StepValues`, the `StepFrequency` property is used to specify the interval between snap points.

XML

```
<editors:SfRangeSlider
Width="200"
```

```
VerticalAlignment="Center"  
Maximum="100"  
Minimum="0"  
StepFrequency="20"  
Value="40" />
```

C#

```
Grid parentGrid = new Grid();  
SfRangeSlider rangeSlider = new SfRangeSlider()  
{  
    Width = 200,  
    Maximum = 100,  
    Minimum = 0,  
    StepFrequency = 20,  
    Value = 40  
};  
parentGrid.Children.Add(rangeSlider);  
this.Content = parentGrid;
```

Snaps To

The [SnapsTo](#) property determines whether the [SfRangeSlider](#) snaps to steps or ticks. Available options for this property are

1. StepValues
2. Ticks

Default option is StepValues and StepFrequency property is used to specify the interval between snap points in this case. When the SnapsTo property is set to Ticks, the TickFrequency property is used to specify the interval between snap points.

Tick Placement

The [TickPlacement](#) property is used to determine where to draw tick marks in relation to the track. Available options for this property are

1. BottomRight
2. Inline
3. None
4. Outside
5. TopLeft

The default option is Inline.

BottomRight

Tick marks are placed either below the track in horizontal orientation or right of the track in vertical orientation.

XML

```
<editors:SfRangeSlider  
    Width="300"  
    Maximum="100"
```

```

Minimum="0"
TickFrequency="20"
TickPlacement="BottomRight"
Value="40" />

```

C#

```

Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 200,
    Maximum = 100,
    Minimum = 0,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
    Value = 40
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;

```



Note: In Vertical Orientation, this option places the ticks to right side.

TopLeft

Tick marks are placed either above the track in horizontal orientation or left of the track in vertical orientation.

XML

```

<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
TickFrequency="20"
TickPlacement="TopLeft"
Value="40" />

```

C#

```

Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 200,
    Maximum = 100,
    Minimum = 0,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.TopLeft,
    Value = 40
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;

```



Note: In Vertical Orientation, this option places the ticks to left side.

Outside

Tick marks are placed on both sides of the track either in horizontal or vertical orientation.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
TickFrequency="20"
TickPlacement="Outside"
Value="40" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 200,
    Maximum = 100,
    Minimum = 0,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.Outside,
    Value = 40
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Inline

Ticks are placed inside the track.

XML

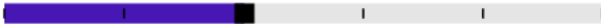
```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
TickFrequency="20"
TickPlacement="Inline"
Value="40" />
```

C#

```
Grid parentGrid = new Grid();
```



```
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 200,
    Maximum = 100,
    Minimum = 0,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.Inline,
    Value = 40
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



None

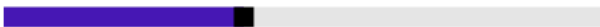
No Tick mark appears.

XML

```
<editors:SfRangeSlider
    Width="300"
    Maximum="100"
    Minimum="0"
    TickFrequency="20"
    TickPlacement="None"
    Value="40" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 200,
    Maximum = 100,
    Minimum = 0,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.None,
    Value = 40
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Range in WPF Range Slider (SfRangeSlider)

The [SfRangeSlider](#) control provides support to select the range of value using two thumbs.

ShowRange

When ShowRange property is set to true, two thumbs are placed in the track. One thumb is used to update the start of the range selection and another thumb is used to update the end of the range selection.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="70"
RangeStart="40"
ShowRange="True" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    ShowRange = true,
    RangeStart = 40,
    RangeEnd = 70
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Note: When the ShowRange property is set to true, both ToolTip will display RangeStart and RangeEnd.

RangeStart

Gets or sets the start value of the range start.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="70"
RangeStart="40"
ShowRange="True" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
```

```
{  
    Width = 300,  
    Maximum = 100,  
    Minimum = 0,  
    ShowRange = true,  
    RangeStart = 40,  
    RangeEnd = 70  
};  
parentGrid.Children.Add(rangeSlider);  
this.Content = parentGrid;
```



RangeEnd

Gets or sets the end value of the range end.

XML

```
<editors:SfRangeSlider  
    Width="300"  
    Maximum="100"  
    Minimum="0"  
    RangeEnd="70"  
    RangeStart="40"  
    ShowRange="True" />
```

C#

```
Grid parentGrid = new Grid();  
SfRangeSlider rangeSlider = new SfRangeSlider()  
{  
    Width = 300,  
    Maximum = 100,  
    Minimum = 0,  
    ShowRange = true,  
    RangeStart = 40,  
    RangeEnd = 70  
};  
parentGrid.Children.Add(rangeSlider);  
this.Content = parentGrid;
```



Drag Selected Range

The `AllowRangeDrag` API allows the range in the Range Slider to be adjusted and the range to be dragged without changing the start and end ranges individually. The default value of `AllowRangeDrag` is false.

XML

```
<editors:SfRangeSlider
Width="300"
AllowRangeDrag="True"
Maximum="100"
Minimum="0"
RangeEnd="20"
RangeStart="0"
ShowRange="True"
ShowValueLabels="True"
TickFrequency="20" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    ShowRange = true,
    RangeStart = 0,
    RangeEnd = 20,
    ShowValueLabels = true,
    TickFrequency = 20,
    AllowRangeDrag = true
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Intermediate Values in WPF Range Slider (SfRangeSlider)

It is possible to get an intermediate value before either the tick value or the step value is snapped when interacting with `SfRangeSlider`.

IntermediateValue

Gets the intermediate value. This works when `ShowRange` is false.

IntermediateRangeStart

Gets the intermediate `RangeStart` value.

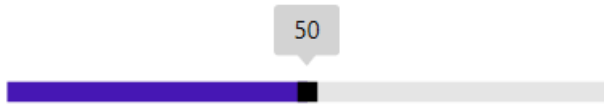
IntermediateRangeEnd

Gets the intermediate `RangeEnd` value.

Note: Above properties are read only properties and cannot set through XAML or code behind.

Thumb ToolTip in WPF Range Slider (SfRangeSlider)

The Thumb tooltip displays the current value where the Thumb stands.



Thumb ToolTip Precision

[ThumbToolTipPrecision](#) property is used to define the precision of the value displayed in the tooltip.

XML

```
<editors:SfRangeSlider
Width="300"
VerticalAlignment="Center"
Maximum="100"
Minimum="0"
ThumbToolTipPrecision="2"
Value="50" />
```



Note: ThumbToolTipPrecision property is only applicable, if ToolTipFormat value is N.

Thumb ToolTipFormat

The [ToolTipFormat](#) property, specifies the format specifier by which to format the ToolTip display value.

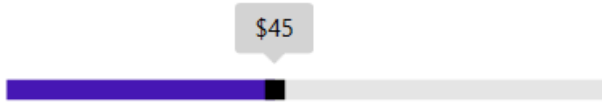
XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
ToolTipFormat="C0"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    ToolTipFormat= C0,
    Value = 50
};
```

```
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Note: Default value of `ToolTipFormat` is N.

Thumb ToolTip Position

The position of the Thumb tooltip in relation to the Thumb can be controlled by the [ThumbToolTipPlacement](#) property. It has the following options.

1. BottomRight
2. TopLeft
3. None

BottomRight

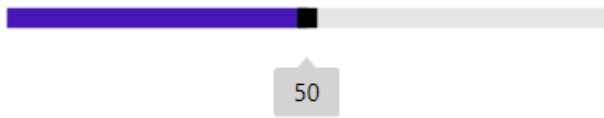
Tooltip is placed either below the Thumb in horizontal orientation or right of the Thumb in vertical orientation.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
ThumbToolTipPlacement="BottomRight"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    ThumbToolTipPlacement = ThumbToolTipPlacement.BottomRight,
    Value = 50
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Note: This option displays the tooltip to right in vertical orientation.

TopLeft

Tooltip is placed either above the Thumb in horizontal orientation or left of the Thumb in vertical orientation.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
ThumbToolTipPlacement="TopLeft"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    ThumbToolTipPlacement = ThumbToolTipPlacement.TopLeft,
    Value = 50
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Note: This option displays the tooltip to left in vertical orientation.

None

No Tooltip appears.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
ThumbToolTipPlacement="None"
```

```
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    ThumbToolTipPlacement = ThumbToolTipPlacement.None,
    Value = 50
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```

**ThumbInterval**

[ThumbInterval](#) is an interval between the two thumbs, the thumbs cannot be moved within this range.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="50"
RangeStart="20"
ShowRange="True"
ThumbInterval="10"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    ShowRange = true,
    RangeStart = 20,
    RangeEnd = 60,
    ThumbInterval = 10,
    Value = 50
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```




Orientation in WPF Range Slider (SfRangeSlider)

The [Orientation](#) property has the following two options.

1. Horizontal
2. Vertical

The default option is Horizontal.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
Orientation="Horizontal"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    Value = 50,
    Orientation = Orientation.Horizontal
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



The following code sample illustrates how to set vertical orientation to [SfRangeSlider](#).

XML

```
<editors:SfRangeSlider
Height="300"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Maximum="100"
Minimum="0"
Orientation="Vertical"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    Value = 50,
    Orientation = Orientation.Vertical
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```

**Direction Reversed in WPF Range Slider (SfRangeSlider)**

The direction of increasing value can be changed using [IsDirectionReversed](#) property. When this property is set to True, the direction of increasing value is towards left in the horizontal orientation and down in the vertical orientation. The default is false.

XML

```
<editors:SfRangeSlider
Width="300"
IsDirectionReversed="True"
Maximum="100"
Minimum="0"
Value="30" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
```

```

Width = 300,
Maximum = 100,
Minimum = 0,
Value = 30,
IsDirectionReversed = true
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;

```



Gestures in WPF Range Slider (SfRangeSlider)

Keyboard

The focused Thumb moves a certain distance on every navigation key press.

Left Key

Thumb moves left and updates the corresponding value.

Right Key

Thumb moves right and updates the corresponding value.

Down Key

Thumb moves down and updates the corresponding value.

Up Key

Thumb moves up and updates the corresponding value.

Note: When the SnapsTo property is set to Ticks then the Thumb snaps to next tick based on the navigation key pressed.

Mouse

RangeSlider allows the mouse gesture to update the value of the SfRangeSlider either by dragging the thumb to a certain distance or by pressing the specified region in the track.

Note: When the thumb is released between two steps or when the pointer is pressed between two steps, then the value and thumb automatically snaps to nearest value.

Label Support in WPF Range Slider (SfRangeSlider)

This feature allows the display of labels for custom values given in the collection of CustomLabels when the ShowCustomLabels property is set to true. When ShowValueLabels is set to true, it also displays labels for all the tick values.

Property Table

Property	Description
CustomLabels	It describes an observable collection of items that contains the custom labels and the values for that the labels should be displayed.

ShowCustomLabels	This property allows the display of custom labels based on the CustomLabels collection for particular values.
ShowValueLabels	This property allows the tick labels to be displayed.
LabelPlacement	This property specifies the position of the custom label placement.
ValuePlacement	This property specifies the position of the label for all of the ticks.
LabelOrientation	LabelOrientation specifies the orientation of the labels as either horizontal or vertical.

CustomLabels

CustomLabels is an observable collection of items that contains the Label and Value properties. Create an observable collection of items by specifying the custom labels for corresponding values as illustrated in the following code example.

Create a ViewModel class with CustomCollection property.

C#

```
public class ViewModel
{
    private ObservableCollection<Items> customCollection = new
    ObservableCollection<Items>();
    public ObservableCollection<Items> CustomCollection
    {
        get { return customCollection; }
        set { customCollection = value; }
    }
    public ViewModel()
    {
        this.customCollection.Add(new Items() { label = "Min", value = 100 });
        this.customCollection.Add(new Items() { label = "Max", value = 200 });
    }
}
```

In the following code example, the CustomCollection property is bound to [CustomLabels](#) property in the SfRangeSlider control that populates the custom labels collection.

XML

```
<editors:SfRangeSlider
Width="300"
CustomLabels="{Binding CustomCollection}"
Maximum="200"
Minimum="100"
ShowCustomLabels="True"
TickFrequency="20"
TickPlacement="Outside"
ThumbToolTipPlacement="BottomRight"/>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 200,
    Minimum = 100,
    ShowCustomLabels = true,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.Outside,
    ThumbToolTipPlacement = ThumbToolTipPlacement.BottomRight
};
Binding binding = new Binding("CustomCollection");
binding.Source = viewModel;
rangeSlider.SetBinding(SfRangeSlider.CustomLabelsProperty, binding);
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



RangeSlider with CustomLabels

ShowCustomLabels

The default value for [ShowCustomLabels](#) is false. When set to true, it displays the custom labels with the corresponding Tool tip for specific values based on the [CustomLabels](#) collection.

XML

```
<editors:SfRangeSlider
    Width="300"
    CustomLabels="{Binding CustomCollection}"
    Maximum="200"
    Minimum="100"
    ShowCustomLabels="True"
    TickFrequency="20"
    TickPlacement="Outside"
    ThumbToolTipPlacement="BottomRight"/>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 200,
    Minimum = 100,
    ShowCustomLabels = true,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.Outside,
    ThumbToolTipPlacement = ThumbToolTipPlacement.BottomRight
};
Binding binding = new Binding("CustomCollection");
binding.Source = viewModel;
```

```
rangeSlider.SetBinding(SfRangeSlider.CustomLabelsProperty, binding);
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



RangeSlider with ShowCustomLabels

LabelPlacement

[LabelPlacement](#) property describes the position of the custom labels for particular values mentioned in the CustomLabels collection. Available options for this property are:

1. BottomRight
2. TopLeft

The following code example illustrates the usage of the LabelPlacement property. The output is displayed in the corresponding images.

XML

```
<editors:SfRangeSlider
Width="300"
CustomLabels="{Binding CustomCollection}"
Maximum="200"
Minimum="100"
ShowCustomLabels="True"
TickFrequency="20"
TickPlacement="Outside"
LabelPlacement="BottomRight"/>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 200,
    Minimum = 100,
    ShowCustomLabels = true,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.Outside,
    LabelPlacement = LabelPlacement.BottomRight
};
Binding binding = new Binding("CustomCollection");
binding.Source = viewModel;
rangeSlider.SetBinding(SfRangeSlider.CustomLabelsProperty, binding);
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



LabelPlacement in BottomRight

XML

```
<editors:SfRangeSlider
Width="300"
CustomLabels="{Binding CustomCollection}"
Maximum="200"
Minimum="100"
ShowCustomLabels="True"
TickFrequency="20"
TickPlacement="Outside"
LabelPlacement="TopLeft"/>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 200,
    Minimum = 100,
    ShowCustomLabels = true,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.Outside,
    LabelPlacement = LabelPlacement.TopLeft
};
Binding binding = new Binding("CustomCollection");
binding.Source = viewModel;
rangeSlider.SetBinding(SfRangeSlider.CustomLabelsProperty, binding);
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



LabelPlacement in TopLeft

ShowValueLabels

The default value of the [ShowValueLabels](#) property is false. When set to true, it displays the label for all the ticks based on the ValuePlacement property.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
ShowValueLabels="True"
TickFrequency="20"
TickPlacement="BottomRight"
Value="40" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    Value = 40,
    ShowValueLabels = true,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



ShowValueLabels property

ValuePlacement

The [ValuePlacement](#) property describes the position of the labels for all the ticks. Available options for this property are:

1. BottomRight
2. TopLeft

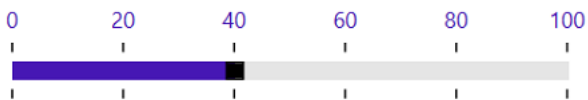
The following code example illustrates the usage of [ValuePlacement](#) property. The output is displayed in the corresponding images.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
ShowValueLabels="True"
TickFrequency="20"
TickPlacement="Outside"
ValuePlacement="TopLeft"
Value="40" />
```


C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    Value = 40,
    ShowValueLabels = true,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
    ValuePlacement = ValuePlacement.TopLeft
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



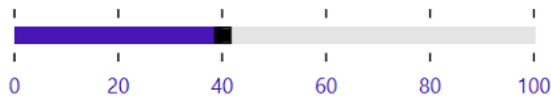
ValuePlacement in TopLeft.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
ShowValueLabels="True"
TickFrequency="20"
TickPlacement="Outside"
ValuePlacement="BottomRight"
Value="40" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    Value = 40,
    ShowValueLabels = true,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
    ValuePlacement = ValuePlacement.BottomRight
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



ValuePlacement in BottomRight

LabelOrientation

The [LabelOrientation](#) property describes the orientation of the labels for both ticks and custom labels. Available options for this property are:

1. Horizontal
2. Vertical

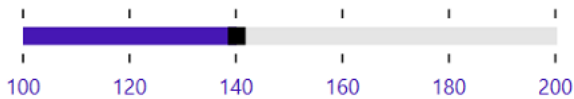
The following code example illustrates the usage of [LabelOrientation](#) property. The output is displayed in the corresponding images.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="200"
Minimum="100"
ShowValueLabels="True"
TickFrequency="20"
TickPlacement="Outside"
ValuePlacement="BottomRight"
LabelOrientation="Horizontal"
Value="140" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 200,
    Minimum = 100,
    Value = 140,
    ShowValueLabels = true,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.Outside,
    ValuePlacement = ValuePlacement.BottomRight,
    LabelOrientation = Orientation.Horizontal
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



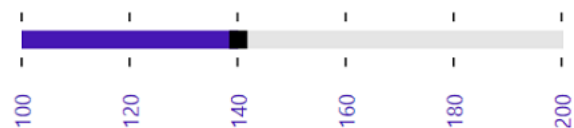
LabelOrientation as Horizontal

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="200"
Minimum="100"
ShowValueLabels="True"
TickFrequency="20"
TickPlacement="Outside"
ValuePlacement="BottomRight"
LabelOrientation="Vertical"
Value="140" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 200,
    Minimum = 100,
    Value = 140,
    ShowValueLabels = true,
    TickFrequency = 20,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.Outside,
    ValuePlacement = ValuePlacement.BottomRight,
    LabelOrientation = Orientation.Vertical
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



LabelOrientation as Vertical

Touch Support in WPF Range Slider (SfRangeSlider)

[MoveToPoint](#) API allows the [SfRangeSlider](#) to move the thumb by tapping or clicking the track of the [SfRangeSlider](#). This property provides the following options:

- MoveToTapPosition

- IncrementBySmallChange
- IncrementByLargeChange
- None

MoveToTapPosition

To move the thumb of [SfRangeSlider](#) to the tapped position, set the [MoveToPoint](#) property to `MoveToTapPosition`, and then tap or click the track of the [SfRangeSlider](#). This moves the thumb to the tapped position.

The following code example and screenshot illustrates the above.

XML

```
<editors:SfRangeSlider
Width="400"
LargeChange="10"
Maximum="100"
Minimum="0"
MoveToPoint="MoveToTapPosition"
SmallChange="5"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 400,
    Maximum = 100,
    Minimum = 0,
    SmallChange = 5,
    LargeChange = 10,
    MoveToPoint = MovePoint.MoveToTapPosition,
    Value = 50
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



IncrementBySmallChange

To move the [SfRangeSlider](#) thumb based on the `SmallChange` value, set the [MoveToPoint](#) property to `IncrementBySmallChange`, and then tap or click the track of the [SfRangeSlider](#). This increments the [SfRangeSlider](#) value by the `SmallChange` value.

The following code example and screenshot illustrates the above.

XML

```
<editors:SfRangeSlider
Width="400"
LargeChange="10"
Maximum="100"
Minimum="0"
MoveToPoint="IncrementBySmallChange"
SmallChange="5"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 400,
    Maximum = 100,
    Minimum = 0,
    SmallChange = 5,
    LargeChange = 10,
    MoveToPoint = MovePoint.IncrementBySmallChange,
    Value = 50
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



IncrementByLargeChange

To move the [SfRangeSlider](#) thumb based on the LargeChange value, set the [MoveToPoint](#) property to [IncrementByLargeChange](#), and then tap the track of the [SfRangeSlider](#). This increments the [SfRangeSlider](#) value by the LargeChange value.

The following code example and screenshot illustrates the above.

XML

```
<editors:SfRangeSlider
Width="400"
LargeChange="10"
Maximum="100"
Minimum="0"
MoveToPoint="IncrementByLargeChange"
SmallChange="5"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
```

```
Width = 400,
Maximum = 100,
Minimum = 0,
SmallChange = 5,
LargeChange = 10,
MoveToPoint = MovePoint.IncrementByLargeChange,
Value = 50
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



None

To fix the thumb movement of the [SfRangeSlider](#), set the `MoveToPoint` property to *None*. This does not allow thumb movement of the [SfRangeSlider](#).

The following code example and screenshot illustrates the above.

XML

```
<editors:SfRangeSlider
Width="400"
LargeChange="10"
Maximum="100"
Minimum="0"
MoveToPoint="None"
SmallChange="5"
Value="50" />
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
Width = 400,
Maximum = 100,
Minimum = 0,
SmallChange = 5,
LargeChange = 10,
MoveToPoint = MovePoint.None,
Value = 50
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Styling and Appearance in WPF Range Slider (SfRangeSlider)

[SfRangeSlider](#) makes it possible to change the appearance by providing certain properties.

InactiveTrackStyle

Modify the appearance of the inactive track using the [InactiveTrackStyle](#) property.

XML

```
<Grid>
<Grid.Resources>
<ResourceDictionary>
<Style x:Key="InactiveTrackStyle" TargetType="Rectangle">
<Setter Property="Height" Value="3" />
<Setter Property="Fill" Value="#a8a8a8" />
<Setter Property="RadiusX" Value="2" />
<Setter Property="RadiusY" Value="2" />
</Style>
</ResourceDictionary>
</Grid.Resources>
<editors:SfRangeSlider
Width="300"
InactiveTrackStyle="{StaticResource InactiveTrackStyle}"
Maximum="100"
Minimum="0" />
</Grid>
```

C#

```
Grid parentGrid = new Grid();
Style inactiveTrackStyle = new Style(typeof(Rectangle));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.FillProperty, new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#a8a8a8"))));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.HeightProperty,
(double)3));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.RadiusXProperty,
(double)2));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.RadiusYProperty,
(double)2));
Resources.Add("inactiveTrackStyle", inactiveTrackStyle);
SfRangeSlider rangeSlider = new SfRangeSlider()
{
Width = 300,
Minimum = 0,
Maximum = 100,
InactiveTrackStyle = inactiveTrackStyle
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



ActiveTrackStyle

Modify the appearance of the active track by using the [ActiveTrackStyle](#) property.

XML

```

<Grid>
<Grid.Resources>
<ResourceDictionary>
<Style x:Key="InactiveTrackStyle" TargetType="Rectangle">
<Setter Property="Height" Value="3" />
<Setter Property="Fill" Value="#a8a8a8" />
<Setter Property="RadiusX" Value="2" />
<Setter Property="RadiusY" Value="2" />
</Style>
<Style x:Key="ActiveTrackStyle" TargetType="Rectangle">
<Setter Property="Height" Value="3" />
<Setter Property="Fill" Value="#505050" />
</Style>
</ResourceDictionary>
</Grid.Resources>
<editors:SfRangeSlider
Width="300"
ActiveTrackStyle="{StaticResource ActiveTrackStyle}"
InactiveTrackStyle="{StaticResource InactiveTrackStyle}"
Maximum="100"
Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True" />
</Grid>

```

C#

```

Grid parentGrid = new Grid();
Style inactiveTrackStyle = new Style(typeof(Rectangle));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.FillProperty, new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#a8a8a8"))));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.HeightProperty,
(double)3));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.RadiusXProperty,
(double)2));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.RadiusYProperty,
(double)2));
Resources.Add("inactiveTrackStyle", inactiveTrackStyle);
Style activeTrackStyle = new Style(typeof(Rectangle));
activeTrackStyle.Setters.Add(new Setter(Rectangle.FillProperty, new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#505050"))));
activeTrackStyle.Setters.Add(new Setter(Rectangle.HeightProperty,
(double)3));
Resources.Add("activeTrackStyle", activeTrackStyle);
SfRangeSlider rangeSlider = new SfRangeSlider()
{
Width = 300,
Minimum = 0,
Maximum = 100,
ShowRange = true,
RangeStart = 20,
RangeEnd = 60,
ActiveTrackStyle = activeTrackStyle,

```



```

InactiveTrackStyle = inactiveTrackStyle
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;

```



ThumbStyle

Modify the appearance of the thumb by using the [ThumbStyle](#) property.

XML

```

<Grid>
<Grid.Resources>
<ResourceDictionary>
<Style x:Key="InactiveTrackStyle" TargetType="Rectangle">
<Setter Property="Height" Value="3" />
<Setter Property="Fill" Value="#a8a8a8" />
<Setter Property="RadiusX" Value="2" />
<Setter Property="RadiusY" Value="2" />
</Style>
<Style x:Key="ActiveTrackStyle" TargetType="Rectangle">
<Setter Property="Height" Value="3" />
<Setter Property="Fill" Value="#505050" />
</Style>
<Style x:Key="ThumbStyle" TargetType="Thumb">
<Setter Property="Width" Value="13" />
<Setter Property="Height" Value="13" />
<Setter Property="Background" Value="#0095ff" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="Thumb">
<Border
x:Name="ThumbBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
CornerRadius="12" />
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>
</Grid.Resources>
<editors:SfRangeSlider
Width="300"
ActiveTrackStyle="{StaticResource ActiveTrackStyle}"
InactiveTrackStyle="{StaticResource InactiveTrackStyle}"
Maximum="100"
Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True"
ThumbStyle="{StaticResource ThumbStyle}" />
</Grid>

```

C#

```

Grid parentGrid = new Grid();
Style inactiveTrackStyle = new Style(typeof(Rectangle));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.FillProperty, new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#a8a8a8"))));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.HeightProperty,
(double)3));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.RadiusXProperty,
(double)2));
inactiveTrackStyle.Setters.Add(new Setter(Rectangle.RadiusYProperty,
(double)2));
Resources.Add("inactiveTrackStyle", inactiveTrackStyle);
Style activeTrackStyle = new Style(typeof(Rectangle));
activeTrackStyle.Setters.Add(new Setter(Rectangle.FillProperty, new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#505050"))));
activeTrackStyle.Setters.Add(new Setter(Rectangle.HeightProperty,
(double)3));
Resources.Add("activeTrackStyle", activeTrackStyle);
ControlTemplate template = new ControlTemplate(typeof(Thumb));
FrameworkElementFactory elemFactory = new
FrameworkElementFactory(typeof(Border));
elemFactory.Name = "Border";
elemFactory.SetValue(Border.CornerRadiusProperty, new CornerRadius(12));
elemFactory.SetValue(Border.BackgroundProperty, new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#0095ff")));
elemFactory.SetValue(Border.BorderBrushProperty, new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#0095ff")));
template.VisualTree = elemFactory;
Style thumbStyle = new Style(typeof(Thumb));
thumbStyle.Setters.Add(new Setter(Thumb.BackgroundProperty, new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#0095ff"))));
thumbStyle.Setters.Add(new Setter(Thumb.HeightProperty, (double)13));
thumbStyle.Setters.Add(new Setter(Thumb.WidthProperty, (double)13));
thumbStyle.Setters.Add(new Setter(Thumb.TemplateProperty, template));
Resources.Add("thumbStyle", thumbStyle);
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Minimum = 0,
    Maximum = 100,
    ShowRange = true,
    RangeStart = 20,
    RangeEnd = 60,
    ActiveTrackStyle = activeTrackStyle,
    InactiveTrackStyle = inactiveTrackStyle,
    ThumbStyle = thumbStyle
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;

```

**Tick Customization**

Modify the appearance of the ticks by using the following properties:

- TickStroke
- ActiveTickStroke
- TickLength
- TickStrokeThickness
- MinorTickStroke
- ActiveMinorTickStroke
- MinorTickLength
- MinorTickStrokeThickness

TickStroke

Use the [TickStroke](#) property, to change the color of major ticks.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True"
TickFrequency="10"
TickPlacement="BottomRight"
TickStroke="#FF0000" />
</Grid>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    RangeEnd = 60,
    RangeStart = 20,
    ShowRange = true,
    TickFrequency = 10,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
    TickStroke = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF0000"))
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



ActiveTickStroke

Use the [ActiveTickStroke](#) property, to change the active major ticks color.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True"
TickFrequency="10"
TickPlacement="BottomRight"
TickStroke="#FF0000"
ActiveTickStroke="#02C9F3"/>
</Grid>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    RangeEnd = 60,
    RangeStart = 20,
    ShowRange = true,
    TickFrequency = 10,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
    TickStroke = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF0000")),
    ActiveTickStroke = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#02C9F3"))
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```

*TickLength*

Use the [TickLength](#) property, to change the height of the major ticks.

XML

```
<Grid>
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True"
TickLength="8"
TickStroke="#FF0000"
ActiveTickStroke="#02C9F3"/>
```

```

TickFrequency="10"
TickPlacement="BottomRight"
TickStroke="#FF0000" />
</Grid>

```

C#

```

Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    RangeEnd = 60,
    RangeStart = 20,
    ShowRange = true,
    TickFrequency = 10,
    TickLength = 8,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
    TickStroke = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF0000"))
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;

```

*TickStrokeThickness*

Use the [TickStrokeThickness](#) property, to change the thickness of the major ticks.

XML

```

<Grid>
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True"
TickFrequency="10"
TickLength="8"
TickPlacement="BottomRight"
TickStroke="#FF0000"
TickStrokeThickness="2" />
</Grid>

```

C#

```

Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()

```

```
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    RangeEnd = 60,
    RangeStart = 20,
    ShowRange = true,
    TickFrequency = 10,
    TickLength = 8,
    TickStrokeThickness = 2,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
    TickStroke = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF0000"))
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



MinorTickStroke

Use the [MinorTickStroke](#) property, to change the minor ticks color.

XML

```
<editors:SfRangeSlider
    Width="300"
    Maximum="100"
    Minimum="0"
    RangeEnd="60"
    RangeStart="20"
    ShowRange="True"
    TickFrequency="10"
    MinorTickFrequency="3"
    TickPlacement="BottomRight"
    MinorTickStroke="#FF0000" />
</Grid>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    RangeEnd = 60,
    RangeStart = 20,
    ShowRange = true,
    TickFrequency = 10,
    MinorTickFrequency=3,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
```

```
MinorTickStroke = new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF0000"))
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



ActiveMinorTickStroke

Use the [ActiveMinorTickStroke](#) property, to change the color of the active minor ticks.

XML

```
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True"
TickFrequency="10"
MinorTickFrequency="3"
TickPlacement="BottomRight"
ActiveMinorTickStroke="#FF0000" />
</Grid>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
Width = 300,
Maximum = 100,
Minimum = 0,
RangeEnd = 60,
RangeStart = 20,
ShowRange = true,
TickFrequency = 10,
MinorTickFrequency=3,
TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
ActiveMinorTickStroke = new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF0000"))
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



MinorTickLength

Use the [MinorTickLength](#) property, to change the height of the minor ticks.

XML

```
<Grid>
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True"
MinorTickLength="6"
TickLength="10"
TickFrequency="10"
MinorTickFrequency="2"
TickPlacement="BottomRight" />
</Grid>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    RangeEnd = 60,
    RangeStart = 20,
    ShowRange = true,
    MinorTickLength = 6,
    TickLength = 10,
    TickFrequency = 10,
    MinorTickFrequency = 2,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```

*MinorTickStrokeThickness*

Use the [MinorTickStrokeThickness](#) property, to change the thickness of the minor ticks.

XML

```
<Grid>
<editors:SfRangeSlider
Width="300"
Maximum="100"
```



```

Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True"
TickFrequency="10"
MinorTickFrequency="2"
TickLength="8"
MinorTickLength="5"
TickPlacement="BottomRight"
TickStroke="#FF0000"
MinorTickStroke="#FF0000"
ActiveTickStroke="#02C9F3"
ActiveMinorTickStroke="#02C9F3"
MinorTickStrokeThickness="2" />
</Grid>

```

C#

```

Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    RangeEnd = 60,
    RangeStart = 20,
    ShowRange = true,
    TickFrequency = 10,
    MinorTickFrequency= 2,
    TickLength = 8,
    MinorTickStrokeThickness = 2,
    MinorTickLength= 5,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
    TickStroke = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF0000"))
    MinorTickStroke = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF0000"))
    ActiveMinorTickStroke = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#02C9F3"))
    ActiveTickStroke = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#02C9F3"))
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;

```



Value label customization

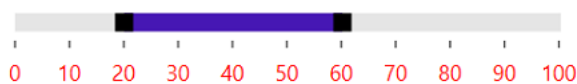
Customize the value label using the [TickBarItem](#) style.

XML

```
<Grid>
<Grid.Resources>
<ResourceDictionary>
<Style TargetType="editors:TickBarItem">
<Setter Property="FontSize" Value="12" />
<Setter Property="Foreground" Value="Red" />
</Style>
</ResourceDictionary>
</Grid.Resources>
<editors:SfRangeSlider
Width="300"
Maximum="100"
Minimum="0"
RangeEnd="60"
RangeStart="20"
ShowRange="True"
ShowValueLabels="True"
TickFrequency="10"
TickLength="4"
TickPlacement="BottomRight"
TickStroke="#505050"
TickStrokeThickness="1" />
</Grid>
```

C#

```
Grid parentGrid = new Grid();
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    RangeEnd = 60,
    ShowValueLabels = true,
    RangeStart = 20,
    ShowRange = true,
    TickFrequency = 10,
    TickLength = 4,
    TickStrokeThickness = 1,
    TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.BottomRight,
    TickStroke = new
    SolidColorBrush((Color) ColorConverter.ConvertFromString("#505050"))
};
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
```



Events in WPF RangeSlider(SfRangeSlider)

How to trigger LabelLoaded event?

The [LabelLoaded](#) event is triggered when the slider label is created. The argument contains the label content.

Content - Used to gets or sets the content of the label.

XML

```
<editors:SfRangeSlider
Width="300"
LabelLoaded="SfRangeSlider_LabelLoaded"
Maximum="200"
Minimum="100"
ShowValueLabels="True"
TickFrequency="20"
TickPlacement="Outside" />
```

C#

```
public MainWindow()
{
    InitializeComponent();
    Grid parentGrid = new Grid();
    SfRangeSlider rangeSlider = new SfRangeSlider()
    {
        Width = 300,
        Maximum = 200,
        Minimum = 100,
        Value = 140,
        ShowValueLabels = true,
        TickFrequency = 20,
        TickPlacement = Syncfusion.Windows.Controls.Input.TickPlacement.Outside
    };
    rangeSlider.LabelLoaded += SfRangeSlider_LabelLoaded;
    parentGrid.Children.Add(rangeSlider);
    this.Content = parentGrid;
}

private void SfRangeSlider_LabelLoaded(object sender, LabelLoadedEventArgs e)
{
    e.Content = e.Content + "%";
}
```

How to trigger RangeChangedEvent?

The [RangeChanged](#) event is triggered when either **RangeStart** or **RangeEnd** values are changed. The argument contains the following information.

NewEndValue – Gets or sets the new end value of range slider.

NewStartValue – Gets or sets the new start value of range slider.

OldStartValue – Gets or sets the old start value of range slider.

OldEndValue – Gets or set the old end value of range slider.

XML

```
<editors:SfRangeSlider
Width="300"
RangeChanged="SfRangeSlider_RangeChanged"
Maximum="100"
Minimum="0"
ShowRange="True"
RangeStart="30"
RangeEnd="60"/>
```

C#

```
public MainWindow()
{
    InitializeComponent();
    Grid parentGrid = new Grid();
    SfRangeSlider rangeSlider = new SfRangeSlider()
    {
        Width = 300,
        Maximum = 100,
        Minimum = 0,
        Value = 140,
        ShowRange = true,
        RangeStart = 30,
        RangeEnd = 60
    };
    rangeSlider.RangeChanged += SfRangeSlider_RangeChanged;
    parentGrid.Children.Add(rangeSlider);
    this.Content = parentGrid;
}

private void SfRangeSlider_RangeChanged(object sender, RangeChangedEventArgs e)
{
    var newStartValue = e.NewStartValue;
    var newEndValue = e.NewEndValue;
    var oldStartValue = e.OldStartValue;
    var oldEndValue = e.OldEndValue;
}
```

How to trigger RangeStartChanged event?

The [RangeStartChanged](#) event is triggered when **RangeStart** value is changed. The argument contains the following information.

OldStartValue – Gets or sets the old start value of range slider.

NewStartValue – Gets or sets the new start value of range slider.

XML

```
<editors:SfRangeSlider
Width="300"
RangeStartChanged="SfRangeSlider_RangeStartChanged"
Maximum="100"
Minimum="0"
ShowRange="True"
```

```
RangeStart="30"
RangeEnd="60"/>
```

C#

```
public MainWindow()
{
    InitializeComponent();
    Grid parentGrid = new Grid();
    SfRangeSlider rangeSlider = new SfRangeSlider()
    {
        Width = 300,
        Maximum = 100,
        Minimum = 0,
        Value = 140,
        ShowRange = true,
        RangeStart = 30,
        RangeEnd = 60
    };
    rangeSlider.RangeStartChanged += SfRangeSlider_RangeStartChanged;
    parentGrid.Children.Add(rangeSlider);
    this.Content = parentGrid;
}

private void SfRangeSlider_RangeStartChanged(object sender,
RangeStartChagedEventArgs e)
{
    var newStartValue = e.NewStartValue;
    var oldStartValue = e.OldStartValue;
}
```

How to trigger RangeEndChanged event?

The [RangeEndChanged](#) event is triggered when **RangeEnd** value is changed. The argument contains the following information.

OldEndValue – Gets or sets the old end value of range slider.

NewEndValue – Gets or sets the new end value of range slider.

XML

```
<editors:SfRangeSlider
Width="300"
RangeEndChanged="SfRangeSlider_RangeEndChanged"
Maximum="100"
Minimum="0"
ShowRange="True"
RangeStart="30"
RangeEnd="60"/>
```

C#

```
public MainWindow()
{
    InitializeComponent();
    Grid parentGrid = new Grid();
```

```
SfRangeSlider rangeSlider = new SfRangeSlider()
{
    Width = 300,
    Maximum = 100,
    Minimum = 0,
    Value = 140,
    ShowRange = true,
    RangeStart = 30,
    RangeEnd = 60
};
rangeSlider.RangeEndChanged += SfRangeSlider_RangeEndChanged;
parentGrid.Children.Add(rangeSlider);
this.Content = parentGrid;
}
private void SfRangeSlider_RangeEndChanged(object sender,
RangeEndChagedEventArgs e)
{
    var newEndValue = e.NewEndValue;
    var oldEndValue = e.OldEndValue;
}
```

SfRating

WPF Rating (SfRating) Overview

The SfRating control for WPF provides the number of stars that represent a rating. It is used to configure the item size, and the number of displayed items in the SfRating control. The SfRating control can be used in various scenarios such as rating movies, rating applications, etc.

Movie Rating



The Walk (2015)

PG | 2h 20min



In 1973, French street performer Philippe Petit is trying to make a living in Paris with juggling acts and wire walking, much to the chagrin of his father. During one performance, he eats a hard candy which was given to him by an audience member and injures his tooth.

Rate



Rating: 3/5

Key features

- **Precision:** Options to decide the accuracy level of rating.
- **Item count:** Supports to determine the number of rating items to be displayed.

Getting Started with WPF Rating (SfRating)

This section explains how to configure the [SfRating](#) control in a real-time scenario and also provides a walk-through on some of the customization features available in the SfRating 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 [SfRating](#) control in any application.

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

[How to install nuget packages](#)

Creating Application with SfRating control

In this walk through, user will create a WPF application that contains [SfRating](#) control.

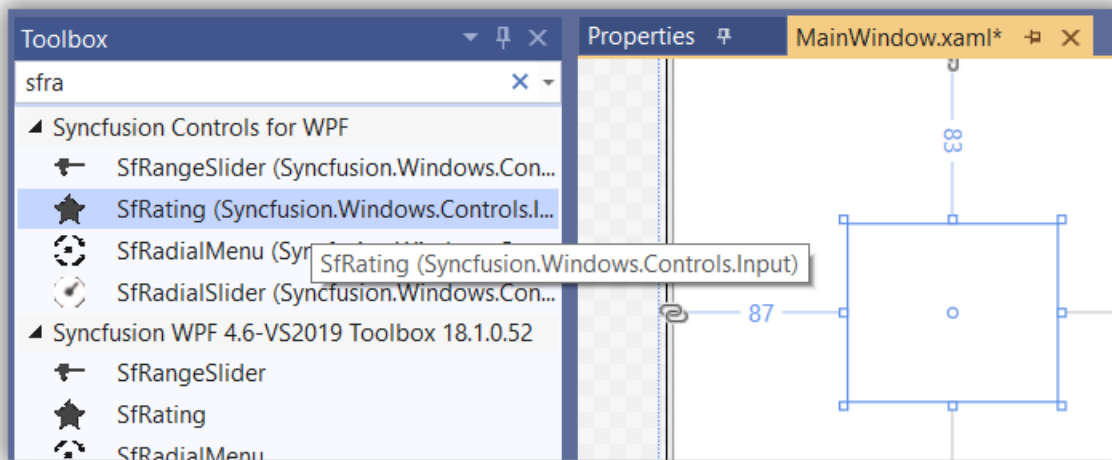
1. [Creating project](#)
2. [Adding control via designer](#)
3. [Adding control manually in XAML](#)
4. [Adding control manually in C#](#)

Creating project

Below section provides detailed information to create new project in Visual Studio to display [SfRating](#) control.

Adding control via designer

The [SfRating](#) control can be added to the application by dragging it from Toolbox and dropping it in designer. The required assembly will be added automatically.



Adding control manually in XAML

In order to add [SfRating](#) control manually in XAML, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.SfShared.WPF
 - Syncfusion.SfInput.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page.
3. Declare [SfRating](#) in XAML page.

XML

```
<Window x:Class="SfRating_GettingStarted.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:SfRating_GettingStarted"
mc:Ignorable="d"
Title="SfRating Application" Height="450" Width="800"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf">
<Grid>
<syncfusion:SfRating ItemsCount="5" Width="150"/>
</Grid>
</Window>
```

Add control manually in C#

In order to add [SfRating](#) control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.SfShared.WPF
 - Syncfusion.SfInput.WPF
2. Import SfRating namespace **Syncfusion.Windows.Controls.Input**.
3. Create SfRating control instance and add it to window.

C#

```
using Syncfusion.Windows.Controls.Input;
namespace SfRating_GettingStarted
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Creating an instance of SfRating control.
            SfRating newrating = new SfRating()
            {
                ItemsCount = 5,
                Width = 150
            };
            //Adding SfRating as window content.
            this.Content = newrating;
        }
    }
}
```



```
}  
}
```

Customize number of rating items

The number of rating items to be displayed can be customized using [ItemCounts](#) property in [SfRating](#) control.

Note: The default value of ItemsCount is 0.

XML

```
<syncfusion:SfRating ItemsCount="5" Width="150"/>
```

C#

```
SfRating rating = new SfRating()  
{  
    ItemsCount = 5,  
    Width = 150  
};
```

Set value

The display value to be selected among the rating items can be set in the [SfRating](#) control using [Value](#) property.

Note: By default, value of this property is 0.

XML

```
<syncfusion:SfRating ItemsCount="5" Value="3" Width="150"/>
```

C#

```
SfRating rating = new SfRating()  
{  
    ItemsCount = 5,  
    Width = 150,  
    Value = 3  
};
```

Precision of selection

The [SfRating](#) control provides option to rate the items in full, half, and exact value using the [Precision](#) property.

Note: By default, the value of Precision property is Standard.

XML

```
<syncfusion:SfRating ItemsCount="5" Precision="Exact" Width="150"/>
```

C#

```
SfRating rating = new SfRating()
```

```
{
    ItemsCount = 5,
    Width = 150,
    Precision = Syncfusion.Windows.Primitives.Precision.Standard
};
```

Movie Rating



The Walk (2015)

PG | 2h 20min



In 1973, French street performer Philippe Petit is trying to make a living in Paris with juggling acts and wire walking, much to the chagrin of his father. During one performance, he eats a hard candy which was given to him by an audience member and injures his tooth.

Rate



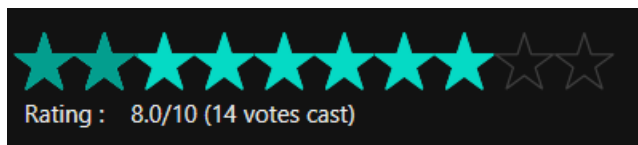
Rating: 3/5

[View Sample in GitHub](#)

Theme

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

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



Precision in WPF Rating (SfRating)

The precision mode defines the accuracy level of the SfRating control. It has Standard, Half, and Exact options. By default, the precision mode of the SfRating control is set to **Standard**.

Standard

When the precision mode of the SfRating control is set to **Standard**, the rating item will be filled completely based on the rating value.

XML

```
<rating:SfRating ItemsCount="5" Precision="Standard"></rating:SfRating>
```

C#

```
SfRating rating;  
public MainWindow()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.ItemCount = 5;  
    rating.Precision = Precision.Standard;  
    Content = rating;  
}
```

*Half*

When the precision mode of the SfRating control is set to **Half**, the rating item will be filled partially based on the rating value.

XML

```
<rating:SfRating ItemCount="5" Precision="Half"></rating:SfRating>
```

C#

```
SfRating rating;  
public MainWindow()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    ItemCount = 5;  
    rating.Precision = Precision.Half;  
    Content = rating;  
}
```

*Exact*

If the precision mode of SfRating is set to **Exact**, the rating item will be filled exactly based on the rating value.

XML

```
<rating:SfRating ItemCount="5" Precision="Exact"></rating:SfRating>
```

C#

```
SfRating rating;  
public MainWindow()  
{  
    InitializeComponent();  
    rating = new SfRating();  
}
```

```
rating.ItemsCount = 5;
rating.Precision = Precision.Exact;
Content = rating;
}
```



ToolTip in WPF Rating (SfRating)

ToolTip provides additional information about objects that are unfamiliar to users and are not directly displayed in UI. In the WPF SfRating control, tooltip shows the data of **Value** property. It will be displayed when the mouse is hovered over the rating items, and it will be disappeared when the mouse is moved from the rating items. The tooltip is enabled or disabled using the **ShowToolTip** property. The default value of this property is true.

XML

```
<rating:SfRating ItemsCount="5" ShowToolTip="True" >
</rating:SfRating>
```

C#

```
SfRating rating;
public MainWindow()
{
    InitializeComponent();
    rating = new SfRating();
    rating.ItemsCount = 5;
    rating.ShowToolTip = true;
    Content = rating;
}
```



Set tooltip precision

The **AutoToolTipPrecision** property sets the number precisions to be displayed after decimal point in tooltip. To set **AutoToolTipPrecision**, set the precision mode to **Exact**.

Note: The default value of this property is 1.

XML

```
<rating:SfRating ItemsCount="5" Precision="Exact" AutoToolTipPrecision="6" >
</rating:SfRating>
```

C#

```
SfRating rating;
public MainWindow()
```

```
{
  InitializeComponent();
  rating = new SfRating();
  rating.ItemsCount = 5;
  rating.Precision = Precision.Exact;
  rating.AutoToolTipPrecision = 6;
  Content = rating;
}
```



Restrict User Selection in WPF Rating (SfRating)

The SfRating control provides support to changeable or unchangeable values. This is achieved using the `IsReadOnly` property. When this property is set to true, the rating value becomes unchangeable. By default, value of this property is false.

XML

```
<rating:SfRating ItemsCount="5" IsReadOnly="True"/>
```

C#

```
SfRating rating;
public MainWindow()
{
  InitializeComponent();
  rating = new SfRating();
  rating.IsReadOnly = true;
  Content = rating;
}
```



Appearance and Styling in WPF Rating (SfRating)

When the default view is not needed, you can customize the view of WPF SfRating control. The SfRating control provides support to customize the size, item count, and space between rating items.

Note: These properties are available in SfRatingItem. To set this property, use the ItemContainerStyle property of the SfRating control.

Set fill color

The SfRating control supports to set the fill color to the selected and unselected items.

Selected items

The `RatedFill` property fills the rated area with the specified solid color in the SfRating control.

XML

```
<rating:SfRating ItemsCount="5" Value="2" >
```

```
<rating:SfRating.ItemContainerStyle>
<Style TargetType="rating:SfRatingItem">
<Setter Property="RatedFill" Value="Green"></Setter>
</Style>
</rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```



Unselected items

The **UnRatedFill** property fills the unrated area with the specified solid color in the SfRating control.

XML

```
<rating:SfRating ItemsCount="5" Value="2" >
<rating:SfRating.ItemContainerStyle>
<Style TargetType="rating:SfRatingItem">
<Setter Property="UnratedFill" Value="Green">
</Setter>
</Style>
</rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```



Set stroke color

The SfRating control supports to set the stroke color to the selected and unselected items.

Selected items

The **RatedStroke** property sets the stroke color to the rated area with the specified solid color to the selected items in the SfRating control.

XML

```
<rating:SfRating ItemsCount="5" Value="2" >
<rating:SfRating.ItemContainerStyle>
<Style TargetType="rating:SfRatingItem">
<Setter Property="RatedStroke" Value="Green">
</Setter>
</Style>
</rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```



Unselected items

The **UnratedStroke** property sets the stroke color to the unrated area with the specified solid color in the SfRating control.

XML

```
<rating:SfRating ItemsCount="5" Value="2" >
```

```
<rating:SfRating.ItemContainerStyle>
<Style TargetType="rating:SfRatingItem">
<Setter Property="UnratedStroke" Value="Green">
</Setter>
</Style>
</rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```



Set stroke thickness

The SfRating control supports to set the stroke thickness to the selected and unselected items.

Selected items

The **RatedStrokeThickness** property sets the stroke thickness to the rated area with the specified value in the SfRating control.

XML

```
<rating:SfRating ItemsCount="5" Value="2" >
<rating:SfRating.ItemContainerStyle>
<Style TargetType="rating:SfRatingItem">
<Setter Property="RatedStroke" Value="Green"></Setter>
<Setter Property="RatedStrokeThickness" Value="2"></Setter>
</Style>
</rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```



Unselected items

The **UnratedStrokeThickness** property sets the stroke thickness to the unrated area with the specified value in the SfRating control.

XML

```
<rating:SfRating ItemsCount="5" Value="2" >
<rating:SfRating.ItemContainerStyle>
<Style TargetType="rating:SfRatingItem">
<Setter Property="UnratedStroke" Value="Green"></Setter>
<Setter Property="UnratedStrokeThickness" Value="2"></Setter>
</Style>
</rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```



PointerOverFill

The **PointerOverFill** property fills the mouse over area with the specified solid color in the SfRating control.

XML

```
<rating:SfRating ItemsCount="5">
  <rating:SfRating.ItemContainerStyle>
    <Style TargetType="rating:SfRatingItem">
      <Setter Property="PointerOverFill" Value="Green"></Setter>
    </Style>
  </rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```

*PointerOverStroke*

The **PointerOverStroke** property sets the stroke color to the mouse over area with the specified solid color in the SfRating control.

XML

```
<rating:SfRating ItemsCount="5">
  <rating:SfRating.ItemContainerStyle>
    <Style TargetType="rating:SfRatingItem">
      <Setter Property="PointerOverStroke" Value="Green"></Setter>
    </Style>
  </rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```

*PointerOverStrokeThickness*

The **PointerOverStrokeThickness** property sets the stroke thickness to the mouse over area with the specified value in the SfRating control.

XML

```
<rating:SfRating ItemsCount="5">
  <rating:SfRating.ItemContainerStyle>
    <Style TargetType="rating:SfRatingItem">
      <Setter Property="PointerOverStroke" Value="Green"></Setter>
      <Setter Property="PointerOverStrokeThickness" Value="2"></Setter>
    </Style>
  </rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```

*Height*

You can set the height of each SfRatingItem using the **Height** property.

XML

```
<rating:SfRating Value="3" >
```



```
<rating:SfRatingItem Height="20"/>
<rating:SfRatingItem Height="18"/>
<rating:SfRatingItem Height="16"/>
<rating:SfRatingItem Height="14"/>
<rating:SfRatingItem Height="12"/>
</rating:SfRating>
```

C#

```
SfRating rating;
public MainWindow()
{
    InitializeComponent();
    rating = new SfRating();
    rating.Value = 3;
    rating.Items.Add(new SfRatingItem() { Height = 20 });
    rating.Items.Add(new SfRatingItem() { Height = 18 });
    rating.Items.Add(new SfRatingItem() { Height = 16 });
    rating.Items.Add(new SfRatingItem() { Height = 14 });
    rating.Items.Add(new SfRatingItem() { Height = 12 });
    Content = rating;
}
```

SfRatingItems with variable sizes

To set same height to each SfRatingItem, use the `ItemContainerStyle` property.

XML

```
<rating:SfRating Value="4" ItemsCount="10" >
  <rating:SfRating.ItemContainerStyle>
    <Style TargetType="rating:SfRatingItem">
      <Setter Property="Height" Value="12"></Setter>
    </Style>
  </rating:SfRating.ItemContainerStyle>
</rating:SfRating>
```

SfRatingItems with similar sizes

ReportViewer

WPF ReportViewer Overview

Essential ReportViewer is a rendering component to display reports defined in the Microsoft's RDL format (2008 or 2008 R2) on both WPF and Silverlight applications. By using the ReportViewer, you can

display tabular, graphical or free-form reports that make use of relational, multi-dimensional, XML and object data source. It supports both Server and Client reports.

The important features of WPF ReportViewer are listed as follows:

- RDL Specification - Supports RDL Specification for SQL Server 2008 and RDL Specification for SQL Server 2008 R2 only. You can refer MSDN for list of available specifications - [msdn.microsoft.com/library/dd297486\(SQL.100\)](https://msdn.microsoft.com/library/dd297486(SQL.100)).
- Data sources - You can use advanced Database server data sources in the ReportViewer (SQL, Oracle and Azure).
- Charts - Show all basic types of Charts that are available in Microsoft RDL reports.
- Tablix - Shows the summaries and simple tables.
- Gauge - Shows measurement values by using expression values.
- Textbox - Shows textbox data with expression support.
- Report Parameter - View the report based on report parameter value.
- Expression - You can use expression to handle reports.
- Printing - Prints the assigned document.
- Customization - You can customize the ReportViewer appearance.

By using the ReportViewer, you can show variety of interactive Microsoft Report Definition Language (RDL) standard reports. The reports can be exported to PDF and displayed by using ReportViewer.

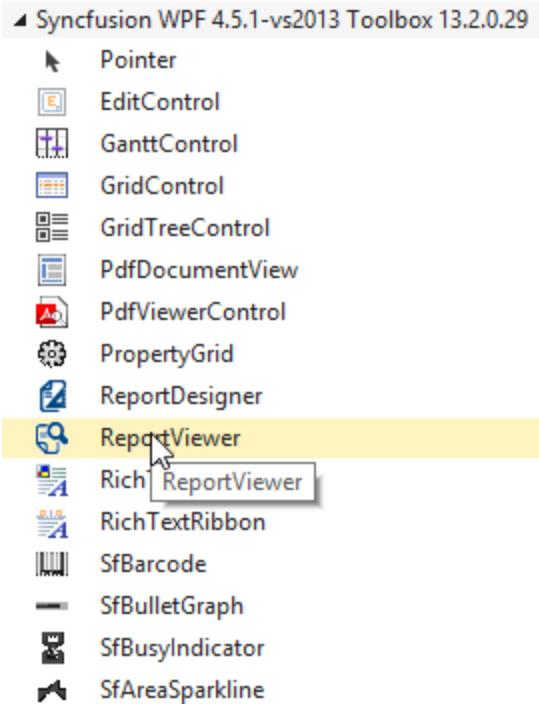
Getting Started with WPF Report Viewer

Creating ReportViewer through Visual Studio

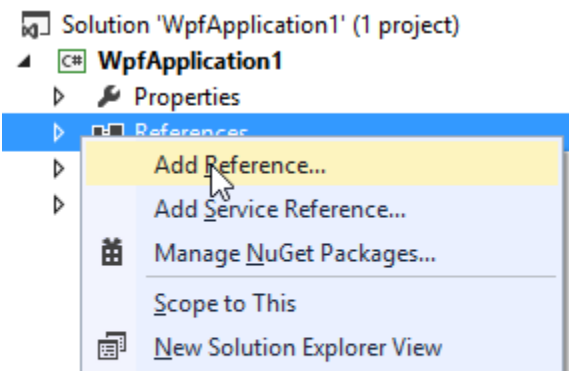
You can create a simple application through the Visual Studio Designer with the Syncfusion WPF ReportViewer control by using the following steps.

1. Create new WPF application in Visual Studio.
2. Add the ReportViewer from the Toolbox.

Drag and drop the ReportViewer control from the Toolbox to the XAML Page.



3. To add references, Right-click on References and select Add Reference.



4. Add the following assemblies.
 - Syncfusion.Chart.Wpf
 - Syncfusion.Compression.Base
 - Syncfusion.DocIO.Base
 - Syncfusion.Gauge.Wpf
 - Syncfusion.Grid.Wpf
 - Syncfusion.Linq.Base
 - Syncfusion.Pdf.Base
 - Syncfusion.PropertyGrid.Wpf
 - Syncfusion.ReportControls.Wpf

- Syncfusion.ReportDesigner.Wpf
- Syncfusion.ReportViewer.Wpf
- Syncfusion.SfMaps.Wpf
- Syncfusion.SfSkinManager.Wpf
- Syncfusion.Shared.Wpf
- Syncfusion.Tools.Wpf
- Syncfusion.XlsIO.Base

Note: Refer the above assemblies from the installed location, C:\Program Files (x86)\Syncfusion\Essential Studio\WPF\{{ site.releaseversion }}\Assemblies

5. Add the following code for creating ReportViewer using code.

XML

```
<sync:ChromelessWindow x:Class="Syncfusion.Samples.ReportView"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:sync="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:Syncfusion.Samples.ViewModel"
xmlns:localUtil="clr-namespace:Syncfusion.Samples.Util"
Title="Product Catalog" WindowStartupLocation="CenterScreen"
SnapsToDevicePixels="True" WindowState="Maximized"
TitleTextAlignment="Center" ShowIcon="False" UseNativeChrome="True"
Icon="App.ico">
<Grid Name="gridSkeleton" localUtil:EventAttachUtil.AddWindowLoaded="True">
<Border Name="groupBoxReportViewer" BorderThickness="0" >
<Grid>
<sync:ReportViewer Name="viewer" />
</Grid>
</Border>
</Grid>
</sync:ChromelessWindow>
```

6. Set **ReportPath** and **ProcessingMode** to the ReportViewer.

XML

```
<sync:ReportViewer Name="viewer"
ReportPath="..\ReportTemplate\InvoiceTemplate.rdl" ProcessingMode="Remote"
/>
```

7. Set Visual Style to the ReportViewer.

CSHARP

```
SkinStorage.SetVisualStyle(this, "Metro");
```

- Run the sample application and you can see the ReportViewer as displayed in the following screenshot.

Invoice

Invoice ID: 10248 View Report

NORTHWIND TRADERS
67 rue des Cinquante Otages
Elgin,
United States.

Invoice No : 10248
Date : 7/16/1996

Bill To: Vins et alcools
Chevalier
59 rue de l'Abbaye
Reims
France

Ship To: Vins et alcools
Chevalier
59 rue de l'Abbaye
Reims
France

Ship Name	Ship Address	Freight	Shipped Date	Ship City	Ship Country
Vins et alcools Chevalier	59 rue de l'Abbaye	\$32.38	7/16/1996	Reims	France

Product ID	Product Name	Quantity	Unit Price	Discount	Price
11	Queso Cabrales	12	\$14.00	0.00%	\$168.00
42	Singaporean Hokkien Fried Mee	10	\$9.80	0.00%	\$98.00
72	Mozzarella di Giovanni	5	\$34.80	0.00%	\$174.00
Total					\$440.00

Thank you for your Business

Show RDLC Reports

- Assign `ReportPath` and `ProcessingMode` to ReportViewer.

CSHARP

```
Syncfusion.Windows.Reports.Viewer.ReportViewer viewer= new
Syncfusion.Windows.Reports.Viewer.ReportViewer;
viewer.ReportPath=@"../ReportTemplate/RDLC/ProductCatalog.rdlc";
```

```
viewer.ProcessingMode = ProcessingMode.Local;
```

2. Set Datasource to the RDLC Report and invoke `RefreshReport` to render the report.

CSHARP

```
viewer.DataSources.Clear();  
viewer.DataSources.Add(new ReportDataSource { Name = "ProductCatalog", Value  
= ProductCatalog.GetData() });  
viewer.RefreshReport();
```

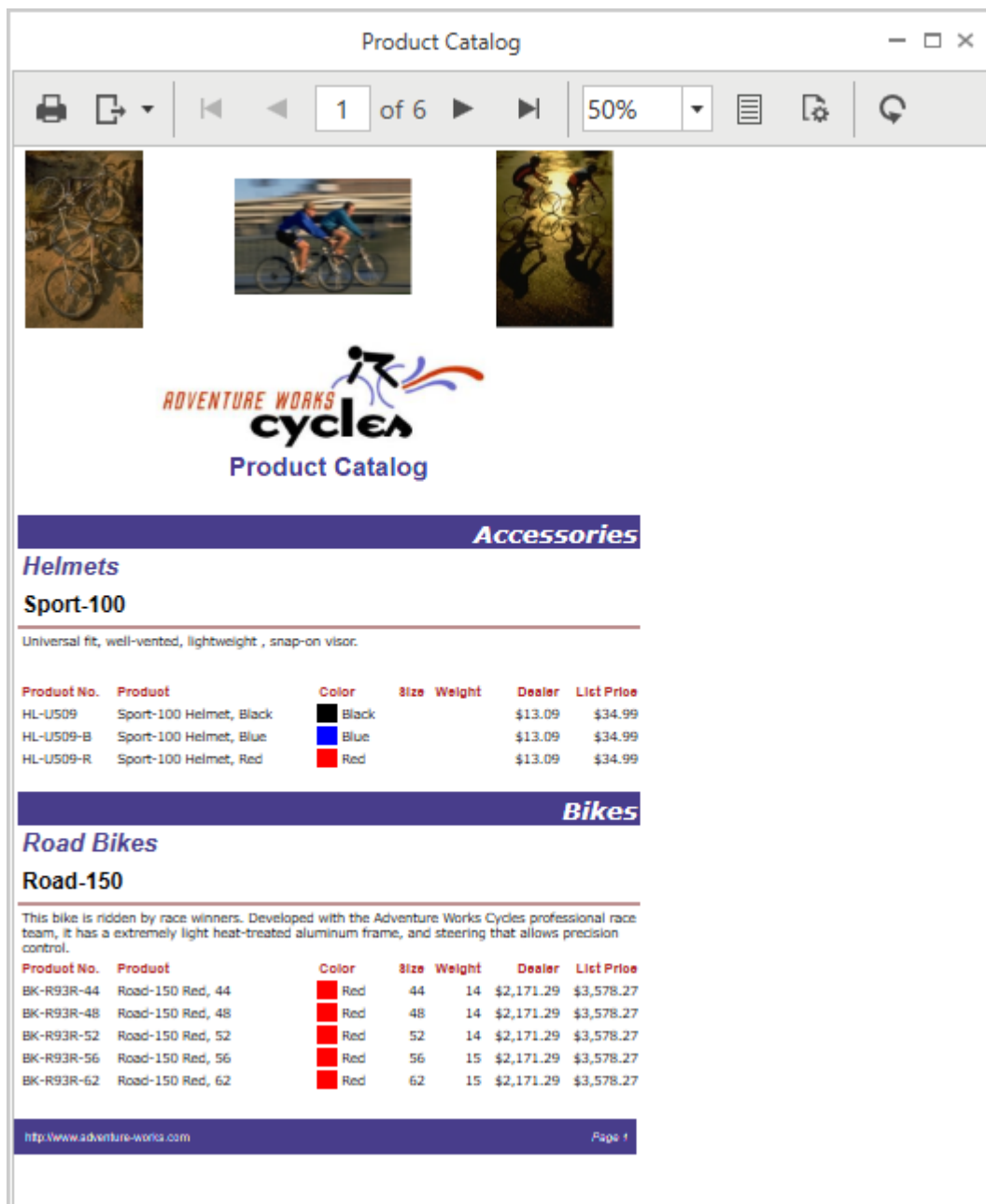
3. Assign values for the datasource in RDLC.

CSHARP

```
#region Product details  
public class ProductCatalog  
{  
    public string ProdSubCat { get; set; }  
    public string ProdModel { get; set; }  
    public string ProdCat { get; set; }  
    public string Description { get; set; }  
    public string ProdName { get; set; }  
    public string ProductNumber { get; set; }  
    public string Color { get; set; }  
    public string Size { get; set; }  
    public double? Weight { get; set; }  
    public double? StandardCost { get; set; }  
    public string Style { get; set; }  
    public string Class { get; set; }  
    public double? ListPrice { get; set; }  
    public static IList GetData()  
    {  
        List<ProductCatalog> datas = new List<ProductCatalog>();  
        ProductCatalog data = null;  
        data = new ProductCatalog()  
        {  
            ProdSubCat = "Road Frames",  
            ProdModel = "HL Road Frame",  
            ProdCat = "Components",  
            Description = "Our lightest and best quality aluminum frame made from the  
newest alloy; it is welded and heat-treated for strength. Our innovative  
design results in maximum comfort and performance.",  
            ProdName = "HL Road Frame - Black, 58",  
            ProductNumber = "FR-R92B-58",  
            Color = "Black",  
            Size = "58",  
            Weight = 2.24,  
            StandardCost = 1059.3100,  
            Style = "U",  
            Class = "H",  
            ListPrice = 1431.5000  
        };  
    }  
}
```

```
datas.Add(data);
data = new ProductCatalog()
{
    ProdSubCat = "Road Frames",
    ProdModel = "HL Road Frame",
    ProdCat = "Components",
    Description = "Our lightest and best quality aluminum frame made from the
newest alloy; it is welded and heat-treated for strength. Our innovative
design results in maximum comfort and performance.",
    ProdName = "HL Road Frame - Red, 58",
    ProductNumber = "FR-R92R-58",
    Color = "Red",
    Size = "58",
    Weight = 2.24,
    StandardCost = 1059.3100,
    Style = "U ",
    Class = "H ",
    ListPrice = 1431.5000
};
datas.Add(data);
data = new ProductCatalog()
{
    ProdSubCat = "Helmets",
    ProdModel = "Sport-100",
    ProdCat = "Accessories",
    Description = "Universal fit, well-vented, lightweight , snap-on visor.",
    ProdName = "Sport-100 Helmet, Red",
    ProductNumber = "HL-U509-R",
    Color = "Red",
    Size = "",
    Weight = null,
    StandardCost = 13.0863,
    Style = "",
    Class = "",
    ListPrice = 34.9900
};
datas.Add(data);
return datas;
}
}
#endregion
```

4. Run the application. The following output is displayed.



Load SSRS Reports

1. To load SSRS reports, initialize ReportViewer control and set the `ReportPath`, `ProcessingMode` and `ReportServerUrl`.

CSHARP

```
Syncfusion.Windows.Reports.Viewer.ReportViewer viewer = new
Syncfusion.Windows.Reports.Viewer.ReportViewer();
viewer.ReportPath = @"//SSRSSamples/Territory Sales";
viewer.ReportServerUrl = @"http://mvc.syncfusion.com/reportserver";
viewer.ProcessingMode = ProcessingMode.Remote;
```



```
viewer.ReportServerCredential = new System.Net.NetworkCredential("ssrs",  
"RDLReport1");
```

2. Add the credential information to the datasource.


CSHARP

```
List<DataSourceCredentials> credentials = new List<DataSourceCredentials>();  
foreach (var dataSource in viewer.GetDataSources())  
{  
    DataSourceCredentials credn = new DataSourceCredentials();  
    credn.Name = dataSource.Name;  
    credn.UserId = "ssrs1";  
    credn.Password = "RDLReport1";  
    credentials.Add(credn);  
}  
viewer.SetDataSourceCredentials(credentials);  
viewer.RefreshReport();
```

3. Run the application. The following output displays.

SSRS Report

1 of 5 50%



Territory Sales

Territory	Sales Person	Order Number	Total Sales
Australia			\$1,943,016.45
	Lynn Tsoffias		\$1,943,016.45
		SO51087	\$4,552.24
		SO51094	\$8,975.46
		SO51134	\$8,278.58
		SO51150	\$40,274.01
		SO51152	\$13,275.10
		SO51173	\$2,254.60
		SO51174	\$502.02
		SO51710	\$27,864.76
		SO51744	\$431.83
		SO51804	\$1,157.26
		SO51806	\$20,704.12
		SO51814	\$90,908.64
		SO51834	\$4,439.73
		SO51840	\$1,004.04
		SO51871	\$272.64
		SO51872	\$486.39
		SO51875	\$61,447.12
		SO53468	\$38,489.23
		SO53498	\$2,606.35
		SO53532	\$2,899.60
		SO53541	\$43,433.62
		SO53555	\$1,322.92
		SO53557	\$545.28
		SO53565	\$43,973.12
		SO53566	\$84,825.90
		SO53579	\$1,696.57
		SO53619	\$1,365.96
		SO55240	\$1,181.23
		SO55244	\$7,122.52
		SO55258	\$9,482.42
		SO55300	\$36,528.62
		SO55307	\$15,775.69
		SO55326	\$1,436.66
		SO55327	\$1,896.77
		SO57032	\$24,175.77
		SO57075	\$67,937.01
		SO57127	\$34,106.14
		SO57147	\$6,585.55
		SO57171	\$1,641.79

ReportViewer API of Essential Studio WPF

Properties

Property	Description	Type	Data Type
ReportPath	Gets or sets the file Reporting Server Report Path or local system path.	Dependency Property	string

DataSources	Get a collection of data sources used by the report.	-	ReportDataSourceCollection
CurrentPage	Gets or sets the current page	Dependency Property	Int
ProcessingMode	Gets or sets the processing mode namely Remote(from ReportingService or process DataSource from Database server) or Local	Dependency Property	Enum
ReportServerUrl	Gets or sets the ReportServerUrl of the Report Server	Dependency Property	String
ShowContextMenu	Gets or sets the ContextMenu visibility	Dependency Property	Boolean
ShowPrintButton	Gets or sets a value that indicates whether Print button is visible on the toolbar.	Dependency Property	Boolean
ShowRefreshButton	Gets or sets a value that indicates whether the Refresh button is visible.	Dependency Property	Boolean
ShowToolBar	Gets or sets a value that indicates whether the toolbar is visible on the control.	Dependency Property	Boolean
ShowZoomControl	Gets or sets a value that indicates whether the Zoom list box is visible.	Dependency Property	Boolean
ViewMode	Gets or sets a value that indicates whether it is Normal or Print View	Dependency Property	enum
ShowPdfExportButton	Gets or sets a value that indicates whether the PDF button is visible	Dependency Property	Boolean
ShowXPSEExportButton	Gets or sets a value that indicates whether the	Dependency Property	Boolean

	XPS Export Button is Visible		
ShowExportControls	Get or set a value that indicates whether the Export control is Visible	Dependency Property	Boolean
ShowPageNavigationControls	Get or set a value that indicates whether the page navigation controls is visible	Dependency Property	Boolean
ReportServerCredential	Credential access to Report Server	Dependency Property	ICredentials
ShowPageLayoutControl	Gets or sets a value that indicates whether the page layout control is visible	Dependency Property	Boolean
ShowParametersBlock	Gets or sets a value that indicates whether the parameter block control is visible	Dependency Property	Boolean

Methods

Method	Description	Parameters	Return Type
RefreshReport	Causes the local report to be rendered with new data.	-	Void
GetParameters	Get the necessary parameters for the report	-	ReportParameterInfoCollection
GetTotalPage	Gets the total pages of the report	-	Void
GetDataSetNames	Get the dataset names from the local report	-	ICollection<string>
LoadReport	Loads the Local report for processing	Stream	void
Print	Displays the Print dialog box.	-	Void
ShowNormalView	Displays the Normal view of the Report	-	Void
SetParameters	Set the necessary parameters for the report	ReportParameter[]	void

Events

Event	Description
ViewModeChanged	The event is triggered when the view is changed to normal and print view
ViewButtonClick	The event is triggered when the view button is clicked
SubreportProcessing	The event is triggered when the report is RDLC and contains with sub report.

ReportViewer Theme support

Theme support for ReportViewer is provided by modifying the UserControl as a ControlTemplate. You can now modify the ReportViewer in your own style, by writing a customized control template for ReportViewer.

To customize a ReportViewer, you have to modify the following control parts.

- PART_exportControl
- PART_comboBoxPageZoom
- PART_Zoom
- PART_ShowError
- PART_PageViewBody
- PART_PageFooterBorder
- PART_PageBodyBorder
- PART_PageHeaderBorder
- PART_scrollViewer
- PART_scrollIDSCredentialBlock
- PART_groupBoxExpandedExceptionScroll
- PART_scrollViewerParamBlock
- PART_canvasContentPage
- PART_CanvasFooter
- PART_CanvasHeader
- PART_treeItemArea
- PART_renderArea
- PARTgridReportParameterBlock
- PART_gridRenderingRegion
- PART_ExceptionGrid
- PART_gridException
- PART_gridLoadingIndicator
- PART_MainGrid
- PART_viewerSplitter
- PART_PageView
- PARTsPanel/Head
- PART_PageViewContainer
- PART_toolBar
- PART_DocumentMap
- PART_btnViewReport1
- PART_buttonNext
- PART_buttonParameters

- PART_buttonPrint
- PART_buttonFirst
- PART_buttonLast
- PART_buttonPrevious
- PART_buttonShowOrHideDocumentMap
- PART_buttonPrintLayout
- PART_buttonPageSetup
- PART_buttonRefresh
- PART_buttonViewReport
- PART_buttonFind
- PART_button back
- PART_toggleShowDetails
- PART_textBoxTotalPages
- PART_labelOf
- PART_textBlockException
- PART_textBoxFind
- PART_textBoxCurrentPage
- PART_textBlockStackTrace
- PART_gridExceptionRow
- PART_loadingIndicatorRow
- PART_toolBarGridRow
- PART_CredentialRow
- PART_parameterGridRow
- PART_viewerContentRow

The following code example illustrates a ReportViewer control template.

XML

```
<Style TargetType="local:ReportViewer">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="local:ReportViewer">
        <Grid x:Name="PART_MainGrid" Height="Auto" Margin="0,0,0,0"
          Background="Transparent">
          <Grid.RowDefinitions>
            <RowDefinition Name="PART_toolBarGridRow" Height="28"/>
            <RowDefinition Name="PART_dsCredentialRow" Height="Auto"/>
            <RowDefinition Name="PART_parameterGridRow" Height="Auto"/>
            <RowDefinition Name="PART_viewerContentRow" Height="*" />
            <RowDefinition Name="PART_gridExceptionRow" Height="0"/>
            <RowDefinition Name="PART_loadingIndicatorRow" Height="0"/>
          </Grid.RowDefinitions>
          <!--Toolbar-->
          <StackPanel Name="PART_toolBar" Height="28" Orientation="Horizontal"
            Grid.ColumnSpan="2" Margin="0">
            <StackPanel.Background>
              <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                <GradientStop Color="#FFFEFBF4" Offset="0.027"/>
                <GradientStop Color="#FFEAEDEF" Offset="0.029"/>
                <GradientStop Color="#FFDCE4F1" Offset="0.498"/>
                <GradientStop Color="#FFE6EAF3" Offset="0.966"/>
                <GradientStop Color="FloralWhite" Offset="0.968"/>
              </LinearGradientBrush>
            </StackPanel.Background>
          </StackPanel>
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

```

<GradientStop Color="#FFD4DBEB" Offset="0.503"/>
</LinearGradientBrush>
</StackPanel.Background>
<!--Print-->
<Button Style="{StaticResource ButtonStyle}" Name="PART_buttonPrint"
Width="22" Height="22" ToolTip="{x:Static prop:Resources.menuItemPrint}"
IsEnabled="False" Margin="2,3,2,3" >
<Image Stretch="None">
<Image.Style>
<Style>
<Style.Triggers>
<Trigger Property="Button.IsEnabled" Value="False">
<Setter Property="Image.Source" Value="{StaticResource PrintDisabled}"/>
</Trigger>
<Trigger Property="Button.IsEnabled" Value="True">
<Setter Property="Image.Source" Value="{StaticResource Print}"/>
</Trigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<!--PrintDisabled-->
</Button>
<ComboBox Name="PART_exportControl" Height="22" Width="35"
Style="{StaticResource ExportComboBox}" ItemContainerStyle="{StaticResource
ItemTemplate}" IsEnabled="False" ToolTip="Export" />
<Rectangle HorizontalAlignment="Center" Height="21.599"
VerticalAlignment="Center" Width="1.6">
<Rectangle.Fill>
<LinearGradientBrush EndPoint="0.662,0.5" StartPoint="0.338,0.5">
<GradientStop Color="#FFBCBCBC" Offset="0.508"/>
<GradientStop Color="White" Offset="0.525"/>
</LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<!--Navigates to First-->
<Button Style="{StaticResource ButtonStyle}" Name="PART_buttonFirst"
ToolTip="{x:Static prop:Resources.menuItemFirst}" Width="22" Height="22"
IsEnabled="False" Margin="0,3,0,3" >
<Image Stretch="None">
<Image.Style>
<Style>
<Style.Triggers>
<Trigger Property="Button.IsEnabled" Value="False">
<Setter Property="Image.Source" Value="{StaticResource First_NavDisabled}"/>
</Trigger>
<Trigger Property="Button.IsEnabled" Value="True">
<Setter Property="Image.Source" Value="{StaticResource First_Nav}"/>
</Trigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<!--First_NavDisabled-->
</Button>
<!--Navigates to Previous-->

```

```

<Button Style="{StaticResource ButtonStyle}" Name="PART_buttonPrevious"
Width="22" Height="22" ToolTip="{x:Static prop:Resources.menuItemPrevious}"
IsEnabled="False" Margin="2,3,2,3" >
<Image>
<Image.Style>
<Style>
<Style.Triggers>
<Trigger Property="Button.IsEnabled" Value="False">
<Setter Property="Image.Source" Value="{StaticResource
Previous_NavDisabled}"/>
</Trigger>
<Trigger Property="Button.IsEnabled" Value="True">
<Setter Property="Image.Source" Value="{StaticResource Previous_Nav}"/>
</Trigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<!--Previous_NavDisabled-->
</Button>
<!--Current Page-->
<TextBox Name="PART_textBoxCurrentPage" Width="60" Height="22"
IsReadOnly="False" IsEnabled="False" BorderBrush="Gray" Margin="2,3,2,3" >
</TextBox>
<TextBlock Name="PART_labelOf" Width="Auto" VerticalAlignment="Center"
Foreground="Black" Margin="2,3,5,3">of</TextBlock>
<!--Total number of pages-->
<TextBlock Name="PART_textBoxTotalPages" Width="Auto" xml:space="preserve"
IsEnabled="false" Margin="2,3,5,3" VerticalAlignment="Center"
Foreground="Black" TextAlignment="Center" HorizontalAlignment="Center"/>
<!--Navigates to Next-->
<Button Style="{StaticResource ButtonStyle}" Name="PART_buttonNext"
Width="22" Height="22" ToolTip="{x:Static prop:Resources.menuItemNext}"
IsEnabled="False" Margin="2,3,2,3" >
<Image Stretch="None">
<Image.Style>
<Style>
<Style.Triggers>
<Trigger Property="Button.IsEnabled" Value="False">
<Setter Property="Image.Source" Value="{StaticResource Next_NavDisabled}"/>
</Trigger>
<Trigger Property="Button.IsEnabled" Value="True">
<Setter Property="Image.Source" Value="{StaticResource Next_Nav}"/>
</Trigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<!--Next_NavDisabled-->
</Button>
<!--Navigates to Last-->
<Button Style="{StaticResource ButtonStyle}" Name="PART_buttonLast"
Width="22" Height="22" ToolTip="{x:Static prop:Resources.menuItemLast}"
IsEnabled="False" Margin="2,3,2,3" >
<Image Stretch="None">
<Image.Style>
<Style>

```



```

<Style.Triggers>
<Trigger Property="Button.IsEnabled" Value="False">
<Setter Property="Image.Source" Value="{StaticResource Last_NavDisabled}"/>
</Trigger>
<Trigger Property="Button.IsEnabled" Value="True">
<Setter Property="Image.Source" Value="{StaticResource Last_Nav}"/>
</Trigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<!--Last_NavDisabled-->
</Button>
<!--Back Parent Report-->
<Button Style="{StaticResource ButtonStyle}" Name="PART_btnback" Width="22"
Height="22" ToolTip="{x:Static prop:Resources.toolTipBack}" IsEnabled="True"
Margin="2,3,2,3" Visibility="Collapsed">
<Image Source="{StaticResource ReportBack}" Stretch="None"/>
</Button>
<Rectangle HorizontalAlignment="Center" Height="21.599"
VerticalAlignment="Center" Width="1.6">
<Rectangle.Fill>
<LinearGradientBrush EndPoint="0.662,0.5" StartPoint="0.338,0.5">
<GradientStop Color="#FFBCBCBC" Offset="0.508"/>
<GradientStop Color="White" Offset="0.525"/>
</LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<!--Zooming-->
<Border Name="PART_comboBoxExternalBorder" BorderBrush="Gray" Width="70"
Height="22" ToolTip="{x:Static prop:Resources.toolTipZoom}"
BorderThickness="0" Margin="2,3,2,3">
<ComboBox Name="PART_comboBoxPageZoom" Width="70" Height="22"
SelectedIndex="2" IsEnabled="False">
<ComboBoxItem Content="25%"/>
<ComboBoxItem Content="50%"/>
<ComboBoxItem Content="100%"/>
<ComboBoxItem Content="200%"/>
<ComboBoxItem Content="300%"/>
<ComboBoxItem Content="400%"/>
<ComboBoxItem Content="500%"/>
</ComboBox>
</Border>
<!--Prints Layout-->
<ToggleButton Style="{StaticResource ViewerToggleButtonStyle}"
Name="PART_buttonPrintLayout" Width="22" Height="22" ToolTip="{x:Static
prop:Resources.menuItemPrintLayout}" IsEnabled="False" Margin="2,3,2,3">
<Image Stretch="None" >
<Image.Style>
<Style>
<Style.Triggers>
<Trigger Property="ToggleButton.IsEnabled" Value="False">
<Setter Property="Image.Source" Value="{StaticResource
PrintLayoutXDisabled}"/>
</Trigger>
<Trigger Property="ToggleButton.IsEnabled" Value="True">
<Setter Property="Image.Source" Value="{StaticResource PrintLayoutX}"/>

```

```

</Trigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<!--PrintLayoutXDisabled-->
</ToggleButton>
<!--Prints Setup-->
<Button Style="{StaticResource ButtonStyle}" Name="PART_buttonPageSetup"
Width="22" Height="22" ToolTip="{x:Static prop:Resources.toolTipPageSetup}"
IsEnabled="False" Margin="2,3,2,3" Visibility="Visible" >
<Image Source="{StaticResource PageSetup}" Stretch="None"/>
<!--PrintSetupDisabled-->
</Button>
<Rectangle HorizontalAlignment="Center" Height="21.599"
VerticalAlignment="Center" Width="1.6">
<Rectangle.Fill>
<LinearGradientBrush EndPoint="0.662,0.5" StartPoint="0.338,0.5">
<GradientStop Color="#FFBCBCBC" Offset="0.508"/>
<GradientStop Color="White" Offset="0.525"/>
</LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<!--Refresh -->
<Button Style="{StaticResource ButtonStyle}" Height="22" Width="22"
Name="PART_buttonRefresh" ToolTip="{x:Static
prop:Resources.menuItemRefresh}" IsEnabled="False" Margin="2,3,2,3">
<Image HorizontalAlignment="Center" VerticalAlignment="Center" Height="16"
Width="16" Stretch="Fill" >
<Image.Style>
<Style>
<Style.Triggers>
<Trigger Property="Button.IsEnabled" Value="False">
<Setter Property="Image.Source" Value="{StaticResource RefreshDisabled}"/>
</Trigger>
<Trigger Property="Button.IsEnabled" Value="True">
<Setter Property="Image.Source" Value="{StaticResource Refresh}"/>
</Trigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<!--RefreshDisabled-->
</Button>
<!--Shows/Hides Document map-->
<Button Style="{StaticResource ButtonStyle}"
Name="PART_buttonShowOrHideDocumentMap" Width="22" ToolTip="{x:Static
prop:Resources.menuItemShowHide}" IsEnabled="False" Visibility="Collapsed"
Margin="2,3,2,3">
<Image Source="{StaticResource ShowHideDisabled}" Stretch="None"
HorizontalAlignment="Center" VerticalAlignment="Center" Height="16"
Width="16"/>
<!--ShowHideDisabled-->
</Button>
<!--Parameterised query-->
<Button Style="{StaticResource ButtonStyle}" Name="PART_buttonParameters"
Width="22" Height="22" ToolTip="{x:Static prop:Resources.toolTipParameters}"

```

```

IsEnabled="False" Visibility="Collapsed" Margin="2,3,2,3"
Background="Transparent" >
<Image Stretch="None" >
<Image.Style>
<Style>
<Style.Triggers>
<Trigger Property="Button.IsEnabled" Value="False">
<Setter Property="Image.Source" Value="{StaticResource
ParametersDisabled}"/>
</Trigger>
<Trigger Property="Button.IsEnabled" Value="True">
<Setter Property="Image.Source" Value="{StaticResource Parameters}"/>
</Trigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<!--ParametersDisabled-->
</Button>
<Rectangle HorizontalAlignment="Center" Height="21.599"
VerticalAlignment="Center" Width="1.6">
<Rectangle.Fill>
<LinearGradientBrush EndPoint="0.662,0.5" StartPoint="0.338,0.5">
<GradientStop Color="#FFBCBCBC" Offset="0.508"/>
<GradientStop Color="White" Offset="0.525"/>
</LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<TextBox Name="PART_textBoxFind" Width="60" Height="22" xml:space="preserve"
IsEnabled="False" BorderBrush="Gray" ToolTip="{x:Static
prop:Resources.toolTipFindTextBox}" Margin="2,3,2,3"
Visibility="Collapsed"/>
<!--Find-->
<Button Style="{StaticResource ButtonStyle}" Name="PART_buttonFind"
Width="22" Height="22" ToolTip="{x:Static prop:Resources.toolTipFind}"
IsEnabled="False" Visibility="Collapsed" Margin="2,3,2,3">
<Image Stretch="None">
<Image.Style>
<Style>
<Style.Triggers>
<Trigger Property="Button.IsEnabled" Value="False">
<Setter Property="Image.Source" Value="{StaticResource FindDisabled}"/>
</Trigger>
<Trigger Property="Button.IsEnabled" Value="True">
<Setter Property="Image.Source" Value="{StaticResource Find}"/>
</Trigger>
</Style.Triggers>
</Style>
</Image.Style>
</Image>
<!--FindDisabled-->
</Button>
</StackPanel>
<!-- DataSource Credential Prompt -->
<Grid Grid.Row="1" FlowDirection="LeftToRight">
<Grid.RowDefinitions>

```

```

<RowDefinition Height="Auto"
x:Name="PART_rowDefinitionFordsCredentialBlock"/>
</Grid.RowDefinitions>
<ScrollView Name="PART_scrollDSCredentialBlock"
HorizontalAlignment="Stretch" VerticalScrollBarVisibility="Auto"
HorizontalScrollBarVisibility="Disabled" Visibility="Collapsed">
<StackPanel Name="PART_stackPaneldsCredential" Grid.Row="0"
Background="LightBlue" Width="Auto" Height="Auto">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="300*" />
<ColumnDefinition Width="100*" />
</Grid.ColumnDefinitions>
<Grid Background="#efefef" x:Name="PART_grid_DsCredentialblock"
Grid.Column="0" Width="Auto">
<StackPanel Width="Auto" Name="PART_sPanel_Head">
</StackPanel>
</Grid>
<StackPanel Grid.Column="1" Background="#efefef">
<Button Content="{x:Static prop:Resources.btnViewReport}"
HorizontalAlignment="Right" x:Name="PART_btnViewReport1" Width="Auto"
Margin="0,5,20,5"/>
</StackPanel>
</Grid>
</StackPanel>
</ScrollView>
</Grid>
<Grid Grid.Row="2" FlowDirection="LeftToRight">
<!--Parameter prompt panel-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" x:Name="PART_rowDefinitionForParameterBlock"/>
</Grid.RowDefinitions>
<ScrollView x:Name="PART_scrollViewerParamBlock"
HorizontalAlignment="Stretch" VerticalScrollBarVisibility="Auto"
HorizontalScrollBarVisibility="Disabled" Visibility="Collapsed">
<StackPanel Name="PART_stackPanelParameters" Grid.Row="0"
Background="LightBlue" Width="Auto" Height="Auto">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="100*" />
</Grid.ColumnDefinitions>
<Grid Background="#efefef" x:Name="PART_grid_ReportParameterBlock"
Grid.Column="0" >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
</Grid.RowDefinitions>
</Grid>
<StackPanel Grid.Column="1" Background="#efefef">
<Button Content="{x:Static prop:Resources.btnViewReport}"
HorizontalAlignment="Right" x:Name="PART_btnViewReport" Width="Auto"
Margin="0,5,20,5"/>

```

```

</StackPanel>
</Grid>
</StackPanel>
</ScrollView>
</Grid>
<Grid Name="PART_gridRenderingRegion" Grid.Row="3" Height="Auto"
Margin="0,0,0,0" FlowDirection="LeftToRight">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"></RowDefinition>
<RowDefinition Height="*"></RowDefinition>
</Grid.RowDefinitions>
<Grid Name="PART_ExceptionGrid" Grid.Row="0" Visibility="Collapsed"
Background="LightBlue">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="24"></ColumnDefinition>
<ColumnDefinition Width="749"></ColumnDefinition>
<ColumnDefinition Width="0" />
</Grid.ColumnDefinitions>
<Image Source="{StaticResource error}" Height="24" Width="24"
Margin="5,5,5,5" Grid.ColumnSpan="2" HorizontalAlignment="Left">
</Image>
<TextBlock Grid.Column="1" HorizontalAlignment="Stretch"
VerticalAlignment="Center" Height="23" Margin="10,7,10,0" Width="730">
ReportViewer encountered some issues loading this report. Please click
<Hyperlink x:Name="PART_hyperlink">here </Hyperlink> to see details of the
issues.
</TextBlock>
</Grid>
<Grid Grid.Row="1" x:Name="PART_renderArea">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"></ColumnDefinition>
<ColumnDefinition Width="Auto"></ColumnDefinition>
<ColumnDefinition Width="*"></ColumnDefinition>
</Grid.ColumnDefinitions>
<Grid Grid.Column="0" Height="Auto" Width="Auto" x:Name="PART_treeItemArea"
Visibility="Collapsed">
<TreeView Name="PART_DocumentMap" Height="Auto" Width="Auto"
BorderThickness="0"/>
</Grid>
<GridSplitter x:Name="PART_viewerSplitter" Visibility="Collapsed"
Grid.Column="1" ShowsPreview="True" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" ResizeBehavior="PreviousAndNext" Width="5"
Background="#FFD4DBEB"/>
<!--Viewer Page-->
<ScrollView x:Name="PART_scrollViewer" Grid.Column="2" Height="Auto"
Width="Auto" HorizontalScrollBarVisibility="Auto"
VerticalScrollBarVisibility="Auto" HorizontalContentAlignment="Center"
VerticalContentAlignment="Center">
<ScrollView.Resources>
<DataTemplate x:Key="ReportViewerCellTeaemplate">
<ContentControl Content="{Binding Path=CellBoundValue}"></ContentControl>
</DataTemplate>
</ScrollView.Resources>
<StackPanel Orientation="Vertical" Name="PART_PageView" Height="Auto"
Width="Auto" Background="White">
<StackPanel.LayoutTransform>

```

```

<ScaleTransform x:Name="PART_Zoom"></ScaleTransform>
</StackPanel.LayoutTransform>
<Border x:Name="PART_PageViewBody" Background="Transparent" Height="Auto"
Width="Auto">
<StackPanel x:Name="PART_PageViewContainer" Background="Transparent"
Height="Auto" Width="Auto">
<Border Name="PART_PageHeaderBorder" Height="Auto" Width="Auto"
Background="Transparent">
<Canvas x:Name="PART_CanvasHeader" Visibility="Collapsed"
Background="Transparent"/>
</Border>
<Border Name="PART_PageBodyBorder" Height="Auto" Width="Auto"
Background="Transparent">
<Canvas x:Name="PART_canvasContentPage" Grid.Row="2"
Background="Transparent" Width="Auto" Height="Auto"/>
</Border>
<Border Name="PART_PageFooterBorder" Height="Auto" Width="Auto"
Background="Transparent">
<Canvas x:Name="PART_CanvasFooter" Visibility="Collapsed"
Background="Transparent"/>
</Border>
</StackPanel>
</Border>
</StackPanel>
</ScrollViewer>
</Grid>
</Grid>
</Grid>
<Grid x:Name="PART_gridException" Grid.Row="4" Background="White"
Visibility="Collapsed" FlowDirection="LeftToRight">
<StackPanel Orientation="Vertical" HorizontalAlignment="Center">
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" >
<Image Name="PART_imageException" Source="{StaticResource Exception}"
Width="48" HorizontalAlignment="Center"/>
<TextBlock Name="PART_textBlockException">
</TextBlock>
</StackPanel>
<ScrollViewer Name="PART_groupBoxExpandedExceptionScroll" Grid.Row="3"
VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto"
Height="400">
<GroupBox Name="PART_groupBoxExpandedException" Width="420" Margin="5,0,5,0"
FlowDirection="RightToLeft" BorderThickness="0">
<GroupBox.Header>
<ToggleButton Name="PART_toggleShowDetails" HorizontalAlignment="Right"
Margin="0,0,0,0" VerticalAlignment="Bottom">Show Details</ToggleButton>
</GroupBox.Header>
<WrapPanel Name="PART_ShowError" Visibility="Collapsed" Width="auto">
<TextBox Name="PART_textBlockStackTrace" TextWrapping="Wrap"
FlowDirection="LeftToRight" Width="390" IsReadOnly="True" ></TextBox>
<TextBox Name="PART_textBlockStackTraceMessage" TextWrapping="Wrap"
FlowDirection="LeftToRight" Width="390" IsReadOnly="True" ></TextBox>
</WrapPanel>
</GroupBox>
</ScrollViewer>
</StackPanel>
</Grid>

```

```

<Grid x:Name="PART_gridLoadingIndicator" Grid.Row="5" Background="White"
Visibility="Collapsed" FlowDirection="LeftToRight">
<loading:LoadingIndicator></loading:LoadingIndicator>
</Grid>
</Grid>
<ControlTemplate.Triggers>
<Trigger Property="local:ReportViewer.ViewMode" Value="Normal">
<Setter TargetName="PART_gridRenderingRegion" Property="Background"
Value="White"/>
</Trigger>
<Trigger Property="local:ReportViewer.ViewMode" Value="Print">
<Setter TargetName="PART_gridRenderingRegion" Property="Background"
Value="LightGray"/>
</Trigger>
</ControlTemplate.Triggers>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

Limitations in WPF ReportViewer control

RDL Specification support

- ReportViewer control does not support RDL Specification for SQL Server 2000 and RDL Specification for SQL Server 2005.

Layout Process

- Syncfusion ReportViewer control has some limitations in Tablix Cell split Layout process in comparison with MS ReportViewer. When table cell width value exceeds the page width, the entire cell moves to the next page in order to display the complete cell items.

Unsupported expression

- Object function and VB function do not have complete support in ReportViewer.
- VB Code functions are not supported in Silverlight and WinRT ReportViewer.

HTML Formatted Data with Report

Report viewer supports showing the HTML formatted data with Textbox report items along with the inline CSS. This section provides the information about supported tags and limitations.

Supported HTML Tags

- Hyperlinks:
- Fonts:
- Header, style and block elements: `,

`,`

- ,
- ,
- ,`
- Text format: ,,,

Limitations of Cascading Style Sheet Attributes

The following is a list of attributes that are supported:

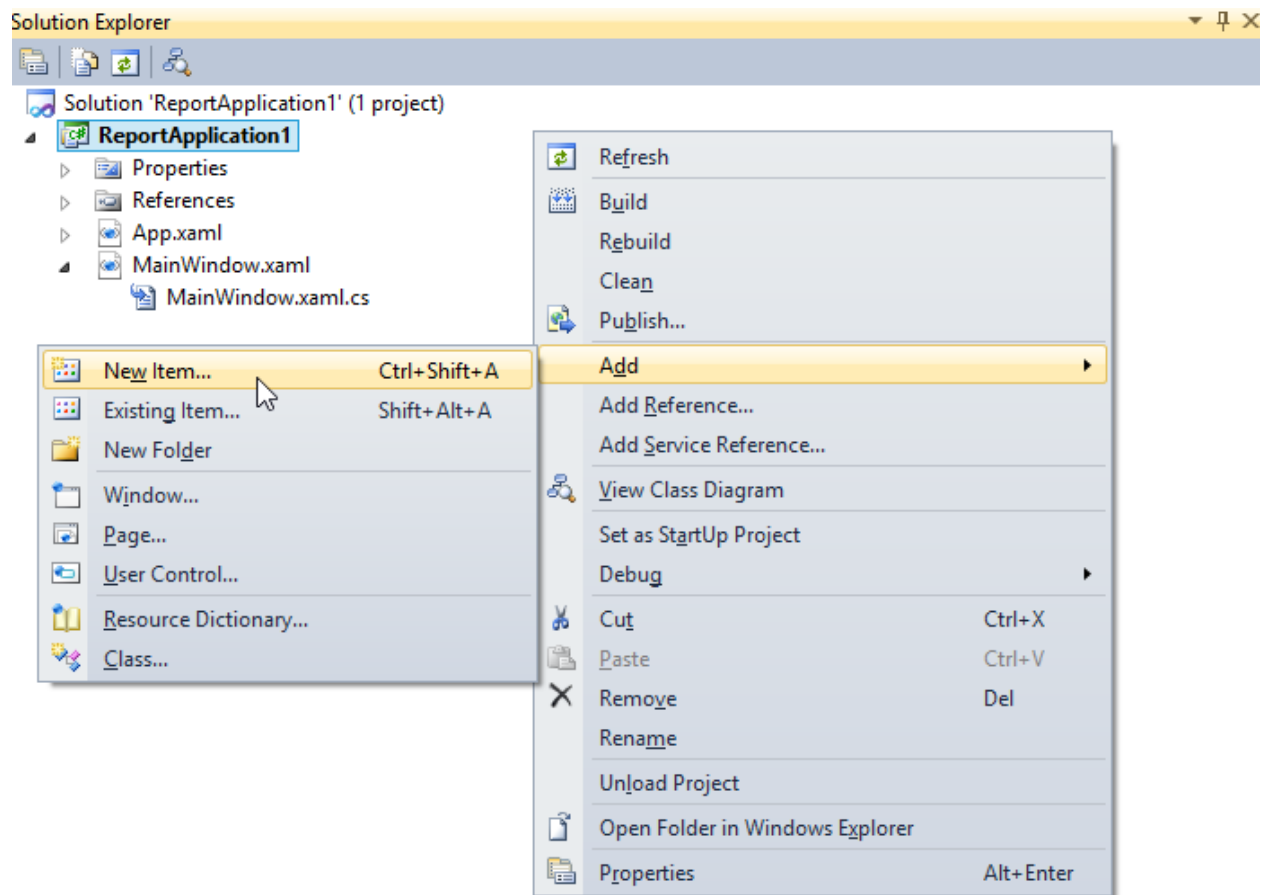
- text-align, text-indent
- font-family
- font-size
- Only valid RDL size values are supported in absolute CSS length units. Supported units are: in, cm, mm, pt, pc, px, ex, and em.
- Relative CSS length units are ignored and are not supported. Unsupported units include percentage (%), and rem.
- color
- padding, padding-bottom, padding-top, padding-right, and padding-left
- font-weight

How to

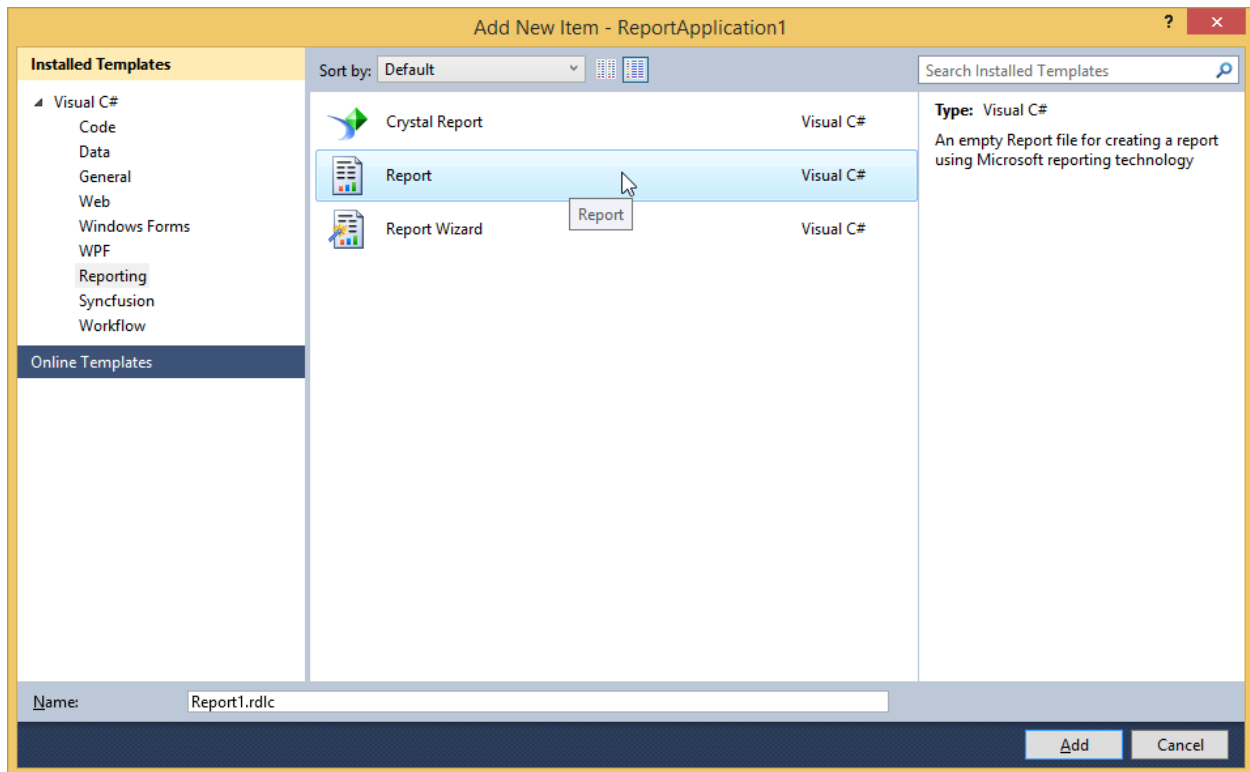
Create RDLC Report In VS2010 and Show It in Report Viewer

This section covers the steps to create RDLC report in VS2010 and shows the created RDLC report in the Report viewer.

1. Create a new WPF application with .NET Framework 4. The Solution Explorer dialog opens.
2. To add a new RDLC report in the WPF application, right-click on the newly added WPF application in the Solution Explorer.
3. Select Add, and then click New Item.



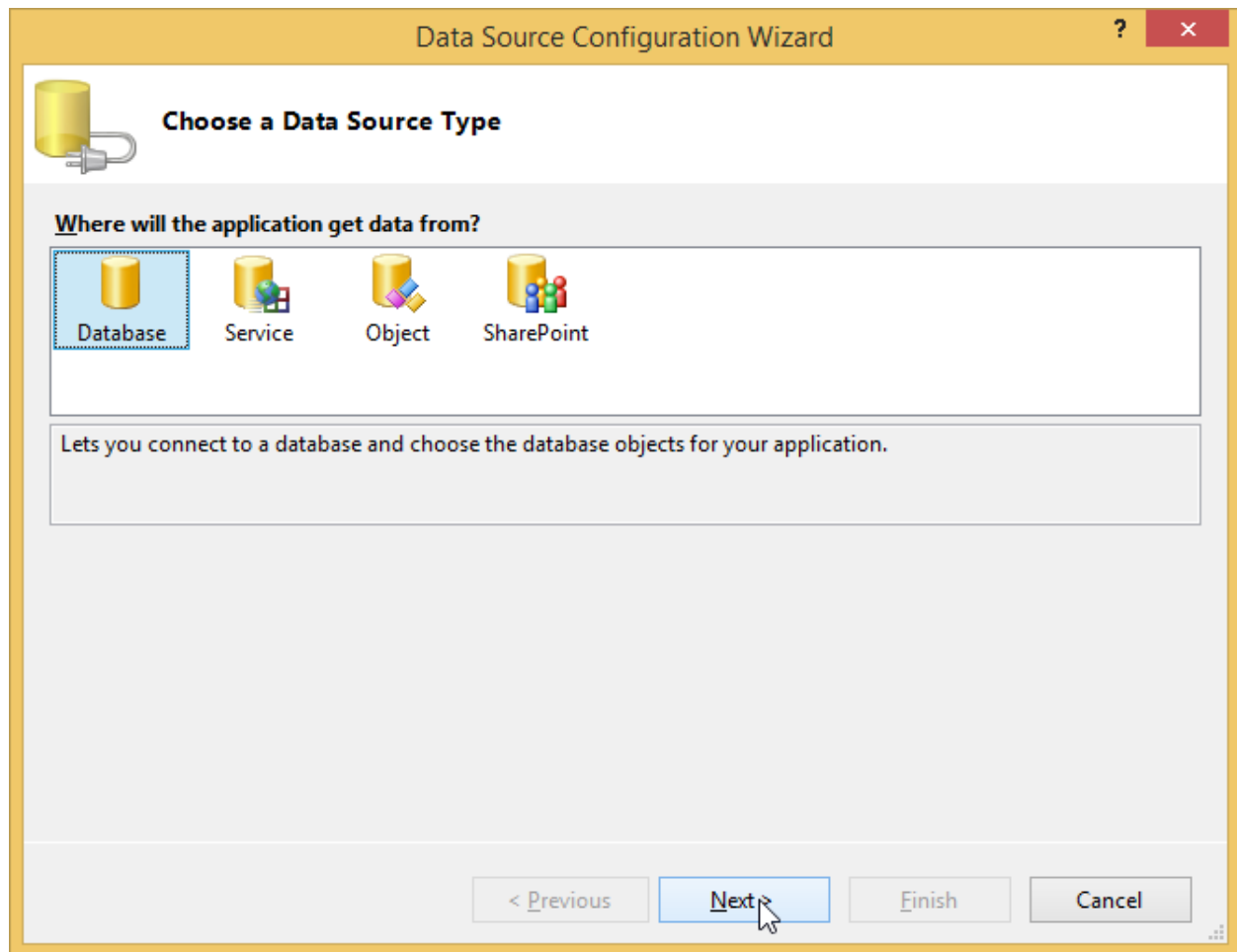
4. Then Add New Item dialog opens.
5. To create a dataset for the RDLC report, click Reporting under Visual C# Items.
6. Click Report, and then click Add.



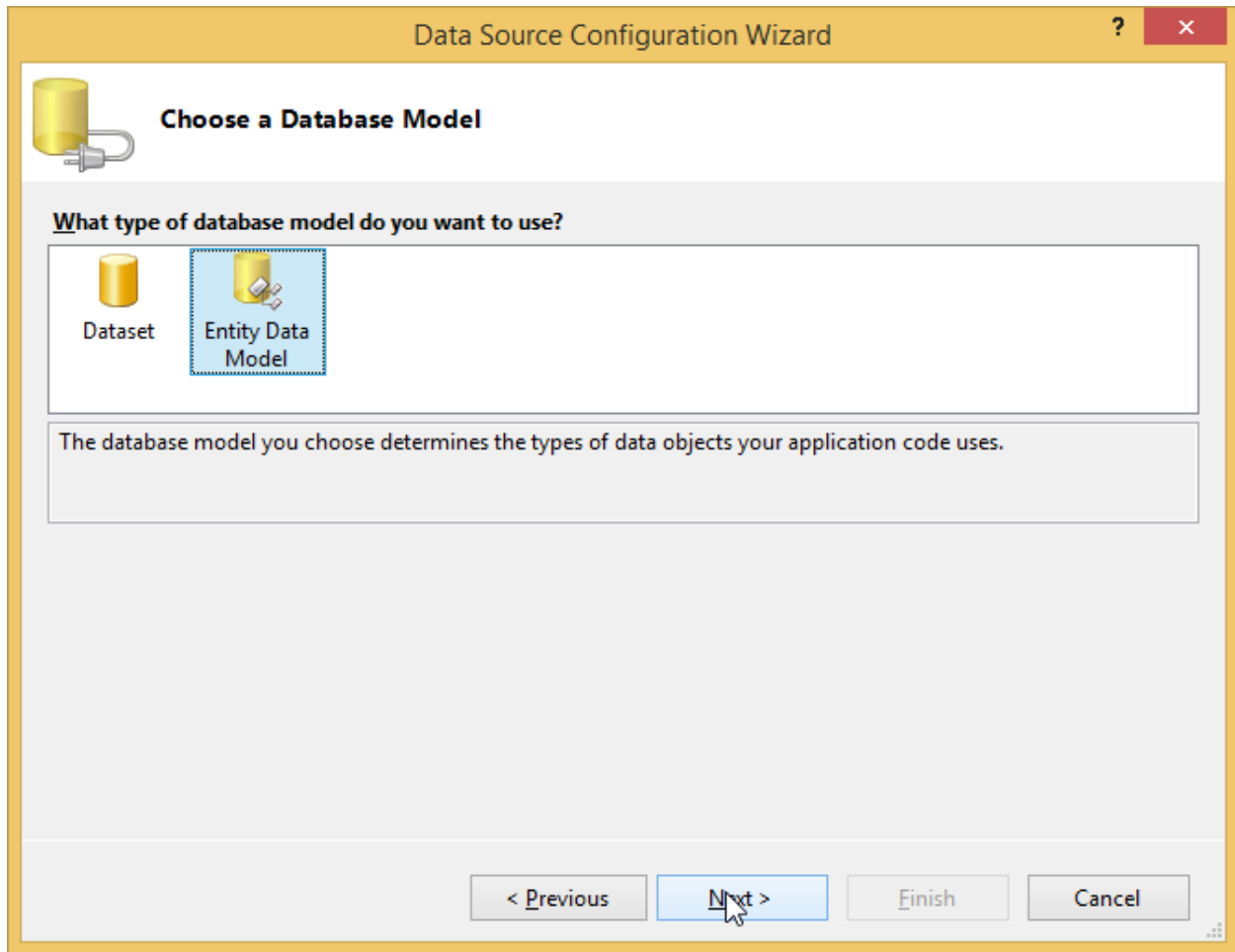
7. The Report Wizard opens.
8. Enter a dataset name in Name field. To choose a data source for the dataset, click New on the right of Data source drop-down combo box.

[illegible]

9. Then Data Source Configuration Wizard opens.
10. Click Database under Where will the application get data from? and then click Next.

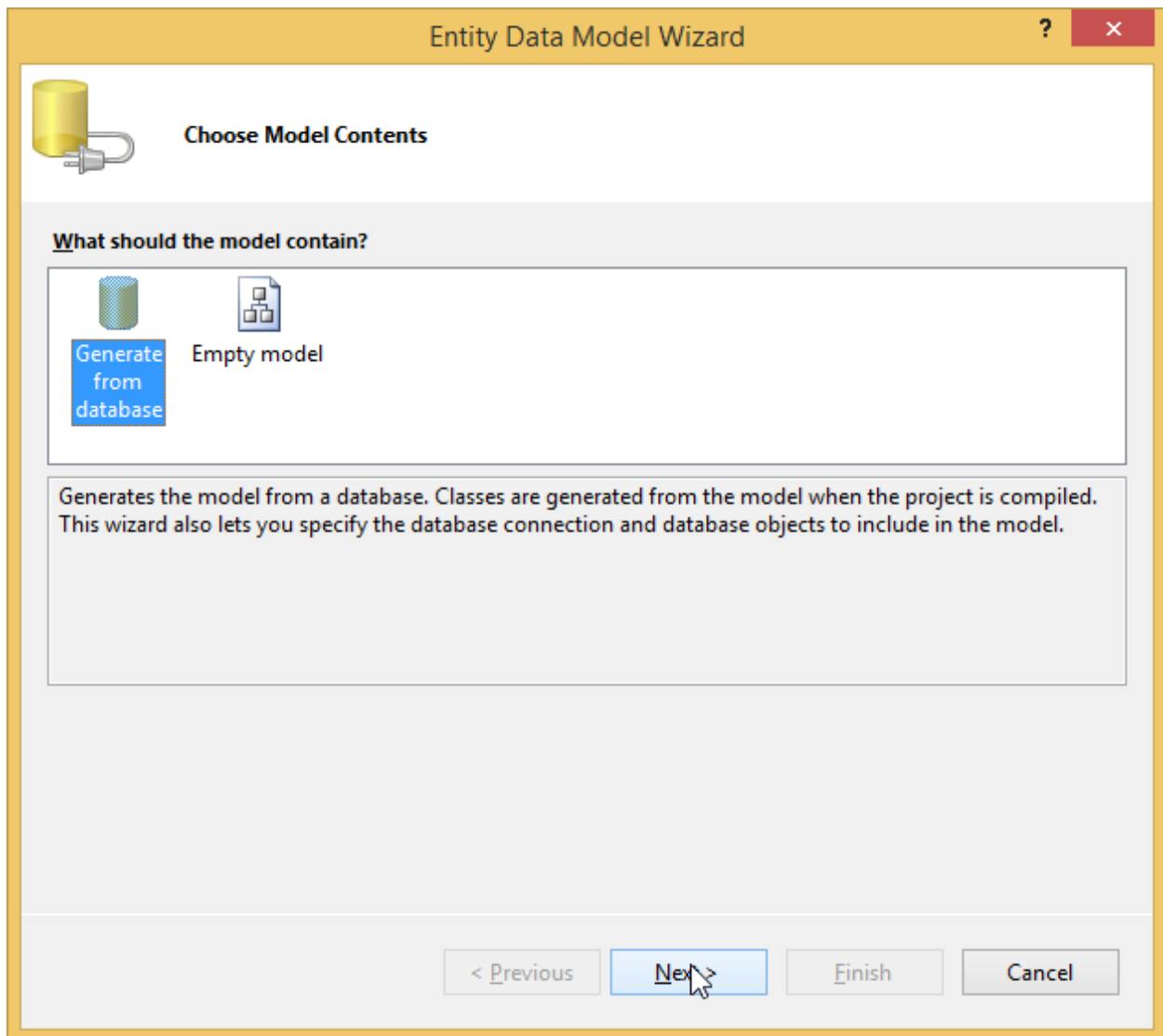


11. Click Entity Data Model under What type of database model do you want to use? and click Next.

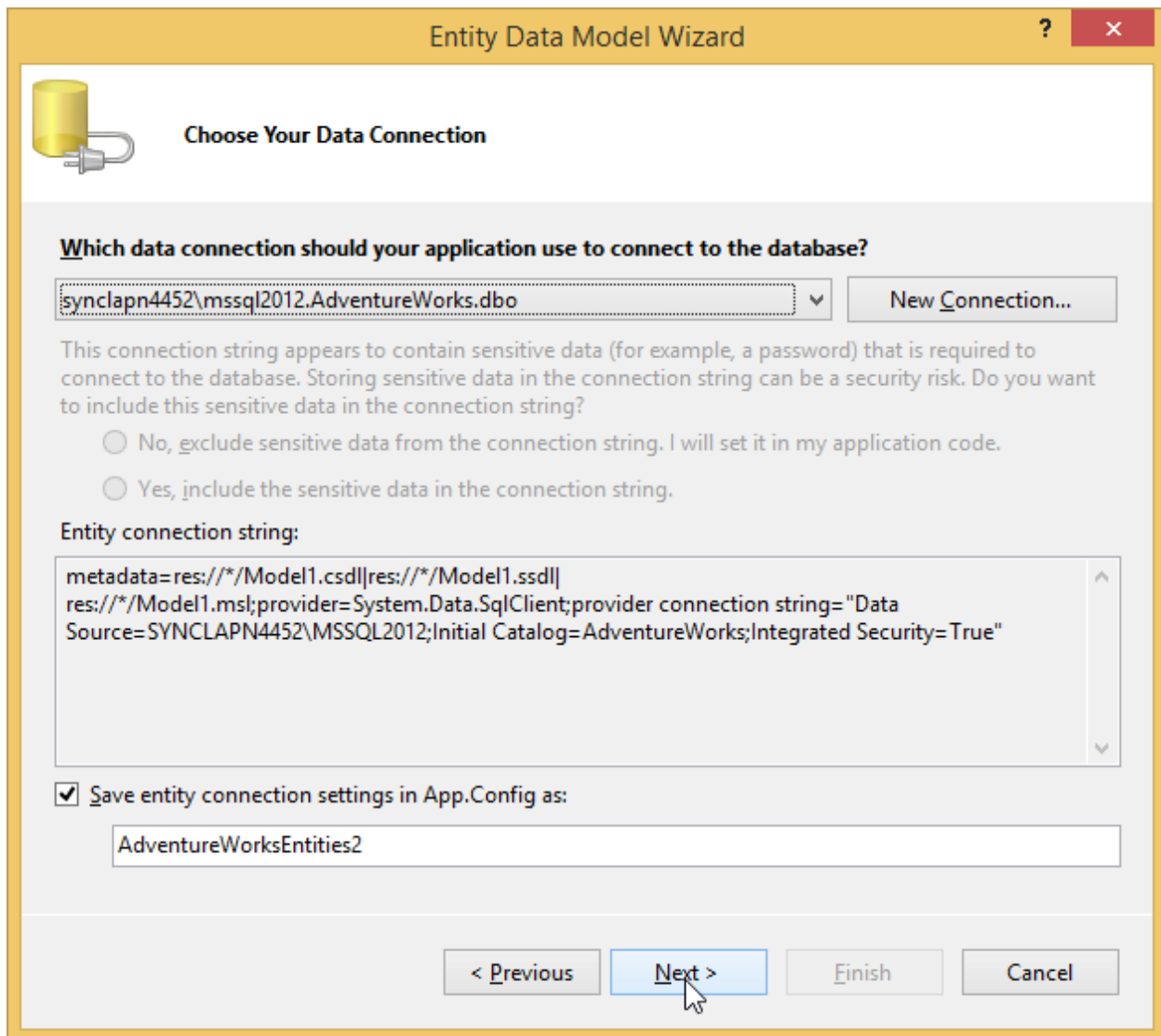


12. Then Entity Data Model Wizard opens.

13. Click Generate from database under what should the model contain? and click Next.



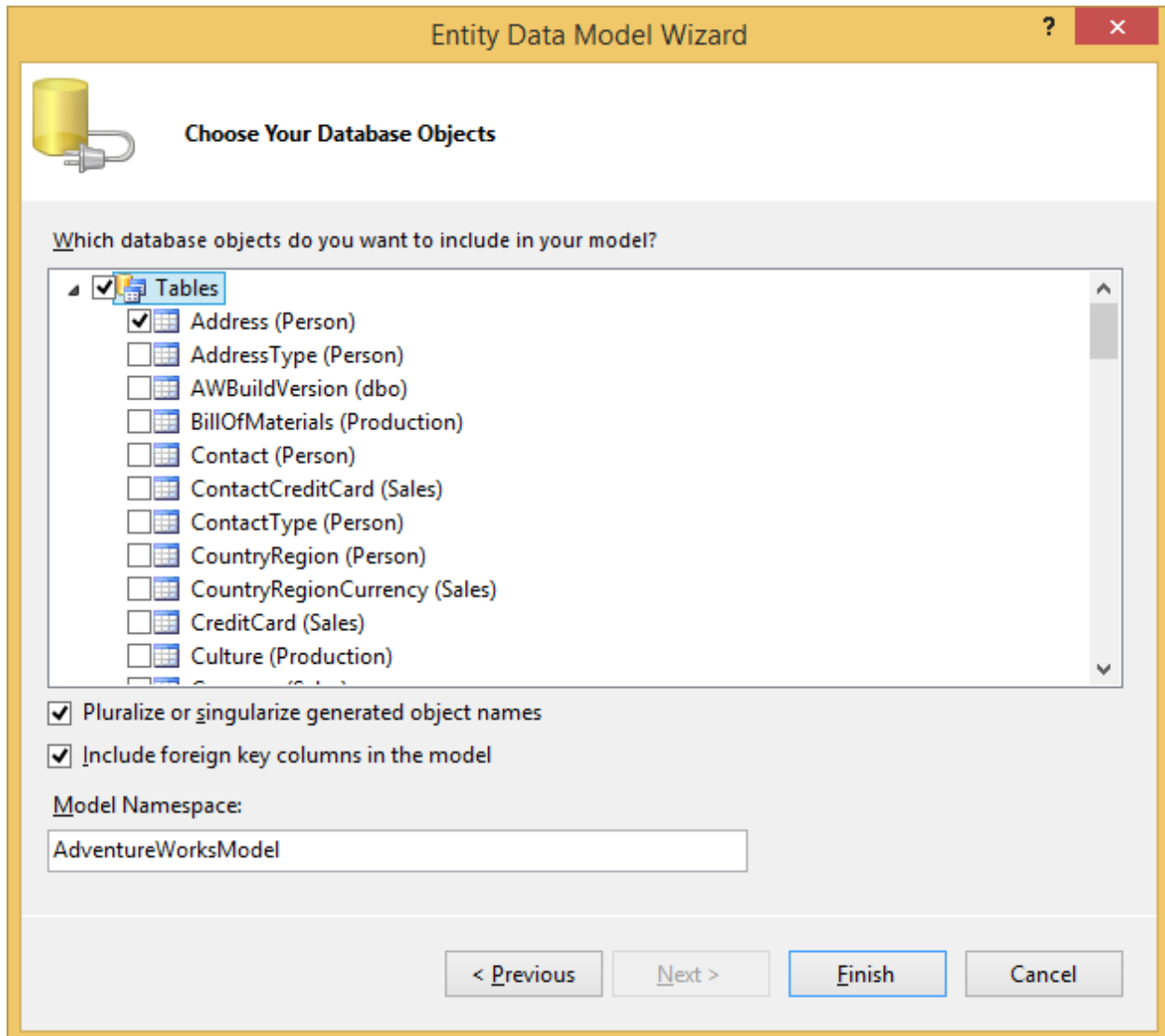
14. Select a data connection from Which data connection should your application use to connect to the database? drop-down combo box. Click Next.



The image shows a screenshot of the 'Entity Data Model Wizard' window, specifically the 'Choose Your Data Connection' step. The window has a yellow title bar with the text 'Entity Data Model Wizard' and standard window controls. Below the title bar is a yellow icon of a cylinder with a plug. The main content area is white and contains the following elements:

- Section Header:** 'Choose Your Data Connection' with a yellow icon.
- Question:** 'Which data connection should your application use to connect to the database?'
- Dropdown Menu:** A dropdown menu showing 'syncclapn4452\mssql2012.AdventureWorks.dbo' with a downward arrow.
- Button:** 'New Connection...'.
- Text:** 'This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?'
- Radio Buttons:** Two radio buttons: 'No, exclude sensitive data from the connection string. I will set it in my application code.' and 'Yes, include the sensitive data in the connection string.'.
- Text:** 'Entity connection string:'.
- Text Area:** A text area containing the connection string: 'metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;provider=System.Data.SqlClient;provider connection string="Data Source=SYNCLAPN4452\MSSQL2012;Initial Catalog=AdventureWorks;Integrated Security=True"'. The text area has a vertical scrollbar on the right.
- Checkbox:** A checked checkbox labeled 'Save entity connection settings in App.Config as:'.
- Text Field:** A text field containing 'AdventureWorksEntities2'.
- Buttons:** Four buttons at the bottom: '< Previous', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted in blue and has a mouse cursor pointing to it.

15. Select the required object and click Next. The Data Source Configuration Wizard opens.



The screenshot shows the 'Entity Data Model Wizard' window, specifically the 'Choose Your Database Objects' step. The window has a yellow title bar with the text 'Entity Data Model Wizard' and standard window controls. Below the title bar is a yellow icon of a database cylinder with a plug. The main area is titled 'Choose Your Database Objects' and contains the question 'Which database objects do you want to include in your model?'. A tree view on the left shows 'Tables' selected. The right pane lists various tables with checkboxes: 'Address (Person)' is checked, while others like 'AddressType (Person)', 'AWBuildVersion (dbo)', 'BillOfMaterials (Production)', 'Contact (Person)', 'ContactCreditCard (Sales)', 'ContactType (Person)', 'CountryRegion (Person)', 'CountryRegionCurrency (Sales)', 'CreditCard (Sales)', and 'Culture (Production)' are unchecked. Below the list are two checked options: 'Pluralize or singularize generated object names' and 'Include foreign key columns in the model'. A text box for 'Model Namespace:' contains 'AdventureWorksModel'. At the bottom are four buttons: '< Previous', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

Entity Data Model Wizard

Choose Your Database Objects

Which database objects do you want to include in your model?

☒ Tables

- ☒ Address (Person)
- ☐ AddressType (Person)
- ☐ AWBuildVersion (dbo)
- ☐ BillOfMaterials (Production)
- ☐ Contact (Person)
- ☐ ContactCreditCard (Sales)
- ☐ ContactType (Person)
- ☐ CountryRegion (Person)
- ☐ CountryRegionCurrency (Sales)
- ☐ CreditCard (Sales)
- ☐ Culture (Production)

☒ Pluralize or singularize generated object names

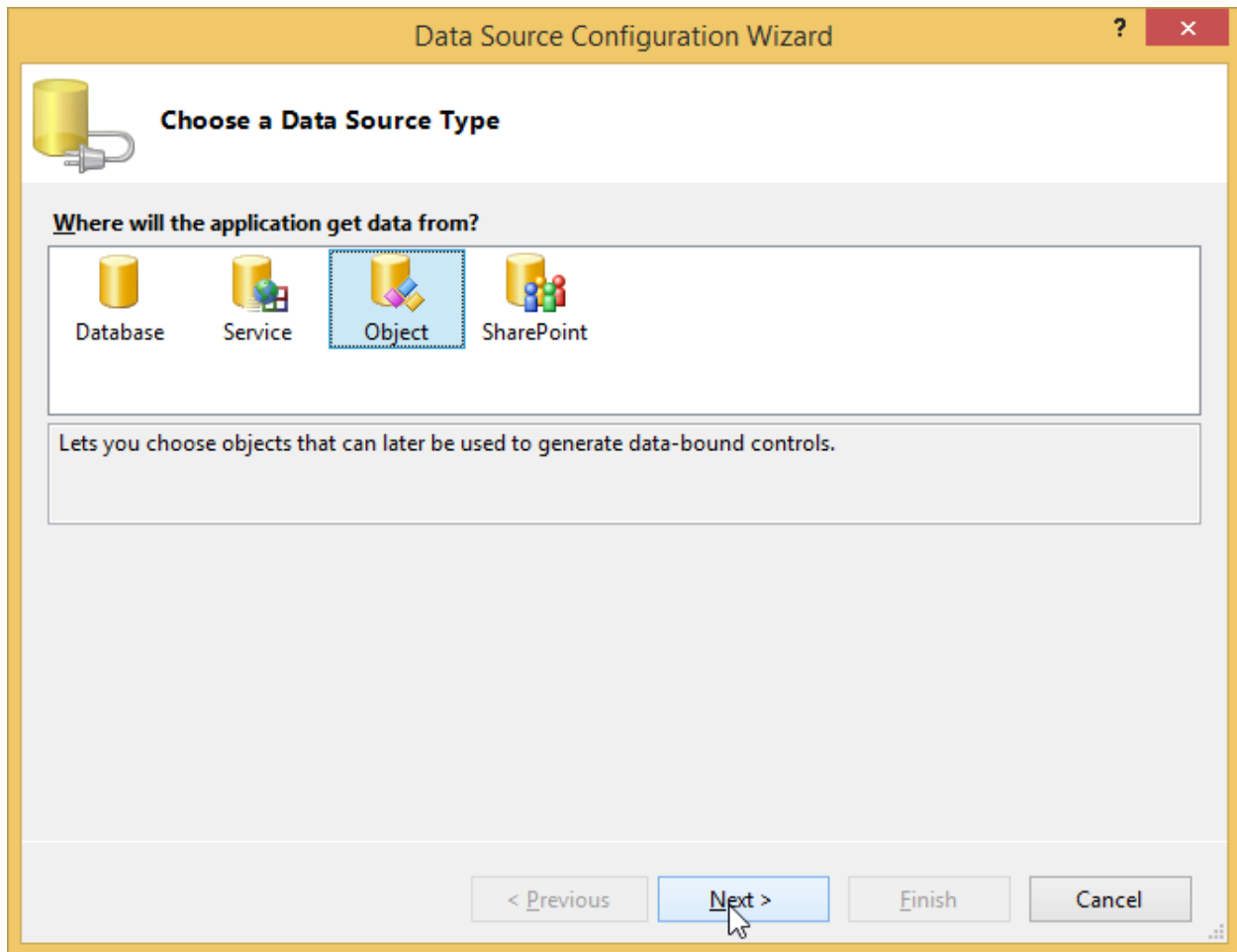
☒ Include foreign key columns in the model

Model Namespace:

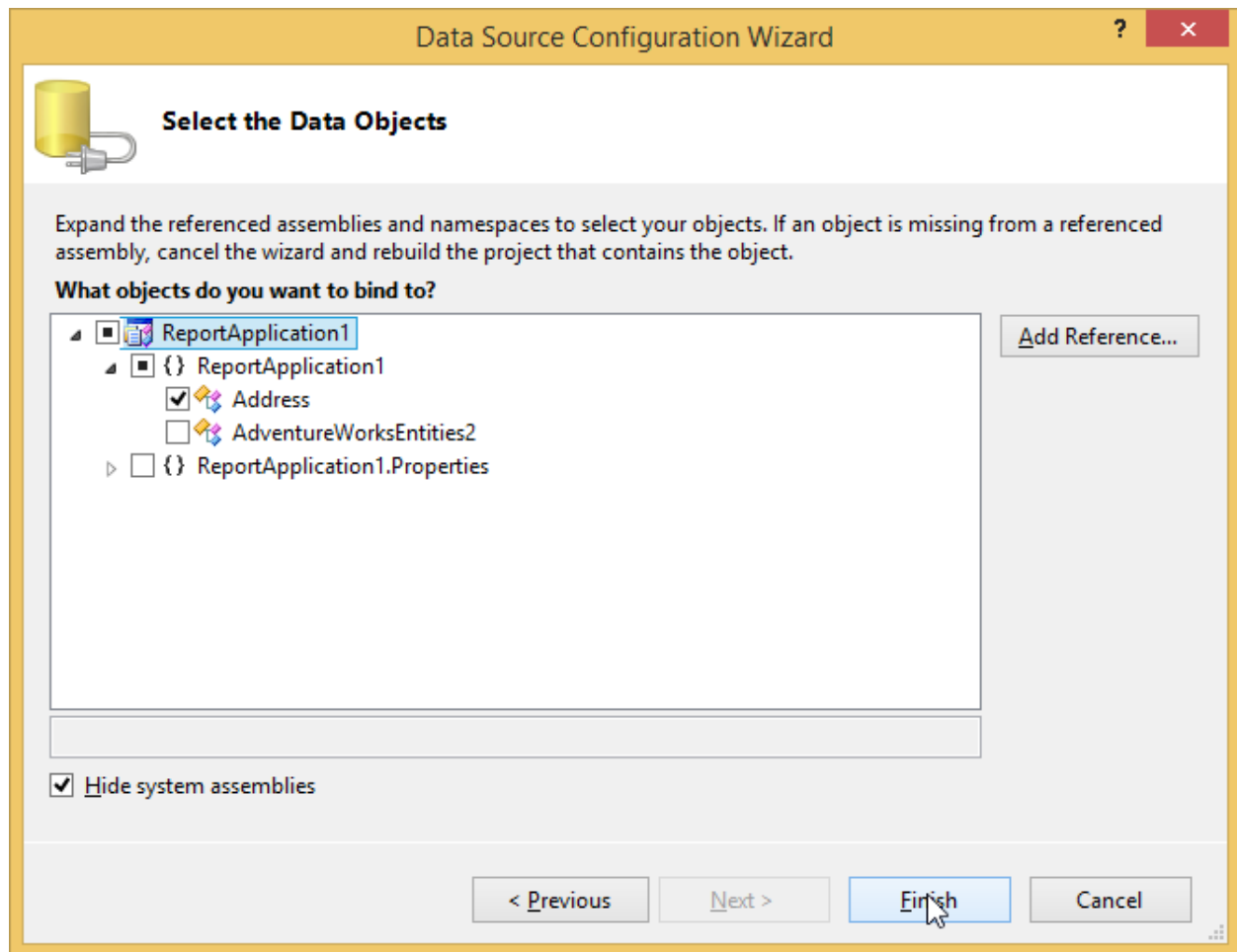
AdventureWorksModel

< Previous Next > Finish Cancel

16. Select Object under Where will the application get data from? and click Next.



17. Select the object under What objects do you want to bind to? and click Finish.



18. The Report Wizard shows the details of the dataset under Fields and Click Next.

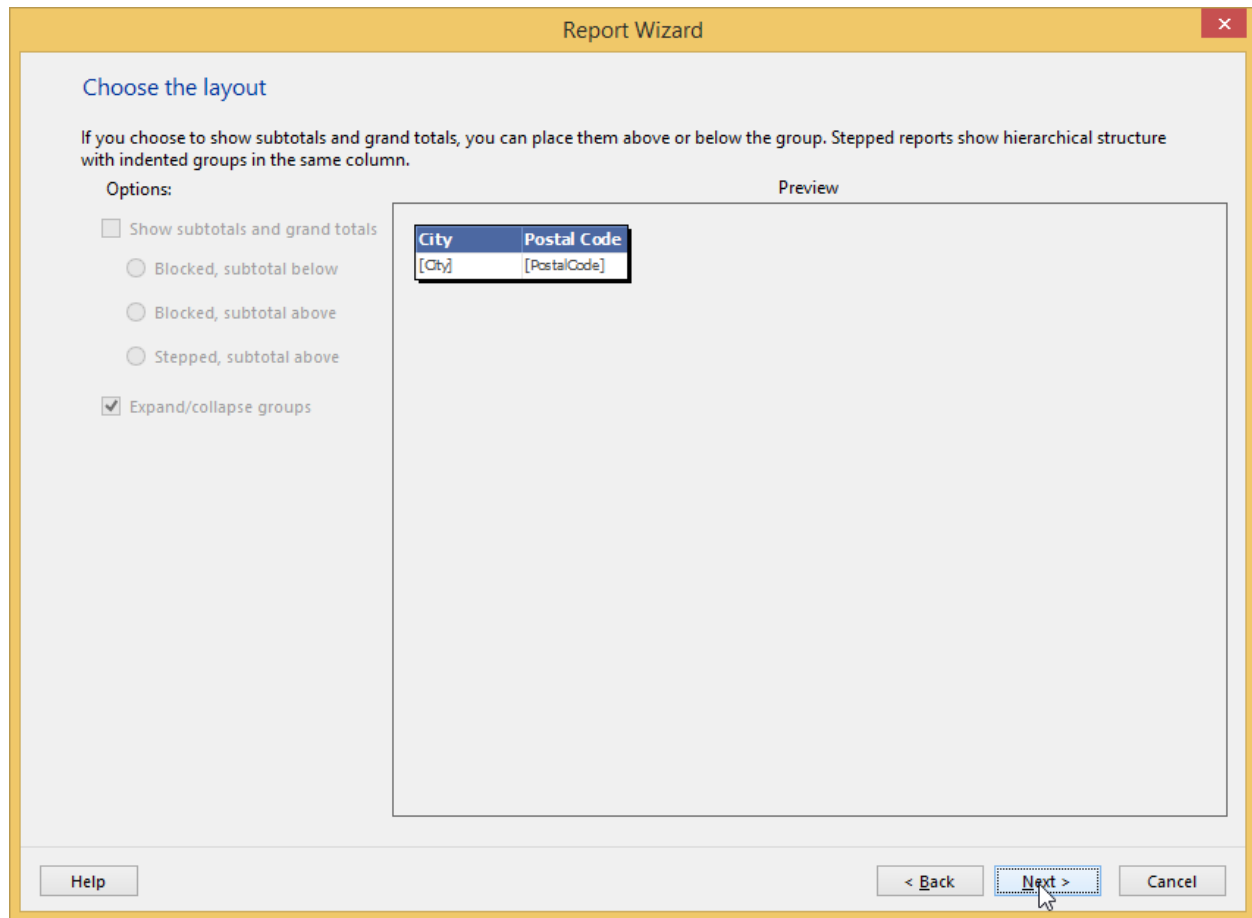
[illegible]

19. Arrange the available fields to row groups, column groups and values field. Click Next.

The screenshot shows the 'Report Wizard' window with the 'Arrange fields' step selected. The window has a yellow title bar and a red close button. The main area is light gray and contains the following elements:

- Available fields:** A list box on the left containing the following fields: AddressID, AddressLine1, AddressLine2, City, ModifiedDate, PostalCode, rowguid, and StateProvinceID.
- Row groups:** An empty box in the center with a grid icon and the label 'Row groups'.
- Column groups:** An empty box on the top right with a grid icon and the label 'Column groups'.
- Values:** A box on the bottom right with a summation symbol (Σ) and the label 'Values'. It contains two dropdown menus: 'City' and 'PostalCode'.
- Buttons:** At the bottom, there are three buttons: 'Help' (gray), '< Back' (blue with a dotted border), and 'Next >' (blue with a dotted border and a mouse cursor pointing to it). A 'Cancel' button is also present on the far right.

20. Choose the layout design in the next window and click Next.



The image shows a 'Report Wizard' dialog box with a yellow title bar. The main area is titled 'Choose the layout'. Below the title, there is a paragraph: 'If you choose to show subtotals and grand totals, you can place them above or below the group. Stepped reports show hierarchical structure with indented groups in the same column.' Under the heading 'Options:', there are four radio button options: 'Show subtotals and grand totals' (unchecked), 'Blocked, subtotal below' (unchecked), 'Blocked, subtotal above' (unchecked), and 'Stepped, subtotal above' (unchecked). There is also a checked checkbox for 'Expand/collapse groups'. To the right of the options is a 'Preview' section containing a table with two columns: 'City' and 'Postal Code'. The table has one data row with values '[City]' and '[PostalCode]'. At the bottom of the dialog, there are three buttons: 'Help', '< Back', and 'Next >', with a 'Cancel' button to the right of 'Next >'. A mouse cursor is pointing at the 'Next >' button.

Report Wizard

Choose the layout

If you choose to show subtotals and grand totals, you can place them above or below the group. Stepped reports show hierarchical structure with indented groups in the same column.

Options:

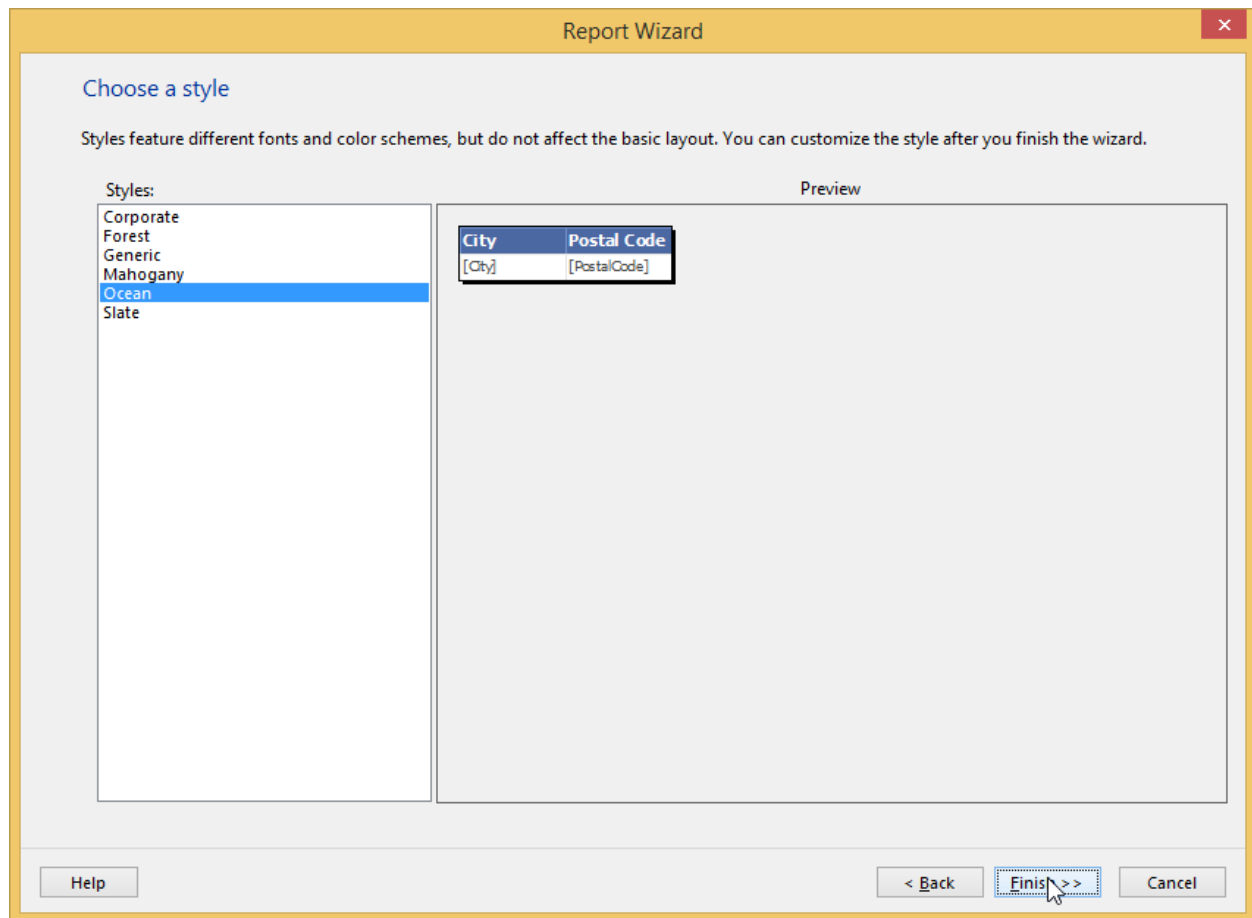
- ☐ Show subtotals and grand totals
- ☐ Blocked, subtotal below
- ☐ Blocked, subtotal above
- ☐ Stepped, subtotal above
- ☒ Expand/collapse groups

Preview

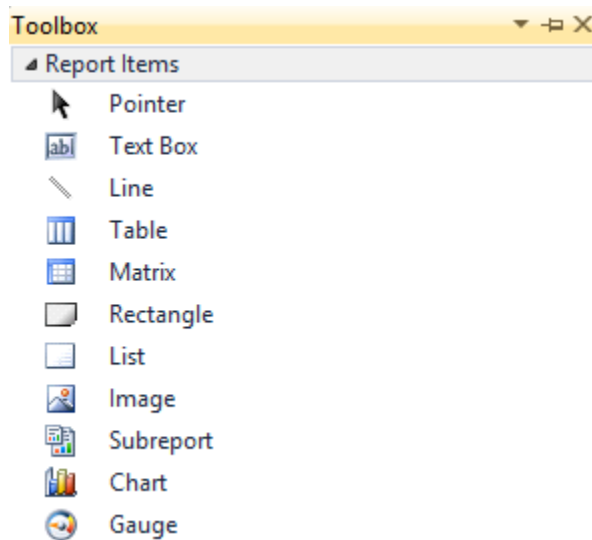
City	Postal Code
[City]	[PostalCode]

Help < Back Next > Cancel

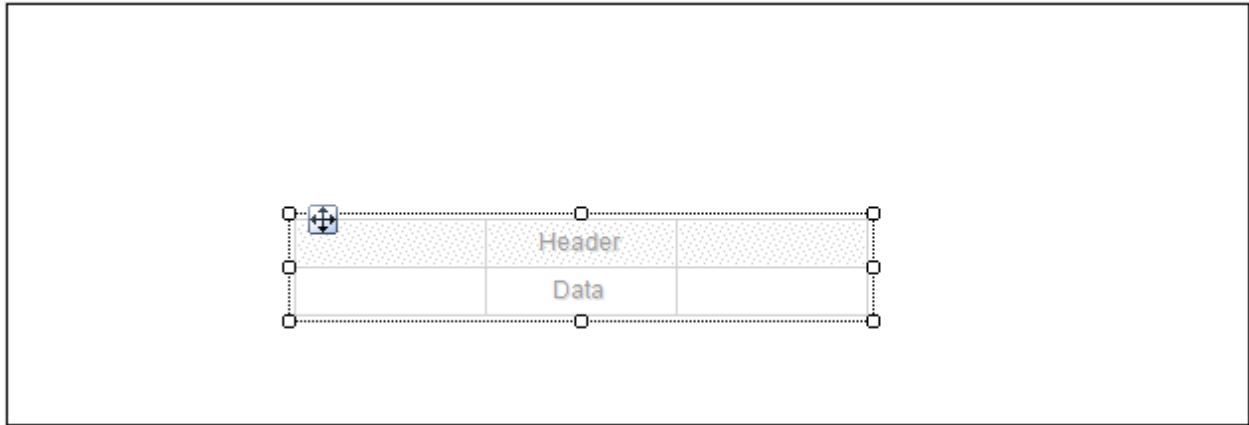
21. In the next window, choose the style and click Finish.



22. On Toolbox window, in Report Items, select Table.



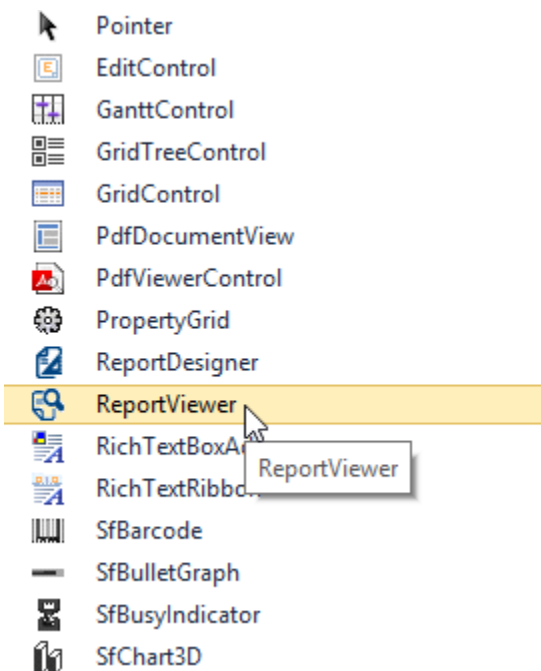
23. Draw a table on the WPF Designer window.



24. Drag the dataset field on the Table.

Address ID	Address Line1	City
[AddressID]	[AddressLine1]	[City]

25. To add Report Viewer in the WPF application, select ReportViewer under Reporting.



26. Set the ReportPath and the ProcessingMode as local in the Report Viewer.

XML

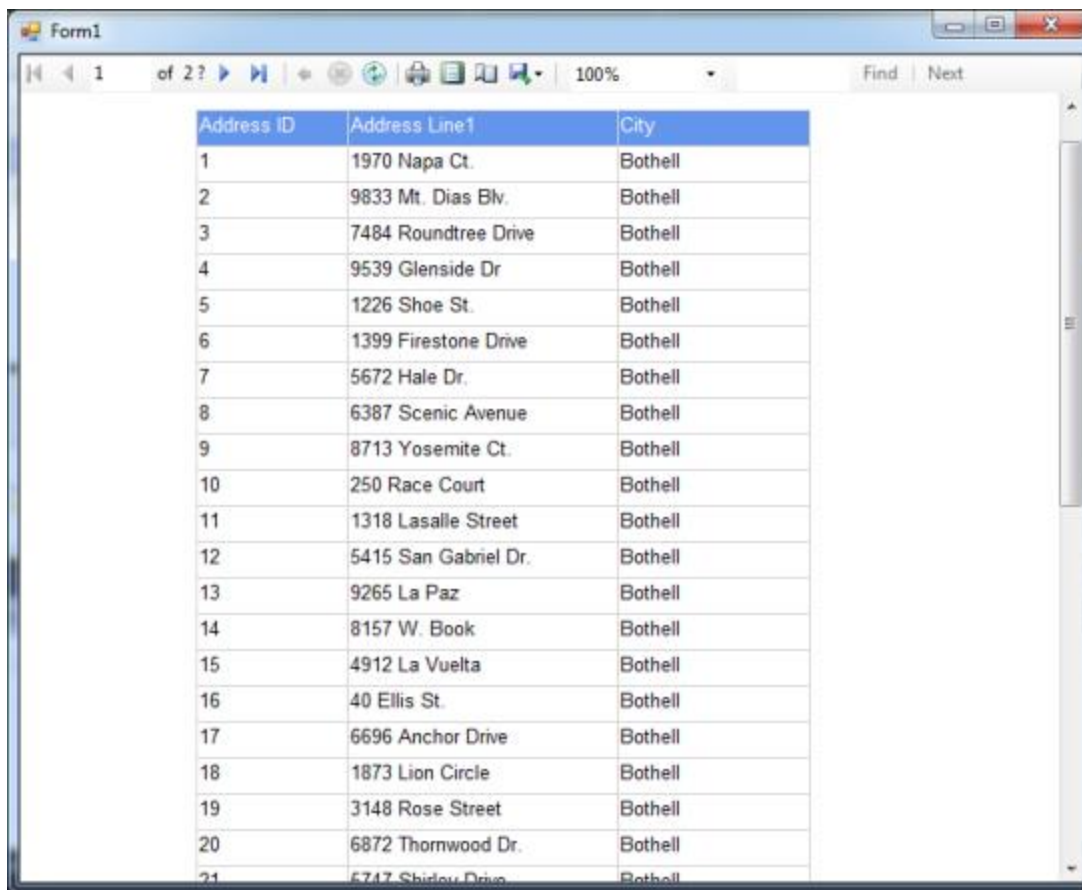
```
<Window x:Class="WpfApplication15.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf">
<Grid>
<syncfusion:ReportViewer Name="viewer" ProcessingMode="Local"
ReportPath="..\..\ProductCatalog.rdlc" />
</Grid>
</Window>
```

27. Set the DataSource information in the code to view the report in the Report Viewer.

CSHARP

```
public MainWindow()
{
InitializeComponent();
this.Loaded += new RoutedEventHandler(MainWindow_Loaded);
}
void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
this.viewer.DataSources.Clear();
this.viewer.DataSources.Add(new
Syncfusion.Windows.Reports.ReportDataSource()
{
Name = "Employee",
Value = new AdventureWorksEntities().Addresses.Take(100)
});
this.viewer.RefreshReport();
}
```

28. Run the application. The following output is displayed.



Address ID	Address Line1	City
1	1970 Napa Ct.	Bothell
2	9833 Mt. Dias Blv.	Bothell
3	7484 Roundtree Drive	Bothell
4	9539 Glenside Dr	Bothell
5	1226 Shoe St.	Bothell
6	1399 Firestone Drive	Bothell
7	5672 Hale Dr.	Bothell
8	6387 Scenic Avenue	Bothell
9	8713 Yosemite Ct.	Bothell
10	250 Race Court	Bothell
11	1318 Lasalle Street	Bothell
12	5415 San Gabriel Dr.	Bothell
13	9265 La Paz	Bothell
14	8157 W. Book	Bothell
15	4912 La Vuelta	Bothell
16	40 Ellis St.	Bothell
17	6696 Anchor Drive	Bothell
18	1873 Lion Circle	Bothell
19	3148 Rose Street	Bothell
20	6872 Thornwood Dr.	Bothell
21	5717 Shiloh Drive	Bothell

SSRS shared/embedded datasource credential information in ReportViewer

RDL reports have limitations for retrieving a password from the SQL Reporting Service (SSRS) for security reasons. Therefore, the credential information is provided when the data source credential is saved in the Reporting Server for shared or embedded data sources.

The following code provides SSRS shared/embedded data source credential information.

C#

```
void reportViewerControl_ReportLoaded(object sender, EventArgs e)
{
    var dataSources = this.reportViewerControl.GetDataSources();
    List<DataSourceCredentials> credentials=new List<DataSourceCredentials> ();
    foreach (var dataSource in dataSources)
    {
        DataSourceCredentials credential= new DataSourceCredentials ();
        credential.Name = dataSource.Name; // Sets the credential based on the data source.
        credential.UserId = "userName";
        credential.Password = "password";
        credentials.Add (credential);
    }
    this.reportViewerControl.SetDataSourceCredentials (credentials);
}
```

Provide the Report Parameters in code behind

Use the SetParameters method in the Report Viewer to provide the parameter values for rendering reports. Note that this method can also be used in a ReportLoaded event to avoid unexpected issues.

C#

```
this.reportViewerControl.ReportLoaded += new
Syncfusion.Windows.Reports.ReportLoadedEventHandler (reportViewerControl_Repo
rtLoaded);
void reportViewerControl_ReportLoaded(object sender, EventArgs e)
{
    //this.reportViewerControl.SetParameters
}
```

Provide the Data Source credential information in code behind

Use SetDataSourceCredentials to provide the data source credential for the Report Viewer. Use this method in a ReportLoaded event to avoid unexpected issues.

C#

```
this.reportViewerControl.ReportLoaded += new ReportLoadedEventHandler
(reportViewerControl_ReportLoaded);
void reportViewerControl_ReportLoaded(object sender, EventArgs e)
{
    var dataSources = this.reportViewerControl.GetDataSources();
    List<DataSourceCredentials> credentials=new List<DataSourceCredentials> ();
    foreach (var dataSource in dataSources)
    {
        DataSourceCredentials credential= new DataSourceCredentials ();
        credential.Name = dataSource.Name; // Sets the credential based on the data
        source.
        credential.UserId = "userName";
        credential.Password = "password";
        credentials.Add (credential);
    }
    this.reportViewerControl.SetDataSourceCredentials (credentials);
}
```

Can you use SharePoint Integrated Mode in ReportViewer?

Yes, Syncfusion Report Viewer supports viewing and exporting SharePoint Integrated Mode Reporting Service reports.

C#

```
this.reportViewer1.ReportLoaded += (sen, arg) =>
{
    IList < DataSourceCredentials > credentials = new List <
    DataSourceCredentials > ();
    foreach(var datasource in this.reportViewer1.GetDataSources())
    {
        DataSourceCredentials creden = new DataSourceCredentials();
        creden.Name = datasource.Name;
        creden.UserId = "username";
        creden.Password = "password";
        credentials.Add(creden);
    }
}
```

```
}  
this.reportViewer1.SetDataSourceCredentials(credentials);  
};  
this.reportViewer1.ReportPath = @" http://ServerName/testreport.rdl";  
this.reportViewer1.ReportServerUrl = @"http: //ServerName/ReportServer";  
this.reportViewer1.RefreshReport();
```

Can you use Azure SSRS reports in ReportViewer?

Yes, Syncfusion Report Viewer supports viewing and exporting Azure hosted reports. By default, Azure Reporting Service works with a Forms credential. So, provide your Forms credential in the ReportServerFormsCredential to view the report.

C#

```
this.reportViewer1.ReportLoaded += (sen, arg) =>  
{  
    IList < DataSourceCredentials > credentials = new List <  
        DataSourceCredentials > ();  
    foreach(var datasource in this.reportViewer1.GetDataSources())  
    {  
        DataSourceCredentials creden = new DataSourceCredentials();  
        creden.Name = datasource.Name;  
        creden.UserId = "username";  
        creden.Password = "password";  
        credentials.Add(creden);  
    }  
    this.reportViewer1.SetDataSourceCredentials(credentials);  
};  
this.reportViewer1.ReportPath = " / AzureReportProject / Reports";  
this.reportViewer1.ReportServerUrl = @"http://ServerName/ReportServer";  
this.reportViewer1.ReportServerFormsCredential = new  
    System.Net.NetworkCredential("userID", "Password");  
this.reportViewer1.RefreshReport();
```

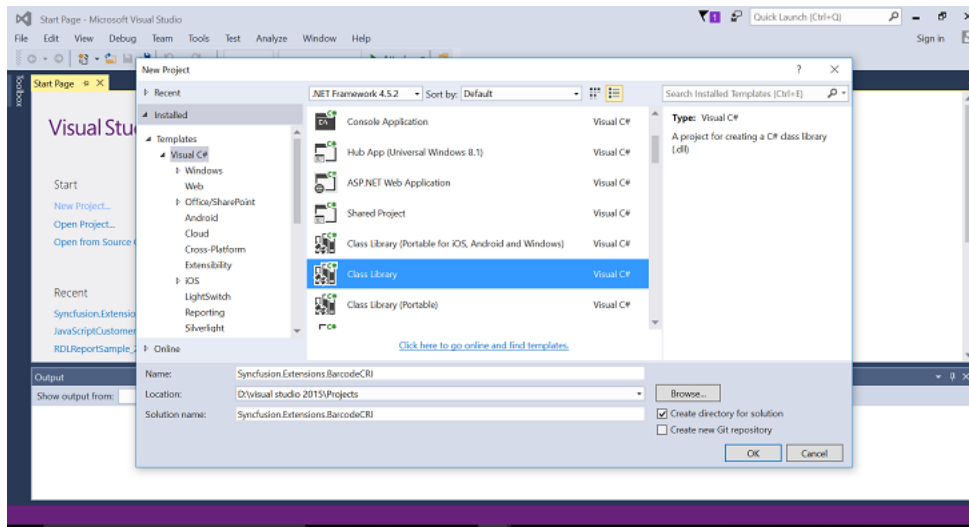
A custom report item allows you to add the functionality that is not natively supported in the RDL or extend the functionality of existing controls in the RDL standard. The run-time component allows to render the custom report item in report viewer.

Creating a custom report item run-time component

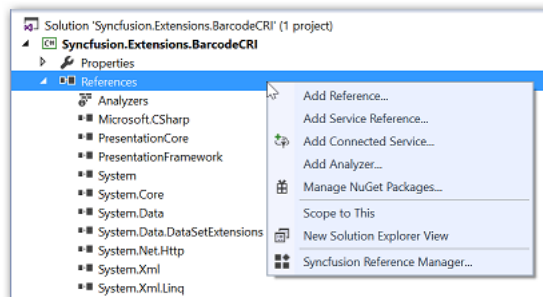
The run-time component of the custom report item is implemented by using CLS-compliant language, and is called by the report processor at run-time. The below section provides detail to create run-time component with the barcode custom report item in report viewer.

Create report item assembly

1. Open the Visual Studio and select the class library project type, then name the project as "Syncfusion.Extensions.BarcodeCRI" for the run-time component.



2. Add the reference "Syncfusion.ReportControls.Wpf", "Syncfusion.SfBarcode.WPF" and "Syncfusion.Shared.WPF" for the extension project.



Note: Refer the above assemblies from the below installed location. For Essential Studio: C:\Program Files (x86)\Syncfusion\Essential Studio\{ site.releaseversion }\Assemblies

3. Add a class "BarcodeCustomReportItem" by inheriting the `ICustomReportItem` interface.

Implementing the `ICustomReportItem` interface

To create a CustomReportItem run-time component, you should implement the `ICustomReportItem` interface, it generates the following two method stubs.

Interface methods	Definition
GenerateReportItemDefinition	Called first and is used for setting definition properties and creating the image object that contains both the definition and instance properties that are used for rendering the item.
EvaluateReportItemInstance	Called after the definition objects have been evaluated, and it provides the instance objects that will be used for rendering the item.

C#

```
namespace Syncfusion.Extensions.BarcodeCRI
{
```

```

public class BarcodeCustomReportItem : RDL.Data.ICustomReportItem
{
    #region ICustomReportItem Members
    public void GenerateReportItemDefinition(CustomReportItem cri)
    {
        //It will create the Image object
        cri.CreateCriImageDefinition();
    }
    public void EvaluateReportItemInstance(CustomReportItem cri)
    {
        Thread thread = new Thread(delegate ()
        {
            RDL.Data.Image imageReportItem = (RDL.Data.Image)cri.GeneratedReportItem;
            imageReportItem.ImageData = DrawImage(cri);
        }, (1024 * 1024 * 64));
        thread.SetApartmentState(ApartmentState.STA);
        thread.Start();
        thread.Join();
    }
    #endregion
    //To create image from custom report item control
    private byte[] DrawImage(CustomReportItem customReportItem)
    {
        try
        {
            byte[] imageData = null;
            SfBarcode barcodeControl = new SfBarcode();
            barcodeControl.Background = new SolidColorBrush(Colors.Transparent);
            barcodeControl.Height = customReportItem.Height.ToPixels();
            barcodeControl.Width = customReportItem.Width.ToPixels();
            barcodeControl.Text =
                (string)LookupCustomProperty(customReportItem.CustomProperties,
                "BarcodeValue");
            barcodeControl.InvalidateArrange();
            barcodeControl.UpdateLayout();
            MemoryStream stream = new ImageConversion().CovertToImage(barcodeControl);
            imageData = new byte[(int)stream.Length];
            stream.Seek(0, SeekOrigin.Begin);
            stream.Read(imageData, 0, (int)stream.Length);
            return imageData;
        }
        catch
        {
            return null;
        }
    }
}

```

Convert custom report item as image

The custom report item is rendered as image in report viewer, so that the run-time component need to be converted as an image. The following converter is used to generate the image for rendering.

C#

```

internal partial class ImageConversion : UserControl

```

```

{
    Canvas InnerCanvas;
    public MemoryStream CovertToImage(Control innerControl)
    {
        try
        {
            this.InnerCanvas = new Canvas();
            this.Content = this.InnerCanvas;
            innerControl.Margin = new Thickness(0);
            InnerCanvas.Children.Add(innerControl);
            InnerCanvas.Width = innerControl.Width;
            InnerCanvas.Height = innerControl.Height;
            Canvas canvas = this.InnerCanvas;
            canvas.Measure(new Size((int)canvas.Width, (int)canvas.Height));
            canvas.Arrange(new Rect(new Size((int)canvas.Width, (int)canvas.Height)));
            int Height = ((int)(InnerCanvas.ActualHeight));
            int Width = ((int)(InnerCanvas.ActualWidth));
            this.Height = InnerCanvas.Height;
            this.Width = InnerCanvas.Width;
            InnerCanvas.LayoutTransform = null;
            Size size = new Size(InnerCanvas.ActualWidth, InnerCanvas.ActualHeight);
            InnerCanvas.Background = Brushes.White;
            InnerCanvas.Arrange(new Rect(size));
            InnerCanvas.UpdateLayout();
            RenderTargetBitmap rtb = new RenderTargetBitmap(Width, Height, 300, 300,
                PixelFormats.Default);
            rtb.Render(InnerCanvas);
            var Source = new MemoryStream();
            BitmapEncoder encoder = new BmpBitmapEncoder();
            encoder.Frames.Add(BitmapFrame.Create(rtb));
            encoder.Save(Source);
            return Source;
        }
        catch
        {
            return null;
        }
    }
}

```

Build project

You can clean and build the extension project, it will generate the run-time component assembly "Syncfusion.Extensions.BarcodeCRI.dll" in the bin folder of the project.

Note: You can create a standalone report viewer application with the help of given [Getting Started Documentation](#).

You can download the extension project with barcode custom report item for report viewer from [here](#).

Deploy a custom report item

To deploy a custom report item, you must modify the application configuration files or create "ReportExtensions.config" file and copy the run-time component assembly (Syncfusion.Extensions.BarcodeCRI) and its dependent assemblies to the bin folder of your application. The deployment requires configuration to process the extensions, and the following describes about configuration settings.

Create a "ReportExtensions.config" file in your application. The following "configSections" section is mandatory to process the extension in the control, so add it as shown in the following code.

XML

```
<configSections>
<section name="ReportingExtensions" type="Syncfusion.Reporting.Extensions,
Syncfusion.ReportControls.WPF" allowLocation="true"
allowDefinition="Everywhere" />
</configSections>
```

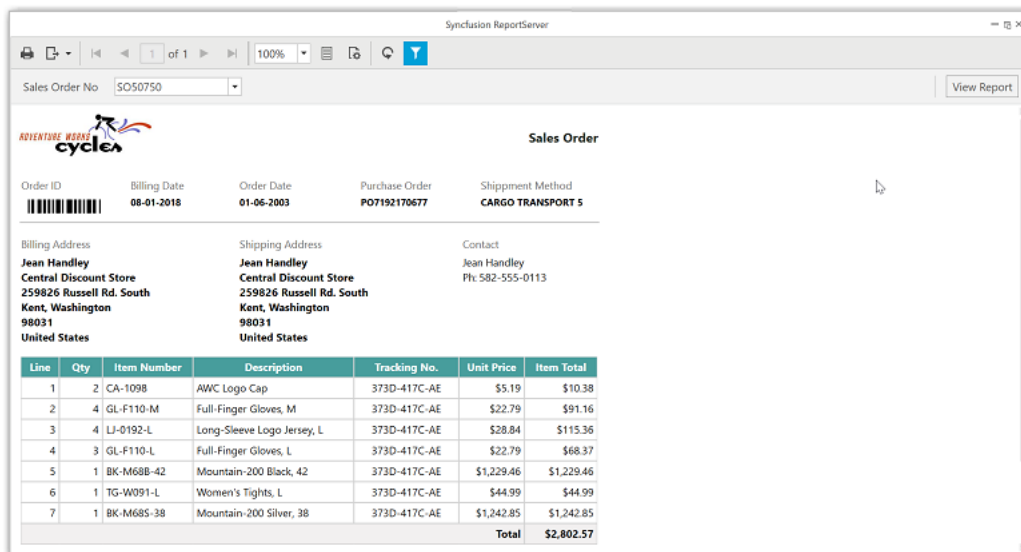
You must add the **ReportItem** tag for all newly added custom report item with the following attributes.

Attribute	Description
Name	Name of your report item that is going to be displayed in the list.
Assembly	Name of the newly created report item assembly.
Type	Report item class name with the namespace.

XML

```
<ReportingExtensions>
<ReportItems>
<ReportItem Name="Barcode" Assembly="Syncfusion.Extensions.BarcodeCRI"
Type="Syncfusion.Extensions.BarcodeCRI.BarcodeCustomReportItem" />
</ReportItems>
</ReportingExtensions>
```

After creating the config file, add it to the report viewer application. Run the application, output with the barcode custom report item is rendered as below.



Shows the invoice report rendered with the barcode custom report item.

ReportWriter

WPF ReportWriter Overview

Report Writer is a class library that enables the user to render reports defined in Microsoft's RDL format (2008 or 2008 R2) as PDF, Word, Excel or HTML documents.

The important features of WPF Report Writer are listed as follows:

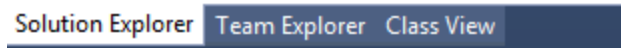
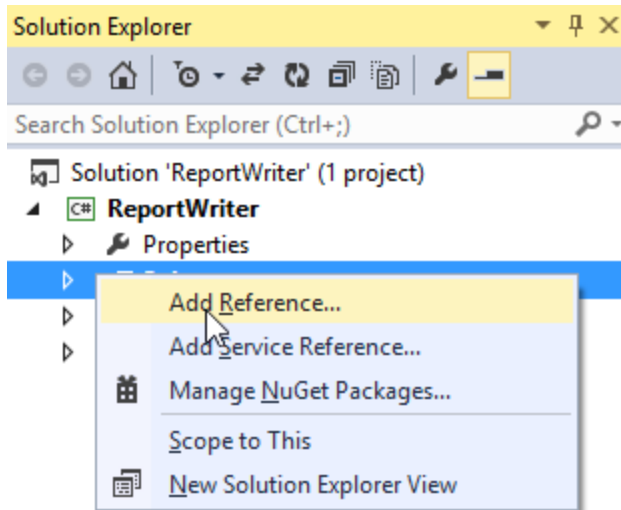
- RDL Specification - Supports RDL specification for the SQL Server 2008 and RDL specification for the SQL Server 2008 R2 only. List of available report definition formats: [msdn.microsoft.com/library/dd297486\(SQL.100\)](https://msdn.microsoft.com/library/dd297486(SQL.100)).
- Data sources - You can use advanced database servers Data Sources in Report Writer (SQL and Oracle).
- Charts - Show all basic types of charts that are available in Microsoft RDL reports.
- Tablix - Shows the summaries and simple tables.
- Gauge - Shows measurement values by using expression values.
- Textbox - Shows textbox data with expression support.
- Export - Export report as PDF, Word, Excel and HTML.
- Report Parameter - Views the report based on the report parameter value.

Getting Started with WPF ReportWriter

Adding ReportWriter to an application

This section illustrates how to add ReportWriter to the WPF application. It includes the following steps.

1. Create a new WPF application in Visual Studio.
2. In the Solution Explorer, Right-click the References folder and then click Add Reference.



3. Add the following references
 - Syncfusion.Chart.Wpf
 - Syncfusion.Compression.Base
 - Syncfusion.DocIO.Base
 - Syncfusion.Gauge.Wpf
 - Syncfusion.Linq.Base
 - Syncfusion.Pdf.Base
 - Syncfusion.ReportControls.Wpf
 - Syncfusion.ReportWriter.Base
 - Syncfusion.SfMaps.Wpf
 - Syncfusion.Shared.Wpf
 - Syncfusion.XlsIO.Base

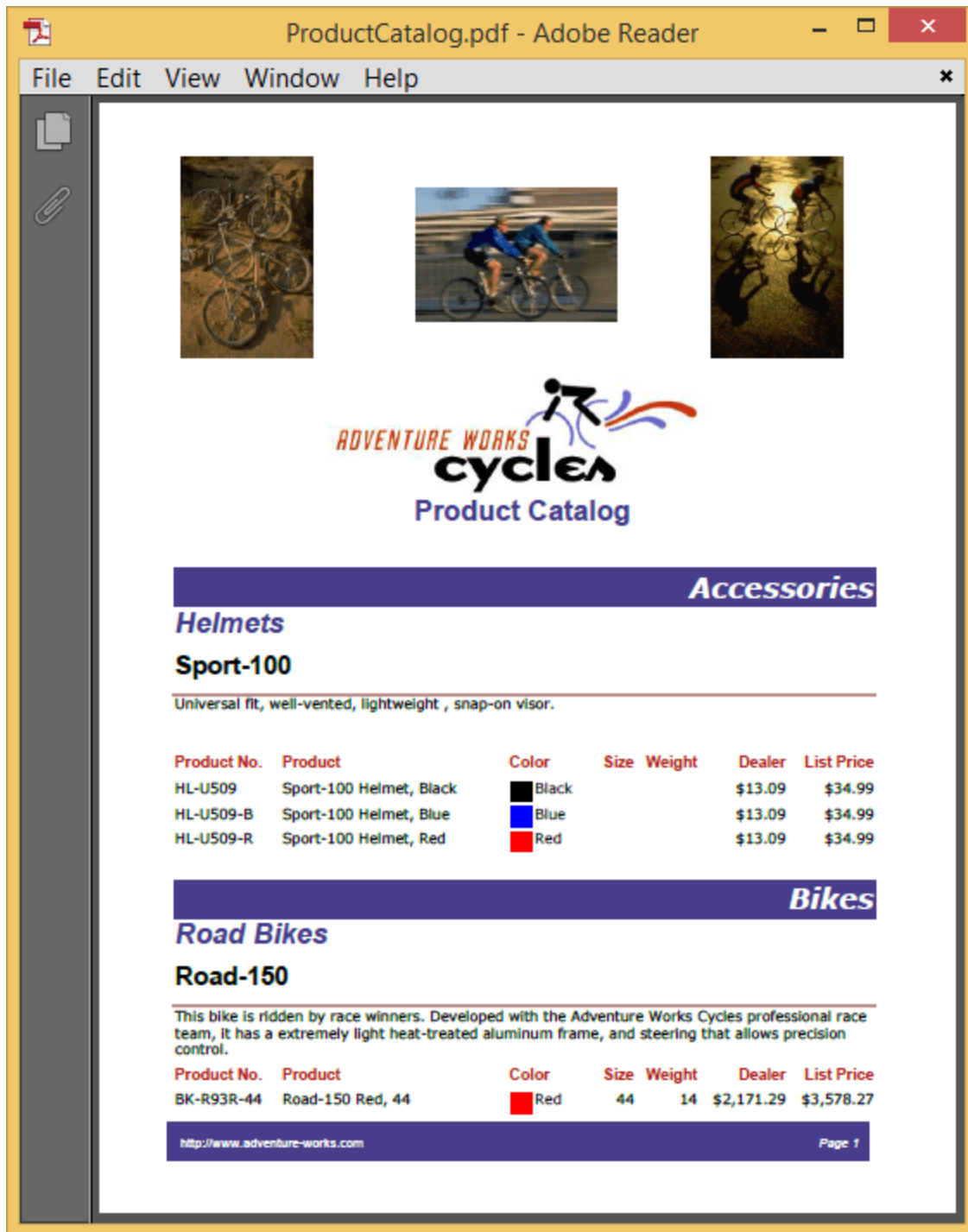
Note: Refer the above assemblies from the installed location, C:\Program Files (x86)\Syncfusion\Essential Studio\WPF\{{ site.releaseversion }}\Assemblies

4. Set the following properties of ReportWriter.
 - ReportPath - Set the local file system path of the report.
 - ReportProcessingMode - Set ProcessingMode as Remote for RDL report and Local for RDLC report.
 - Export format - Set Export format as PDF, Excel, Word or HTML.
5. Add the following code to export RDL report to the required export formats.

CSHARP

```
string fileName = null;
WriterFormat format;
string reportPath = @"..\ReportTemplate\Product Catalog.rdl";
//Step 1 : Instantiate the report writer with the parameter "ReportPath".
ReportWriter reportWriter = new ReportWriter(reportPath);
reportWriter.ReportProcessingMode = ProcessingMode.Remote;
//Step 2 : Save the report as Pdf or Word or Excel
if (pdf.IsChecked == true)
{
    fileName = "ProductCatalog.pdf";
    format = WriterFormat.PDF;
}
else if (word.IsChecked == true)
{
    fileName = "ProductCatalog.doc";
    format = WriterFormat.Word;
}
else if (excel.IsChecked == true)
{
    fileName = "ProductCatalog.xls";
    format = WriterFormat.Excel;
}
else
{
    fileName = "ProductCatalog.html";
    format = WriterFormat.HTML;
}
reportWriter.Save(fileName, format);
```

5. Run the application. The following output displays exported report in PDF format.



Export RDL Reports

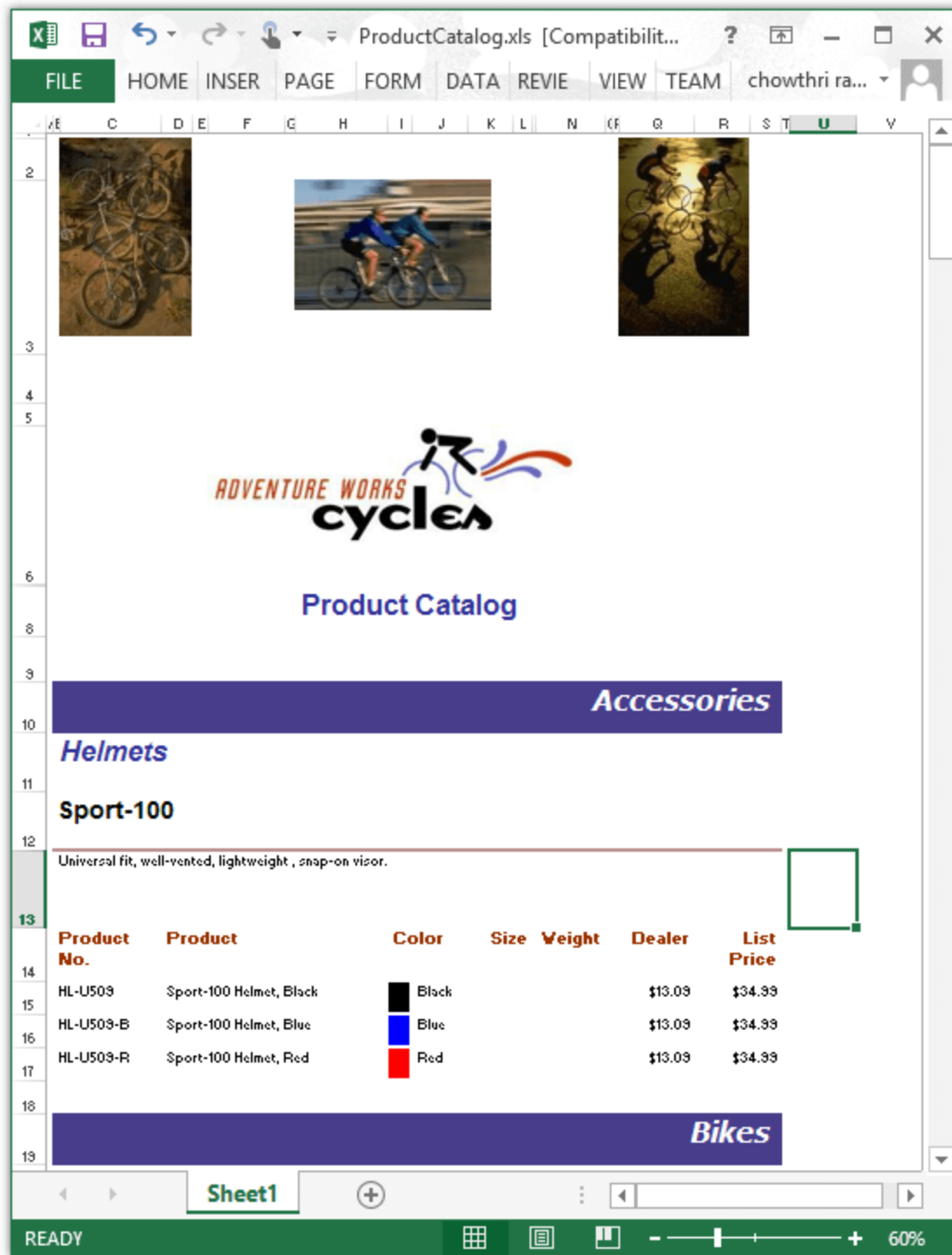
The ReportWriter allows you to export the RDL report to popular file formats PDF, WORD, EXCEL and HTML. The following code helps you to bind data to ReportWriter.

1. Assign `ReportPath`, `ReportProcessingMode` and `ExportFormat` to ReportWriter.

CSHARP

```
string reportPath = @"..\ReportTemplate\Product Catalog.rdl";  
ReportWriter reportWriter = new ReportWriter(reportPath);  
reportWriter.ReportProcessingMode = ProcessingMode.Remote;  
reportWriter.Save("ProductCatalog.xls", WriterFormat.Excel);
```

2. Run the application. The following output displays exported report in Excel format.



Export RDLC Reports

The ReportWriter allows you to export the RDLC report to popular file formats PDF, WORD, EXCEL and HTML. The following code helps you to bind data to ReportWriter.

1. Assign `ReportPath`, `ReportProcessingMode` and `ExportFormat` to ReportWriter.

CSHARP

```
string reportPath = @"..\ReportTemplate\RDLC\ProductCatalog.rdlc";
ReportWriter reportWriter = new ReportWriter(reportPath, dataSources);
reportWriter.ReportProcessingMode = ProcessingMode.Local;
reportWriter.Save("ProductCatalog.doc", WriterFormat.WORD);
```

2. Add data source to the RDLC report.

CSHARP

```
ReportDataSourceCollection dataSources = new ReportDataSourceCollection();
dataSources.Add(new ReportDataSource { Name = "ProductCatalog", Value =
ProductCatalogSource.GetData() });
```

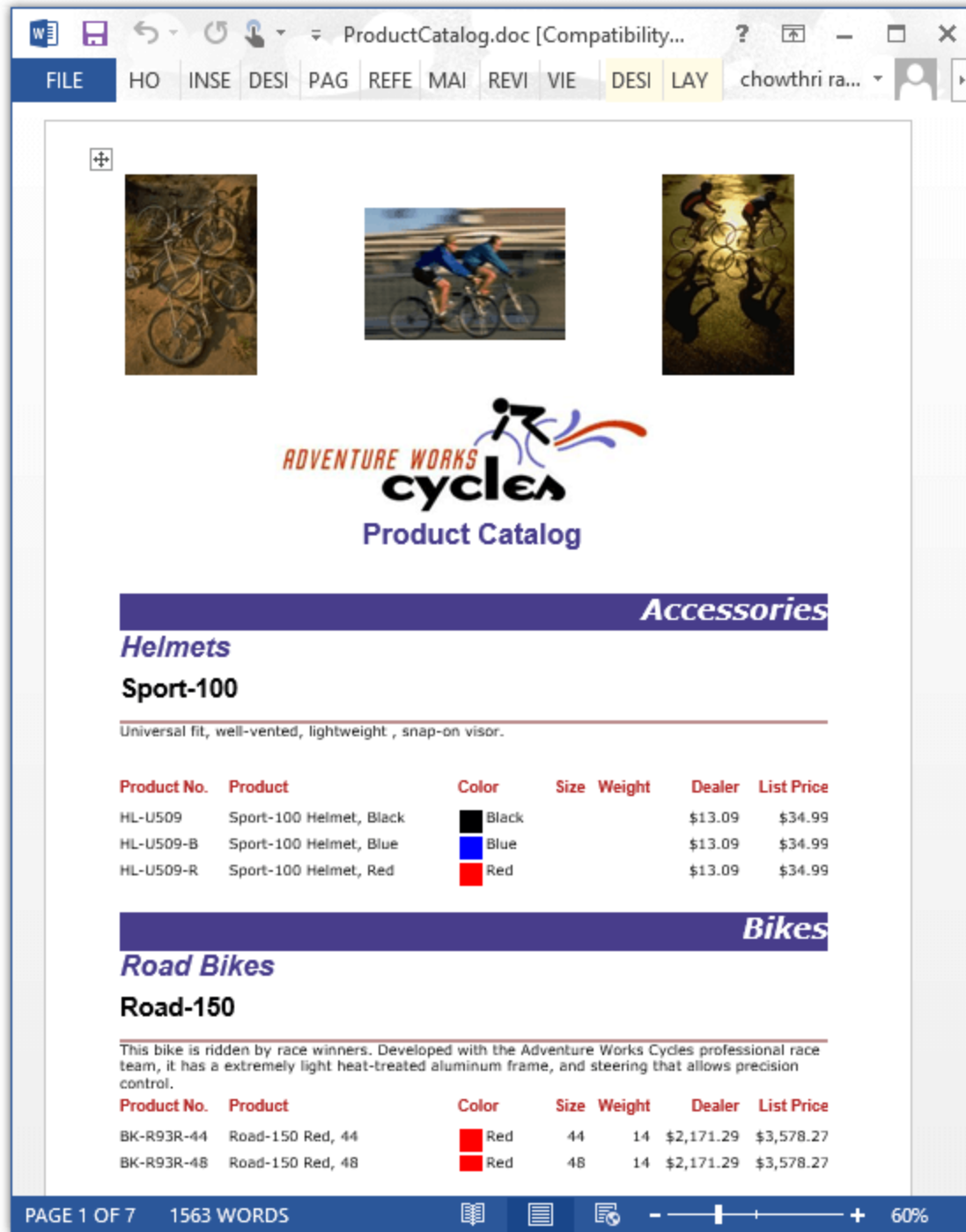
3. Assign values for the data source which is given in the RDLC.

CSHARP

```
#region ProductCatalog Details
public class ProductCatalogSource
{
    public string ProdSubCat { get; set; }
    public string ProdModel { get; set; }
    public string ProdCat { get; set; }
    public string Description { get; set; }
    public string ProdName { get; set; }
    public string ProductNumber { get; set; }
    public string Color { get; set; }
    public string Size { get; set; }
    public double? Weight { get; set; }
    public double? StandardCost { get; set; }
    public string Style { get; set; }
    public string Class { get; set; }
    public double? ListPrice { get; set; }
    public static IList GetData()
    {
        List<ProductCatalogSource> datas = new List<ProductCatalogSource>();
        ProductCatalogSource data = null;
        data = new ProductCatalogSource()
        {
            ProdSubCat = "Road Frames",
            ProdModel = "HL Road Frame",
            ProdCat = "Components",
            Description = "Our lightest and best quality aluminum frame made from the
newest alloy; it is welded and heat-treated for strength. Our innovative
design results in maximum comfort and performance.",
            ProdName = "HL Road Frame - Black, 58",
            ProductNumber = "FR-R92B-58",
            Color = "Black",
            Size = "58",
            Weight = 2.24,
            StandardCost = 1059.3100,
```

```
Style = "U ",
Class = "H ",
ListPrice = 1431.5000
};
datas.Add(data);
data = new ProductCatalogSource()
{
    ProdSubCat = "Road Frames",
    ProdModel = "HL Road Frame",
    ProdCat = "Components",
    Description = "Our lightest and best quality aluminum frame made from the
newest alloy; it is welded and heat-treated for strength. Our innovative
design results in maximum comfort and performance.",
    ProdName = "HL Road Frame - Red, 58",
    ProductNumber = "FR-R92R-58",
    Color = "Red",
    Size = "58",
    Weight = 2.24,
    StandardCost = 1059.3100,
    Style = "U ",
    Class = "H ",
    ListPrice = 1431.5000
};
datas.Add(data);
data = new ProductCatalogSource()
{
    ProdSubCat = "Helmets",
    ProdModel = "Sport-100",
    ProdCat = "Accessories",
    Description = "Universal fit, well-vented, lightweight , snap-on visor.",
    ProdName = "Sport-100 Helmet, Red",
    ProductNumber = "HL-U509-R",
    Color = "Red",
    Size = "",
    Weight = null,
    StandardCost = 13.0863,
    Style = "",
    Class = "",
    ListPrice = 34.9900
};
datas.Add(data);
return datas;
}
}
#endregion
```

4. Run the application. The following output displays exported report in Word format.



Saving reports in WPF ReportWriter Control

Essential ReportWriter provides support for saving a report as a PDF, Word, Excel and HTML documents with the help of the class ReportWriter. The report elements such as Tablix, matrices, charts, gauges, shapes, and text boxes are supported in this feature.

Saving report as PDF

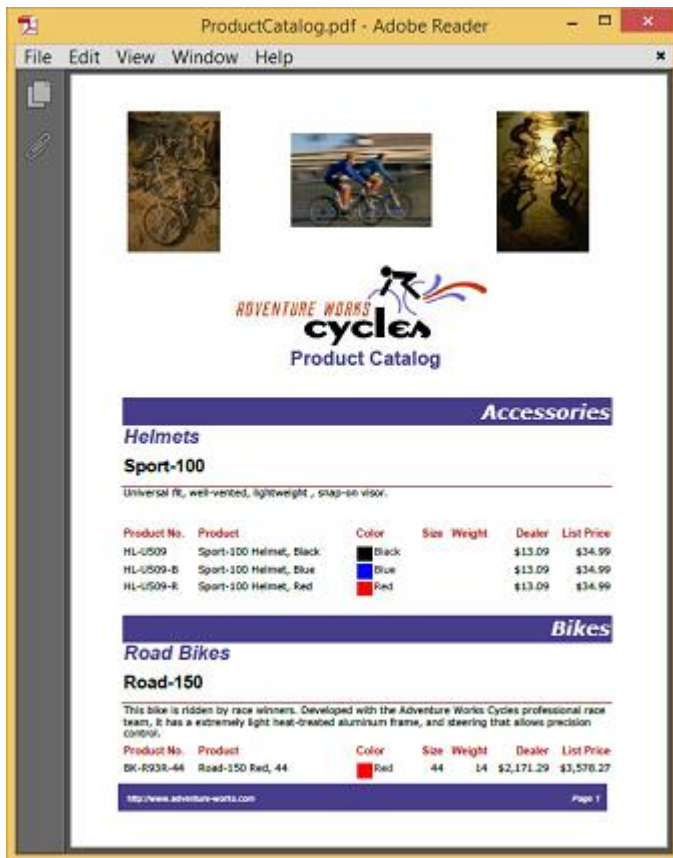
The report generated using the ReportDesigner can be exported as a PDF document using the following code.

C#

```
//Instantiate the report writer with the parameter "ReportPath" and
"ReportDataSourceCollection".
ReportWriter reportWriter = new ReportWriter(reportpath, dataSources);
reportWriter.Save("Sample.pdf", WriterFormat.PDF);
```

VB.NET

```
'Instantiate the report writer with the parameter "ReportPath" and
"ReportDataSourceCollection".
Dim reportWriter As New ReportWriter (reportpath, dataSources)
reportWriter.Save("Sample.pdf", WriterFormat.PDF)
```



Saving report as Excel

The report generated using the ReportDesigner can be exported as an Excel document using the following code example.

C#

```
//Instantiate the report writer with the parameter "ReportPath" and
"ReportDataSourceCollection".
ReportWriter reportWriter = new ReportWriter(reportPath, dataSources);
reportWriter.Save("Sample.xls", WriterFormat.Excel);
```

VB.NET

'Instantiate the report writer with the parameter "ReportPath" and "ReportDataSourceCollection".

```
Dim reportWriter As New ReportWriter (reportPath, dataSources)
reportWriter.Save("Sample.xls", WriterFormat.Excel)
```



Saving report as Word

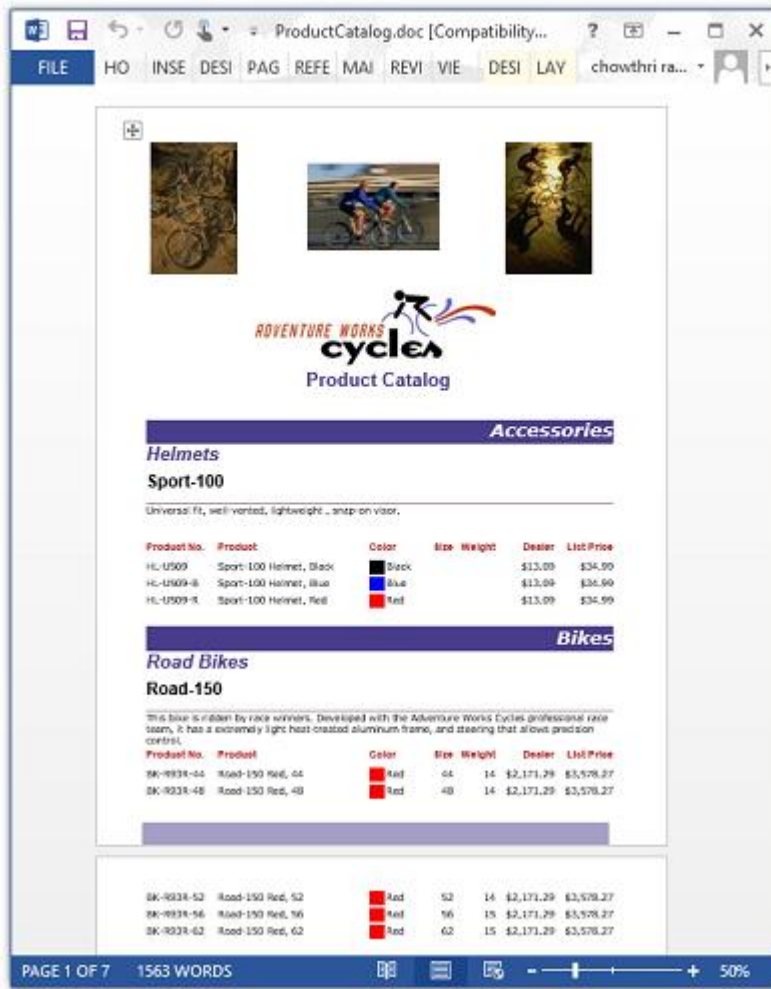
The report generated using the ReportDesigner can also be exported as a Word document using the following code example.

C#

```
// Instantiate the report writer with the parameter "ReportPath" and
// "ReportDataSourceCollection".
ReportWriter reportWriter = new ReportWriter (reportPath, dataSources);
reportWriter.Save("Sample.doc", WriterFormat.WORD);
```

VB.NET

```
'Instantiate the report writer with the parameter "ReportPath" and
'ReportDataSourceCollection".
Dim reportWriter As New ReportWriter (reportPath, dataSources)
reportWriter.Save("Sample.doc", WriterFormat.WORD)
```



Saving report as an HTML

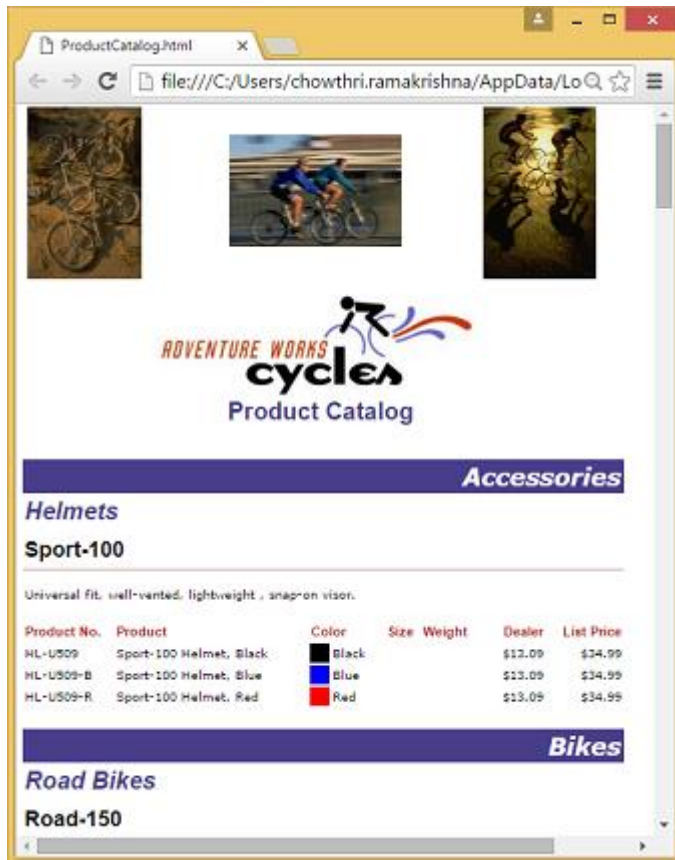
The report generated using the ReportDesigner can be exported as an HTML document using the following code example.

C#

```
//Instantiate the report writer with the parameter "ReportPath" and
"ReportDataSourceCollection".
ReportWriter reportWriter = new ReportWriter(reportPath, dataSources);
reportWriter.Save("Sample.html", WriterFormat.HTML);
```

VB.NET

```
'Instantiate the report writer with the parameter "ReportPath" and
"ReportDataSourceCollection".
Dim reportWriter As New ReportWriter (reportPath, dataSources)
reportWriter.Save("Sample.html", WriterFormat.HTML)
```



ReportWriter API of Essential Studio WPF

Constructors

Name	Description
ReportWriter()	Initializes a new instance of the ReportWriter.
ReportWriter(string)	Initializes a new instance of the ReportWriter with the specified path of the report.
ReportWriter(stream)	Initializes a new instance of the ReportWriter with the stream of the report.
ReportWriter(string, ReportDataSourceCollection)	Initializes a new instance of the ReportWriter with the specified path of the report and data source collection.
ReportWriter(stream, ReportDataSourceCollection)	Initializes a new instance of the ReportWriter with the stream of the report and data source collection.

Properties

Property Name	Description
DataSources	Gets or sets the data source collection.

ReportPath	Gets or sets the report path (file name of the report with its full path).
ReportProcessingMode	Gets or sets the processing mode (either local or remote).
ReportServerCredential	Gets or sets the Report Server credential.
ReportServerFormsCredential	Gets or sets the Report Server form credential. This is used to export SQL Azure RDL reports.
ReportServerUrl	Gets or sets the ReportServerURL.

Methods

Name	Description
GetDataSetNames()	Returns the names of the data set as IList <string>.
GetParameters()	Returns the parameters of the report as ReportParameterInfoCollection.
SetParameters(IEnumerable<ReportParameters>)	Sets the parameters of the report.
LoadReport(stream)	Loads the report with the specified stream that contains the RDL contents.
Save(string,writer type)	Saves the report as a PDF, Word, Excel and HTML documents in the mentioned path.
Save(stream)	Saves the report as a PDF, Word, Excel and HTML documents in the stream.

Events

Events	Description
ReportError	This event is triggered when Report throws Error.
ExportCompleted	This event is triggered when export is completed.
SubreportProcessing	This event is triggered when the report is RDLC and contains with subreport.

How to

Save a report as stream

You can save a report to stream using Save(Stream, WriterFormat) (overloaded method). This method is useful when generating the report on the server side and sending it to a client-side application. The following code explains how to save a report as a stream.

C#

```
//Step 1: create the report data source
ReportDataSourceCollection dataSources = new ReportDataSourceCollection();
```

```
dataSources.Add(new ReportDataSource() { Name = "Sales", Value =  
GetDataSource() });  
//Step 2: Instantiate the report writer with the parameter "ReportPath" and  
"ReportDataSourceCollection"  
ReportWriter reportWriter = new ReportWriter(reportPath, dataSources);  
MemoryStream stream = new MemoryStream();  
//Step 3: Save the report as PDF, Word or Excel document in the form of  
stream contents  
reportWriter.Save(stream, WriterFormat.PDF);
```

VB.NET

```
'Step 1: Create the report data source  
Dim dataSources As New ReportDataSourceCollection()  
dataSources.Add(New ReportDataSource() With {.Name = "Sales", .Value =  
GetDataSource()})  
'Step 2: Instantiate the report writer with the parameter "ReportPath" and  
"ReportDataSourceCollection"  
Dim reportWriter As New ReportWriter(reportPath, dataSources)  
Dim stream As New MemoryStream()  
'Step 3: Save the report as PDF, Word or Excel document in the form of  
stream contents  
reportWriter.Save(stream, WriterFormat.PDF)
```

Ribbon

WPF Ribbon control Overview

The [WPF Ribbon](#) illustrates the implementation of Office UI with Ribbon items and Backstage. Also RibbonWindow has been implemented for giving a themed Office UI look and feel for the traditional window. Functionality of keyboard navigation, Tooltips and Key tips are provided for access to items present in the ribbon.

Components of Ribbon controls

- Ribbon Tabs and Items
- Backstage Button
- Quick Access Toolbar (QAT)
- RibbonStatusBar

Key features

- Provides a ribbon window that overrides the default window and can set visual styles of ribbon window to different theme styles.
- The Ribbon control is available in both normal and simplified layout.
- QuickAccessToolbar provides placing of frequently used items.
- RibbonTab is available to add different Tabs like in Microsoft Outlook.
- RibbonBar is available to structure the layouts.
- Ribbon has a custom implementation of button control as RibbonButton.
- Can minimize / maximize the ribbon.
- Keyboard navigation made easy through KeyTips.

- RibbonStatusBar provides placing of StatusBar items.

Getting Started with WPF Ribbon

This section explains how to implement a similar UI as Microsoft Outlook using Ribbon.

Add ribbon

There are several ways to add Syncfusion control in to Visual Studio WPF project, the following steps will helps to add a Ribbon control through XAML Code.

- Create a WPF project in Visual Studio and refer the following assemblies.
 1. Syncfusion.Tools.Wpf
 2. Syncfusion.Shared.Wpf
- Include an XML namespace for the above assemblies to the Main window.

XML

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

- Change the Window as **RibbonWindow**.

XML

```
<syncfusion:RibbonWindow
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="RibbonControl.MainWindow">
</syncfusion:RibbonWindow>
```

- Add following namespace and inherit MainWindow from **RibbonWindow** in code behind.

C#

```
using Syncfusion.Windows.Tools.Controls;
public partial class MainWindow : RibbonWindow
```

VB.NET

```
Imports Syncfusion.Windows.Tools.Controls
Public class As partial
```

- Now, Add the Ribbon control with a required optimal name, using the included namespace in XAML.

XML

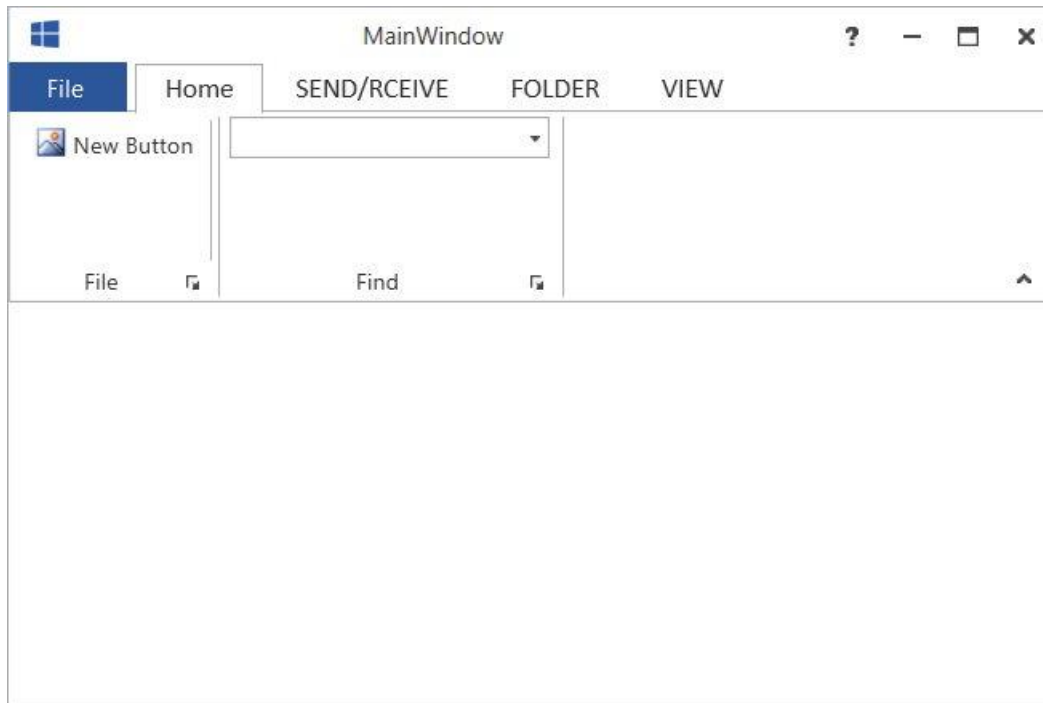
```
<syncfusion:RibbonWindow
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="RibbonControl.MainWindow">
<Grid>
<syncfusion:Ribbon x:Name="_ribbon"/>
</Grid>
</syncfusion:RibbonWindow>
```

Set icon for RibbonWindow

- Icon of the RibbonWindow can be set using the property named **Office2010Icon**. Please refer to the below code.

XML

```
<syncfusion:RibbonWindow
x:Class="WPF.Ribbon.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:local="clr-namespace:WPF.Ribbon"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:sfSkinManager="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow"
Office2010Icon="Assets/New Microsoft.png"
sfSkinManager:SfSkinManager.VisualStudio="Office2016White"
mc:Ignorable="d">
<Grid>
<syncfusion:Ribbon x:Name="_ribbon"/>
</Grid>
</syncfusion:RibbonWindow>
```

Set visual styles

Ribbon supports various visual styles by using the `SfSkinManager`. To apply Visual Studio style on the current layout, refer to the following steps.

- Refer the following assemblies with the project
 1. Syncfusion.SfSkinManager.Wpf
 2. Syncfusion.Themes.Office2013White.Wpf
- Include an XML namespace for the `SfSkinManager` assembly to the `MainWindow`.

XML

```
<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon"/>
</Grid>
</syncfusion:RibbonWindow>
```

- Now apply the value as `Office2013White` to the `VisualStyle` property of the `SfSkinManager` for the `RibbonWindow`.

XML

```
<syncfusion:RibbonWindow
```

```

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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStudio="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon"/>
</Grid>
</syncfusion:RibbonWindow>

```



Add RibbonTab

Ribbon control accept **RibbonTab** as children, Here four **RibbonTab** are added and that can hold **RibbonItems** with **RibbonBar**.

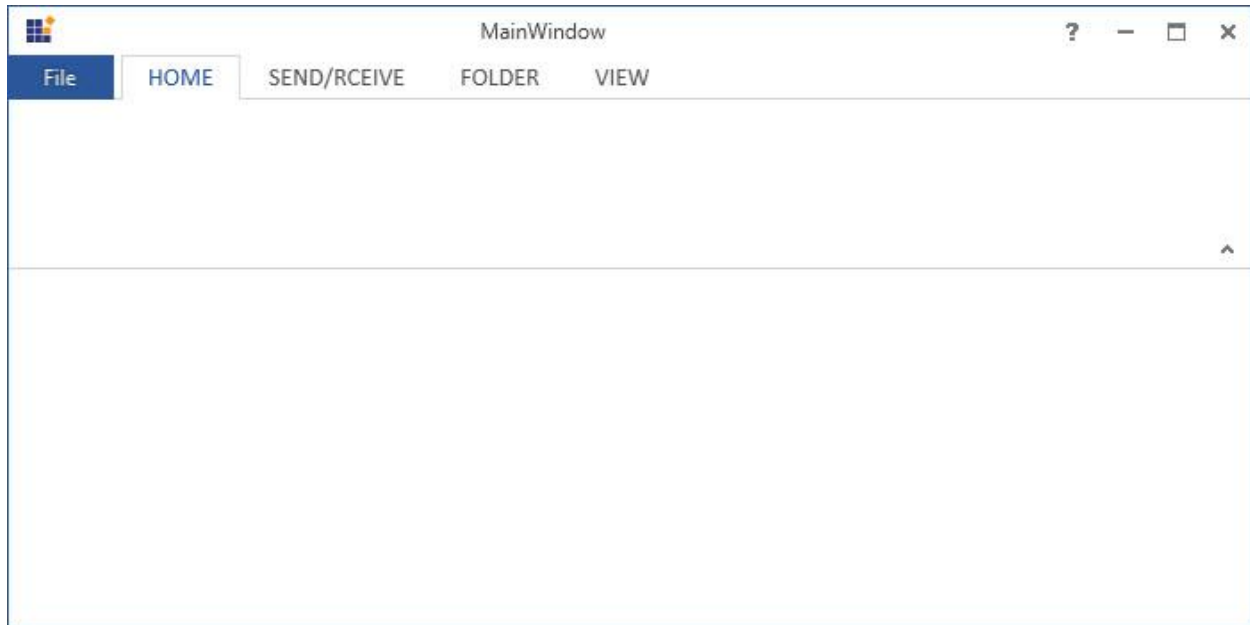
XML

```

<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStudio="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True"/>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>

```

```
</syncfusion:RibbonWindow>
```

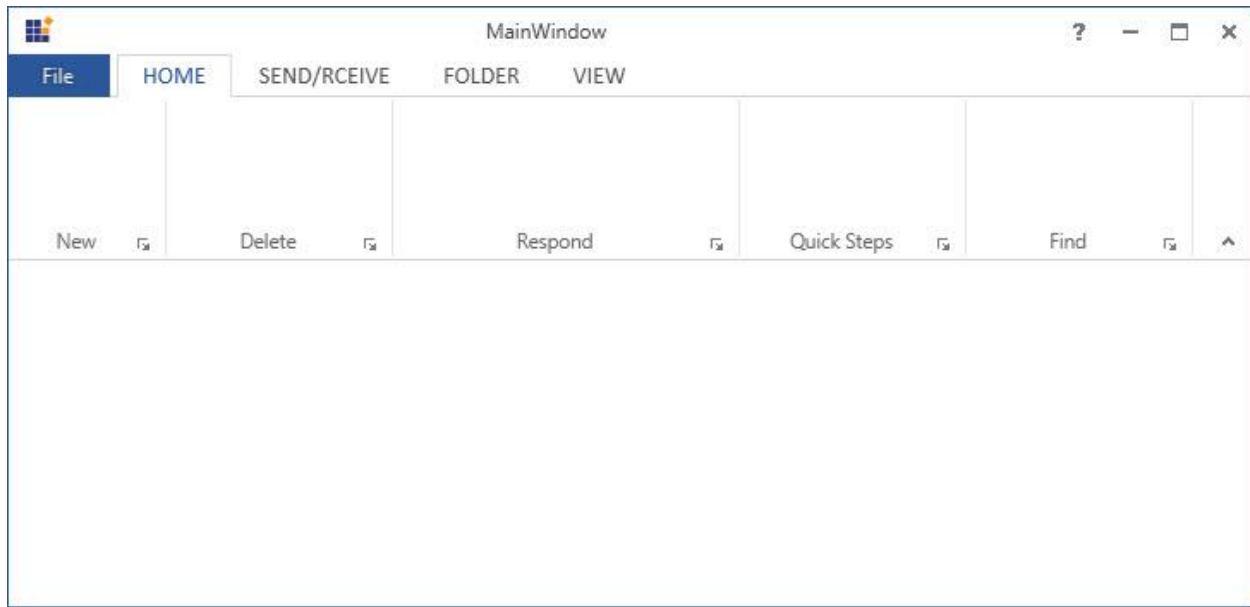


Add RibbonBar

RibbonTab accepts **RibbonBar** as children, here five **RibbonBar** Controls are added inside "HOME" RibbonTab. To set header for RibbonBar, use its **Header** property.

XML

```
<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStudio="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New"/>
<syncfusion:RibbonBar Name="Delete" Width="150" Header="Delete"/>
<syncfusion:RibbonBar Name="Respond" Width="90" Header="Respond"/>
<syncfusion:RibbonBar Name="Quicksteps" Width="90" Header="Quick Steps"/>
<syncfusion:RibbonBar Name="Find" Width="90" Header="Find"/>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```



Add RibbonButton

RibbonButton provides functionalities like normal **Button**. It can place inside the **RibbonBar**. Here, "New" **RibbonBar** holds a **RibbonButton** with a caption as "New Email" using its **Label** property. It also provides **SizeForm** property for different sizes.

If the value of **SizeForm** is large, then image of 16 16 size *has been expanded to 32 32* automatically. Similarly, image of 32 32 *has been compressed to 16 16* if **SizeForm** is ExtraSmall.

XML

```
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
  <syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
</syncfusion:RibbonBar>
```

Also add several other Ribbon Button as per the requirement

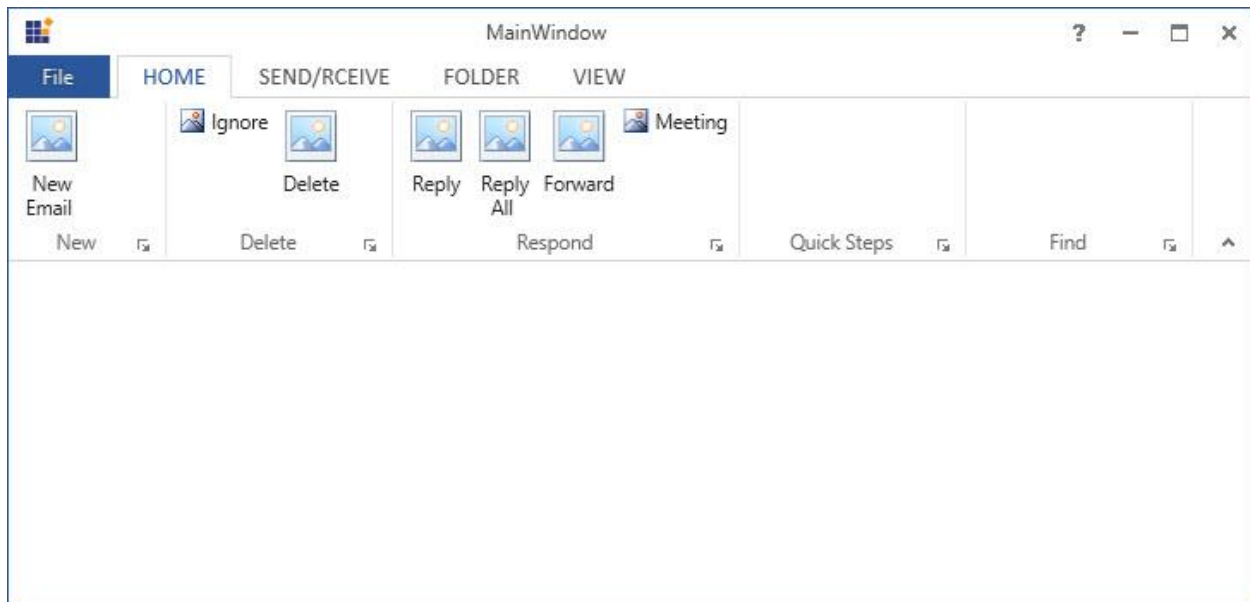
XML

```
<syncfusion:RibbonWindow
  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="RibbonControl.MainWindow"
  xmlns:syncfusionskin="clr-
  namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
  syncfusionskin:SfSkinManager.VisualStyle="Office2013White" >
  <Grid>
    <syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
      <syncfusion:RibbonTab Caption="HOME" IsChecked="True">
        <syncfusion:RibbonBar Name="New" Width="90" Header="New">
          <syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
        </syncfusion:RibbonBar>
        <syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
          <syncfusion:RibbonButton Label="Ignore"/>
          <syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
        </syncfusion:RibbonBar>
      </syncfusion:RibbonTab>
    </syncfusion:Ribbon>
  </Grid>
</syncfusion:RibbonWindow>
```

```

</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Quicksteps" Width="90" Header="Quick Steps"/>
<syncfusion:RibbonBar Name="Find" Width="90" Header="Find"/>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Note: Image of any size has been used for `RibbonButton` and also it supports image of all formats.

Add DropDownButton

`DropDownButton` appears like normal button that contains a drop arrow. It displays some items, while click on it. It accepts `DropDownMenuItem` as its children. Here, "New" RibbonBar holds a `DropDownButton` with a caption as "New Items" using its `Label` property also holding items like "E-mail Message", "Appointment", "Meeting", "Contact" and "Task". It also provides `SizeForm` property for different sizes.

XML

```

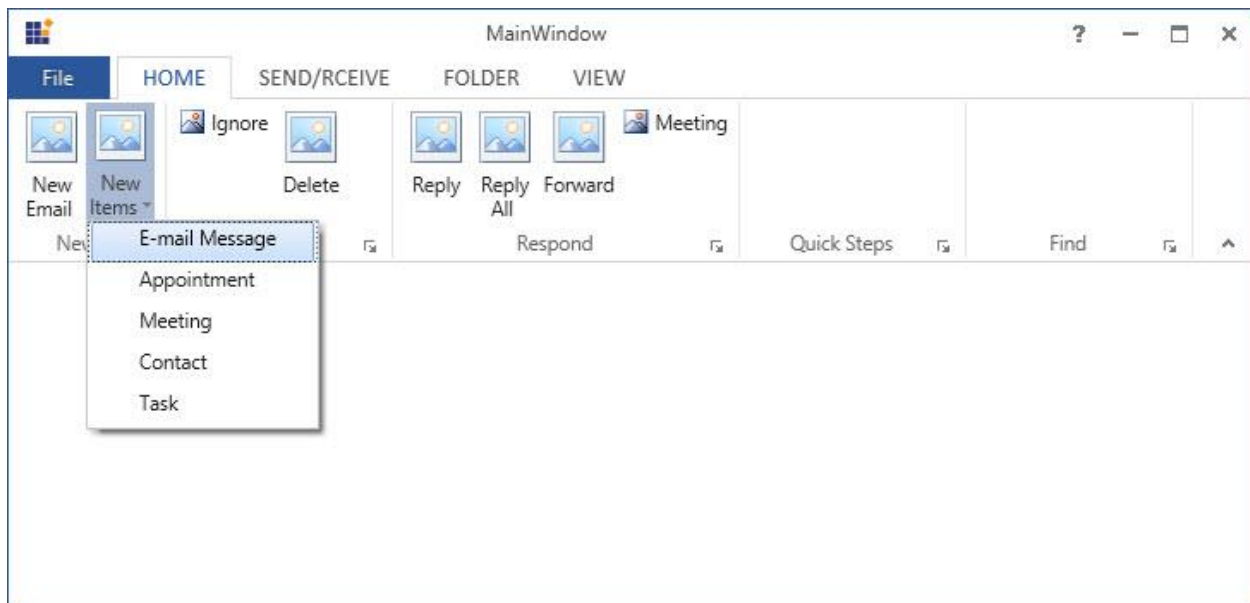
<syncfusion:RibbonWindow
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"

```

```

syncfusionskin:SfSkinManager.VisualStyle="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Quicksteps" Width="90" Header="Quick Steps"/>
<syncfusion:RibbonBar Name="Find" Width="90" Header="Find"/>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

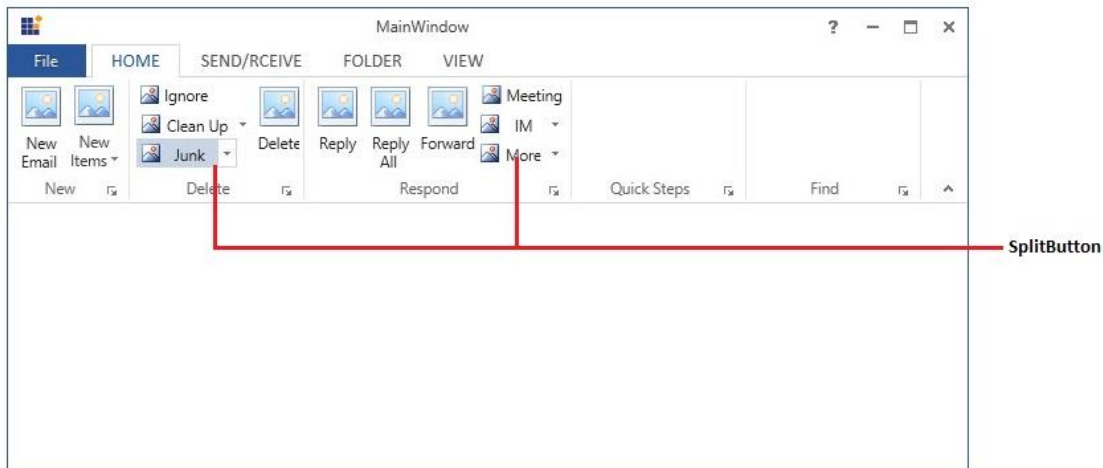


Add SplitButton

SplitButton also appears with a drop arrow. It displays some items while click on it. It accepts `DropDownMenuItem` as its children. Here, "Delete" RibbonBar holds a `SplitButton` with a caption as "Clean Up" using its `Label` property also holding items like "Clean Up Folder", "Clean Up Conversation" and "Clean Up Folder/SubFolder". It also provides `SizeForm` property for different sizes.

XML

```
<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStyle="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Quicksteps" Width="90" Header="Quick Steps"/>
<syncfusion:RibbonBar Name="Find" Width="90" Header="Find"/>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```



Add RibbonGallery

RibbonGallery displays items with good look and feel and it also used to classify the items as groups for easy navigation. Here, "QuickSteps" RibbonBar holds a **RibbonGallery** with **InRibbon** visual mode to place gallery items with in Ribbon. Also **ItemHeight**, **ItemWidth** properties are used to set height and width respectively.

XML

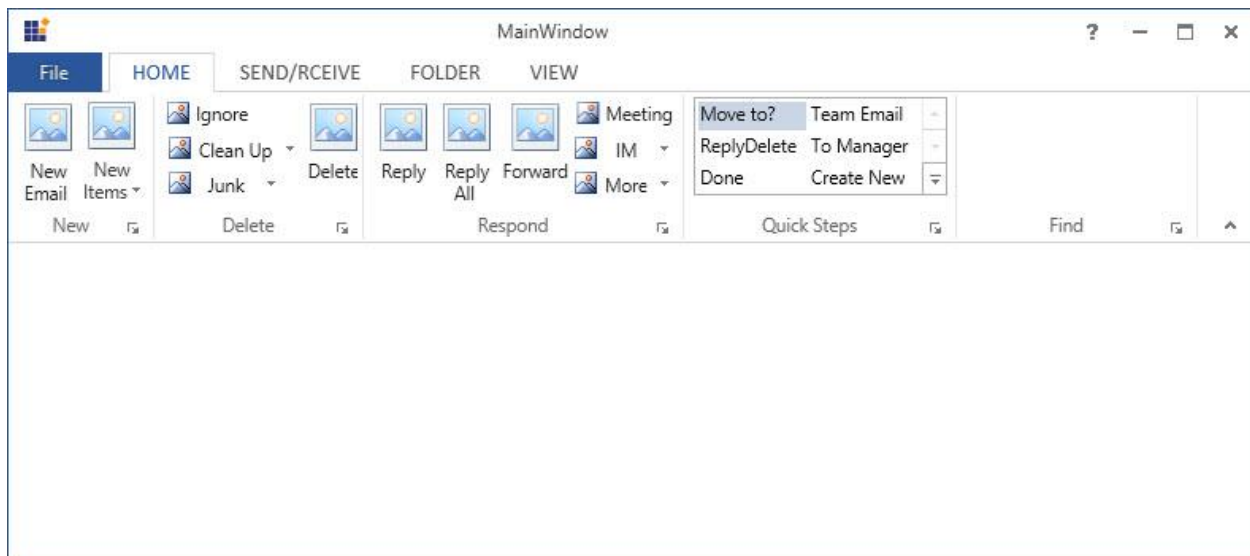
```
<syncfusion:RibbonWindow
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStyle="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
```



```

<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="20" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Find" Width="90" Header="Find"/>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Add RibbonComboBox

RibbonComboBox is used to display the list of items. It accepts **ComboBoxItems** as its children. Here, "Find" RibbonBar holds a **RibbonComboBox** with a caption as "Filter Email" using its **Label** property also holding items like "Person1@mail.com", "Person1@mail.com" and "Person1@mail.com".

XML

```
<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStyle="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="20" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

```

</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

Add QAT

QuickAccessToolBar (QAT) is used to group the frequently used commands and access the commands easily without having to search for the command in the menu bar. Also it can be placed above or below the ribbon.

XML

```

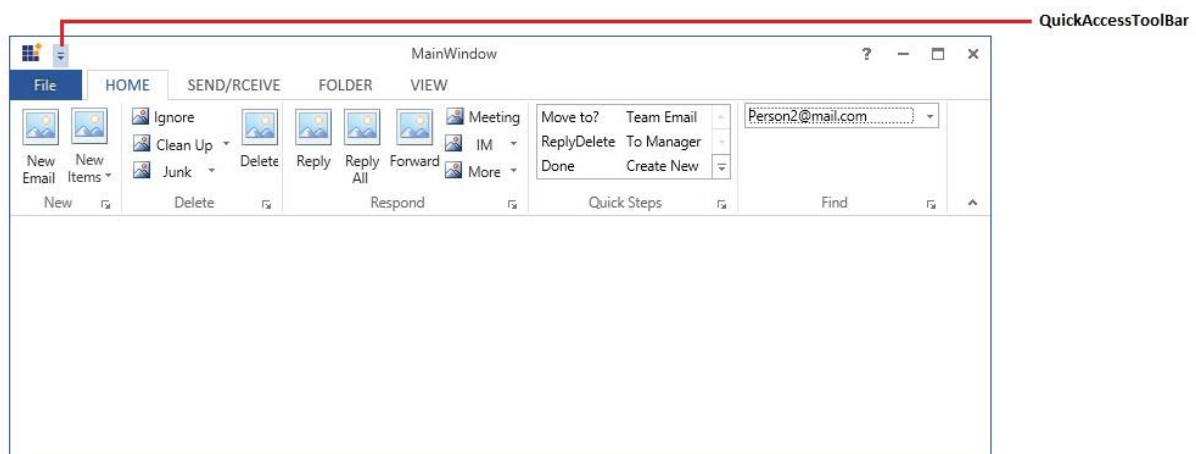
<syncfusion:RibbonWindow
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStudio="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>

```

```

<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="20" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Add items to Quick Access toolbar

QATMenuItems property of **QuickAccessToolBar** used to add items to Dropdown menu of QAT.

XML

```

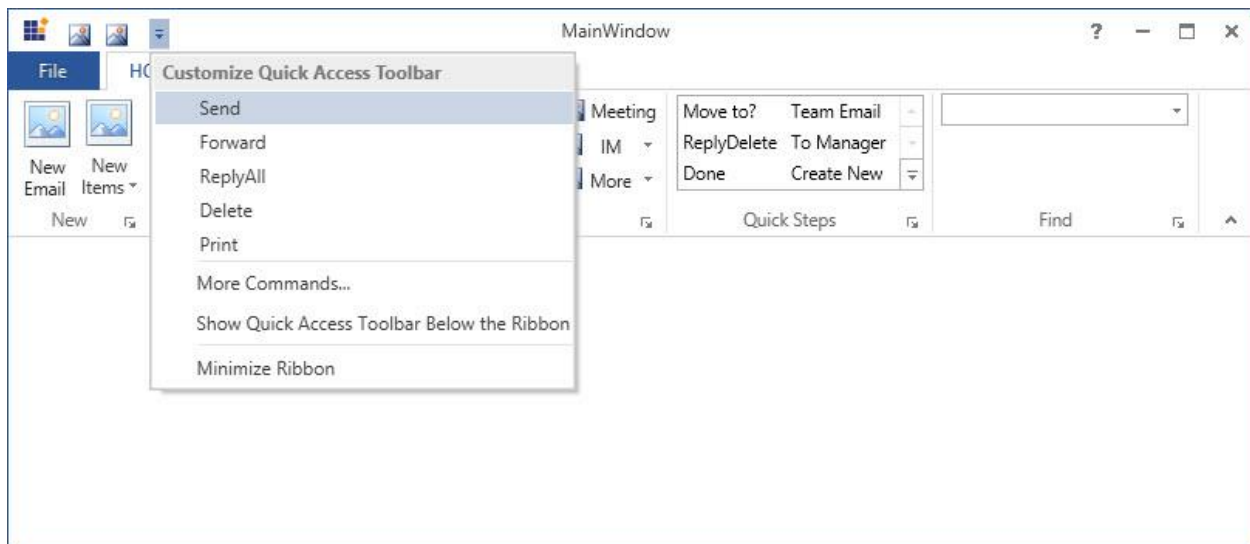
<syncfusion:RibbonWindow
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStudio="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="20" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>

```

```

</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall"/>
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Send" />
<syncfusion:RibbonButton Label="Forward" />
<syncfusion:RibbonButton Label="ReplyAll" />
<syncfusion:RibbonButton Label="Delete" />
<syncfusion:RibbonButton Label="Print" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Add BackStage

BackStage can be added by using **BackStage** property of Ribbon. To show the BackStage by, click the **FILE** Menu in Ribbon like in Microsoft Outlook.

XML

```

<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStudio="Office2013White" >

```



```

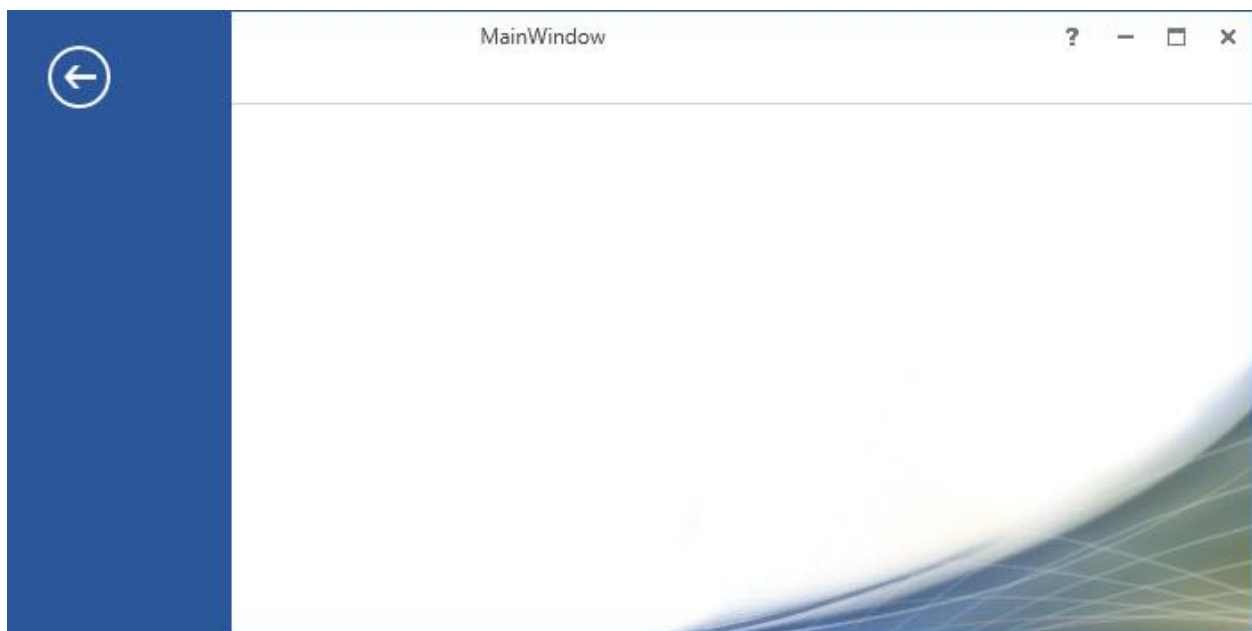
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="20" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" />

```

```

<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Send" />
<syncfusion:RibbonButton Label="Forward" />
<syncfusion:RibbonButton Label="ReplyAll" />
<syncfusion:RibbonButton Label="Delete" />
<syncfusion:RibbonButton Label="Print" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage/>
</syncfusion:Ribbon.BackStage>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Add application menu

An Application Menu contains standard commands that are performed like in Microsoft Outlook 2003 UI. Using the `ApplicationMenu` property of the Ribbon, this Menu is viewed by added at the top-left corner of the window.

Note: Visual style is set to `Default` for RibbonWindow in SkinStorage. and BackStage is not applicable when `ApplicationMenu` is used in Ribbon.

XML

```

<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
syncfusion:SkinStorage.VisualStyle="Default" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">

```



```

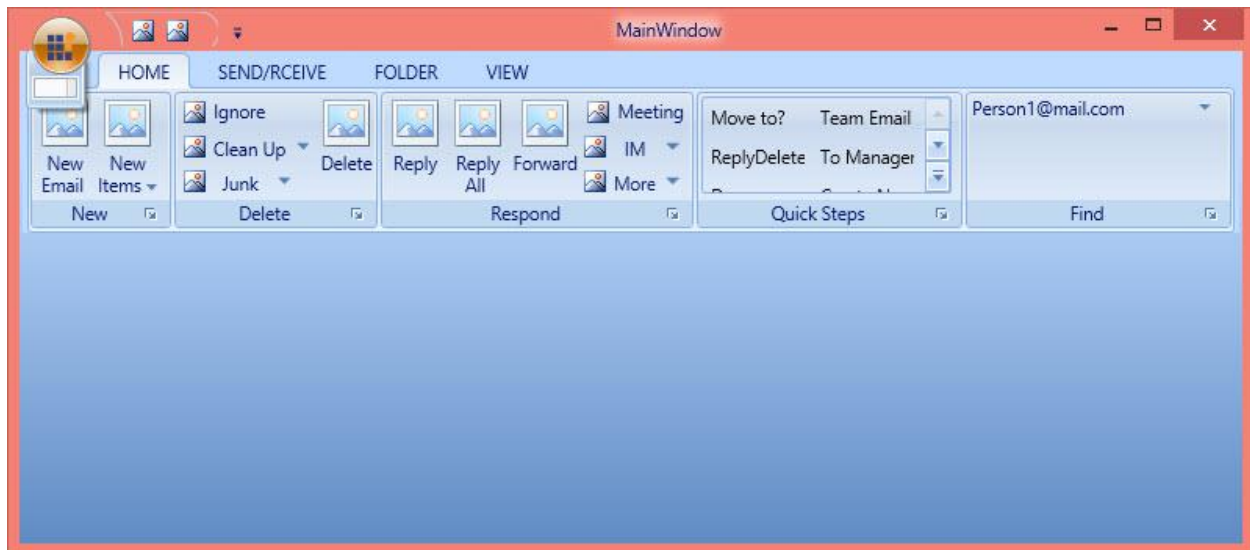
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="30" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" />
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Send" />

```

```

<syncfusion:RibbonButton Label="Forward" />
<syncfusion:RibbonButton Label="ReplyAll" />
<syncfusion:RibbonButton Label="Delete" />
<syncfusion:RibbonButton Label="Print" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.ApplicationMenu>
<syncfusion:ApplicationMenu Name="_applicationMenu" Width="38" Height="38"
syncfusion:Ribbon.KeyTip="F" IsPopupOpen="False"
ApplicationButtonImage="syncfusion.png">
</syncfusion:ApplicationMenu>
</syncfusion:Ribbon.ApplicationMenu>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Add application items to the application menu

ApplicationItems are displayed in bottom of the Application menu. Different ApplicationItems are added to an application menu using its ApplicationItems property. Here two RibbonButton with its Label property as "Options" and "Exits" are added as ApplicationMenuItems.

XML

```

<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
syncfusion:SkinStorage.VisualStudio="Default"
>
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">

```

```

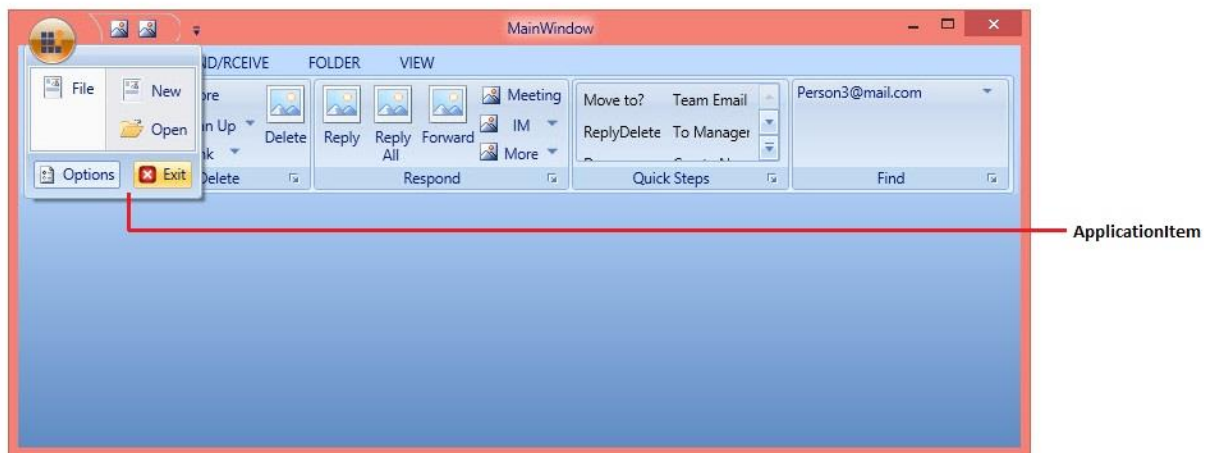
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="30" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" />
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Send" />
<syncfusion:RibbonButton Label="Forward" />
<syncfusion:RibbonButton Label="ReplyAll" />
<syncfusion:RibbonButton Label="Delete" />
<syncfusion:RibbonButton Label="Print" />

```

```

</syncfusion:QuickAccessToolBar.QATMenuItems>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.ApplicationMenu>
<syncfusion:ApplicationMenu Name="_applicationMenu" Width="38" Height="38"
syncfusion:Ribbon.KeyTip="F" IsPopupOpen="False"
ApplicationButtonImage="syncfusion.png">
<syncfusion:SimpleMenuButton Label="File" Icon="Document32.png"/>
<syncfusion:ApplicationMenu.MenuItems>
<syncfusion:SimpleMenuButton Label="New" Icon="Document32.png"/>
<syncfusion:SimpleMenuButton Label="Open" Icon="Open32.png" />
</syncfusion:ApplicationMenu.MenuItems>
<syncfusion:ApplicationMenu.ApplicationItems>
<syncfusion:RibbonButton SizeForm = "Small" Label="Options"
SmallIcon="Options.png"/>
<syncfusion:RibbonButton SizeForm = "Small" Label="Exit"
SmallIcon="Exit.png"/>
</syncfusion:ApplicationMenu.ApplicationItems>
</syncfusion:ApplicationMenu>
</syncfusion:Ribbon.ApplicationMenu>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Adding custom controls to the title bar

The `RibbonWindow` allows to load any custom controls into the right side of the title bar by using both [HeaderItems](#) and [HeaderItemsSource](#) property.

Adding items using `HeaderItems`

The [HeaderItems](#) property of the `RibbonWindow` allows you to load any controls directly into the title bar.

XML

```

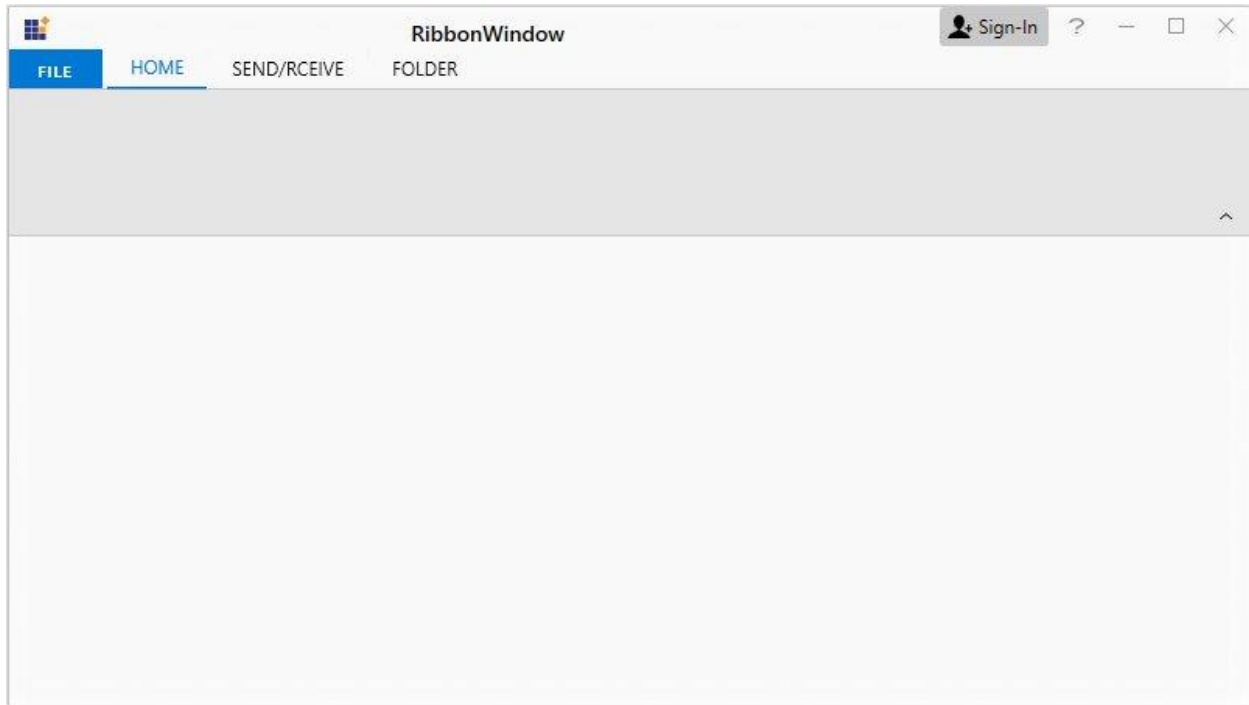
<syncfusion:RibbonWindow x:Class="CustomControl.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

```

```

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:CustomControl"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.Theme="{skin:SkinManagerExtension ThemeName=FluentLight}"
mc:Ignorable="d"
Title="CustomControl in the TitleBar"
Height="450" Width="800"
IsGlassActive="False"
WindowStartupLocation="CenterScreen"
WindowState="Normal">
<syncfusion:RibbonWindow.Resources>
<DataTemplate x:Key="icontemplate">
<Grid>
<Path Data="M26.199951,12.300006L28.399963,12.300006 28.399963,15.899998
32,15.899998 32,18.000004 28.399963,18.000004 28.399963,21.600012
26.199951,21.600012 26.199951,18.000004 22.599976,18.000004
22.599976,15.800007 26.199951,15.800007z M14.799988,0C18.599976,-7.6615834E-
08 21.699951,3.8000038 21.699951,8.6000081 21.699951,12.500003
19.699951,15.399998 17,16.800007 16.599976,16.899998 16,17.399998
16,18.000004 16,18.500004 16.599976,19.000004 16.899963,19.10001
22.599976,21.100012 27.5,23.9 29.5,29.400002L0,29.400002C2,23.800009
6.8999634,21.300009 12.599976,19.10001 12.899963,19.000004
13.399963,18.500004 13.399963,18.000004 13.399963,17.500004
12.899963,17.000004 12.599976,16.899998 9.7999878,15.399998
7.8999634,12.600009 7.8999634,8.6000081 7.7999878,3.8000038 10.899963,-
7.6615834E-08 14.799988,0z" Stretch="Uniform" Fill="{Binding
RelativeSource={RelativeSource Mode=Self}, Path=(TextBlock.Foreground)}"
Width="16" Height="16" 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>
</DataTemplate>
</syncfusion:RibbonWindow.Resources>
<syncfusion:RibbonWindow.HeaderItems>
<syncfusion:ButtonAdv x:Name="headerItem" Height="25" Label="Sign-In"
SizeMode="Normal" IconTemplate="{StaticResource icontemplate}" />
</syncfusion:RibbonWindow.HeaderItems>
<Grid>
<syncfusion:Ribbon x:Name="ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True"/>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Adding items using HeaderItemsSource

The [HeaderItemsSource](#) property of the RibbonWindow allows you to bind a collection of objects which used to load custom controls into the right side of the title bar.

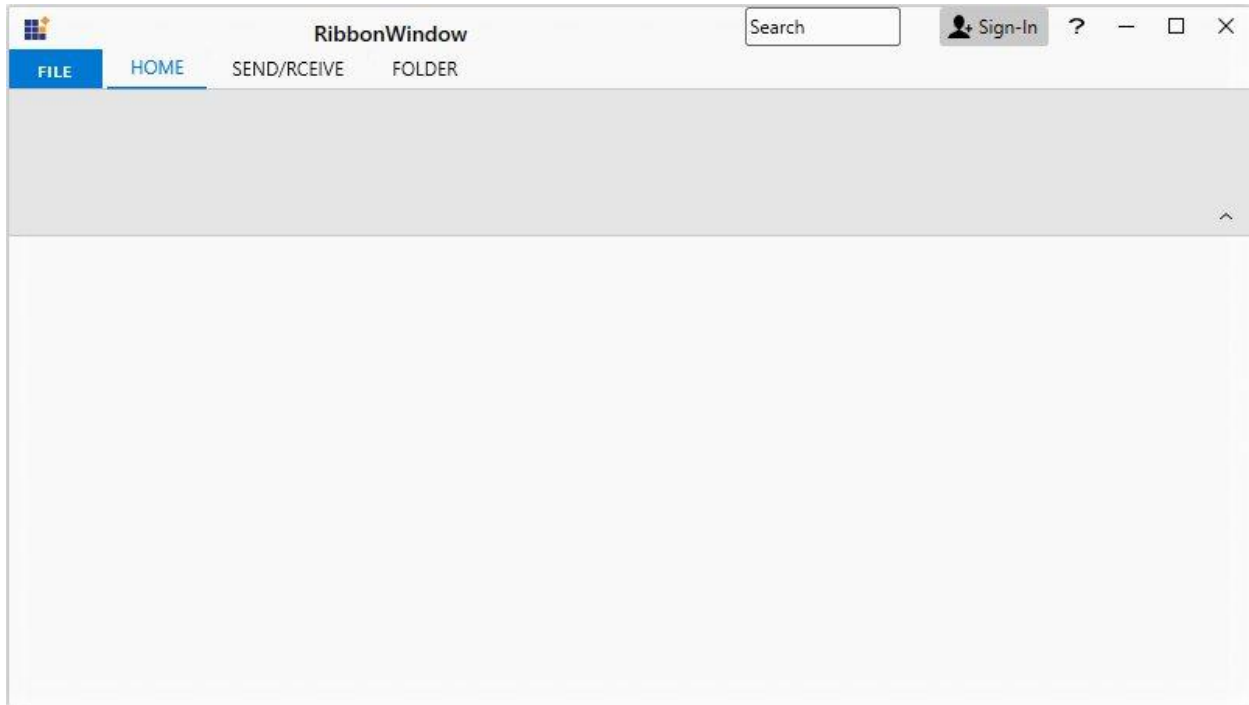
XML

```
<syncfusion:RibbonWindow x:Class="CustomControl.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:CustomControl"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.Theme="{skin:SkinManagerExtension ThemeName=FluentLight}"
xmlns:listCollection="clr-namespace:System.Collections;assembly=mscorlib"
mc:Ignorable="d"
Title="RibbonWindow"
Height="450" Width="800"
IsGlassActive="False"
WindowStartupLocation="CenterScreen"
WindowState="Normal">
<syncfusion:RibbonWindow.Resources>
<DataTemplate x:Key="icontemplate">
<Grid>
<Path Data="M26.199951,12.300006L28.399963,12.300006 28.399963,15.899998
32,15.899998 32,18.000004 28.399963,18.000004 28.399963,21.600012
26.199951,21.600012 26.199951,18.000004 22.599976,18.000004
22.599976,15.800007 26.199951,15.800007z M14.799988,0C18.599976,-7.6615834E-
08 21.699951,3.8000038 21.699951,8.6000081 21.699951,12.500003
19.699951,15.399998 17,16.800007 16.599976,16.899998 16,17.399998
16,18.000004 16,18.500004 16.599976,19.000004 16.899963,19.10001
```

```

22.599976,21.100012 27.5,23.9 29.5,29.400002L0,29.400002C2,23.800009
6.8999634,21.300009 12.599976,19.10001 12.899963,19.000004
13.399963,18.500004 13.399963,18.000004 13.399963,17.500004
12.899963,17.000004 12.599976,16.899998 9.7999878,15.399998
7.8999634,12.600009 7.8999634,8.6000081 7.7999878,3.8000038 10.899963,-
7.6615834E-08 14.799988,0z" Stretch="Uniform" Fill="{Binding
RelativeSource={RelativeSource Mode=Self}, Path=(TextBlock.Foreground)}"
Width="16" Height="16" 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>
</DataTemplate>
</syncfusion:RibbonWindow.Resources>
<syncfusion:RibbonWindow.HeaderItemsSource>
<listCollection:ArrayList>
<TextBox x:Name="textBox" Text="Search" Width="100"/>
<syncfusion:ButtonAdv x:Name="headerItem" Height="25" Margin="25,0,0,0"
Label="Sign-In" SizeMode="Normal" IconTemplate="{StaticResource
icontemplate}" />
</listCollection:ArrayList>
</syncfusion:RibbonWindow.HeaderItemsSource>
<Grid>
<syncfusion:Ribbon x:Name="ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True"/>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Setting HeaderItemTemplate

The RibbonWindow allows you to customize the visual appearance of the custom items stored in the [HeaderItemsSource](#) using [HeaderItemTemplate](#) property.

XML

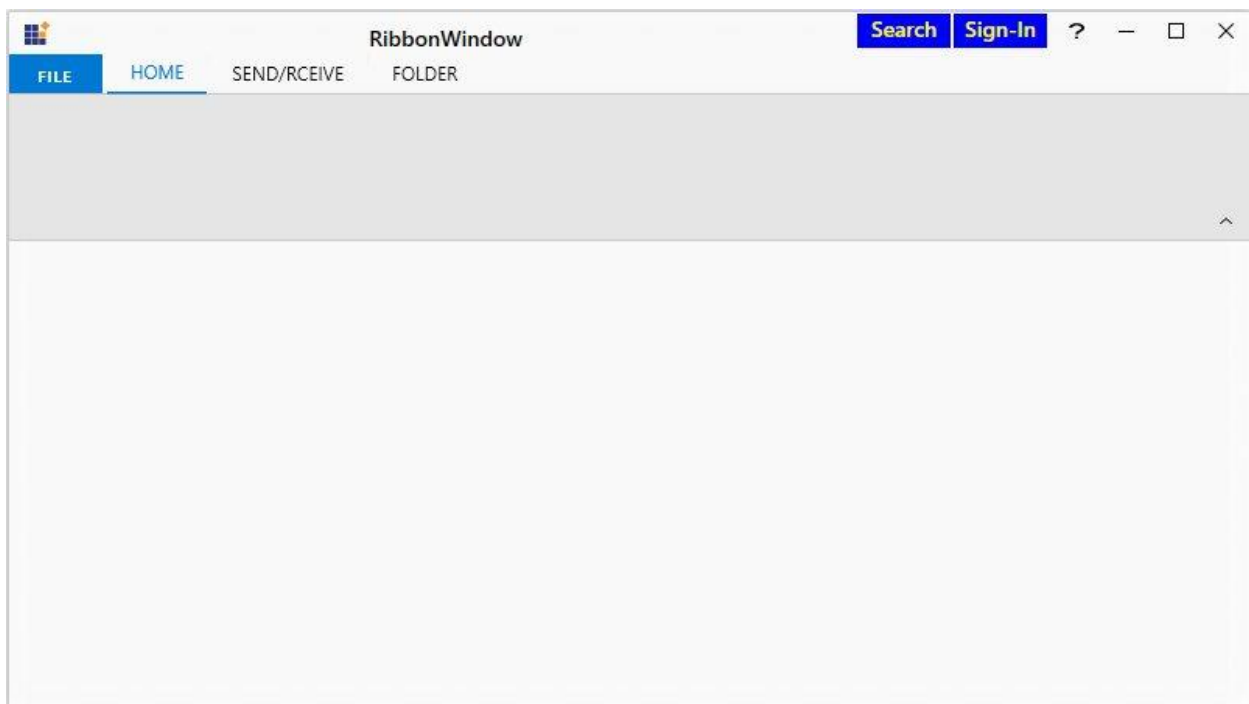
```
<syncfusion:RibbonWindow x:Class="CustomControl.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:CustomControl"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.Theme="{skin:SkinManagerExtension ThemeName=FluentLight}"
xmlns:listCollection="clr-namespace:System.Collections;assembly=microsoft"
xmlns:system="clr-namespace:System;assembly=microsoft"
mc:Ignorable="d"
Title="RibbonWindow"
Height="450" Width="800"
IsGlassActive="False"
WindowStartupLocation="CenterScreen"
WindowState="Normal">
<syncfusion:RibbonWindow.HeaderItemTemplate>
<DataTemplate>
<Grid>
<Border Margin="1" Width="60" Height="22" Background="Blue">
<TextBlock Foreground="Yellow" Text="{Binding}" TextAlignment="Center" />
</Border>
</Grid>
</DataTemplate>
</syncfusion:RibbonWindow.HeaderItemTemplate>
```



```

<syncfusion:RibbonWindow.HeaderItemsSource>
<listCollection:ArrayList>
<system:String>Search</system:String>
<system:String>Sign-In</system:String>
</listCollection:ArrayList>
</syncfusion:RibbonWindow.HeaderItemsSource>
<Grid>
<syncfusion:Ribbon x:Name="ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True"/>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Setting HeaderItemTemplateSelector

The RibbonWindow allows you to customize the visual appearance of each item with different templates based on specific constraints by using the [HeaderItemTemplateSelector](#).

XML

```

<syncfusion:RibbonWindow x:Class="CustomControl.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:CustomControl"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.Theme="{skin:SkinManagerExtension ThemeName=FluentLight}"
xmlns:listCollection="clr-namespace:System.Collections;assembly=mscorlib"

```

```

xmlns:system="clr-namespace:System;assembly=mscorlib"
mc:Ignorable="d"
Title="RibbonWindow"
Height="450" Width="800"
IsGlassActive="False"
WindowStartupLocation="CenterScreen"
WindowState="Normal">
<syncfusion:RibbonWindow.Resources>
<DataTemplate x:Key="BlueBorderTemplate" >
<Border Margin="1" Width="60" Height="22" Background="Blue">
<TextBlock Foreground="Yellow" Text="{Binding}" TextAlignment="Center" />
</Border>
</DataTemplate>
<DataTemplate x:Key="OrangeBorderTemplate">
<Border Margin="1" Width="60" Height="22" Background="Orange">
<TextBlock Foreground="Green" Text="{Binding}" TextAlignment="Center" />
</Border>
</DataTemplate>
</syncfusion:RibbonWindow.Resources>
<syncfusion:RibbonWindow.HeaderItemTemplateSelector>
<local:ItemTemplateSelector/>
</syncfusion:RibbonWindow.HeaderItemTemplateSelector>
<syncfusion:RibbonWindow.HeaderItemsSource>
<listCollection:ArrayList>
<system:String>Search</system:String>
<system:String>Sign-In</system:String>
</listCollection:ArrayList>
</syncfusion:RibbonWindow.HeaderItemsSource>
<Grid>
<syncfusion:Ribbon x:Name="ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True"/>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

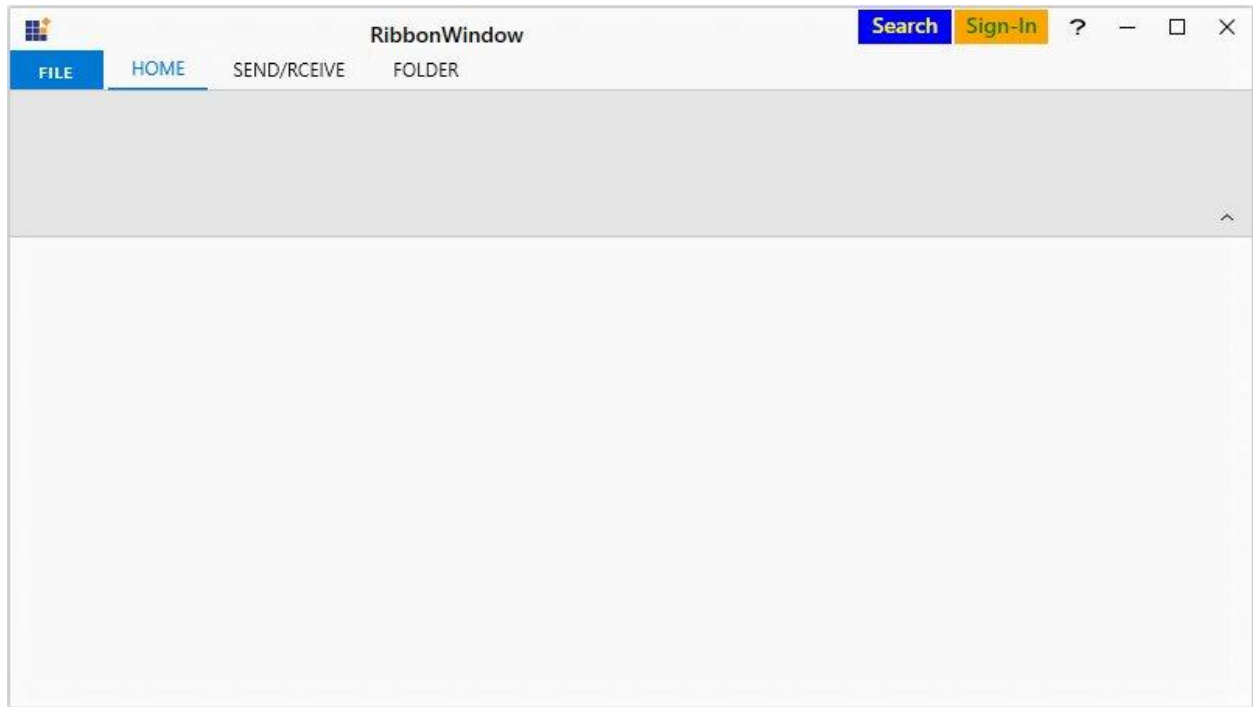
C#

```

public class ItemTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject
    container)
    {
        if (item.ToString() == "Search")
        {
            return (container as FrameworkElement).TryFindResource("BlueBorderTemplate")
            as DataTemplate;
        }
        if (item.ToString() == "Sign-In")
        {
            return (container as
            FrameworkElement).TryFindResource("OrangeBorderTemplate") as DataTemplate;
        }
        return base.SelectTemplate(item, container);
    }
}

```

```
}
```



Note: [View sample in GitHub](#)

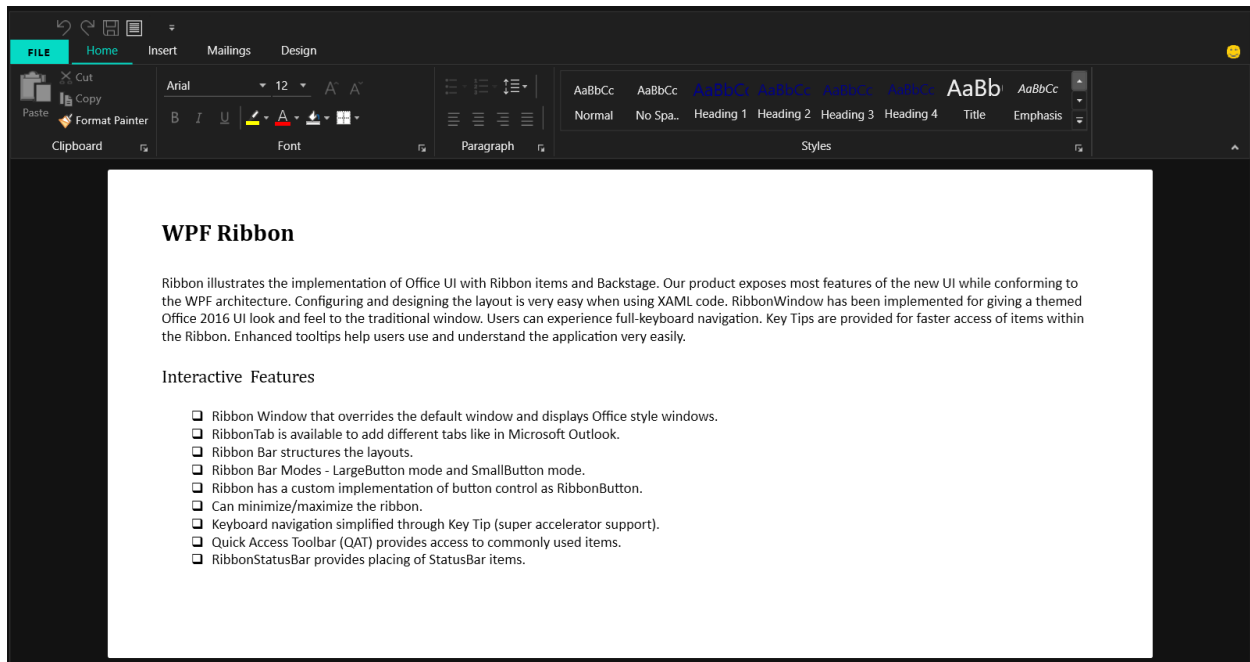
Set simplified layout

The simplified layout is designed to display the most commonly used Ribbon commands in a single line interface, allowing more screen space for compact content viewing, while other commands are placed inside the overflow menu. To know more about the simplified layout, refer [here](#).

Theme

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

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



BackStage in WPF Ribbon

Backstage is a separate view containing tabs and buttons that can show an application's information and basic settings. The backstage items can also be arranged at either the top or bottom. It provides different types of animations such as fade, scale, and zoom as well as support to customize the animation duration.

BackStage settings in Ribbon

The BackStage can be added by using [BackStage](#) property of Ribbon. To show the BackStage by, click the FILE Menu in Ribbon like in Microsoft Outlook.

XML

```
<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStyle="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</Grid>
```

```

<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
  <syncfusion:RibbonButton Label="Ignore"/>
  <syncfusion:SplitButton Label="Clean Up">
    <syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
    <syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
    <syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
  </syncfusion:SplitButton>
  <syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
  <syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
  <syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
  <syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
  <syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
  <syncfusion:RibbonButton Label="Meeting"/>
  <syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
  <syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
  <syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
  ItemHeight="20" ItemWidth="70">
    <syncfusion:RibbonGalleryItem Content="Move to?"/>
    <syncfusion:RibbonGalleryItem Content="Team Email"/>
    <syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
    <syncfusion:RibbonGalleryItem Content="To Manager"/>
    <syncfusion:RibbonGalleryItem Content="Done"/>
    <syncfusion:RibbonGalleryItem Content="Create New"/>
  </syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
  <syncfusion:RibbonComboBox Label="Filter Email" Width="160">
    <ComboBoxItem>Person1@mail.com</ComboBoxItem>
    <ComboBoxItem>Person2@mail.com</ComboBoxItem>
    <ComboBoxItem>Person3@mail.com</ComboBoxItem>
  </syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
  <syncfusion:QuickAccessToolBar>
    <syncfusion:RibbonButton SizeForm="ExtraSmall"/>
    <syncfusion:RibbonButton SizeForm="ExtraSmall" />
    <syncfusion:QuickAccessToolBar.QATMenuItems>
      <syncfusion:RibbonButton Label="Send" />
      <syncfusion:RibbonButton Label="Forward" />
      <syncfusion:RibbonButton Label="ReplyAll" />
      <syncfusion:RibbonButton Label="Delete" />
      <syncfusion:RibbonButton Label="Print" />
    </syncfusion:QuickAccessToolBar.QATMenuItems>
  </syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
  <syncfusion:Backstage/>
</syncfusion:Ribbon.BackStage>
</syncfusion:Ribbon>

```

```
</Grid>
</syncfusion:RibbonWindow>
```



Customize the BackStage Visibility

The [IsBackStageVisible](#) property of Ribbon enables you to show/hide the BackStage. The following code example illustrates how to show or hide BackStage.

1) Through Property

XML

```
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top"
IsBackStageVisible ="True"/>
```

C#

```
_ribbon.IsBackStageVisible = true;
```

2) Through Methods

C#

```
private void ShowBackstage_Click(object sender, RoutedEventArgs e)
{
    //to show back stage
    _ribbon.ShowBackStage();
}
private void HideBackstage_Click(object sender, RoutedEventArgs e)
{
    //to hide the back stage
    _ribbon.HideBackStage();
}
```

VB.NET

```

Private Sub ShowBackstage_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    'to show back stage
    _ribbon.ShowBackStage()
End Sub
Private Sub HideBackstage_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    'to hide the back stage
    _ribbon.HideBackStage()
End Sub

```

3) Through Commands

Ribbon control provides the static command [OpenBackStage](#) to show/hide the BackStage based on the parameter passed to it.

XML

```

<syncfusion:RibbonButton x:Name="ribbonButton" Label="Open Backstage"
Command="{x:Static syncfusion:RibbonCommands.OpenBackStage}"
CommandTarget="{Binding ElementName=ribbon}">
<syncfusion:RibbonButton.CommandParameter>
<!--If we passes false it will hide the backstage otherwise it will show the
backstage-->
<sys:Boolean>true</sys:Boolean>
</syncfusion:RibbonButton.CommandParameter>
</syncfusion:RibbonButton>

```

Note: In order to bind the [OpenBackStage](#) command, CommandTarget and CommandParameter must be defined.

Add BackStageCommandButton

The `BackStageCommandButton` can be added to BackStage as BackStage Element. Here four BackStageCommandButtons are added with `Header` property value as "Save", "SaveAttachments", "Options" and "Exit".

XML

```

<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStudio="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>

```

```

<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="20" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" />
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Send" />
<syncfusion:RibbonButton Label="Forward" />
<syncfusion:RibbonButton Label="ReplyAll" />
<syncfusion:RibbonButton Label="Delete" />
<syncfusion:RibbonButton Label="Print" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
</syncfusion:QuickAccessToolBar>

```



```
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>
<syncfusion:BackStageCommandButton Header="Save" />
<syncfusion:BackStageCommandButton Header="SaveAttachments" />
<syncfusion:BackStageCommandButton Header="Options" />
<syncfusion:BackStageCommandButton Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```



Setting image to BackStageCommandButton

The [BackStageCommandButton](#) allows to display any type of image such as glyph, font or any custom content using the [IconTemplate](#) property, which is the preferred option. It also allows to display a normal image or vector image using the [IconType](#) enumeration property. The default value of the [IconType](#) property is [Icon](#). The [IconType](#) enumeration has the following values:

- **Icon** - Gets the details of the icon from the [Icon](#) property and sets it to the [BackStageCommandButton](#).
- **VectorImage** - Gets the details of the icon path from the [VectorImage](#) property and sets it to the [BackStageCommandButton](#).

Note: The [BackStageCommandButton](#) loads icon in the following priority order,

- [IconTemplate](#)
- [VectorImage](#)
- [Icon](#)

Setting icon template

The [IconTemplate](#) property provides support to set any type of image such as glyph, font or any custom content to the [BackStageCommandButton](#). The [BackStageCommandButton](#) displays the [IconTemplate](#) in 16 * 16 size.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top"
EnableSimplifiedLayoutMode="True">
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>
<syncfusion:BackStageCommandButton Header="Save" >
<syncfusion:BackStageCommandButton.IconTemplate>
<DataTemplate>
<Path Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M5.0000019,11 L5.0000019,15 11.000002,15 11.000002,11 z M4.0000019,1
L4.0000019,6 12.000002,6 12.000002,1 z M1,1 L1,13.174 2.7160001,15
4.0000019,15 4.0000019,10 12.000002,10 12.000002,15 15,15 15,1 13.000002,1
13.000002,7 3.0000019,7 3.0000019,1 z M0,0 L3.0000019,0 13.000002,0 16,0
16,16 12.000002,16 4.0000019,16 2.2840004,16 0,13.57 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground)}" Stretch="Uniform" />
</DataTemplate>
</syncfusion:BackStageCommandButton.IconTemplate>
</syncfusion:BackStageCommandButton>
<syncfusion:BackStageCommandButton Header="Close" >
<syncfusion:BackStageCommandButton.IconTemplate>
<DataTemplate>
<Grid Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center" SnapsToDevicePixels="true">
<Path
Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M1.4139423,0L7.0029922,5.5845888 12.592018,0 14.006015,1.4149939
8.4180527,6.9985202 14.006,12.582007 12.591996,13.997001 7.0030056,8.4124444
1.4140122,13.997001 1.5026823E-05,12.582007 5.5879484,6.9985092 0,1.4149939z
"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground)}"
SnapsToDevicePixels="True" Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:BackStageCommandButton.IconTemplate>
</syncfusion:BackStageCommandButton>
```

```

</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Clipboard">
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

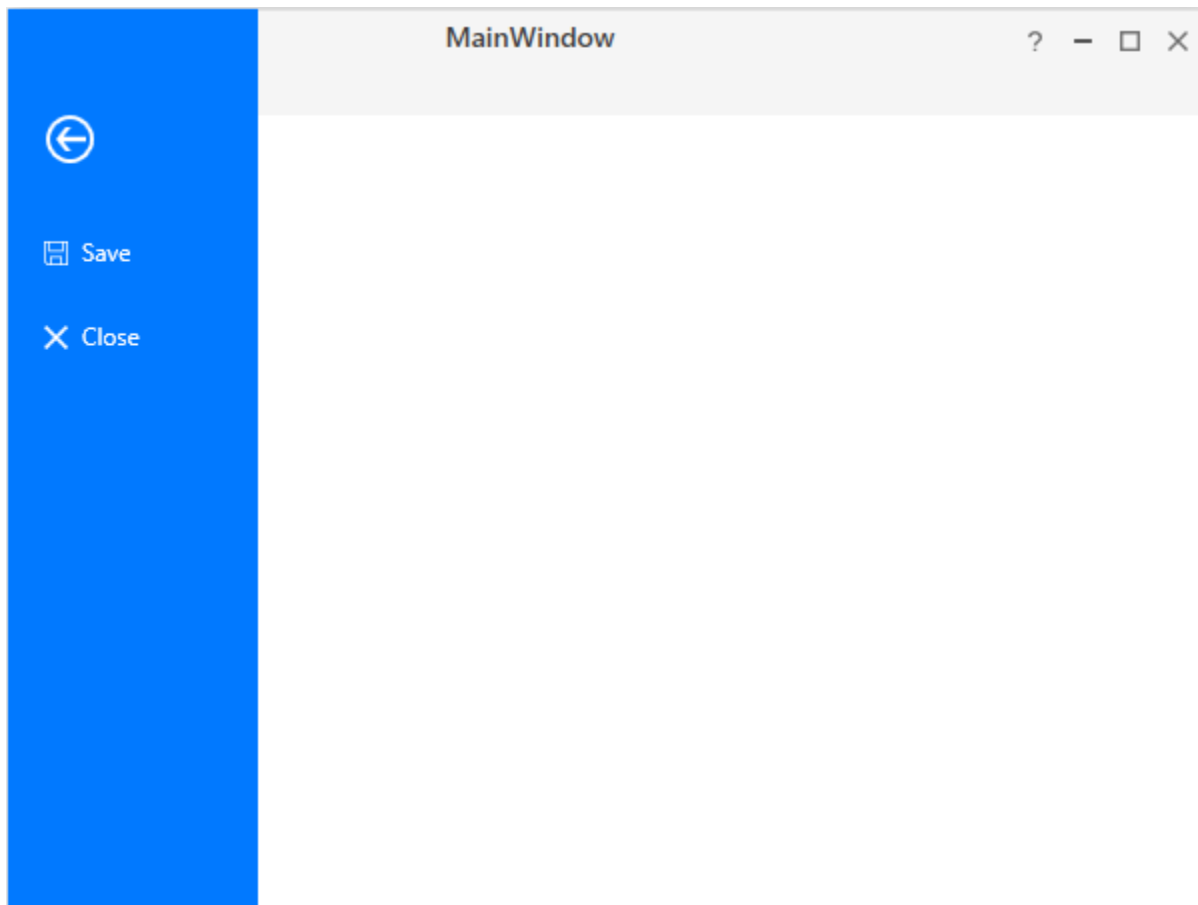
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
Backstage backstage = new Backstage();
BackstageCommandButton backStageCommandButton = new
BackstageCommandButton();
backStageCommandButton.Header = "Save";
DataTemplate iconDataTemplate = new DataTemplate();
FrameworkElementFactory gridElement = new
FrameworkElementFactory(typeof(Grid));
FrameworkElementFactory pathElement1 = new
FrameworkElementFactory(typeof(Path));
pathElement1.SetValue(Path.DataProperty, Geometry.Parse("M5.0000019,11
L5.0000019,15 11.000002,15 11.000002,11 z M4.0000019,1 L4.0000019,6
12.000002,6 12.000002,1 z M1,1 L1,13.174 2.7160001,15 4.0000019,15
4.0000019,10 12.000002,10 12.000002,15 15,15 15,1 13.000002,1 13.000002,7
3.0000019,7 3.0000019,1 z M0,0 L3.0000019,0 13.000002,0 16,0 16,16
12.000002,16 4.0000019,16 2.2840004,16 0,13.57 z"));
pathElement1.SetValue(Path.HeightProperty, (double)12);
pathElement1.SetValue(Path.WidthProperty, (double)12);
pathElement1.SetValue(Path.FillProperty, new SolidColorBrush(Colors.White));
pathElement1.SetValue(Path.StretchProperty, Stretch.Fill);
gridElement.AppendChild(pathElement1);
iconDataTemplate.VisualTree = gridElement;
backStageCommandButton.IconTemplate = iconDataTemplate;
backstage.Items.Add(backStageCommandButton);
BackstageCommandButton backStageCommandButton2 = new
BackstageCommandButton();
backStageCommandButton2.Header = "Close";
DataTemplate iconDataTemplate2 = new DataTemplate();
FrameworkElementFactory gridElement2 = new
FrameworkElementFactory(typeof(Grid));
FrameworkElementFactory pathElement2 = new
FrameworkElementFactory(typeof(Path));
gridElement2.SetValue(Grid.WidthProperty, (double)12);
gridElement2.SetValue(Grid.HeightProperty, (double)12);
pathElement2.SetValue(Path.DataProperty,
Geometry.Parse("M1.4139423,0L7.0029922,5.5845888 12.592018,0

```

```

14.006015,1.4149939 8.4180527,6.9985202 14.006,12.582007 12.591996,13.997001
7.0030056,8.4124444 1.4140122,13.997001 1.5026823E-05,12.582007
5.5879484,6.9985092 0,1.4149939z")));
pathElement2.SetValue(Path.HeightProperty, (double)12);
pathElement2.SetValue(Path.WidthProperty, (double)12);
pathElement2.SetValue(Path.FillProperty, new SolidColorBrush(Colors.White));
pathElement2.SetValue(Path.StretchProperty, Stretch.Fill);
gridElement2.AppendChild(pathElement2);
iconDataTemplate2.VisualTree = gridElement2;
backStageCommandButton2.IconTemplate = iconDataTemplate2;
backstage.Items.Add(backStageCommandButton2);
//Setting backstage to ribbon
ribbon.BackStage = backstage;
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Note: [View sample in GitHub](#)

Setting image path

The [BackStageCommandButton](#) allows to set the image using its [Icon](#) property. The [BackStageCommandButton](#) displays the [Icon](#) in 16 * 16 size.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
```

```

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStyle="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>
<syncfusion:BackStageCommandButton Header="Save" Icon="Resources/Save.png"
/>
<syncfusion:BackStageCommandButton Header="Close"
Icon="Resources/Close.png" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Clipboard">
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

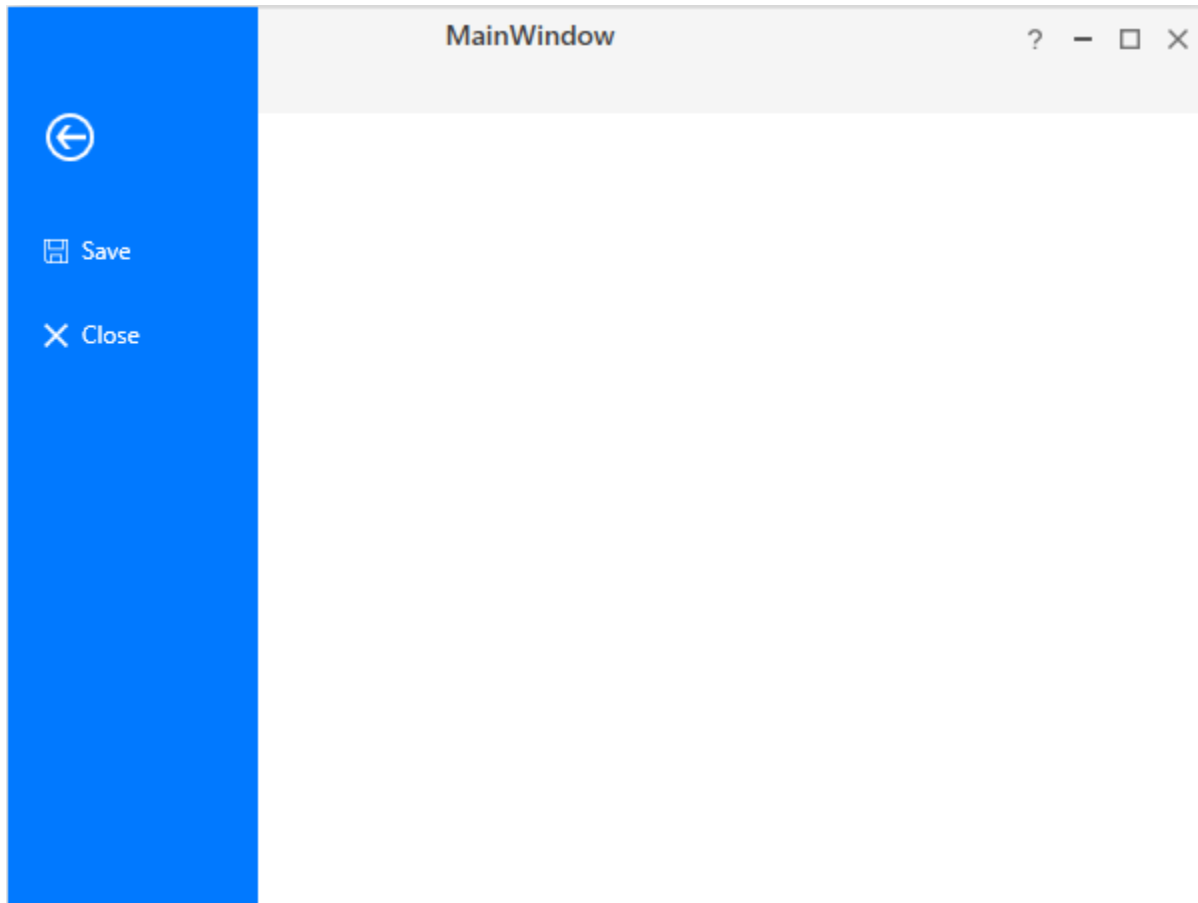
C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
Backstage backstage = new Backstage();
BackStageCommandButton backStageCommandButton = new
BackStageCommandButton();
backStageCommandButton.Header = "Save";
backStageCommandButton.Icon = new BitmapImage(new
Uri(@"Resources/Save.png", UriKind.RelativeOrAbsolute));
backstage.Items.Add(backStageCommandButton);
BackStageCommandButton backStageCommandButton2 = new
BackStageCommandButton();
backStageCommandButton2.Header = "Close";
backStageCommandButton2.Icon = new BitmapImage(new
Uri(@"Resources/Close.png", UriKind.RelativeOrAbsolute));
backstage.Items.Add(backStageCommandButton2);
//Setting backstage to ribbon
ribbon.BackStage = backstage;
ribbon.Items.Add(homeTab);

```

```
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
```



Setting vector image

The [VectorImage](#) property is of type `ObservableCollection<Path>` which allows the image to be set as path type. The [BackStageCommandButton](#) displays the [VectorImage](#) in 16 * 16 size.

Note: The [IconTemplate](#) property is the preferred option to set any type of image such as glyph, font or any custom content when compared to the [VectorImage](#) property.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStyle="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<Grid x:Name="grid">
```

```

<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>
<syncfusion:BackStageCommandButton Header="Save" IconType="VectorImage" >
<syncfusion:BackStageCommandButton.VectorImage>
<Path Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M5.0000019,11 L5.0000019,15 11.000002,15 11.000002,11 z M4.0000019,1
L4.0000019,6 12.000002,6 12.000002,1 z M1,1 L1,13.174 2.7160001,15
4.0000019,15 4.0000019,10 12.000002,10 12.000002,15 15,15 15,1 13.000002,1
13.000002,7 3.0000019,7 3.0000019,1 z M0,0 L3.0000019,0 13.000002,0 16,0
16,16 12.000002,16 4.0000019,16 2.2840004,16 0,13.57 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" Stretch="Uniform" />
</syncfusion:BackStageCommandButton.VectorImage>
</syncfusion:BackStageCommandButton>
<syncfusion:BackStageCommandButton Header="Close" IconType="VectorImage">
<syncfusion:BackStageCommandButton.VectorImage>
<Path
Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M1.4139423,0L7.0029922,5.5845888 12.592018,0 14.006015,1.4149939
8.4180527,6.9985202 14.006,12.582007 12.591996,13.997001 7.0030056,8.4124444
1.4140122,13.997001 1.5026823E-05,12.582007 5.5879484,6.9985092
0,1.4149939z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
SnapsToDevicePixels="True" Stretch="Fill" />
</syncfusion:BackStageCommandButton.VectorImage>
</syncfusion:BackStageCommandButton>
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Clipboard">
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

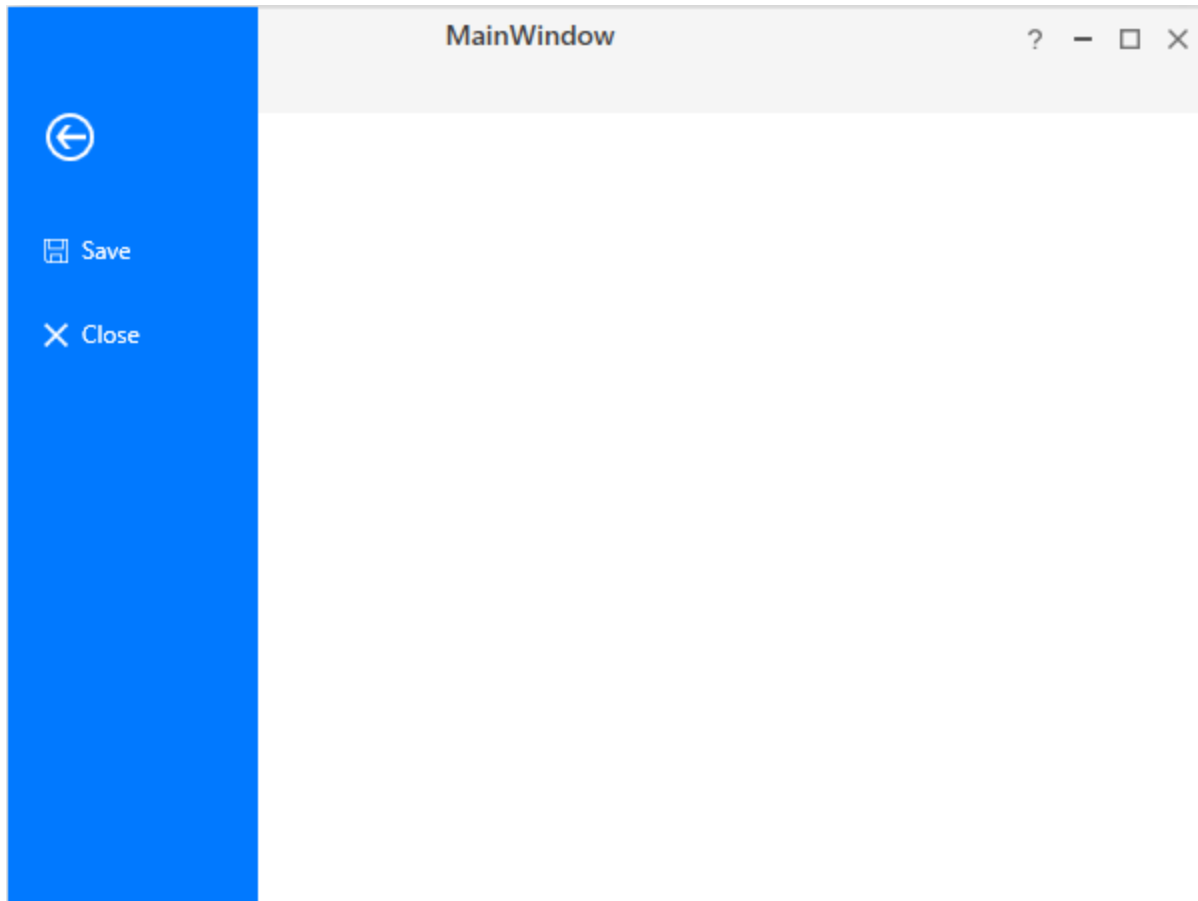
C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
Backstage backstage = new Backstage();
BackStageCommandButton backStageCommandButton = new
BackStageCommandButton();
backStageCommandButton.Header = "Save";

```

```
Path savePath1 = new Path();
savePath1.Data = Geometry.Parse("M5.0000019,11 L5.0000019,15 11.000002,15
11.000002,11 z M4.0000019,1 L4.0000019,6 12.000002,6 12.000002,1 z M1,1
L1,13.174 2.7160001,15 4.0000019,15 4.0000019,10 12.000002,10 12.000002,15
15,15 15,1 13.000002,1 13.000002,7 3.0000019,7 3.0000019,1 z M0,0
L3.0000019,0 13.000002,0 16,0 16,16 12.000002,16 4.0000019,16 2.2840004,16
0,13.57 z");
savePath1.Height = 12;
savePath1.Width = 12;
savePath1.Fill = new SolidColorBrush(Colors.White);
savePath1.Stretch = Stretch.Fill;
backStageCommandButton.VectorImage.Add(savePath1);
backstage.Items.Add(backStageCommandButton);
BackStageCommandButton backStageCommandButton2 = new
BackStageCommandButton();
backStageCommandButton2.Header = "Close";
Path closePath1 = new Path();
closePath1.Data = Geometry.Parse("M1.4139423,0L7.0029922,5.5845888
12.592018,0 14.006015,1.4149939 8.4180527,6.9985202 14.006,12.582007
12.591996,13.997001 7.0030056,8.4124444 1.4140122,13.997001 1.5026823E-
05,12.582007 5.5879484,6.9985092 0,1.4149939z");
closePath1.Height = 12;
closePath1.Width = 12;
closePath1.Fill = new SolidColorBrush(Colors.White);
closePath1.Stretch = Stretch.Fill;
backStageCommandButton2.VectorImage.Add(closePath1);
backstage.Items.Add(backStageCommandButton2);
//Setting backstage to ribbon
ribbon.BackStage = backstage;
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
```

Add BackStageTabItem

The `BackStageTabItem` can also be added inside the BackStage Element. Here four `BackStageCommandButtons` are added with `Header` property value as "Open", "Print" and "Office Account".

XML

```
<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStudio="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
```

```

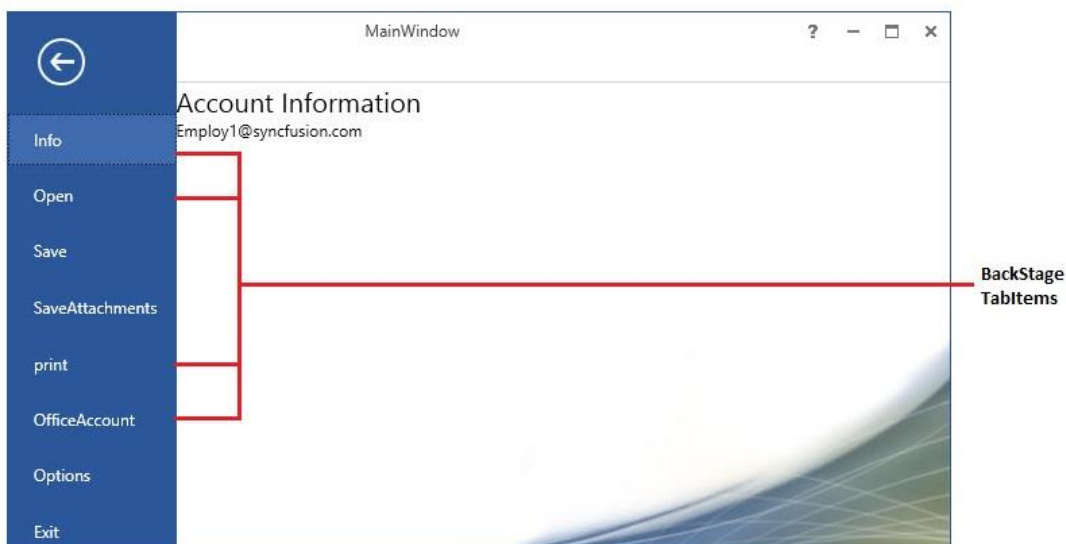
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="20" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" />
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Send" />
<syncfusion:RibbonButton Label="Forward" />
<syncfusion:RibbonButton Label="ReplyAll" />
<syncfusion:RibbonButton Label="Delete" />
<syncfusion:RibbonButton Label="Print" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>

```

```

<syncfusion:BackstageTabItem Header="Info">
<StackPanel>
<TextBlock FontSize="20" Text="Account Information"/>
<TextBlock FontSize="12" Text="Employ1@syncfusion.com"/>
</StackPanel>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem Header="Open"/>
<syncfusion:BackstageCommandButton Header="Save" />
<syncfusion:BackstageCommandButton Header="SaveAttachments" />
<syncfusion:BackstageTabItem Header="print"/>
<syncfusion:BackstageTabItem Header="OfficeAccount"/>
<syncfusion:BackstageCommandButton Header="Options" />
<syncfusion:BackstageCommandButton Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Add BackStage separator

The BackStageSeparator is used to group out similar backstage elements in BackStage. Here BackStageCommandButtons, BackStageTabItems are separated by [BackStageSeparator](#) according to their use.

XML

```

<syncfusion:RibbonWindow
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="RibbonControl.MainWindow"
xmlns:syncfusion:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"

```

```

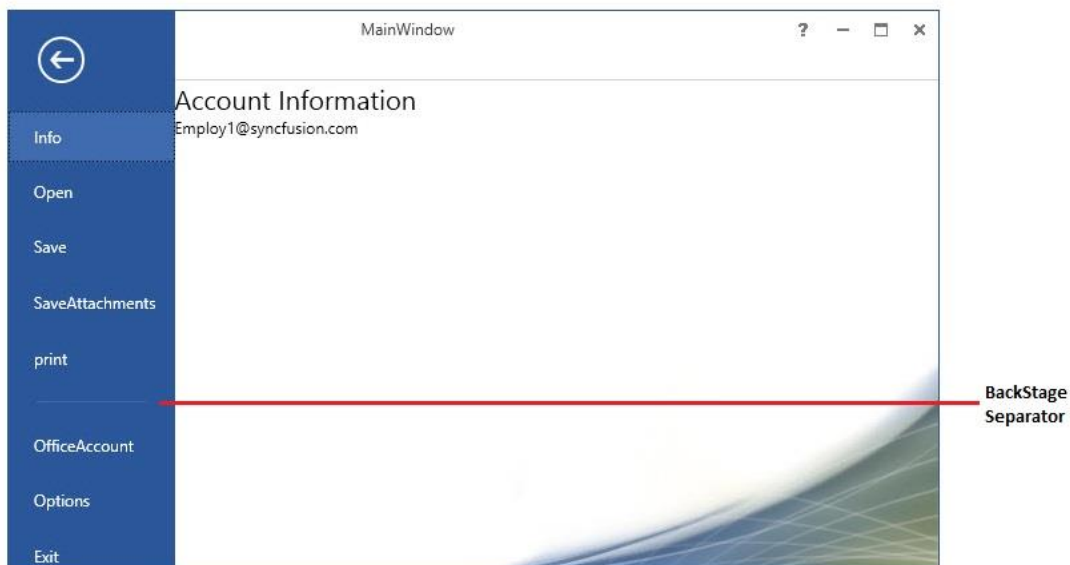
syncfusionskin:SfSkinManager.VisualStyle="Office2013White" >
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="20" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall"/>

```

```

<syncfusion:RibbonButton SizeForm="ExtraSmall" />
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Send" />
<syncfusion:RibbonButton Label="Forward" />
<syncfusion:RibbonButton Label="ReplyAll" />
<syncfusion:RibbonButton Label="Delete" />
<syncfusion:RibbonButton Label="Print" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>
<syncfusion:BackstageTabItem Header="Info">
<StackPanel>
<TextBlock FontSize="20" Text="Account Information"/>
<TextBlock FontSize="12" Text="Employ1@syncfusion.com"/>
</StackPanel>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem Header="Open"/>
<syncfusion:BackStageCommandButton Header="Save" />
<syncfusion:BackStageCommandButton Header="SaveAttachments" />
<syncfusion:BackstageTabItem Header="print"/>
<syncfusion:BackStageSeparator/>
<syncfusion:BackstageTabItem Header="OfficeAccount"/>
<syncfusion:BackStageCommandButton Header="Options" />
<syncfusion:BackStageCommandButton Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



BackStage items position

The **BackStage** items listed below can be positioned either at top or bottom by using the [Position](#) property.

- [BackStageCommandButton](#)
- [BackStageTabItem](#)
- [BackStageSeparator](#)

The following code example illustrates how to position the **BackStage** items either at top or bottom.

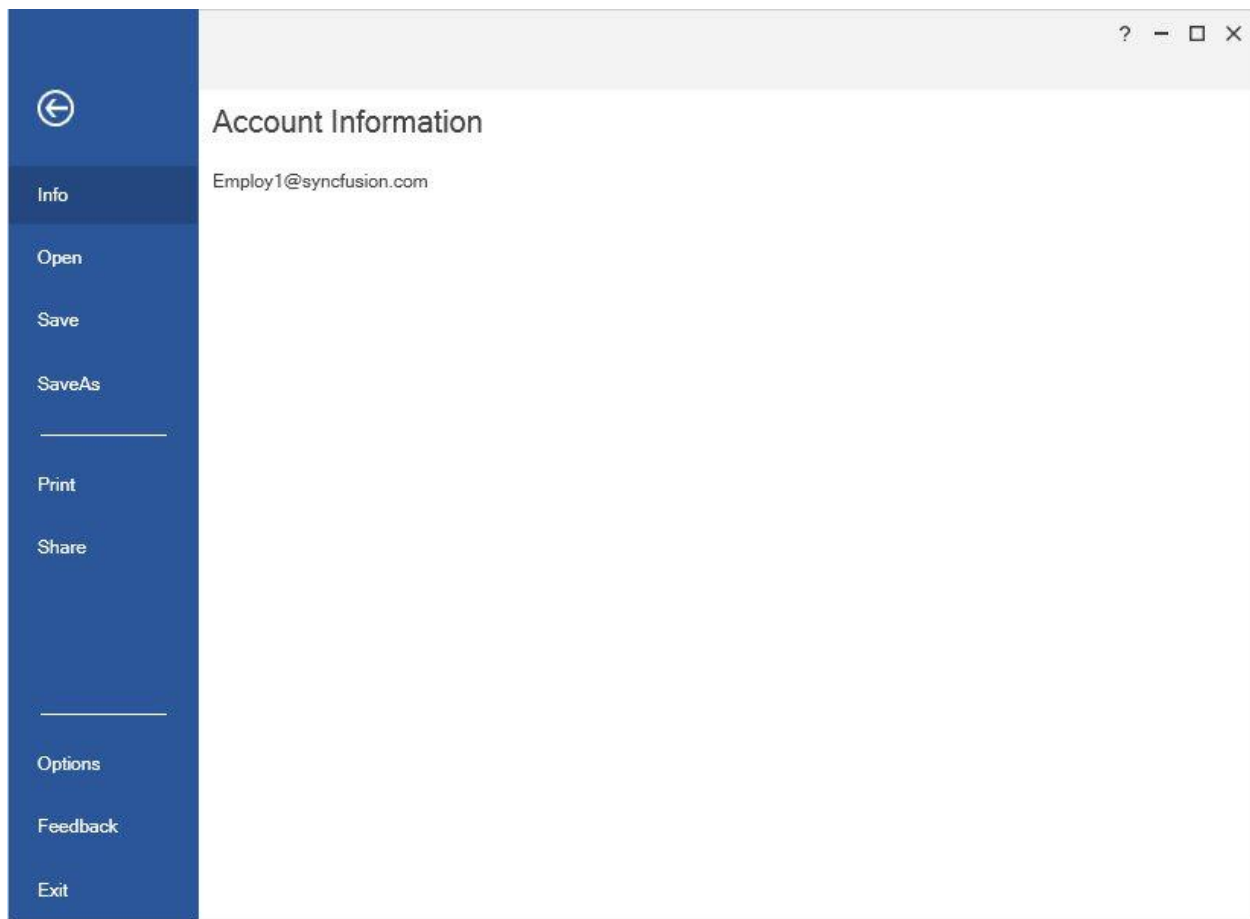
XML

```
<syncfusion:RibbonWindow
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="BackStage.MainWindow"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStudio="Office2019Colorful" Width="820"
Height="600" WindowStartupLocation="CenterScreen">
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top"
IsBackStageVisible="True">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="New" Width="90" Header="New">
<syncfusion:RibbonButton SizeForm="Large" Label="New Email"/>
<syncfusion:DropDownButton SizeForm="Large" Label="New Items">
<syncfusion:DropDownMenuItem Header="E-mail Message"/>
<syncfusion:DropDownMenuItem Header="Appointment"/>
<syncfusion:DropDownMenuItem Header="Meeting"/>
<syncfusion:DropDownMenuItem Header="Contact"/>
<syncfusion:DropDownMenuItem Header="Task"/>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Width="130" Header="Delete">
<syncfusion:RibbonButton Label="Ignore"/>
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder"/>
<syncfusion:DropDownMenuItem Header="Clean Up Conversation"/>
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Junk" Margin="0,0,12,0" Width="76"/>
<syncfusion:RibbonButton Label="Delete" SizeForm="Large"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Respond" Width="200" Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Meeting"/>
<syncfusion:SplitButton Label="IM" Margin="-2,0,6,0" Width="68"/>
<syncfusion:SplitButton Label="More" Margin="-2,0,6,0" Width="68"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Name="QuickSteps" Header="Quick Steps">
```

```

<syncfusion:RibbonGallery Width="160" VisualMode="InRibbon"
ItemHeight="20" ItemWidth="70">
<syncfusion:RibbonGalleryItem Content="Move to?"/>
<syncfusion:RibbonGalleryItem Content="Team Email"/>
<syncfusion:RibbonGalleryItem Content="ReplyDelete"/>
<syncfusion:RibbonGalleryItem Content="To Manager"/>
<syncfusion:RibbonGalleryItem Content="Done"/>
<syncfusion:RibbonGalleryItem Content="Create New"/>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Width="170" Header="Find" Name="Find" >
<syncfusion:RibbonComboBox Label="Filter Email" Width="160">
<ComboBoxItem>Person1@mail.com</ComboBoxItem>
<ComboBoxItem>Person2@mail.com</ComboBoxItem>
<ComboBoxItem>Person3@mail.com</ComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" />
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Send" />
<syncfusion:RibbonButton Label="Forward" />
<syncfusion:RibbonButton Label="ReplyAll" />
<syncfusion:RibbonButton Label="Delete" />
<syncfusion:RibbonButton Label="Print" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>
<syncfusion:BackstageTabItem Header="Info">
<StackPanel>
<TextBlock FontSize="20" Margin="10" Text="Account Information"/>
<TextBlock FontSize="12" Margin="10" Text="Employ1@syncfusion.com"/>
</StackPanel>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem Header="Open"/>
<syncfusion:BackstageCommandButton Header="Save" />
<syncfusion:BackstageCommandButton Header="SaveAs" />
<syncfusion:BackstageSeparator/>
<syncfusion:BackstageTabItem Header="Print"/>
<syncfusion:BackstageTabItem Header="Share"/>
<syncfusion:BackstageSeparator Position="Bottom"/>
<syncfusion:BackstageTabItem Position="Bottom" Header="Options"/>
<syncfusion:BackstageCommandButton Position="Bottom" Header="Feedback" />
<syncfusion:BackstageCommandButton Position="Bottom" Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

Note: [View sample in GitHub](#)

Different types of Animation

Backstage provides different types of animations such as fade, scale, and zoom by using the [AnimationType](#) enumeration property.

- None
- Slide
- Fade
- Zoom

Animation duration

The [AnimationDuration](#) property is used to set the duration for animation in milliseconds.

The following code example illustrates how to animate the BackStage using the [AnimationType](#) property.

XML

```
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage x:Name="ribbonBackStage"
AnimationDuration="00:00:00.250" syncfusion:Ribbon.KeyTip="B"
AnimationType="{Binding
```



```

ElementName=AnimationTypes,Path=SelectedItem,UpdateSourceTrigger=PropertyChanged}"/>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="I" Header="Info">
<Grid Background="{Binding ElementName=mainRibbon,Path=Background}">
<Grid.RowDefinitions>
<RowDefinition Height="auto" />
<RowDefinition Height="auto" />
</Grid.RowDefinitions>
<local:Information x:Name="informationView" />
</Grid>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="R" Header="Recent">
<local:Recent x:Name="recentView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="N" Header="New">
<local:New x:Name="newView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageSeparator />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="A"
Command="{Binding SaveAsCommand}"
Header="Save As"
IconTemplate="{StaticResource Save}" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="E"
Command="{Binding OpenCommand}"
Header="Open"
Icon="/BackStage;component/Assets/Ribbon/Open32.png" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="C"
Command="{Binding CloseCommand}"
Header="Close"
IconTemplate="{StaticResource CloseTab}" />
<syncfusion:BackStageSeparator />
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="P" Header="Print">
<local:Print x:Name="printView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageSeparator Position="Bottom"/>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="H" Position="Bottom"
Header="Help">
<local:Help x:Name="helpView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageCommandButton
VerticalAlignment="Bottom" Position="Bottom"
syncfusion:Ribbon.KeyTip="X"
Command="{Binding ExitCommand}"
CommandParameter="{Binding ElementName=mainWindow}"
Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>

```

C#

```
ribbon.BackStage.AnimationDuration = TimeSpan.FromMilliseconds(250);
```

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

Placement Customization

The Backstage can be opened or closed within any window or placement target specified using the [PlacementType](#) and [PlacementTarget](#) properties in BackStage. It allows us to open the Backstage under the Ribbon tab or occupies the window or placement target's entire client area.

The [PlacementTarget](#) property specifies the element relative to which the Backstage should be positioned while it is opened.

The [PlacementType](#) property provides the following options which allows to place the backstage either in full screen or below the RibbonTab.

- FullScreen
- BelowTab

Place the backstage in FullScreen

Backstage placed in RibbonWindow

When the [PlacementType](#) is set to [FullScreen](#) and the [PlacementTarget](#) is not set, the Backstage will occupy the entire RibbonWindow while it is opened.

XML

```
<syncfusion:RibbonWindow
x:Class="BackStage.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:interactivity="http://schemas.microsoft.com/xaml/behaviors"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:system="clr-namespace:System;assembly=microsoft.windows.common-base-1.0"
xmlns:local="clr-namespace:BackStage"
xmlns:tools="clr-
namespace:Syncfusion.Windows.Tools;assembly=Syncfusion.Tools.Wpf"
Width="1100"
Height="700"
Title="Backstage Placement Demo"
syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
ThemeName=Office2019Colorful}"
WindowStartupLocation="CenterScreen">
<syncfusion:RibbonWindow.Resources>
<syncfusion:ColorToBrushConverter x:Key="ColorToBrushConverter" />
</syncfusion:RibbonWindow.Resources>
<syncfusion:RibbonWindow.DataContext>
<local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
<Grid>
<Grid.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
```

```

<ResourceDictionary
Source="/BackStage;component/Assets/Ribbon/PathIcon.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Grid.Resources>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid>
<syncfusion:Ribbon
Name="mainRibbon" local:ViewModel.Ribbon="{Binding ElementName=mainRibbon}"
IsBackStageVisible="True"
BackStageColor="{Binding ElementName=backColor, Path=Color, Mode=OneWay,
Converter={StaticResource ColorToBrushConverter}}"
BackStageHeader="File">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Save}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Bold}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Copy}"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage x:Name="ribbonBackStage" PlacementType="FullScreen"
syncfusion:Ribbon.KeyTip="B" >
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="I" Header="Info">
<Grid Background="{Binding ElementName=mainRibbon, Path=Background}">
<Grid.RowDefinitions>
<RowDefinition Height="auto" />
<RowDefinition Height="auto" />
</Grid.RowDefinitions>
<local:Information x:Name="informationView" />
</Grid>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="R" Header="Recent">
<local:Recent x:Name="recentView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="N" Header="New">
<local:New x:Name="newView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackStageSeparator />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="A" Command="{Binding SaveAsCommand}"
Header="Save As" IconTemplate="{StaticResource Save}"/>
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="E" Command="{Binding OpenCommand}"
Header="Open" Icon="/BackStage;component/Assets/Ribbon/Open32.png" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="C" Command="{Binding CloseCommand}"
Header="Close" IconTemplate="{StaticResource CloseTab}"/>
<syncfusion:BackStageSeparator />

```

```

<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="P" Header="Print">
<local:Print x:Name="printView" Background="{Binding
ElementName=mainRibbon,Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageSeparator Position="Bottom"/>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="H" Position="Bottom"
Header="Help">
<local:Help x:Name="helpView" Background="{Binding
ElementName=mainRibbon,Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageCommandButton
VerticalAlignment="Bottom" Position="Bottom"
syncfusion:Ribbon.KeyTip="X"
Command="{Binding ExitCommand}"
CommandParameter="{Binding ElementName=mainWindow}"
Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Name="ribbonFormatBar" syncfusion:Ribbon.KeyTip="FN"
Header="Clipboard">
<syncfusion:RibbonButton
Margin="1" Command="ApplicationCommands.Paste" Label="Paste"
SizeForm="Large" IconTemplate="{StaticResource Paste}"/>
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Cut" Label="Cut"
SizeForm="Small" IconTemplate="{StaticResource Cut}"/>
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Copy" Label="Copy"
SizeForm="Small" IconTemplate="{StaticResource Copy}"/>
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Format Painter"
SizeForm="Small" IconTemplate="{StaticResource FormatPainter}"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="barFont" Header="Font"
IsLargeButtonPanel="False">
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox
Name="ribbonFontName" Width="110" syncfusion:Ribbon.KeyTip="FF"
DisplayMemberPath="FontFamily"
IsEditable="True" ItemsSource="{Binding FontFamilyList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"
CommandParameter="{Binding ElementName=ribbonFontName, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox
Name="ribbonFontSize" Width="40" DisplayMemberPath="FontSize"
IsEditable="True" ItemsSource="{Binding FontSizeList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"

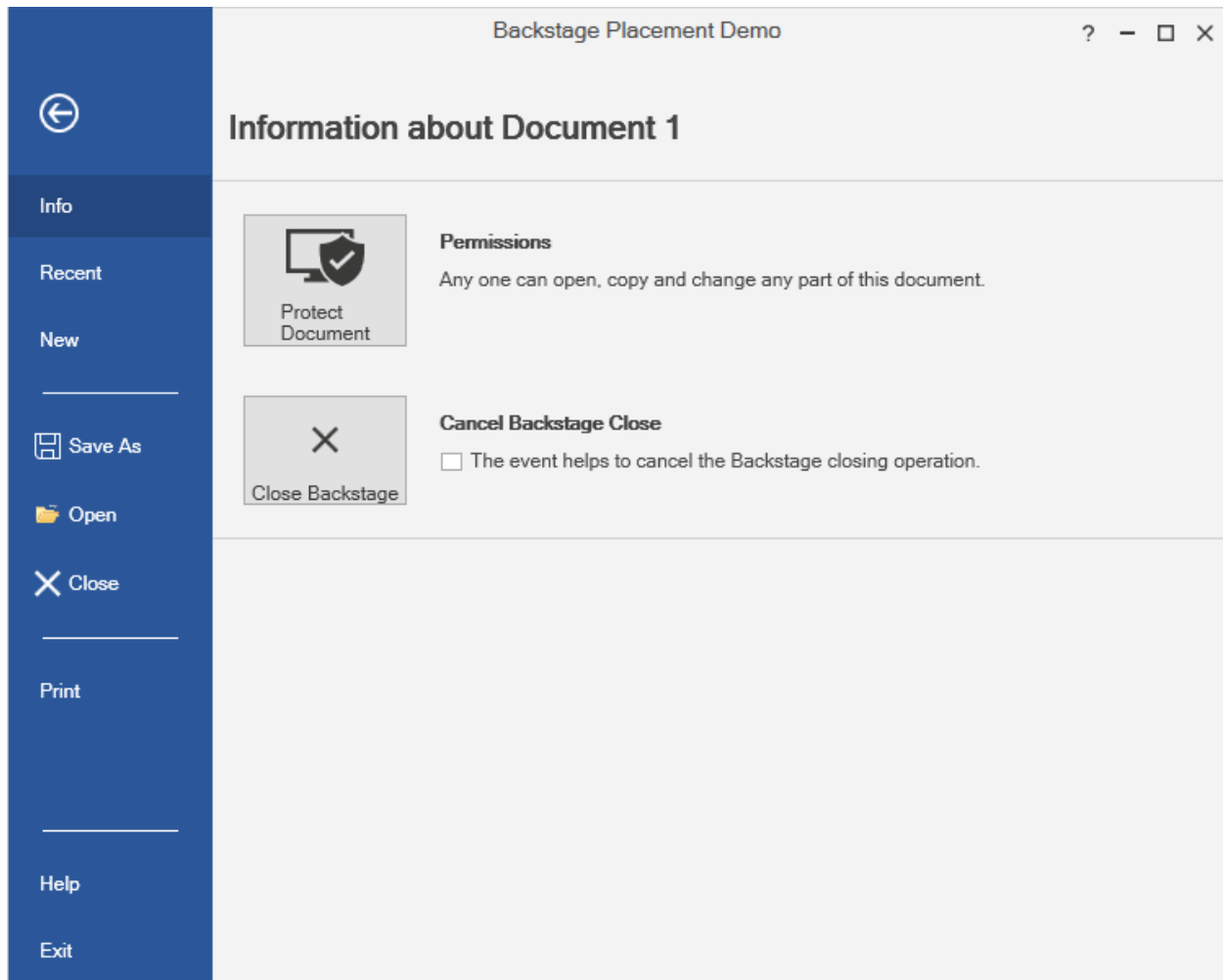
```

```

CommandParameter="{Binding ElementName=ribbonFontSize, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24">
<syncfusion:RibbonButton
Name="ribbonIndentButton" Command="EditingCommands.IncreaseFontSize"
SizeForm="ExtraSmall" IconTemplate="{StaticResource IncreaseFontSize}"/>
<syncfusion:RibbonButton
Command="EditingCommands.DecreaseFontSize" SizeForm="ExtraSmall"
IconTemplate="{StaticResource DecreaseFontSize}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="ribbonBoldButton" Command="EditingCommands.ToggleBold" IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Bold}"/>
<syncfusion:RibbonButton
Name="ribbonItalicButton" Command="EditingCommands.ToggleItalic"
IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Italics}"/>
<syncfusion:RibbonButton
Name="ribbonUnderlineButton" Command="EditingCommands.ToggleUnderline"
IsToggle="True" SizeForm="ExtraSmall" IconTemplate="{StaticResource
Underline}"/>
<syncfusion:RibbonButton
Name="strikeButton" Command="local:RibbonCommand.ButtonCommand"
SizeForm="ExtraSmall" IconTemplate="{StaticResource ClearFormatting}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
FontColor}">
<syncfusion:ColorPickerPalette x:Name="fontColorPicker" Color="Black"
IsExpanded="True" />
</syncfusion:SplitButton>
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
TextHighlight}">
<syncfusion:ColorPickerPalette x:Name="highlightColorPicker"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatShading" syncfusion:Ribbon.KeyTip="FS" SizeForm="ExtraSmall"
IconTemplate="{StaticResource Shading}">
<syncfusion:ColorPickerPalette x:Name="shadingColorPicker" Margin="3"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatBorder" syncfusion:Ribbon.KeyTip="BF" SizeForm="ExtraSmall"
IconTemplate="{StaticResource FormatBorder}">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.DropDownCommand" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>

```

```
<ListBoxItem Margin="5">Full Border</ListBoxItem>
<ListBoxItem Margin="5">Half Border</ListBoxItem>
<ListBoxItem Margin="5">Inside Border</ListBoxItem>
<ListBoxItem Margin="5">Outside Border</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert">
<syncfusion:RibbonBar Header="Pages">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Cover Page"
LargeIcon="/BackStage;component/Assets/Ribbon/CoverPage32.png"
SizeForm="Large" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="BlankPage"
SizeForm="Large"
IconTemplate="{StaticResource BlankPage}" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="PageBreak"
IconTemplate="{StaticResource PageBreak}" SizeForm="Large" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Illustrations">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Picture"
SizeForm="Large" IconTemplate="{StaticResource Picture}" />
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
<Grid x:Name="backstageGrid" Margin="10" Grid.Row="1"/>
</Grid>
</syncfusion:RibbonWindow>
```



Backstage placed in MS Window

When the [PlacementType](#) is set to [FullScreen](#) and the [PlacementTarget](#) is not set, the Backstage will occupy the entire MS Window while it is opened.

XML

```
<Window
x:Class="BackStage.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:interactivity="http://schemas.microsoft.com/xaml/behaviors"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:system="clr-namespace:System;assembly=microsoft.windows.common-base-1.0"
xmlns:local="clr-namespace:BackStage"
xmlns:tools="clr-
namespace:Syncfusion.Windows.Tools;assembly=Syncfusion.Tools.Wpf"
Width="1100"
Height="700"
Title="Backstage Placement Demo"
syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
ThemeName=Office2019Colorful}"
```

```

WindowStartupLocation="CenterScreen">
<Window.Resources>
<syncfusion:ColorToBrushConverter x:Key="ColorToBrushConverter" />
</Window.Resources>
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
<Grid.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/BackStage;component/Assets/Ribbon/PathIcon.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Grid.Resources>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid>
<syncfusion:Ribbon
Name="mainRibbon" local:ViewModel.Ribbon="{Binding ElementName=mainRibbon}"
IsBackStageVisible="True"
BackStageColor="{Binding ElementName=backColor, Path=Color, Mode=OneWay,
Converter={StaticResource ColorToBrushConverter}}"
BackStageHeader="File">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Save}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Bold}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Copy}"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage x:Name="ribbonBackStage" PlacementType="FullScreen"
syncfusion:Ribbon.KeyTip="B" >
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="I" Header="Info">
<Grid Background="{Binding ElementName=mainRibbon, Path=Background}">
<Grid.RowDefinitions>
<RowDefinition Height="auto" />
<RowDefinition Height="auto" />
</Grid.RowDefinitions>
<local:Information x:Name="informationView" />
</Grid>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="R" Header="Recent">
<local:Recent x:Name="recentView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="N" Header="New">
<local:New x:Name="newView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>

```



```

<syncfusion:BackStageSeparator />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="A" Command="{Binding SaveAsCommand}"
Header="Save As" IconTemplate="{StaticResource Save}" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="E" Command="{Binding OpenCommand}"
Header="Open" Icon="/BackStage;component/Assets/Ribbon/Open32.png" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="C" Command="{Binding CloseCommand}"
Header="Close" IconTemplate="{StaticResource CloseTab}" />
<syncfusion:BackStageSeparator />
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="P" Header="Print">
<local:Print x:Name="printView" Background="{Binding
ElementName=mainRibbon, Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageSeparator Position="Bottom"/>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="H" Position="Bottom"
Header="Help">
<local:Help x:Name="helpView" Background="{Binding
ElementName=mainRibbon, Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageCommandButton
VerticalAlignment="Bottom" Position="Bottom"
syncfusion:Ribbon.KeyTip="X"
Command="{Binding ExitCommand}"
CommandParameter="{Binding ElementName=mainWindow}"
Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Name="ribbonFormatBar" syncfusion:Ribbon.KeyTip="FN"
Header="Clipboard">
<syncfusion:RibbonButton
Margin="1" Command="ApplicationCommands.Paste" Label="Paste"
SizeForm="Large" IconTemplate="{StaticResource Paste}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Cut" Label="Cut"
SizeForm="Small" IconTemplate="{StaticResource Cut}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Copy" Label="Copy"
SizeForm="Small" IconTemplate="{StaticResource Copy}" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Format Painter"
SizeForm="Small" IconTemplate="{StaticResource FormatPainter}" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="barFont" Header="Font"
IsLargeButtonPanel="False">
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox
Name="ribbonFontName" Width="110" syncfusion:Ribbon.KeyTip="FF"
DisplayMemberPath="FontFamily"
IsEditable="True" ItemsSource="{Binding FontFamilyList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"

```

```

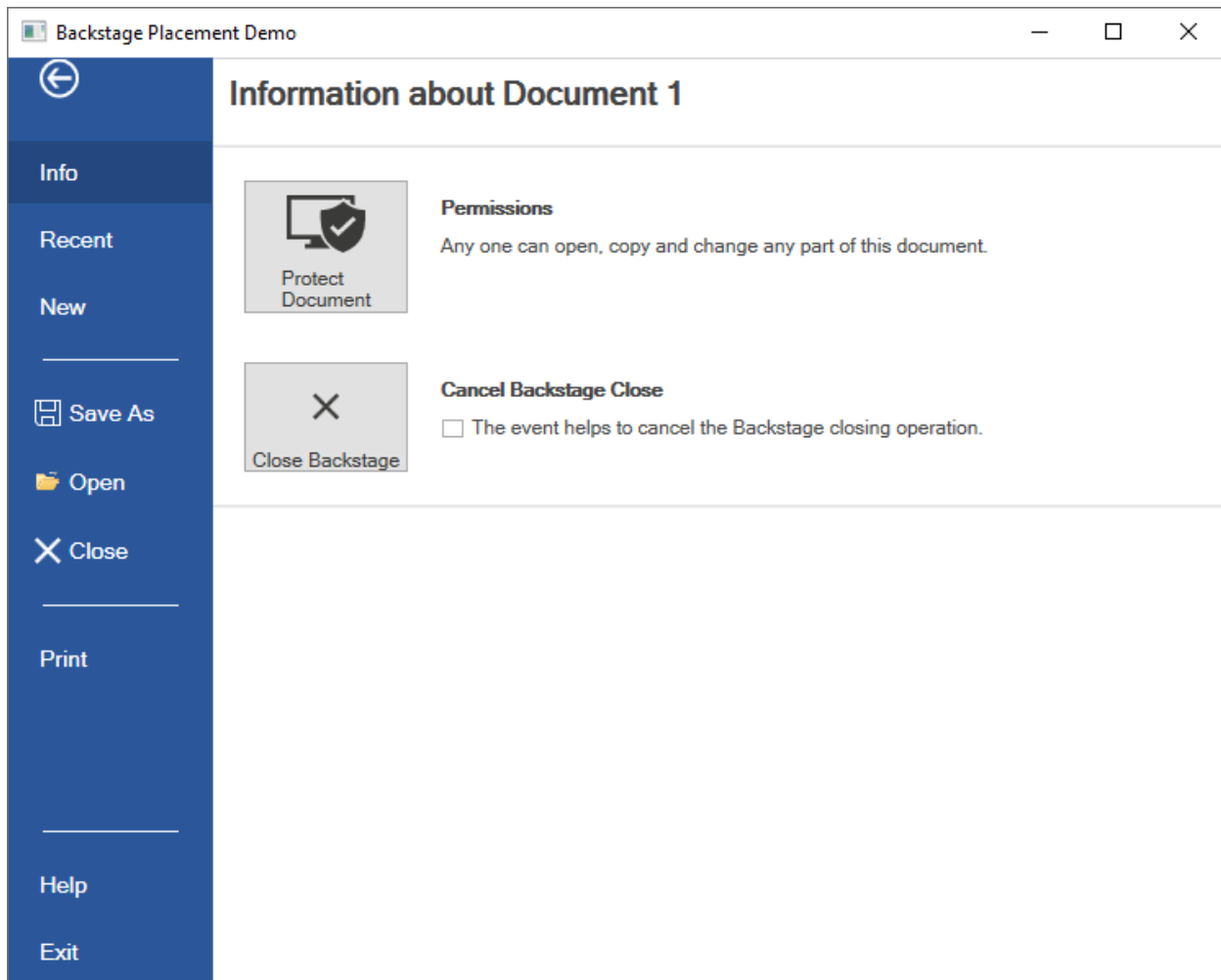
CommandParameter="{Binding ElementName=ribbonFontName, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox
Name="ribbonFontSize" Width="40" DisplayMemberPath="FontSize"
IsEditable="True" ItemsSource="{Binding FontSizeList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"
CommandParameter="{Binding ElementName=ribbonFontSize, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24">
<syncfusion:RibbonButton
Name="ribbonIndentButton" Command="EditingCommands.IncreaseFontSize"
SizeForm="ExtraSmall" IconTemplate="{StaticResource IncreaseFontSize}"/>
<syncfusion:RibbonButton
Command="EditingCommands.DecreaseFontSize" SizeForm="ExtraSmall"
IconTemplate="{StaticResource DecreaseFontSize}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="ribbonBoldButton" Command="EditingCommands.ToggleBold" IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Bold}"/>
<syncfusion:RibbonButton
Name="ribbonItalicButton" Command="EditingCommands.ToggleItalic"
IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Italics}"/>
<syncfusion:RibbonButton
Name="ribbonUnderlineButton" Command="EditingCommands.ToggleUnderline"
IsToggle="True" SizeForm="ExtraSmall" IconTemplate="{StaticResource
Underline}"/>
<syncfusion:RibbonButton
Name="strikeButton" Command="local:RibbonCommand.ButtonCommand"
SizeForm="ExtraSmall" IconTemplate="{StaticResource ClearFormatting}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
FontColor}">
<syncfusion:ColorPickerPalette x:Name="fontColorPicker" Color="Black"
IsExpanded="True" />
</syncfusion:SplitButton>
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
TextHighlight}">
<syncfusion:ColorPickerPalette x:Name="highlightColorPicker"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatShading" syncfusion:Ribbon.KeyTip="FS" SizeForm="ExtraSmall"
IconTemplate="{StaticResource Shading}">

```

```

<syncfusion:ColorPickerPalette x:Name="shadingColorPicker" Margin="3"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatBorder" syncfusion:Ribbon.KeyTip="BF" SizeForm="ExtraSmall"
IconTemplate="{StaticResource FormatBorder}">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.DropDownCommand" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem Margin="5">Full Border</ListBoxItem>
<ListBoxItem Margin="5">Half Border</ListBoxItem>
<ListBoxItem Margin="5">Inside Border</ListBoxItem>
<ListBoxItem Margin="5">Outside Border</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert">
<syncfusion:RibbonBar Header="Pages">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Cover Page"
LargeIcon="/BackStage;component/Assets/Ribbon/CoverPage32.png"
SizeForm="Large" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="BlankPage"
SizeForm="Large"
IconTemplate="{StaticResource BlankPage}"/>
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="PageBreak"
IconTemplate="{StaticResource PageBreak}" SizeForm="Large" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Illustrations">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Picture"
SizeForm="Large" IconTemplate="{StaticResource Picture}"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
<Grid x:Name="backstageGrid" Margin="10" Grid.Row="1"/>
</Grid>
</Window>

```



Backstage placed in Placement target

When the [PlacementType](#) is set to [FullScreen](#) and the [PlacementTarget](#) is set to an element, the Backstage will occupy the entire client area of the target element while it is opened.

XML

```
<syncfusion:RibbonWindow
x:Class="BackStage.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:interactivity="http://schemas.microsoft.com/xaml/behaviors"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:system="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:BackStage"
xmlns:tools="clr-
namespace:Syncfusion.Windows.Tools;assembly=Syncfusion.Tools.Wpf"
Width="1100"
Height="700"
Title="Backstage Placement Demo"
syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
ThemeName=Office2019Colorful}"
```

```

WindowStartupLocation="CenterScreen">
<syncfusion:RibbonWindow.Resources>
<syncfusion:ColorToBrushConverter x:Key="ColorToBrushConverter" />
</syncfusion:RibbonWindow.Resources>
<syncfusion:RibbonWindow.DataContext>
<local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
<Grid>
<Grid.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/BackStage;component/Assets/Ribbon/PathIcon.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Grid.Resources>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid>
<syncfusion:Ribbon
Name="mainRibbon" local:ViewModel.Ribbon="{Binding ElementName=mainRibbon}"
IsBackStageVisible="True"
BackStageColor="{Binding ElementName=backColor, Path=Color, Mode=OneWay,
Converter={StaticResource ColorToBrushConverter}}"
BackStageHeader="File">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Save}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Bold}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Copy}"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage x:Name="ribbonBackStage" PlacementType="FullScreen"
syncfusion:Ribbon.KeyTip="B" >
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="I" Header="Info">
<Grid Background="{Binding ElementName=mainRibbon, Path=Background}">
<Grid.RowDefinitions>
<RowDefinition Height="auto" />
<RowDefinition Height="auto" />
</Grid.RowDefinitions>
<local:Information x:Name="informationView" />
</Grid>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="R" Header="Recent">
<local:Recent x:Name="recentView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="N" Header="New">
<local:New x:Name="newView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>

```

```

<syncfusion:BackStageSeparator />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="A" Command="{Binding SaveAsCommand}"
Header="Save As" IconTemplate="{StaticResource Save}" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="E" Command="{Binding OpenCommand}"
Header="Open" Icon="/BackStage;component/Assets/Ribbon/Open32.png" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="C" Command="{Binding CloseCommand}"
Header="Close" IconTemplate="{StaticResource CloseTab}" />
<syncfusion:BackStageSeparator />
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="P" Header="Print">
<local:Print x:Name="printView" Background="{Binding
ElementName=mainRibbon, Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageSeparator Position="Bottom"/>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="H" Position="Bottom"
Header="Help">
<local:Help x:Name="helpView" Background="{Binding
ElementName=mainRibbon, Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageCommandButton
VerticalAlignment="Bottom" Position="Bottom"
syncfusion:Ribbon.KeyTip="X"
Command="{Binding ExitCommand}"
CommandParameter="{Binding ElementName=mainWindow}"
Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Name="ribbonFormatBar" syncfusion:Ribbon.KeyTip="FN"
Header="Clipboard">
<syncfusion:RibbonButton
Margin="1" Command="ApplicationCommands.Paste" Label="Paste"
SizeForm="Large" IconTemplate="{StaticResource Paste}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Cut" Label="Cut"
SizeForm="Small" IconTemplate="{StaticResource Cut}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Copy" Label="Copy"
SizeForm="Small" IconTemplate="{StaticResource Copy}" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Format Painter"
SizeForm="Small" IconTemplate="{StaticResource FormatPainter}" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="barFont" Header="Font"
IsLargeButtonPanel="False">
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox
Name="ribbonFontName" Width="110" syncfusion:Ribbon.KeyTip="FF"
DisplayMemberPath="FontFamily"
IsEditable="True" ItemsSource="{Binding FontFamilyList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"

```

```

CommandParameter="{Binding ElementName=ribbonFontName, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox
Name="ribbonFontSize" Width="40" DisplayMemberPath="FontSize"
IsEditable="True" ItemsSource="{Binding FontSizeList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"
CommandParameter="{Binding ElementName=ribbonFontSize, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24">
<syncfusion:RibbonButton
Name="ribbonIndentButton" Command="EditingCommands.IncreaseFontSize"
SizeForm="ExtraSmall" IconTemplate="{StaticResource IncreaseFontSize}"/>
<syncfusion:RibbonButton
Command="EditingCommands.DecreaseFontSize" SizeForm="ExtraSmall"
IconTemplate="{StaticResource DecreaseFontSize}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="ribbonBoldButton" Command="EditingCommands.ToggleBold" IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Bold}"/>
<syncfusion:RibbonButton
Name="ribbonItalicButton" Command="EditingCommands.ToggleItalic"
IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Italics}"/>
<syncfusion:RibbonButton
Name="ribbonUnderlineButton" Command="EditingCommands.ToggleUnderline"
IsToggle="True" SizeForm="ExtraSmall" IconTemplate="{StaticResource
Underline}"/>
<syncfusion:RibbonButton
Name="strikeButton" Command="local:RibbonCommand.ButtonCommand"
SizeForm="ExtraSmall" IconTemplate="{StaticResource ClearFormatting}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
FontColor}">
<syncfusion:ColorPickerPalette x:Name="fontColorPicker" Color="Black"
IsExpanded="True" />
</syncfusion:SplitButton>
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
TextHighlight}">
<syncfusion:ColorPickerPalette x:Name="highlightColorPicker"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatShading" syncfusion:Ribbon.KeyTip="FS" SizeForm="ExtraSmall"
IconTemplate="{StaticResource Shading}">

```

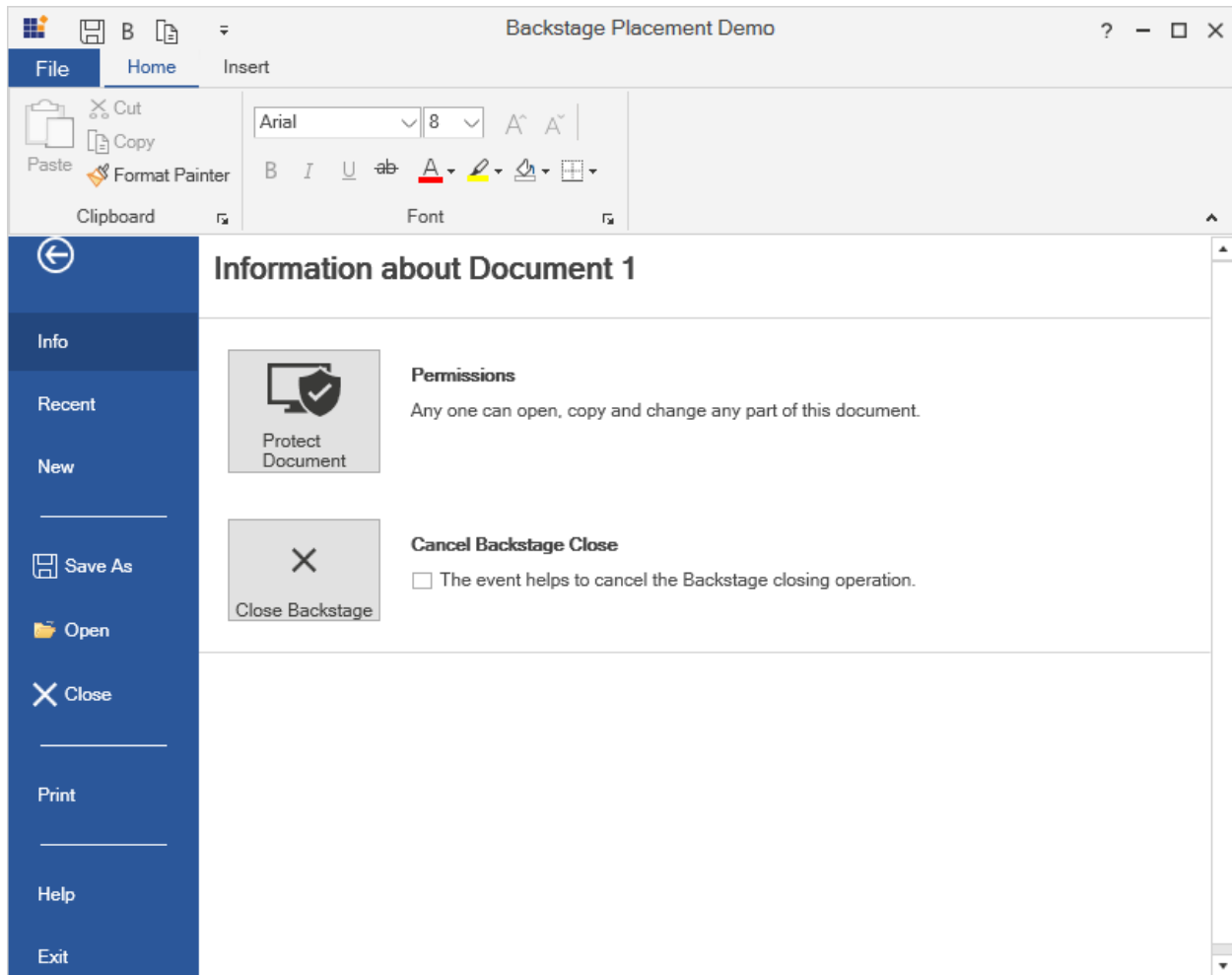
```

<syncfusion:ColorPickerPalette x:Name="shadingColorPicker" Margin="3"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatBorder" syncfusion:Ribbon.KeyTip="BF" SizeForm="ExtraSmall"
IconTemplate="{StaticResource FormatBorder}">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.DropDownCommand" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem Margin="5">Full Border</ListBoxItem>
<ListBoxItem Margin="5">Half Border</ListBoxItem>
<ListBoxItem Margin="5">Inside Border</ListBoxItem>
<ListBoxItem Margin="5">Outside Border</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert">
<syncfusion:RibbonBar Header="Pages">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Cover Page"
LargeIcon="/BackStage;component/Assets/Ribbon/CoverPage32.png"
SizeForm="Large" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="BlankPage"
SizeForm="Large"
IconTemplate="{StaticResource BlankPage}"/>
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="PageBreak"
IconTemplate="{StaticResource PageBreak}" SizeForm="Large" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Illustrations">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Picture"
SizeForm="Large" IconTemplate="{StaticResource Picture}"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
<Grid x:Name="backstageGrid" Grid.Row="1"/>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```
ribbonBackStage.PlacementTarget = backstageGrid;
```

Place the backstage in BelowTab

Backstage placed in RibbonWindow

When the [PlacementType](#) is set to [BelowTab](#) and the [PlacementTarget](#) is not set, the Backstage will appear below the ribbon tabs instead of occupying the entire Ribbon Window while it is opened.

XML

```
<syncfusion:RibbonWindow
x:Class="BackStage.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:interactivity="http://schemas.microsoft.com/xaml/behaviors"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:system="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:BackStage"
xmlns:tools="clr-
namespace:Syncfusion.Windows.Tools;assembly=Syncfusion.Tools.Wpf"
Width="744"
Height="590"
Title="Backstage Placement Demo"
```

```

syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
ThemeName=Office2019Colorful}"
WindowStartupLocation="CenterScreen">
<syncfusion:RibbonWindow.Resources>
<syncfusion:ColorToBrushConverter x:Key="ColorToBrushConverter" />
</syncfusion:RibbonWindow.Resources>
<syncfusion:RibbonWindow.DataContext>
<local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
<Grid>
<Grid.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/BackStage;component/Assets/Ribbon/PathIcon.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Grid.Resources>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid>
<syncfusion:Ribbon
Name="mainRibbon" local:ViewModel.Ribbon="{Binding ElementName=mainRibbon}"
IsBackStageVisible="True"
BackStageColor="{Binding ElementName=backColor, Path=Color, Mode=OneWay,
Converter={StaticResource ColorToBrushConverter}}"
BackStageHeader="File">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Save}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Bold}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Copy}"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage x:Name="ribbonBackStage" PlacementType="BelowTab"
syncfusion:Ribbon.KeyTip="B">
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="I" Header="Info">
<Grid Background="{Binding ElementName=mainRibbon, Path=Background}">
<Grid.RowDefinitions>
<RowDefinition Height="auto" />
<RowDefinition Height="auto" />
</Grid.RowDefinitions>
<local:Information x:Name="informationView" />
</Grid>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="R" Header="Recent">
<local:Recent x:Name="recentView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="N" Header="New">

```

```

<local:New x:Name="newView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackstageSeparator />
<syncfusion:BackstageCommandButton
syncfusion:Ribbon.KeyTip="A" Command="{Binding SaveAsCommand}"
Header="Save As" IconTemplate="{StaticResource Save}" />
<syncfusion:BackstageCommandButton
syncfusion:Ribbon.KeyTip="E" Command="{Binding OpenCommand}"
Header="Open" Icon="/BackStage;component/Assets/Ribbon/Open32.png" />
<syncfusion:BackstageCommandButton
syncfusion:Ribbon.KeyTip="C" Command="{Binding CloseCommand}"
Header="Close" IconTemplate="{StaticResource CloseTab}" />
<syncfusion:BackstageSeparator />
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="P" Header="Print">
<local:Print x:Name="printView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackstageSeparator Position="Bottom" />
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="H" Position="Bottom"
Header="Help">
<local:Help x:Name="helpView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackstageCommandButton
VerticalAlignment="Bottom" Position="Bottom"
syncfusion:Ribbon.KeyTip="X"
Command="{Binding ExitCommand}"
CommandParameter="{Binding ElementName=mainWindow}"
Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Name="ribbonFormatBar" syncfusion:Ribbon.KeyTip="FN"
Header="Clipboard">
<syncfusion:RibbonButton
Margin="1" Command="ApplicationCommands.Paste" Label="Paste"
SizeForm="Large" IconTemplate="{StaticResource Paste}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Cut" Label="Cut"
SizeForm="Small" IconTemplate="{StaticResource Cut}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Copy" Label="Copy"
SizeForm="Small" IconTemplate="{StaticResource Copy}" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Format Painter"
SizeForm="Small" IconTemplate="{StaticResource FormatPainter}" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="barFont" Header="Font"
IsLargeButtonPanel="False">
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox
Name="ribbonFontName" Width="110" syncfusion:Ribbon.KeyTip="FF"
DisplayMemberPath="FontFamily"
IsEditable="True" ItemsSource="{Binding FontFamilyList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">

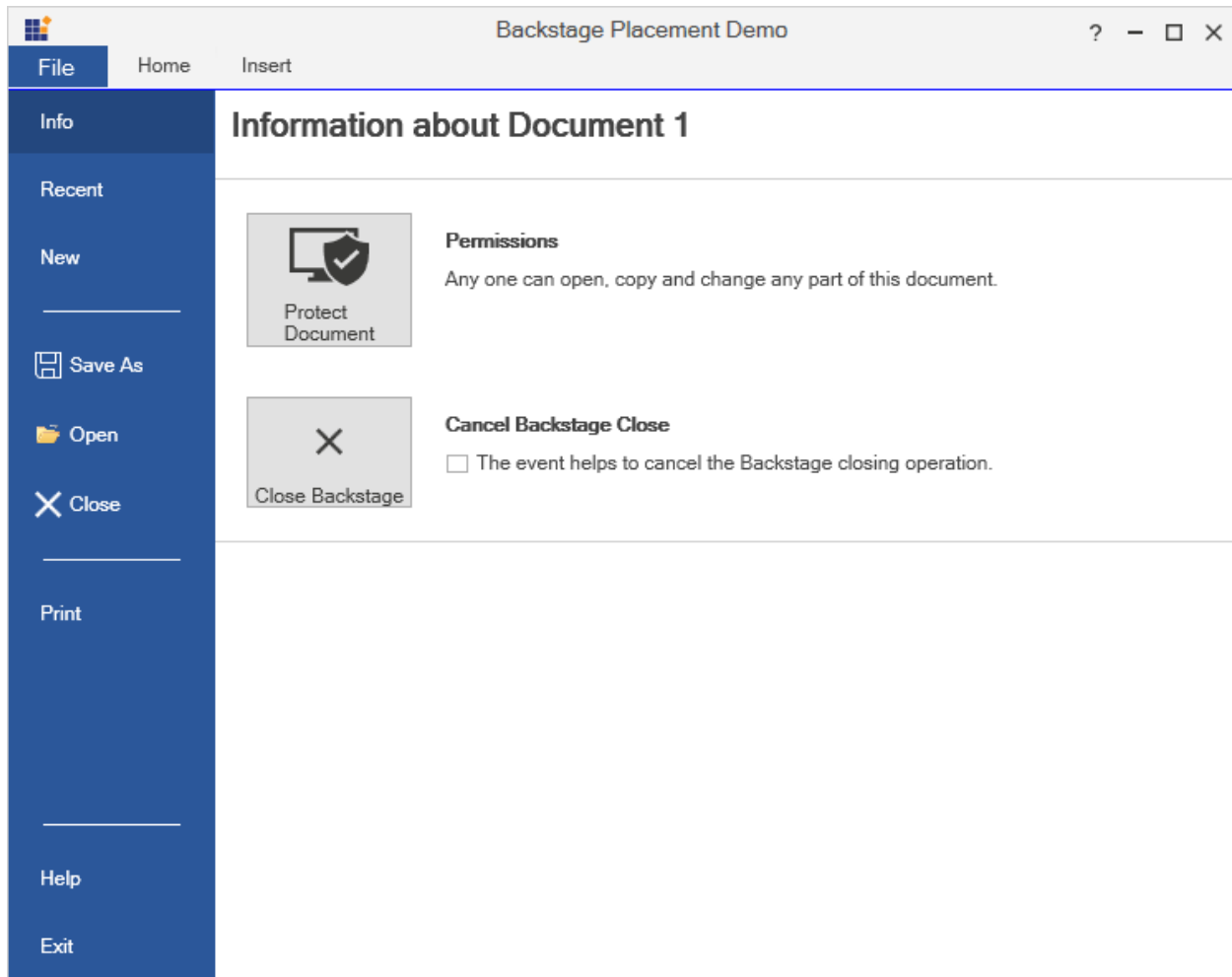
```

```

<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"
CommandParameter="{Binding ElementName=ribbonFontName, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox
Name="ribbonFontSize" Width="40" DisplayMemberPath="FontSize"
IsEditable="True" ItemsSource="{Binding FontSizeList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"
CommandParameter="{Binding ElementName=ribbonFontSize, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24">
<syncfusion:RibbonButton
Name="ribbonIndentButton" Command="EditingCommands.IncreaseFontSize"
SizeForm="ExtraSmall" IconTemplate="{StaticResource IncreaseFontSize}"/>
<syncfusion:RibbonButton
Command="EditingCommands.DecreaseFontSize" SizeForm="ExtraSmall"
IconTemplate="{StaticResource DecreaseFontSize}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="ribbonBoldButton" Command="EditingCommands.ToggleBold" IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Bold}"/>
<syncfusion:RibbonButton
Name="ribbonItalicButton" Command="EditingCommands.ToggleItalic"
IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Italics}"/>
<syncfusion:RibbonButton
Name="ribbonUnderlineButton" Command="EditingCommands.ToggleUnderline"
IsToggle="True" SizeForm="ExtraSmall" IconTemplate="{StaticResource
Underline}"/>
<syncfusion:RibbonButton
Name="strikeButton" Command="local:RibbonCommand.ButtonCommand"
SizeForm="ExtraSmall" IconTemplate="{StaticResource ClearFormatting}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
FontColor}">
<syncfusion:ColorPickerPalette x:Name="fontColorPicker" Color="Black"
IsExpanded="True" />
</syncfusion:SplitButton>
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
TextHighlight}">
<syncfusion:ColorPickerPalette x:Name="highlightColorPicker"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatShading" syncfusion:Ribbon.KeyTip="FS" SizeForm="ExtraSmall"

```

```
IconTemplate="{StaticResource Shading}">
<syncfusion:ColorPickerPalette x:Name="shadingColorPicker" Margin="3"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatBorder" syncfusion:Ribbon.KeyTip="BF" SizeForm="ExtraSmall"
IconTemplate="{StaticResource FormatBorder}">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.DropDownCommand" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem Margin="5">Full Border</ListBoxItem>
<ListBoxItem Margin="5">Half Border</ListBoxItem>
<ListBoxItem Margin="5">Inside Border</ListBoxItem>
<ListBoxItem Margin="5">Outside Border</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert">
<syncfusion:RibbonBar Header="Pages">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Cover Page"
LargeIcon="/BackStage;component/Assets/Ribbon/CoverPage32.png"
SizeForm="Large" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="BlankPage"
SizeForm="Large"
IconTemplate="{StaticResource BlankPage}"/>
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="PageBreak"
IconTemplate="{StaticResource PageBreak}" SizeForm="Large" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Illustrations">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Picture"
SizeForm="Large" IconTemplate="{StaticResource Picture}"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
<Grid x:Name="backstageGrid" Grid.Row="1"/>
</Grid>
</syncfusion:RibbonWindow>
```



Backstage placed in MS Window

When the [PlacementType](#) is set to [BelowTab](#) and the [PlacementTarget](#) is not set, the Backstage will appear below the ribbon tabs instead of occupying the entire MS Window while it is opened.

XML

```
<Window
x:Class="BackStage.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:interactivity="http://schemas.microsoft.com/xaml/behaviors"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:system="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:BackStage"
xmlns:tools="clr-
namespace:Syncfusion.Windows.Tools;assembly=Syncfusion.Tools.Wpf"
Width="1100"
Height="700"
Title="Backstage Placement Demo"
syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
ThemeName=Office2019Colorful}"
WindowStartupLocation="CenterScreen">
```

```

<Window.Resources>
<syncfusion:ColorToBrushConverter x:Key="ColorToBrushConverter" />
</Window.Resources>
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
<Grid.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/BackStage;component/Assets/Ribbon/PathIcon.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Grid.Resources>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid>
<syncfusion:Ribbon
Name="mainRibbon" local:ViewModel.Ribbon="{Binding ElementName=mainRibbon}"
BackStageColor="{Binding ElementName=backColor, Path=Color, Mode=OneWay,
Converter={StaticResource ColorToBrushConverter}}"
BackStageHeader="File">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Save}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Bold}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Copy}"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage x:Name="ribbonBackStage" PlacementType="BelowTab"
syncfusion:Ribbon.KeyTip="B" >
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="I" Header="Info">
<Grid Background="{Binding ElementName=mainRibbon, Path=Background}">
<Grid.RowDefinitions>
<RowDefinition Height="auto" />
<RowDefinition Height="auto" />
</Grid.RowDefinitions>
<local:Information x:Name="informationView" />
</Grid>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="R" Header="Recent">
<local:Recent x:Name="recentView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="N" Header="New">
<local:New x:Name="newView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackStageSeparator />
<syncfusion:BackStageCommandButton

```

```

syncfusion:Ribbon.KeyTip="A" Command="{Binding SaveAsCommand}"
Header="Save As" IconTemplate="{StaticResource Save}" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="E" Command="{Binding OpenCommand}"
Header="Open" Icon="/BackStage;component/Assets/Ribbon/Open32.png" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="C" Command="{Binding CloseCommand}"
Header="Close" IconTemplate="{StaticResource CloseTab}" />
<syncfusion:BackStageSeparator />
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="P" Header="Print">
<local:Print x:Name="printView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageSeparator Position="Bottom"/>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="H" Position="Bottom"
Header="Help">
<local:Help x:Name="helpView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageCommandButton
VerticalAlignment="Bottom" Position="Bottom"
syncfusion:Ribbon.KeyTip="X"
Command="{Binding ExitCommand}"
CommandParameter="{Binding ElementName=mainWindow}"
Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Name="ribbonFormatBar" syncfusion:Ribbon.KeyTip="FN"
Header="Clipboard">
<syncfusion:RibbonButton
Margin="1" Command="ApplicationCommands.Paste" Label="Paste"
SizeForm="Large" IconTemplate="{StaticResource Paste}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Cut" Label="Cut"
SizeForm="Small" IconTemplate="{StaticResource Cut}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Copy" Label="Copy"
SizeForm="Small" IconTemplate="{StaticResource Copy}" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Format Painter"
SizeForm="Small" IconTemplate="{StaticResource FormatPainter}" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="barFont" Header="Font"
IsLargeButtonPanel="False">
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox
Name="ribbonFontName" Width="110" syncfusion:Ribbon.KeyTip="FF"
DisplayMemberPath="FontFamily"
IsEditable="True" ItemsSource="{Binding FontFamilyList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"
CommandParameter="{Binding ElementName=ribbonFontName, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>

```



```

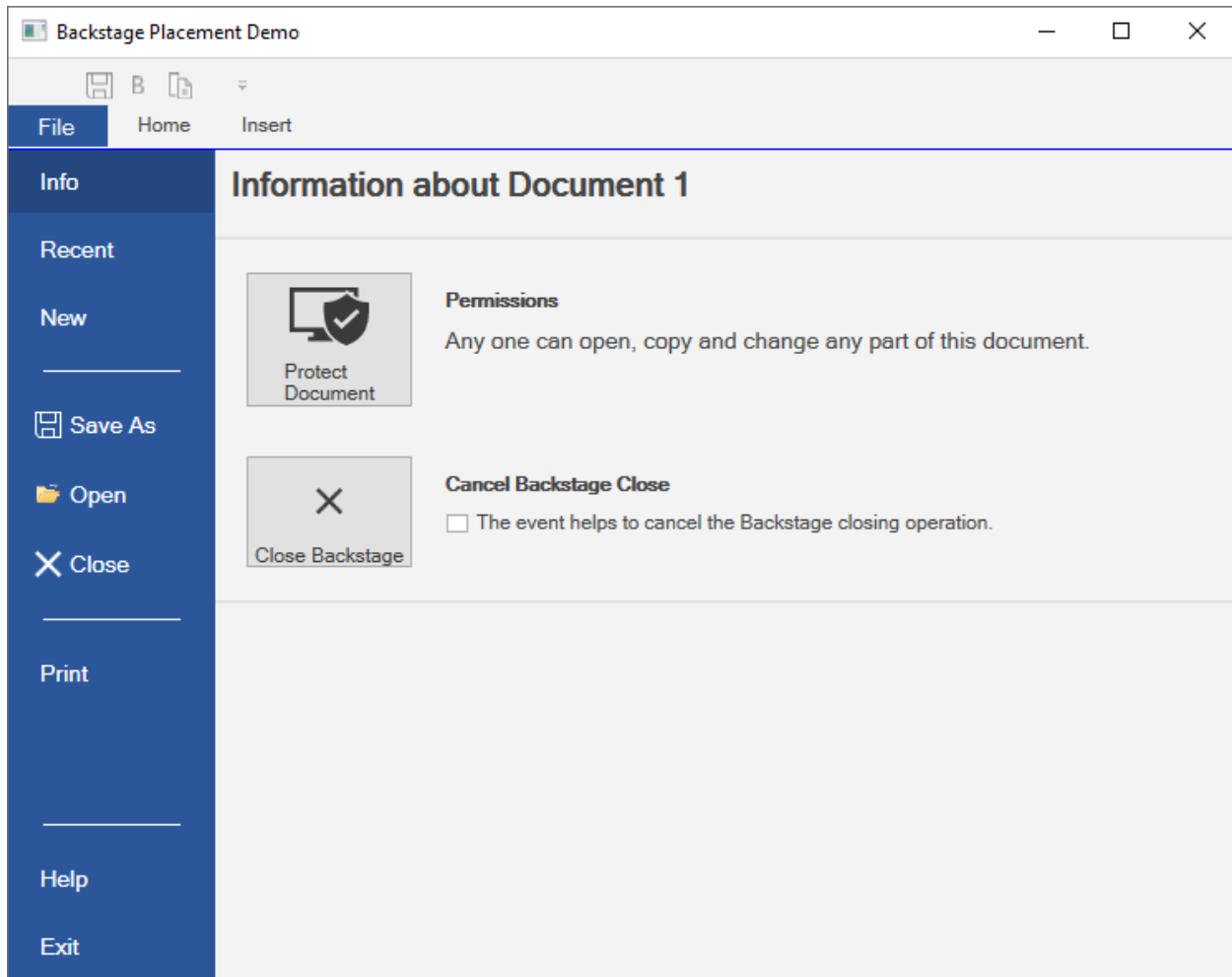
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox
Name="ribbonFontSize" Width="40" DisplayMemberPath="FontSize"
IsEditable="True" ItemsSource="{Binding FontSizeList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"
CommandParameter="{Binding ElementName=ribbonFontSize, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24">
<syncfusion:RibbonButton
Name="ribbonIndentButton" Command="EditingCommands.IncreaseFontSize"
SizeForm="ExtraSmall" IconTemplate="{StaticResource IncreaseFontSize}"/>
<syncfusion:RibbonButton
Command="EditingCommands.DecreaseFontSize" SizeForm="ExtraSmall"
IconTemplate="{StaticResource DecreaseFontSize}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="ribbonBoldButton" Command="EditingCommands.ToggleBold" IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Bold}"/>
<syncfusion:RibbonButton
Name="ribbonItalicButton" Command="EditingCommands.ToggleItalic"
IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Italics}"/>
<syncfusion:RibbonButton
Name="ribbonUnderlineButton" Command="EditingCommands.ToggleUnderline"
IsToggle="True" SizeForm="ExtraSmall" IconTemplate="{StaticResource
Underline}"/>
<syncfusion:RibbonButton
Name="strikeButton" Command="local:RibbonCommand.ButtonCommand"
SizeForm="ExtraSmall" IconTemplate="{StaticResource ClearFormatting}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
FontColor}">
<syncfusion:ColorPickerPalette x:Name="fontColorPicker" Color="Black"
IsExpanded="True" />
</syncfusion:SplitButton>
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
TextHighlight}">
<syncfusion:ColorPickerPalette x:Name="highlightColorPicker"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatShading" syncfusion:Ribbon.KeyTip="FS" SizeForm="ExtraSmall"
IconTemplate="{StaticResource Shading}">
<syncfusion:ColorPickerPalette x:Name="shadingColorPicker" Margin="3"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton

```

```

Name="formatBorder" syncfusion:Ribbon.KeyTip="BF" SizeForm="ExtraSmall"
IconTemplate="{StaticResource FormatBorder}">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.DropDownCommand" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem Margin="5">Full Border</ListBoxItem>
<ListBoxItem Margin="5">Half Border</ListBoxItem>
<ListBoxItem Margin="5">Inside Border</ListBoxItem>
<ListBoxItem Margin="5">Outside Border</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert">
<syncfusion:RibbonBar Header="Pages">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Cover Page"
LargeIcon="/BackStage;component/Assets/Ribbon/CoverPage32.png"
SizeForm="Large" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="BlankPage"
SizeForm="Large"
IconTemplate="{StaticResource BlankPage}" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="PageBreak"
IconTemplate="{StaticResource PageBreak}" SizeForm="Large" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Illustrations">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Picture"
SizeForm="Large" IconTemplate="{StaticResource Picture}" />
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
<Grid x:Name="backstageGrid" Grid.Row="1"/>
</Grid>
</Window>

```



Backstage placed in Placement target

When the [PlacementType](#) is set to [BelowTab](#) and the [PlacementTarget](#) is set to an element, the Backstage will occupy the area of the target element minus the ribbon tab area while it is opened.

XML

```
<Window
  x:Class="BackStage.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:interactivity="http://schemas.microsoft.com/xaml/behaviors"
  xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
  xmlns:syncfusionskin="clr-
    namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
  xmlns:system="clr-namespace:System;assembly=mscorlib"
  xmlns:local="clr-namespace:BackStage"
  xmlns:tools="clr-
    namespace:Syncfusion.Windows.Tools;assembly=Syncfusion.Tools.Wpf"
  Width="1100"
  Height="700"
  Title="Backstage Placement Demo"
  syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
    ThemeName=Office2019Colorful}"
  WindowStartupLocation="CenterScreen">
```

```

<Window.Resources>
<syncfusion:ColorToBrushConverter x:Key="ColorToBrushConverter" />
</Window.Resources>
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
<Grid.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/BackStage;component/Assets/Ribbon/PathIcon.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Grid.Resources>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid>
<syncfusion:Ribbon
Name="mainRibbon" local:ViewModel.Ribbon="{Binding ElementName=mainRibbon}"
BackStageColor="{Binding ElementName=backColor, Path=Color, Mode=OneWay,
Converter={StaticResource ColorToBrushConverter}}"
BackStageHeader="File">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Save}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Bold}"/>
<syncfusion:RibbonButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
Copy}"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage x:Name="ribbonBackStage" PlacementType="BelowTab"
syncfusion:Ribbon.KeyTip="B" >
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="I" Header="Info">
<Grid Background="{Binding ElementName=mainRibbon, Path=Background}">
<Grid.RowDefinitions>
<RowDefinition Height="auto" />
<RowDefinition Height="auto" />
</Grid.RowDefinitions>
<local:Information x:Name="informationView" />
</Grid>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="R" Header="Recent">
<local:Recent x:Name="recentView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="N" Header="New">
<local:New x:Name="newView" Background="{Binding
ElementName=mainRibbon, Path=Background}"/>
</syncfusion:BackstageTabItem>
<syncfusion:BackStageSeparator />
<syncfusion:BackStageCommandButton

```

```

syncfusion:Ribbon.KeyTip="A" Command="{Binding SaveAsCommand}"
Header="Save As" IconTemplate="{StaticResource Save}" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="E" Command="{Binding OpenCommand}"
Header="Open" Icon="/BackStage;component/Assets/Ribbon/Open32.png" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.KeyTip="C" Command="{Binding CloseCommand}"
Header="Close" IconTemplate="{StaticResource CloseTab}" />
<syncfusion:BackStageSeparator />
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="P" Header="Print">
<local:Print x:Name="printView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageSeparator Position="Bottom"/>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="H" Position="Bottom"
Header="Help">
<local:Help x:Name="helpView" Background="{Binding
ElementName=mainRibbon,Path=Background}" />
</syncfusion:BackstageTabItem>
<syncfusion:BackStageCommandButton
VerticalAlignment="Bottom" Position="Bottom"
syncfusion:Ribbon.KeyTip="X"
Command="{Binding ExitCommand}"
CommandParameter="{Binding ElementName=mainWindow}"
Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Name="ribbonFormatBar" syncfusion:Ribbon.KeyTip="FN"
Header="Clipboard">
<syncfusion:RibbonButton
Margin="1" Command="ApplicationCommands.Paste" Label="Paste"
SizeForm="Large" IconTemplate="{StaticResource Paste}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Cut" Label="Cut"
SizeForm="Small" IconTemplate="{StaticResource Cut}" />
<syncfusion:RibbonButton
HorizontalAlignment="Left" Command="ApplicationCommands.Copy" Label="Copy"
SizeForm="Small" IconTemplate="{StaticResource Copy}" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Format Painter"
SizeForm="Small" IconTemplate="{StaticResource FormatPainter}" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="barFont" Header="Font"
IsLargeButtonPanel="False">
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox
Name="ribbonFontName" Width="110" syncfusion:Ribbon.KeyTip="FF"
DisplayMemberPath="FontFamily"
IsEditable="True" ItemsSource="{Binding FontFamilyList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"
CommandParameter="{Binding ElementName=ribbonFontName, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>

```

```

</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox
Name="ribbonFontSize" Width="40" DisplayMemberPath="FontSize"
IsEditable="True" ItemsSource="{Binding FontSizeList}" SelectedIndex="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.RibbonComboBoxCommand"
CommandParameter="{Binding ElementName=ribbonFontSize, Path=SelectedIndex}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24">
<syncfusion:RibbonButton
Name="ribbonIndentButton" Command="EditingCommands.IncreaseFontSize"
SizeForm="ExtraSmall" IconTemplate="{StaticResource IncreaseFontSize}"/>
<syncfusion:RibbonButton
Command="EditingCommands.DecreaseFontSize" SizeForm="ExtraSmall"
IconTemplate="{StaticResource DecreaseFontSize}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="ribbonBoldButton" Command="EditingCommands.ToggleBold" IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Bold}"/>
<syncfusion:RibbonButton
Name="ribbonItalicButton" Command="EditingCommands.ToggleItalic"
IsToggle="True"
SizeForm="ExtraSmall" IconTemplate="{StaticResource Italics}"/>
<syncfusion:RibbonButton
Name="ribbonUnderlineButton" Command="EditingCommands.ToggleUnderline"
IsToggle="True" SizeForm="ExtraSmall" IconTemplate="{StaticResource
Underline}"/>
<syncfusion:RibbonButton
Name="strikeButton" Command="local:RibbonCommand.ButtonCommand"
SizeForm="ExtraSmall" IconTemplate="{StaticResource ClearFormatting}"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
FontColor}">
<syncfusion:ColorPickerPalette x:Name="fontColorPicker" Color="Black"
IsExpanded="True" />
</syncfusion:SplitButton>
<syncfusion:SplitButton SizeForm="ExtraSmall" IconTemplate="{StaticResource
TextHighlight}">
<syncfusion:ColorPickerPalette x:Name="highlightColorPicker"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatShading" syncfusion:Ribbon.KeyTip="FS" SizeForm="ExtraSmall"
IconTemplate="{StaticResource Shading}">
<syncfusion:ColorPickerPalette x:Name="shadingColorPicker" Margin="3"
IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton

```

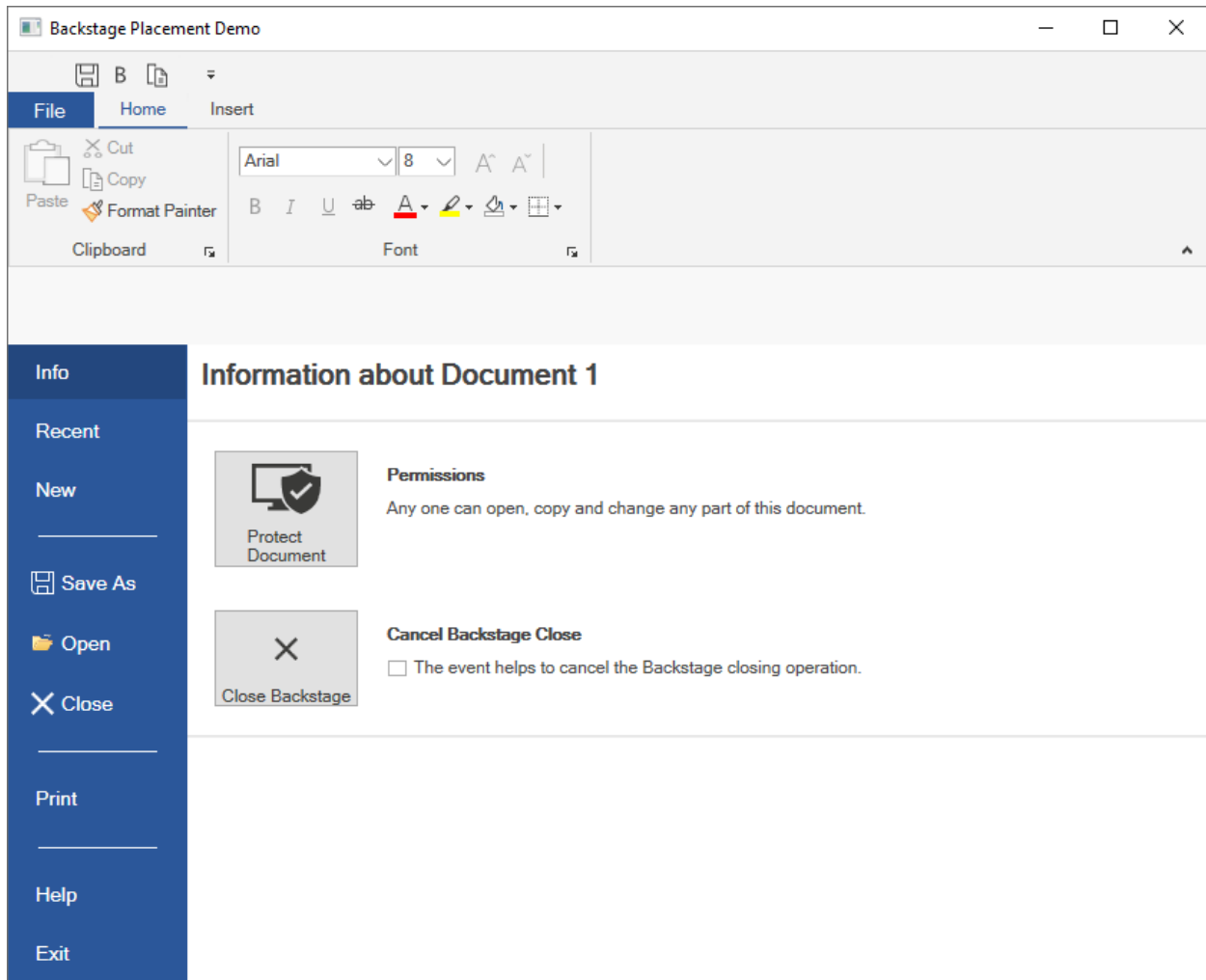
```

Name="formatBorder" syncfusion:Ribbon.KeyTip="BF" SizeForm="ExtraSmall"
IconTemplate="{StaticResource FormatBorder}">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction
Command="local:RibbonCommand.DropDownCommand" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem Margin="5">Full Border</ListBoxItem>
<ListBoxItem Margin="5">Half Border</ListBoxItem>
<ListBoxItem Margin="5">Inside Border</ListBoxItem>
<ListBoxItem Margin="5">Outside Border</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert">
<syncfusion:RibbonBar Header="Pages">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Cover Page"
LargeIcon="/BackStage;component/Assets/Ribbon/CoverPage32.png"
SizeForm="Large" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="BlankPage"
SizeForm="Large"
IconTemplate="{StaticResource BlankPage}" />
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="PageBreak"
IconTemplate="{StaticResource PageBreak}" SizeForm="Large" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Illustrations">
<syncfusion:RibbonButton
Command="local:RibbonCommand.ButtonCommand" Label="Picture"
SizeForm="Large" IconTemplate="{StaticResource Picture}" />
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
<Grid x:Name="backstageGrid" Grid.Row="1"/>
</Grid>
</Window>

```

C#

```
ribbonBackStage.PlacementTarget = backstageGrid;
```



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

Customize the BackStageButton visibility

Ribbon control allows to show or hide the **BackStageButton** by using its **Visibility** property.

The following code example illustrates how to show or hide the **BackStageButton**.

XML

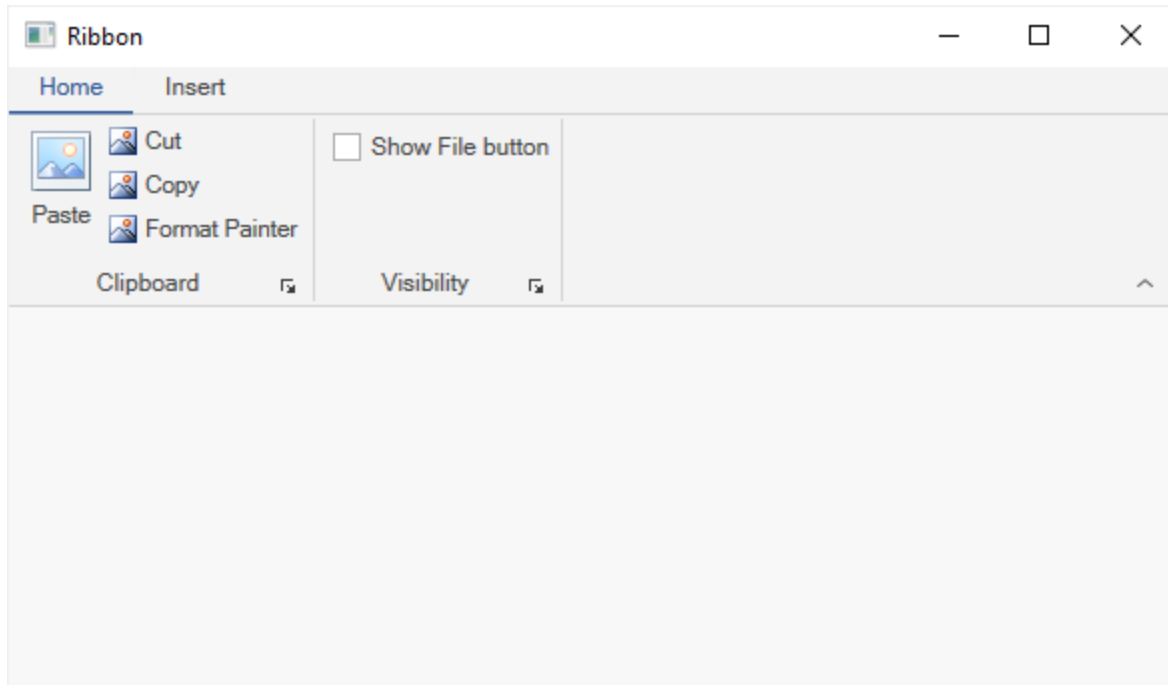
```
<Window
x:Class="BackStageButton.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:BackStageButton"
xmlns:skinManager="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="Ribbon"
Width="800"
Height="450"
skinManager:SfSkinManager.Theme="{skinManager:SkinManagerExtension
ThemeName=Office2019Colorful}">
<Grid>
```



```
<syncfusion:Ribbon Name="ribbon" BackStageHeader="File">
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage x:Name="ribbonBackStage">
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="I" Header="Info" />
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="R" Header="Recent" />
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="N" Header="New" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar
Name="ribbonFormatBar"
syncfusion:Ribbon.KeyTip="FN"
Header="Clipboard">
<syncfusion:RibbonButton
Margin="1"
Label="Paste"
SizeForm="Large" />
<syncfusion:RibbonButton
HorizontalAlignment="Left"
Label="Cut"
SizeForm="Small" />
<syncfusion:RibbonButton
HorizontalAlignment="Left"
Label="Copy"
SizeForm="Small" />
<syncfusion:RibbonButton Label="Format Painter" SizeForm="Small" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Visibility">
<syncfusion:RibbonCheckBox
Margin="3"
Content="Show File button"
IsChecked="True" />
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert" />
</syncfusion:Ribbon>
</Grid>
</Window>
```

C#

```
public MainWindow()
{
InitializeComponent();
this.Loaded += MainWindow_Loaded;
}
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
ribbon.BackStageButton.Visibility = Visibility.Collapsed;
}
```



Application Menu in WPF Ribbon

The [ApplicationMenu](#) can be added by using [ApplicationMenu](#) property of Ribbon. To show the [ApplicationMenu](#), click the **FILE** button in Ribbon like in Microsoft Outlook.

The [IsPopupOpen](#) boolean property is used to show the [ApplicationMenu](#) while launching itself.

Note: The [BackStage](#) is not applicable when the ApplicationMenu is needed.

Adding MenuItems in ApplicationMenu

[MenuItems](#) are displayed in right of the [ApplicationMenu](#). Different [MenuItems](#) are added to an application menu using its [MenuItems](#) property.

XML

```
<syncfusion:RibbonWindow
x:Class="ApplicationMenu.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:interactivity="http://schemas.microsoft.com/xaml/behaviors"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:system="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:ApplicationMenu"
xmlns:tools="clr-
namespace:Syncfusion.Windows.Tools;assembly=Syncfusion.Tools.Wpf"
Width="1100" Height="700"
Title="Application Menu Demo"
syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
ThemeName=Office2019Colorful}"
WindowStartupLocation="CenterScreen">
<syncfusion:RibbonWindow.DataContext>
<local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
```

```

<syncfusion:RibbonWindow.Resources>
<syncfusion:ColorToBrushConverter x:Key="ColorToBrushConverter" />
<syncfusion:RibbonContextMenu
x:Key="GalleryContextMenu"
x:Name="contextMenu"
ItemsSource="{Binding}">
<syncfusion:RibbonMenuItem
Command="{Binding ButtonCommand}"
Header="Apply Style"
IconBarEnabled="True" />
<syncfusion:RibbonMenuItem
Command="{Binding RemoveStyleCommand}"
Header="Remove from Style Gallery"
IconBarEnabled="True" />
<Separator />
<syncfusion:RibbonMenuItem
Command="{Binding MinimizeRibbonCommand}"
Header="Minimize Ribbon"
IconBarEnabled="True" />
</syncfusion:RibbonContextMenu>
</syncfusion:RibbonWindow.Resources>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/ApplicationMenu;component/Assets/Ribbon/PathIcon.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Grid.Resources>
<Grid x:Name="ribbonGrid">
<syncfusion:Ribbon
Name="mainRibbon"
local:ViewModel.Ribbon="{Binding ElementName=mainRibbon}"
syncfusion:Ribbon.IsAutoSizeFormEnabled="True"
BackStageCornerImageVisibility="Collapsed"
BackStageHeader="File" EnableMoreCommands="True" ShowCustomizeRibbon="True">
<syncfusion:Ribbon.ApplicationMenu>
<syncfusion:ApplicationMenu
Name="_applicationMenu"
Width="38"
Height="38"
IsBelowAppButton="True"
syncfusion:Ribbon.KeyTip="F"
ApplicationButtonImage="/ApplicationMenu;component/Assets/Ribbon/App.ico"
IsPopupOpen="False">
<syncfusion:RibbonButton
Width="200" Height="30" Label="New"
SizeForm="Small" Command="local:RibbonCommand.ButtonCommand"
IconTemplate="{StaticResource New}"/>
<syncfusion:RibbonButton
Width="200" Height="30" Command="local:RibbonCommand.OpenCommand"

```

```

Label="Open" SizeForm="Small"
SmallIcon="/ApplicationMenu;component/Assets/Ribbon/Open32.png"/>
<syncfusion:SplitMenuButton
Width="200" Height="30" Label="Save As"
Icon="/ApplicationMenu;component/Assets/Ribbon/Save_20.png">
<syncfusion:ApplicationMenuGroup Header="Save in another format"
IconBarEnabled="False">
<syncfusion:RibbonButton
Height="30" Label="DOC Files" SizeForm="Small"
Command="local:RibbonCommand.SaveAsCommand" IconTemplate="{StaticResource
DOC}"/>
<syncfusion:RibbonButton
Height="30" Label="PDF Files" SizeForm="Small"
Command="local:RibbonCommand.SaveAsCommand" IconTemplate="{StaticResource
PDF}"/>
<syncfusion:RibbonButton
Height="30" Label="XLS Files" SizeForm="Small"
Command="local:RibbonCommand.SaveAsCommand" IconTemplate="{StaticResource
XLS}"/>
<syncfusion:RibbonButton
Height="30" Label="ZIP Files" SizeForm="Small"
Command="local:RibbonCommand.SaveAsCommand" IconTemplate="{StaticResource
ZIP}"/>
</syncfusion:ApplicationMenuGroup>
</syncfusion:SplitMenuButton>
<syncfusion:RibbonButton
Width="200" Height="30" Label="Security" SizeForm="Small"
Command="local:RibbonCommand.ButtonCommand" IconTemplate="{StaticResource
Security}"/>
<Separator />
<syncfusion:RibbonButton
Width="200" Height="30" Label="Share"
Command="local:RibbonCommand.ButtonCommand"
SizeForm="Small" IconTemplate="{StaticResource Sharing}"/>
<syncfusion:ApplicationMenu.MenuItems>
<TextBlock MinWidth="300" FontWeight="Bold">Create New Outlook
Item</TextBlock>
<Separator />
<syncfusion:SimpleMenuButton Label="Recent Document lists"
Command="local:RibbonCommand.ButtonCommand" IconTemplate="{StaticResource
Copy}"/>
<syncfusion:SimpleMenuButton Label="Document lists"
Command="local:RibbonCommand.ButtonCommand" IconTemplate="{StaticResource
OnePage}"/>
</syncfusion:ApplicationMenu.MenuItems>
</syncfusion:ApplicationMenu>
</syncfusion:Ribbon.ApplicationMenu>
<syncfusion:Ribbon.TabPanelItem>
<syncfusion:RibbonButton SizeForm="ExtraSmall"
SmallIcon="/ApplicationMenu;component/Assets/Ribbon/Smile16.png" />
</syncfusion:Ribbon.TabPanelItem>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Save" Label="Save" />

```

```

<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Quick Print" Label="Quick
Print" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Print Preview"
Label="Print Preview" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Undo" Label="Undo" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Redo" Label="Redo" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Paste" Label="Paste" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Undo"
Command="ApplicationCommands.Undo"
Label="Undo"
SizeForm="ExtraSmall"
IconTemplate="{StaticResource Undo}"
ToolTip="Undo">
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Redo"
Command="ApplicationCommands.Redo"
IconTemplate="{StaticResource Redo}"
Label="Redo"
SizeForm="ExtraSmall"
ToolTip="Redo"/>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Save"
Command="{Binding ButtonCommand}"
Label="Save"
IconTemplate="{StaticResource Save}"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Copy"
Label="Copy"
Command="ApplicationCommands.Copy"
SizeForm="ExtraSmall"
ToolTip="Copy"
IconTemplate="{StaticResource Copy}">
</syncfusion:RibbonButton>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab
syncfusion:Ribbon.KeyTip="H"
Caption="Home"
IsChecked="True">
<syncfusion:RibbonBar

```

```

Name="formatBarClipboard" syncfusion:Ribbon.KeyTip="FN"
syncfusion:Ribbon.ShowInMoreCommands="True" Header="Clipboard">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction Command="{Binding Path=ButtonCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:RibbonButton
syncfusion:Ribbon.KeyTip="CP"
syncfusion:RibbonCommandManager.SynchronizedItem="Paste"
Command="ApplicationCommands.Paste" IconTemplate="{StaticResource Paste}"
Label="Paste" SizeForm="Large">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Content="Paste the contents of clipboard."
Description="Paste (Ctrl+V)" />
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
x:Name="cut" HorizontalAlignment="Left" syncfusion:Ribbon.KeyTip="CT"
Command="ApplicationCommands.Cut"
IconTemplate="{StaticResource Cut}" Label="Cut" SizeForm="Small">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Cut (Ctrl+X)">
<TextBlock
Width="130" HorizontalAlignment="Left" Text="Cut the selection and put it on
the clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
x:Name="copy" HorizontalAlignment="Left" syncfusion:Ribbon.KeyTip="CY"
Command="ApplicationCommands.Copy"
IconTemplate="{StaticResource Copy}" Label="Copy" SizeForm="Small">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Copy (Ctrl+C)">
<TextBlock Width="130" HorizontalAlignment="Left" Foreground="#FF4C4C4C"
Text="Copy the selection and put it on the clipboard." TextWrapping="Wrap"
/>
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
x:Name="formatPainter" syncfusion:Ribbon.KeyTip="CR" Command="{Binding
ButtonCommand}"
IconType="VectorImage" Label="Format Painter" SizeForm="Small"
IconTemplate="{StaticResource FormatPainter}">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Format painter (Ctrl+Shift+C)"
HelpText="Press F1 for more help.">
<TextBlock Width="175" HorizontalAlignment="Left" Foreground="#FF4C4C4C"
TextWrapping="Wrap">
<Run Text="Copy formatting from one place and apply it to another." />
<LineBreak />
<LineBreak />
<Run Text="Double-click this button to apply the same formatting to multiple
places in the document." />

```

```

</TextBlock>
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar
Name="barFont" syncfusion:Ribbon.KeyTip="HF" Header="Font"
IsLargeButtonPanel="False" KeyTipOnCollapsed="ZF">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction Command="{Binding Path=ButtonCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox
Name="fontFamilyComboBox" Width="100" syncfusion:Ribbon.KeyTip="FF"
syncfusion:Ribbon.ShowInMoreCommands="False"
DisplayMemberPath="FontFamily" IsEditable="True" ItemsSource="{Binding
FontFamilyList}" SelectedIndex="2">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction Command="{Binding
Path=RibbonComboBoxCommand}" CommandParameter="{Binding
ElementName=fontFamilyComboBox, Path=SelectedItem}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox
Name="fontSizeComboBox"
Width="40"
syncfusion:Ribbon.KeyTip="FZ"
syncfusion:Ribbon.ShowInMoreCommands="False"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
DisplayMemberPath="FontSize"
IsEditable="True"
ItemsSource="{Binding FontSizeList}"
SelectedIndex="5">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction Command="{Binding
Path=RibbonComboBoxCommand}" CommandParameter="{Binding
ElementName=fontSizeComboBox, Path=SelectedItem}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="increaseFontSize" syncfusion:Ribbon.KeyTip="IF"
Command="EditingCommands.IncreaseFontSize"
IconTemplate="{StaticResource IncreaseFontSize}" Label="Increase Font Size"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="decreaseFontSize" syncfusion:Ribbon.KeyTip="DF"
Command="EditingCommands.DecreaseFontSize"
IconTemplate="{StaticResource DecreaseFontSize}" Label="Decrease Font Size"
SizeForm="ExtraSmall"/>

```

```

</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="formatBold" syncfusion:Ribbon.KeyTip="B"
Command="EditingCommands.ToggleBold"
IconTemplate="{StaticResource Bold}" IsSelected="{Binding
FormatBoldIsSelectedProperty}"
IsToggle="True" Label="Bold" SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="formatItalic" syncfusion:Ribbon.KeyTip="I"
Command="EditingCommands.ToggleItalic"
IconTemplate="{StaticResource Italics}" IsSelected="{Binding
FormatItalicIsSelectedProperty}"
IsToggle="True" Label="Italic" SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="formatUnderLine" syncfusion:Ribbon.KeyTip="U"
Command="EditingCommands.ToggleUnderline" IconTemplate="{StaticResource
Underline}"
IsSelected="{Binding FormatUnderLineIsSelectedProperty}" Label="Underline"
SizeForm="ExtraSmall"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
SeparatorVisibility="Collapsed">
<syncfusion:SplitButton
x:Name="textHighlight" syncfusion:Ribbon.KeyTip="TH"
IconTemplate="{StaticResource TextHighlight}"
Label="Text Highlight Color" SizeForm="ExtraSmall">
<syncfusion:ColorPickerPalette
x:Name="highlightColorPicker" IsExpanded="True" Color="#5B99EE" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
x:Name="fontColor" syncfusion:Ribbon.KeyTip="FC"
IconTemplate="{StaticResource FontColor}"
Label="Font Color" SizeForm="ExtraSmall">
<syncfusion:ColorPickerPalette
x:Name="fontColorPicker" IsExpanded="True" Color="Black" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatShading" syncfusion:Ribbon.KeyTip="FS"
IconTemplate="{StaticResource Shading}"
Label="Shading" SizeForm="ExtraSmall">
<syncfusion:ColorPickerPalette
x:Name="shadingColorPicker" Margin="3" IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatBorder" syncfusion:Ribbon.KeyTip="BF"
IconTemplate="{StaticResource FormatBorder}"
Label="Borders" SizeForm="ExtraSmall">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction Command="{Binding Path=DropDownCommand}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem Margin="5">Full Border</ListBoxItem>

```



```

<ListBoxItem Margin="5">Half Border</ListBoxItem>
<ListBoxItem Margin="5">Inside Border</ListBoxItem>
<ListBoxItem Margin="5">Outside Border</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar
Name="formatParagraph" syncfusion:Ribbon.KeyTip="HP" Header="Paragraph"
IsLargeButtonPanel="False" KeyTipOnCollapsed="ZP">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction Command="{Binding Path=ButtonCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:ButtonPanel
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
SeparatorVisibility="Collapsed">
<syncfusion:SplitButton
Name="formatBullet" syncfusion:Ribbon.KeyTip="FN"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
Command="EditingCommands.ToggleBullets" IconTemplate="{StaticResource
Bullets}" Label="Bullets" SizeForm="ExtraSmall">
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatNumbering" syncfusion:Ribbon.KeyTip="FB"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
Command="EditingCommands.ToggleNumbering" IconTemplate="{StaticResource
Numbering}"
Label="Numbering" SizeForm="ExtraSmall">
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatLineSpacing" syncfusion:Ribbon.KeyTip="FA"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
Command="{Binding ButtonCommand}" IconTemplate="{StaticResource
LineSpacing}"
Label="Line Spacing" SizeForm="ExtraSmall">
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="formatLeftAlign" syncfusion:Ribbon.KeyTip="LA"
Command="EditingCommands.AlignLeft"
IconTemplate="{StaticResource AlignLeft}" Label="Align Left"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="formatCenterAlign" syncfusion:Ribbon.KeyTip="CA"
Command="EditingCommands.AlignCenter"
IconTemplate="{StaticResource AlignCenter}" Label="Align Center"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="formatRightAlign" syncfusion:Ribbon.KeyTip="RA"
Command="EditingCommands.AlignRight"
IconTemplate="{StaticResource AlignRight}" Label="Align Right"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton

```

```

Name="formatJustify" syncfusion:Ribbon.KeyTip="JA"
Command="EditingCommands.AlignJustify"
IconTemplate="{StaticResource AlignJustify}" Label="Align Justify"
SizeForm="ExtraSmall"/>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar IconTemplate="{StaticResource IncreaseFontSize}"
Header="Styles">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction Command="{Binding Path=ButtonCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:RibbonGallery
x:Name="gallery" Width="380" ContextMenu="{DynamicResource
GalleryContextMenu}"
IconTemplate="{StaticResource Styling}" ItemHeight="60" ItemWidth="60"
Label="Styles"
MenuIconBarEnabled="True" SelectedItem="{Binding RibbonGallerySelectedItem}"
SizeForm="Large">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectedItemChanged">
<interactivity:InvokeCommandAction Command="{Binding Path=DropDownCommand}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:RibbonGalleryItem
x:Name="firstItem" CheckOnClick="True" Command="{Binding
GalleryItemOneCommand}" Tag="firstItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0" Text="AaBbCc" TextAlignment="Center" />
<TextBlock
Margin="0,11,0,0" Text="Normal" TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="secondItem"
CheckOnClick="True"
Command="{Binding GalleryItemTwoCommand}"
Tag="secondItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,11,0,0"
Text="No Spa.."
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="thirdItem"
CheckOnClick="True"
Command="{Binding GalleryItemThreeCommand}"
Tag="thirdItem">

```

```
<StackPanel>
<TextBlock
Margin="0,5,0,0"
FontSize="18"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,6,0,0"
Text="Heading 1"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="fourthItem"
CheckOnClick="True"
Command="{Binding GalleryItemFourCommand}"
Tag="fourthItem">
<StackPanel>
<TextBlock
Margin="0,6,0,0"
FontSize="16"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,8,0,0"
Text="Heading 2"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="fifthItem"
CheckOnClick="True"
Command="{Binding GalleryItemFiveCommand}"
Tag="fifthItem">
<StackPanel>
<TextBlock
Margin="0,7,0,0"
FontSize="15"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,9,0,0"
Text="Heading 3"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="sixthItem"
CheckOnClick="True"
Command="{Binding GalleryItemSixCommand}"
Tag="sixthItem">
<StackPanel>
<TextBlock
Margin="0,7,0,0"
```

```
FontSize="14"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,9,0,0"
Text="Heading 4"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="seventhItem"
CheckOnClick="True"
Command="{Binding GalleryItemSevenCommand}"
Tag="seventhItem">
<StackPanel>
<TextBlock
FontSize="24"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,5,0,0"
Text="Title"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="eighthItem"
CheckOnClick="True"
Command="{Binding GalleryItemEightCommand}"
Tag="eighthItem">
<StackPanel>
<TextBlock
Margin="0,7,0,0"
FontStyle="Italic"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,13,0,0"
Text="Emphasis"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="ninthItem"
CheckOnClick="True"
Command="{Binding GalleryItemNineCommand}"
Tag="ninthItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
FontWeight="Bold"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Strong"
```

```
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="tenthItem"
CheckOnClick="True"
Command="{Binding GalleryItemTenCommand}"
Tag="tenthItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center"
TextDecorations="Underline" />
<TextBlock
Margin="0,10,0,0"
Text="Underline"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="eleventhItem"
CheckOnClick="True"
Command="{Binding GalleryItemElevenCommand}"
Tag="eleventhItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Foreground="Red"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Important"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="italicItem"
CheckOnClick="True"
Command="{Binding GalleryItemTwelveCommand}"
Tag="italicItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
FontStyle="Italic"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Italic"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="superString"
CheckOnClick="True"
```

```

Command="{Binding GalleryItemThirteenCommand}"
Tag="superString">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
FontWeight="ExtraBold"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Super Str.."
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGallery.MenuItems>
<syncfusion:RibbonButton
Label="Create a style" IconTemplate="{StaticResource IncreaseFontSize}" />
<syncfusion:RibbonButton
Command="{Binding ClearFormattingCommand}"
IconTemplate="{StaticResource ClearFormatting}"
Label="Clear Formatting"/>
</syncfusion:RibbonGallery.MenuItems>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab
syncfusion:Ribbon.KeyTip="I"
Caption="Insert"
IsChecked="False">
</syncfusion:RibbonTab>
<syncfusion:RibbonTab syncfusion:Ribbon.KeyTip="M" Caption="Mailings">
</syncfusion:RibbonTab>
<syncfusion:RibbonTab
syncfusion:Ribbon.KeyTip="D"
Caption="Design"
IsChecked="False">
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
<ScrollViewer
x:Name="scrollViewer"
Grid.Row="1"
Grid.ColumnSpan="2"
VerticalScrollBarVisibility="Auto">
<Grid>
<RichTextBox
Name="editor"
Margin="100,10"
Padding="50"
local:ViewModel.RichTextBox="richTextBoxText"
AcceptsTab="True"
Background="{Binding ElementName=shadingColorPicker, Path=Color,
Mode=OneWay, Converter={StaticResource ColorToBrushConverter}}"
BorderBrush="Transparent"
BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">

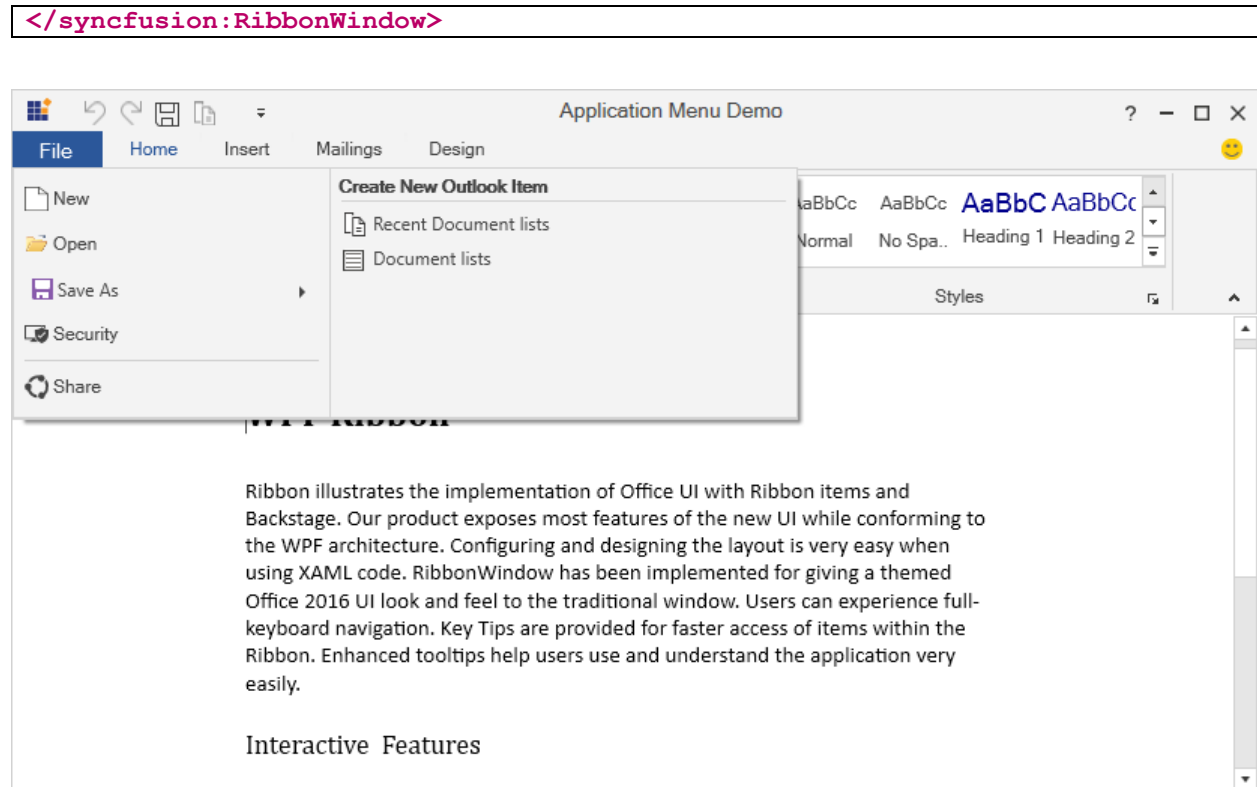
```

```

<interactivity:InvokeCommandAction Command="{Binding
Path=RichTextBoxSelectionChangedCommand}" />
</interactivity:EventTrigger>
<interactivity:EventTrigger EventName="PreviewMouseLeftButtonUp">
<interactivity:InvokeCommandAction Command="{Binding
Path=RichTextBoxPreviewMouseLeftButtonUpCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<FlowDocument
Name="flowDocumentName"
Background="{Binding ElementName=shadingColorPicker, Path=Color,
Mode=OneWay, Converter={StaticResource ColorToBrushConverter}}"
FontFamily="Calibri"
FontSize="14"
Foreground="{Binding ElementName=fontColorPicker, Path=Color, Mode=OneWay,
Converter={StaticResource ColorToBrushConverter}}"
TextAlignment="Left">
<Paragraph
FontFamily="Cambria"
FontSize="24"
FontWeight="Bold">
<Run Text="{Binding TitleText}" />
</Paragraph>
<Paragraph
x:Name="firstPara"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding FirstParagraphText}" />
</Paragraph>
<Paragraph
x:Name="secondPara"
FontFamily="Cambria"
FontSize="18">
<Run Text="{Binding SecondTitleText}" />
</Paragraph>
<List MarkerStyle="Square">
<ListItem>
<Paragraph
x:Name="pointOne"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointOne}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointTwo"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointTwo}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointThree"
FontFamily="Calibri"
FontSize="15">

```

```
<Run Text="{Binding PointThree}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointFour"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointFour}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointFive"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointFive}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointSix"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointSix}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointSeven"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointSeven}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointEight"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointEight}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointNine"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointNine}" />
</Paragraph>
</ListItem>
</List>
</FlowDocument>
</RichTextBox>
</Grid>
</ScrollViewer>
</Grid>
```

Adding ApplicationItems in ApplicationMenu

[ApplicationItems](#) are displayed in bottom of the [ApplicationMenu](#). Different [ApplicationItems](#) are added to an application menu using its [ApplicationItems](#) property.

XML

```
<syncfusion:RibbonWindow
x:Class="ApplicationMenu.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:interactivity="http://schemas.microsoft.com/xaml/behaviors"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:system="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:ApplicationMenu"
xmlns:tools="clr-
namespace:Syncfusion.Windows.Tools;assembly=Syncfusion.Tools.Wpf"
Width="1100" Height="700"
Title="Application Menu Demo"
syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
ThemeName=Office2019Colorful}"
WindowStartupLocation="CenterScreen">
<syncfusion:RibbonWindow.DataContext>
<local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
<syncfusion:RibbonWindow.Resources>
<syncfusion:ColorToBrushConverter x:Key="ColorToBrushConverter" />
<syncfusion:RibbonContextMenu
x:Key="GalleryContextMenu"
```

```

x:Name="contextMenu"
ItemsSource="{Binding}">
<syncfusion:RibbonMenuItem
Command="{Binding ButtonCommand}"
Header="Apply Style"
IconBarEnabled="True" />
<syncfusion:RibbonMenuItem
Command="{Binding RemoveStyleCommand}"
Header="Remove from Style Gallery"
IconBarEnabled="True" />
<Separator />
<syncfusion:RibbonMenuItem
Command="{Binding MinimizeRibbonCommand}"
Header="Minimize Ribbon"
IconBarEnabled="True" />
</syncfusion:RibbonContextMenu>
</syncfusion:RibbonWindow.Resources>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/ApplicationMenu;component/Assets/Ribbon/PathIcon.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Grid.Resources>
<Grid x:Name="ribbonGrid">
<syncfusion:Ribbon
Name="mainRibbon"
local:ViewModel.Ribbon="{Binding ElementName=mainRibbon}"
syncfusion:Ribbon.IsAutoSizeFormEnabled="True"
BackStageCornerImageVisibility="Collapsed"
BackStageHeader="File" EnableMoreCommands="True" ShowCustomizeRibbon="True">
<syncfusion:Ribbon.ApplicationMenu>
<syncfusion:ApplicationMenu
Name="_applicationMenu"
Width="38"
Height="38"
IsBelowAppButton="True"
syncfusion:Ribbon.KeyTip="F"
ApplicationButtonImage="/ApplicationMenu;component/Assets/Ribbon/App.ico"
IsPopupOpen="False">
<syncfusion:RibbonButton
Width="200" Height="30" Label="New"
SizeForm="Small" Command="local:RibbonCommand.ButtonCommand"
IconTemplate="{StaticResource New}"/>
<syncfusion:RibbonButton
Width="200" Height="30" Command="local:RibbonCommand.OpenCommand"
Label="Open" SizeForm="Small"
SmallIcon="/ApplicationMenu;component/Assets/Ribbon/Open32.png"/>
<syncfusion:SplitMenuButton
Width="200" Height="30" Label="Save As"
Icon="/ApplicationMenu;component/Assets/Ribbon/Save_20.png">

```

```

<syncfusion:ApplicationMenuGroup Header="Save in another format"
IconBarEnabled="False">
<syncfusion:RibbonButton
Height="30" Label="DOC Files" SizeForm="Small"
Command="local:RibbonCommand.SaveAsCommand" IconTemplate="{StaticResource
DOC}"/>
<syncfusion:RibbonButton
Height="30" Label="PDF Files" SizeForm="Small"
Command="local:RibbonCommand.SaveAsCommand" IconTemplate="{StaticResource
PDF}"/>
<syncfusion:RibbonButton
Height="30" Label="XLS Files" SizeForm="Small"
Command="local:RibbonCommand.SaveAsCommand" IconTemplate="{StaticResource
XLS}"/>
<syncfusion:RibbonButton
Height="30" Label="ZIP Files" SizeForm="Small"
Command="local:RibbonCommand.SaveAsCommand" IconTemplate="{StaticResource
ZIP}"/>
</syncfusion:ApplicationMenuGroup>
</syncfusion:SplitMenuButton>
<syncfusion:RibbonButton
Width="200" Height="30" Label="Security" SizeForm="Small"
Command="local:RibbonCommand.ButtonCommand" IconTemplate="{StaticResource
Security}"/>
<Separator />
<syncfusion:RibbonButton
Width="200" Height="30" Label="Share"
Command="local:RibbonCommand.ButtonCommand"
SizeForm="Small" IconTemplate="{StaticResource Sharing}"/>
<syncfusion:ApplicationMenu.MenuItems>
<TextBlock MinWidth="300" FontWeight="Bold">Create New Outlook
Item</TextBlock>
<Separator />
<syncfusion:SimpleMenuButton Label="Recent Document lists"
Command="local:RibbonCommand.ButtonCommand" IconTemplate="{StaticResource
Copy}"/>
<syncfusion:SimpleMenuButton Label="Document lists"
Command="local:RibbonCommand.ButtonCommand" IconTemplate="{StaticResource
OnePage}"/>
</syncfusion:ApplicationMenu.MenuItems>
<syncfusion:ApplicationMenu.ApplicationItems>
<syncfusion:ButtonAdv
Background="Transparent"
Label="Options"
SizeMode="Normal"
Command="local:RibbonCommand.ButtonCommand"
SmallIcon="/ApplicationMenu;component/Assets/Ribbon/Display.png"/>
<syncfusion:ButtonAdv
Background="Transparent"
Label="Close"
SizeMode="Normal"
Command="{Binding ApplicationMenuCommand}"
IconTemplate="{StaticResource CloseTab}"/>
</syncfusion:ApplicationMenu.ApplicationItems>
</syncfusion:ApplicationMenu>
</syncfusion:Ribbon.ApplicationMenu>
</syncfusion:Ribbon.TabPanelItem>

```

```

<syncfusion:RibbonButton SizeForm="ExtraSmall"
SmallIcon="/ApplicationMenu;component/Assets/Ribbon/Smile16.png" />
</syncfusion:Ribbon.TabPanelItem>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Save" Label="Save" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Quick Print" Label="Quick
Print" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Print Preview"
Label="Print Preview" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Undo" Label="Undo" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Redo" Label="Redo" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Paste" Label="Paste" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Undo"
Command="ApplicationCommands.Undo"
Label="Undo"
SizeForm="ExtraSmall"
IconTemplate="{StaticResource Undo}"
ToolTip="Undo">
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Redo"
Command="ApplicationCommands.Redo"
IconTemplate="{StaticResource Redo}"
Label="Redo"
SizeForm="ExtraSmall"
ToolTip="Redo"/>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Save"
Command="{Binding ButtonCommand}"
Label="Save"
IconTemplate="{StaticResource Save}"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Copy"
Label="Copy"
Command="ApplicationCommands.Copy"
SizeForm="ExtraSmall"
ToolTip="Copy"
IconTemplate="{StaticResource Copy}">

```

```

</syncfusion:RibbonButton>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab
syncfusion:Ribbon.KeyTip="H"
Caption="Home"
IsChecked="True">
<syncfusion:RibbonBar
Name="formatBarClipboard" syncfusion:Ribbon.KeyTip="FN"
syncfusion:Ribbon.ShowInMoreCommands="True" Header="Clipboard">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction Command="{Binding Path=ButtonCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:RibbonButton
syncfusion:Ribbon.KeyTip="CP"
syncfusion:RibbonCommandManager.SynchronizedItem="Paste"
Command="ApplicationCommands.Paste" IconTemplate="{StaticResource Paste}"
Label="Paste" SizeForm="Large">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Content="Paste the contents of clipboard."
Description="Paste (Ctrl+V)" />
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
x:Name="cut" HorizontalAlignment="Left" syncfusion:Ribbon.KeyTip="CT"
Command="ApplicationCommands.Cut"
IconTemplate="{StaticResource Cut}" Label="Cut" SizeForm="Small">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Cut (Ctrl+X)">
<TextBlock
Width="130" HorizontalAlignment="Left" Text="Cut the selection and put it on
the clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
x:Name="copy" HorizontalAlignment="Left" syncfusion:Ribbon.KeyTip="CY"
Command="ApplicationCommands.Copy"
IconTemplate="{StaticResource Copy}" Label="Copy" SizeForm="Small">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Copy (Ctrl+C)">
<TextBlock Width="130" HorizontalAlignment="Left" Foreground="#FF4C4C4C"
Text="Copy the selection and put it on the clipboard." TextWrapping="Wrap"
/>
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
x:Name="formatPainter" syncfusion:Ribbon.KeyTip="CR" Command="{Binding
ButtonCommand}"
IconType="VectorImage" Label="Format Painter" SizeForm="Small"
IconTemplate="{StaticResource FormatPainter}">
<syncfusion:RibbonButton.ToolTip>

```

```

<syncfusion:ScreenTip Description="Format painter (Ctrl+Shift+C)"
HelpText="Press F1 for more help.">
<TextBlock Width="175" HorizontalAlignment="Left" Foreground="#FF4C4C4C"
TextWrapping="Wrap">
<Run Text="Copy formatting from one place and apply it to another." />
<LineBreak />
<LineBreak />
<Run Text="Double-click this button to apply the same formatting to multiple
places in the document." />
</TextBlock>
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar
Name="barFont" syncfusion:Ribbon.KeyTip="HF" Header="Font"
IsLargeButtonPanel="False" KeyTipOnCollapsed="ZF">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction Command="{Binding Path=ButtonCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox
Name="fontFamilyComboBox" Width="100" syncfusion:Ribbon.KeyTip="FF"
syncfusion:Ribbon.ShowInMoreCommands="False"
DisplayMemberPath="FontFamily" IsEditable="True" ItemsSource="{Binding
FontFamilyList}" SelectedIndex="2">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction Command="{Binding
Path=RibbonComboBoxCommand}" CommandParameter="{Binding
ElementName=fontFamilyComboBox, Path=SelectedItem}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox
Name="fontSizeComboBox"
Width="40"
syncfusion:Ribbon.KeyTip="FZ"
syncfusion:Ribbon.ShowInMoreCommands="False"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
DisplayMemberPath="FontSize"
IsEditable="True"
ItemsSource="{Binding FontSizeList}"
SelectedIndex="5">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction Command="{Binding
Path=RibbonComboBoxCommand}" CommandParameter="{Binding
ElementName=fontSizeComboBox, Path=SelectedItem}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton

```

```

Name="increaseFontSize" syncfusion:Ribbon.KeyTip="IF"
Command="EditingCommands.IncreaseFontSize"
IconTemplate="{StaticResource IncreaseFontSize}" Label="Increase Font Size"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="decreaseFontSize" syncfusion:Ribbon.KeyTip="DF"
Command="EditingCommands.DecreaseFontSize"
IconTemplate="{StaticResource DecreaseFontSize}" Label="Decrease Font Size"
SizeForm="ExtraSmall"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="formatBold" syncfusion:Ribbon.KeyTip="B"
Command="EditingCommands.ToggleBold"
IconTemplate="{StaticResource Bold}" IsSelected="{Binding
FormatBoldIsSelectedProperty}"
IsToggle="True" Label="Bold" SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="formatItalic" syncfusion:Ribbon.KeyTip="I"
Command="EditingCommands.ToggleItalic"
IconTemplate="{StaticResource Italics}" IsSelected="{Binding
FormatItalicIsSelectedProperty}"
IsToggle="True" Label="Italic" SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="formatUnderLine" syncfusion:Ribbon.KeyTip="U"
Command="EditingCommands.ToggleUnderline" IconTemplate="{StaticResource
Underline}"
IsSelected="{Binding FormatUnderLineIsSelectedProperty}" Label="Underline"
SizeForm="ExtraSmall"/>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
SeparatorVisibility="Collapsed">
<syncfusion:SplitButton
x:Name="textHighlight" syncfusion:Ribbon.KeyTip="TH"
IconTemplate="{StaticResource TextHighlight}"
Label="Text Highlight Color" SizeForm="ExtraSmall">
<syncfusion:ColorPickerPalette
x:Name="highlightColorPicker" IsExpanded="True" Color="#5B99EE" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
x:Name="fontColor" syncfusion:Ribbon.KeyTip="FC"
IconTemplate="{StaticResource FontColor}"
Label="Font Color" SizeForm="ExtraSmall">
<syncfusion:ColorPickerPalette
x:Name="fontColorPicker" IsExpanded="True" Color="Black" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatShading" syncfusion:Ribbon.KeyTip="FS"
IconTemplate="{StaticResource Shading}"
Label="Shading" SizeForm="ExtraSmall">
<syncfusion:ColorPickerPalette
x:Name="shadingColorPicker" Margin="3" IsExpanded="True" Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatBorder" syncfusion:Ribbon.KeyTip="BF"
IconTemplate="{StaticResource FormatBorder}"

```

```

Label="Borders" SizeForm="ExtraSmall">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction Command="{Binding Path=DropDownCommand}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem Margin="5">Full Border</ListBoxItem>
<ListBoxItem Margin="5">Half Border</ListBoxItem>
<ListBoxItem Margin="5">Inside Border</ListBoxItem>
<ListBoxItem Margin="5">Outside Border</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar
Name="formatParagraph" syncfusion:Ribbon.KeyTip="HP" Header="Paragraph"
IsLargeButtonPanel="False" KeyTipOnCollapsed="ZP">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction Command="{Binding Path=ButtonCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:ButtonPanel
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
SeparatorVisibility="Collapsed">
<syncfusion:SplitButton
Name="formatBullet" syncfusion:Ribbon.KeyTip="FN"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
Command="EditingCommands.ToggleBullets" IconTemplate="{StaticResource
Bullets}" Label="Bullets" SizeForm="ExtraSmall">
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatNumbering" syncfusion:Ribbon.KeyTip="FB"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
Command="EditingCommands.ToggleNumbering" IconTemplate="{StaticResource
Numbering}"
Label="Numbering" SizeForm="ExtraSmall">
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatLineSpacing" syncfusion:Ribbon.KeyTip="FA"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
Command="{Binding ButtonCommand}" IconTemplate="{StaticResource
LineSpacing}"
Label="Line Spacing" SizeForm="ExtraSmall">
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
Name="formatLeftAlign" syncfusion:Ribbon.KeyTip="LA"
Command="EditingCommands.AlignLeft"
IconTemplate="{StaticResource AlignLeft}" Label="Align Left"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="formatCenterAlign" syncfusion:Ribbon.KeyTip="CA"
Command="EditingCommands.AlignCenter"

```



```

IconTemplate="{StaticResource AlignCenter}" Label="Align Center"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="formatRightAlign" syncfusion:Ribbon.KeyTip="RA"
Command="EditingCommands.AlignRight"
IconTemplate="{StaticResource AlignRight}" Label="Align Right"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Name="formatJustify" syncfusion:Ribbon.KeyTip="JA"
Command="EditingCommands.AlignJustify"
IconTemplate="{StaticResource AlignJustify}" Label="Align Justify"
SizeForm="ExtraSmall"/>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar IconTemplate="{StaticResource IncreaseFontSize}"
Header="Styles">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction Command="{Binding Path=ButtonCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:RibbonGallery
x:Name="gallery" Width="380" ContextMenu="{DynamicResource
GalleryContextMenu}"
IconTemplate="{StaticResource Styling}" ItemHeight="60" ItemWidth="60"
Label="Styles"
MenuIconBarEnabled="True" SelectedItem="{Binding RibbonGallerySelectedItem}"
SizeForm="Large">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectedItemChanged">
<interactivity:InvokeCommandAction Command="{Binding Path=DropDownCommand}"
/>
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:RibbonGalleryItem
x:Name="firstItem" CheckOnClick="True" Command="{Binding
GalleryItemOneCommand}" Tag="firstItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0" Text="AaBbCc" TextAlignment="Center" />
<TextBlock
Margin="0,11,0,0" Text="Normal" TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="secondItem"
CheckOnClick="True"
Command="{Binding GalleryItemTwoCommand}"
Tag="secondItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,11,0,0"
Text="No Spa.."

```

```
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="thirdItem"
CheckOnClick="True"
Command="{Binding GalleryItemThreeCommand}"
Tag="thirdItem">
<StackPanel>
<TextBlock
Margin="0,5,0,0"
FontSize="18"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,6,0,0"
Text="Heading 1"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="fourthItem"
CheckOnClick="True"
Command="{Binding GalleryItemFourCommand}"
Tag="fourthItem">
<StackPanel>
<TextBlock
Margin="0,6,0,0"
FontSize="16"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,8,0,0"
Text="Heading 2"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="fifthItem"
CheckOnClick="True"
Command="{Binding GalleryItemFiveCommand}"
Tag="fifthItem">
<StackPanel>
<TextBlock
Margin="0,7,0,0"
FontSize="15"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,9,0,0"
Text="Heading 3"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
```

```
<syncfusion:RibbonGalleryItem
x:Name="sixthItem"
CheckOnClick="True"
Command="{Binding GalleryItemSixCommand}"
Tag="sixthItem">
<StackPanel>
<TextBlock
Margin="0,7,0,0"
FontSize="14"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,9,0,0"
Text="Heading 4"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="seventhItem"
CheckOnClick="True"
Command="{Binding GalleryItemSevenCommand}"
Tag="seventhItem">
<StackPanel>
<TextBlock
FontSize="24"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,5,0,0"
Text="Title"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="eighthItem"
CheckOnClick="True"
Command="{Binding GalleryItemEightCommand}"
Tag="eighthItem">
<StackPanel>
<TextBlock
Margin="0,7,0,0"
FontStyle="Italic"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,13,0,0"
Text="Emphasis"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="ninthItem"
CheckOnClick="True"
Command="{Binding GalleryItemNineCommand}"
Tag="ninthItem">
<StackPanel>
```

```
<TextBlock
Margin="0,9,0,0"
FontWeight="Bold"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Strong"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="tenthItem"
CheckOnClick="True"
Command="{Binding GalleryItemTenCommand}"
Tag="tenthItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center"
TextDecorations="Underline" />
<TextBlock
Margin="0,10,0,0"
Text="Underline"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="eleventhItem"
CheckOnClick="True"
Command="{Binding GalleryItemElevenCommand}"
Tag="eleventhItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Foreground="Red"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Important"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="italicItem"
CheckOnClick="True"
Command="{Binding GalleryItemTwelveCommand}"
Tag="italicItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
FontStyle="Italic"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
```

```

Margin="0,10,0,0"
Text="Italic"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="superString"
CheckOnClick="True"
Command="{Binding GalleryItemThirteenCommand}"
Tag="superString">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
FontWeight="ExtraBold"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Super Str.."
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGallery.MenuItems>
<syncfusion:RibbonButton
Label="Create a style" IconTemplate="{StaticResource IncreaseFontSize}" />
<syncfusion:RibbonButton
Command="{Binding ClearFormattingCommand}"
IconTemplate="{StaticResource ClearFormatting}"
Label="Clear Formatting"/>
</syncfusion:RibbonGallery.MenuItems>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab
syncfusion:Ribbon.KeyTip="I"
Caption="Insert"
IsChecked="False">
</syncfusion:RibbonTab>
<syncfusion:RibbonTab syncfusion:Ribbon.KeyTip="M" Caption="Mailings">
</syncfusion:RibbonTab>
<syncfusion:RibbonTab
syncfusion:Ribbon.KeyTip="D"
Caption="Design"
IsChecked="False">
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
<ScrollViewer
x:Name="scrollViewer"
Grid.Row="1"
Grid.ColumnSpan="2"
VerticalScrollBarVisibility="Auto">
<Grid>
<RichTextBox
Name="editor"
Margin="100,10"
Padding="50"

```

```

local:ViewModel.RichTextBox="richTextBoxText"
AcceptsTab="True"
Background="{Binding ElementName=shadingColorPicker, Path=Color,
Mode=OneWay, Converter={StaticResource ColorToBrushConverter}}"
BorderBrush="Transparent"
BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction Command="{Binding
Path=RichTextBoxSelectionChangedCommand}" />
</interactivity:EventTrigger>
<interactivity:EventTrigger EventName="PreviewMouseLeftButtonUp">
<interactivity:InvokeCommandAction Command="{Binding
Path=RichTextBoxPreviewMouseLeftButtonUpCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<FlowDocument
Name="flowDocumentName"
Background="{Binding ElementName=shadingColorPicker, Path=Color,
Mode=OneWay, Converter={StaticResource ColorToBrushConverter}}"
FontFamily="Calibri"
FontSize="14"
Foreground="{Binding ElementName=fontColorPicker, Path=Color, Mode=OneWay,
Converter={StaticResource ColorToBrushConverter}}"
TextAlignment="Left">
<Paragraph
FontFamily="Cambria"
FontSize="24"
FontWeight="Bold">
<Run Text="{Binding TitleText}" />
</Paragraph>
<Paragraph
x:Name="firstPara"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding FirstParagraphText}" />
</Paragraph>
<Paragraph
x:Name="secondPara"
FontFamily="Cambria"
FontSize="18">
<Run Text="{Binding SecondTitleText}" />
</Paragraph>
<List MarkerStyle="Square">
<ListItem>
<Paragraph
x:Name="pointOne"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointOne}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointTwo"
FontFamily="Calibri"
FontSize="15">

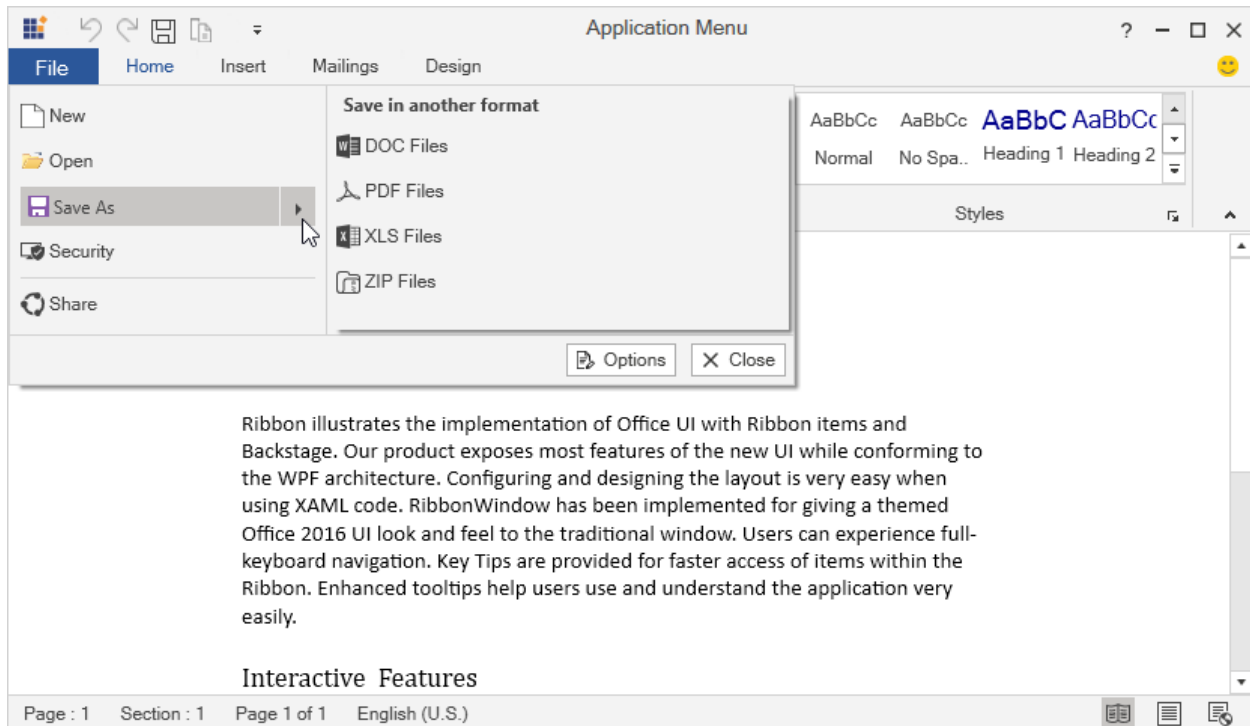
```

```
<Run Text="{Binding PointTwo}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointThree"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointThree}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointFour"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointFour}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointFive"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointFive}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointSix"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointSix}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointSeven"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointSeven}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointEight"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointEight}" />
</Paragraph>
</ListItem>
<ListItem>
<Paragraph
x:Name="pointNine"
FontFamily="Calibri"
FontSize="15">
<Run Text="{Binding PointNine}" />
```

```

</Paragraph>
</ListItem>
</List>
</FlowDocument>
</RichTextBox>
</Grid>
</ScrollViewer>
</Grid>
</syncfusion:RibbonWindow>

```



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

Ribbon Items

RibbonButton in WPF Ribbon

RibbonButton provides functionality similar to a normal button. Additionally, it comes in different sizes and can be easily placed inside the **RibbonBar** in the Ribbon control.

Setting various size modes

RibbonButton supports three types of size modes and it can be set using the [SizeForm](#) property. The different [SizeForm](#) available are as follows:

- **ExtraSmall** - Displays only the image in 16 * 16 size.
- **Small** - Displays the label and the image in 16 * 16 size.
- **Large** - Displays the label and the image in 32 * 32 size.

XML

```

<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.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:RibbonButton_IconTemp"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top" x:Name="ribbon"
ShowCustomizeRibbon="True">
<syncfusion:RibbonTab Caption="Home" IsChecked="True">
<syncfusion:RibbonBar Header="Clipboard">
<syncfusion:RibbonButton Label="Paste" SizeForm="Large"
LargeIcon="/Resources/Paste32.png" />
<syncfusion:RibbonButton Label="Cut" SizeForm="Small"
SmallIcon="/Resources/Cut16.png" />
<syncfusion:RibbonButton Label="Copy" SizeForm="Small"
SmallIcon="/Resources/Copy16.png" />
<syncfusion:RibbonButton Label="Format Painter" SizeForm="Small"
SmallIcon="/Resources/FormatPainter16.png" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Font" >
<syncfusion:RibbonButton Label="Bold" SmallIcon="/Resources/Bold16.png"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton Label="Italic" SmallIcon="/Resources/Italic16.png"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton Label="Underline"
SmallIcon="/Resources/Underline16.png" SizeForm="ExtraSmall"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Send / Receive"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
RibbonTab sendTab = new RibbonTab();
sendTab.Caption = "Send / Receive";
// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Clipboard";
// Creating items
RibbonButton pasteButton = new RibbonButton();
pasteButton.Label = "Paste";
pasteButton.SizeForm = SizeForm.Large;

```

```
pasteButton.LargeIcon = new BitmapImage(new Uri(@"Resources/Paste32.png",
UriKind.RelativeOrAbsolute));
RibbonButton cutButton = new RibbonButton();
cutButton.Label = "Cut";
cutButton.SizeForm = SizeForm.Small;
cutButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Cut16.png",
UriKind.RelativeOrAbsolute));
RibbonButton copyButton = new RibbonButton();
copyButton.Label = "Copy";
cutButton.SizeForm = SizeForm.Small;
copyButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Copy16.png",
UriKind.RelativeOrAbsolute));
RibbonButton formatButton = new RibbonButton();
formatButton.Label = "Format Painter";
cutButton.SizeForm = SizeForm.Small;
formatButton.SmallIcon = new BitmapImage(new
Uri(@"Resources/FormatPainter16.png", UriKind.RelativeOrAbsolute))
// Adding items to bar
clipboardBar.Items.Add(pasteButton);
clipboardBar.Items.Add(cutButton);
clipboardBar.Items.Add(copyButton);
clipboardBar.Items.Add(formatButton);
// Creating new bar
RibbonBar fontBar = new RibbonBar();
fontBar.Header = "Font";
// Creating items
RibbonButton boldButton = new RibbonButton();
boldButton.Label = "Bold";
boldButton.SizeForm = SizeForm.ExtraSmall;
boldButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Bold16.png",
UriKind.RelativeOrAbsolute));
RibbonButton italicButton = new RibbonButton();
italicButton.Label = "Italic";
italicButton.SizeForm = SizeForm.ExtraSmall;
italicButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Italic16.png",
UriKind.RelativeOrAbsolute));
RibbonButton underlineButton = new RibbonButton();
underlineButton.Label = "Underline";
underlineButton.SizeForm = SizeForm.ExtraSmall;
underlineButton.SmallIcon = new BitmapImage(new
Uri(@"Resources/Underline16.png", UriKind.RelativeOrAbsolute));
fontBar.Items.Add(boldButton);
fontBar.Items.Add(italicButton);
fontBar.Items.Add(underlineButton);
// Adding bars to the tabs
homeTab.Items.Add(clipboardBar);
homeTab.Items.Add(fontBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
ribbon.Items.Add(sendTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
```



Note: When **simplified** layout is set, [RibbonButton](#) displays the image in 20 * 20 size irrespective of the size form. Also, the text in the **Large** size form will appear to the right of the image.

Setting image to RibbonButton

The [RibbonButton](#) allows to display any type of image such as glyph, font or any custom content using the [IconTemplateSelector](#) and [IconTemplate](#) property, which are the preferred options. It also allows to display a normal image or vector image using the [IconType](#) enumeration property. The default value of the [IconType](#) property is **Icon**. The [IconType](#) enumeration has the following values:

- **Icon** - Gets the details of the icon from the [SmallIcon](#), [MediumIcon](#) or [LargeIcon](#) properties and sets it to the [RibbonButton](#).
- **VectorImage** - Gets the details of the icon path from the [VectorImage](#) property and sets it to the [RibbonButton](#).

Note: The [RibbonButton](#) loads icon in the following priority order,

- [IconTemplateSelector](#)
- [IconTemplate](#)
- [VectorImage](#)
- [LargeIcon](#)
- [MediumIcon](#)
- [SmallIcon](#)

Setting icon template selector

The [IconTemplateSelector](#) property provides support to specify a different data template based on the value of the [SizeForm](#) or [LayoutMode](#) properties. For simplified layout, the template content will be resized to 20 * 20 size which is the standard.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<syncfusion:RibbonWindow.Resources>
<DataTemplate x:Key="ribbonButtonSmallIconTemplate">
<Grid Width="14" Height="16">
```

```

<Path Width="7" Margin="0,7,0,0" HorizontalAlignment="Right"
Fill="#FF3A3A38" Stretch="Fill"
Data="F1M122,367L127,367L127,360L122,360z M128,368L121,368L121,359L128,359z"
/>
<Path Margin="0,2,2,0" Fill="#FFDE6C00" Stretch="Fill"
Data="M0,0 L12,0 12,4 11,4 11,0.99999994 1.00000002,0.99999994 1.00000002,13
6.00000001,13 6.00000001,14 0,14 z" />
<Path Margin="1,3,3,1" Fill="#FFF8DB8F" Stretch="Fill"
Data="M0,0 L10,0 10,3 9.00000001,3 9.00000001,0.99999994 1.00000001,0.99999994
1.00000001,1.5829999 1.00000001,2.5 1.00000001,11 5.00000001,11 5.00000001,12
0,12 z" />
<Path Margin="2.011,0.5,0.983,0.983" Fill="#FFFAFAFA" Stretch="Fill"
Data="M5.9873815,7.496151 L11.006,7.496151 11.006,14.516999
5.9873815,14.516999 z M0,5.4959998 L3.9880071,5.4964137 3.9880071,13.51695
0,13.51695 z M3.9889999,2.2337155E-15 C4.8170028,-4.4703477E-08
5.4889999,0.67098993 5.4889999,1.5 L5.4889999,2 7.4889999,2 7.4889999,5
0.4889999,5 0.4889999,2 2.4889999,2 2.4889999,1.5 C2.4889999,0.67098993
3.1609969,-4.4703477E-08 3.9889999,2.2337155E-15 z" />
<Path Height="6" Margin="2,0,4,0" VerticalAlignment="Top" Fill="#FF797774"
Stretch="Fill"
Data="M4,1 C3.447998,1 3,1.4490051 3,2 L3,3 1,3 1,5 7,5 7,3 5,3 5,2
C5,1.4490051 4.552002,1 4,1 z M4,0 C5.1029968,0 6,0.89700317 6,2 L8,2 8,6
0,6 0,2 2,2 C2,0.89700317 2.8970032,0 4,0 z" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="ribbonButtonLargeIconTemplate">
<Grid Margin="2">
<Path Data="M17,0 L21,0 21,7 20,7 20,1.00000001 17,1.00000001 z M0,0 L4,0
4,1.00000001 1.00000001,1.00000001 1.00000001,23 12,23 12,24 0,24 z"
Margin="0,4,6,2" Fill="#FFED8733" Stretch="Fill" />
<Path Data="M1.00000002,0.99999994 L1.00000002,17 13,17 13,0.99999994 z M0,0
L14,0 14,18 0,18 z"
Margin="13,11,0,0" Fill="#FF3C3B39" Stretch="Fill" />
<Path Data="M16,0 L17,0 19,0 19,6 17,6 17,2 16,2 z M0,0 L2,0 3,0 3,2 2,2
2,20 11,20 11,22 0,22 z"
Margin="1,5,7,3" Fill="#FFF8DB8F" Stretch="Fill"/>
<Path Data="M10.999956,12.5 L22.999956,12.5 22.999956,28.5 10.999956,28.5 z
M7.4999558,0 C9.1569382,0 10.499956,1.3439941 10.499956,3 L13.499956,3
13.499956,6.5 15,6.5 15,10.5 9.00000001,10.5 9.00000001,24.5 0,24.5 0,6.5
1.4999557,6.5 1.4999557,3 4.4999558,3 C4.4999558,1.3439941 5.8439499,0
7.4999558,0 z"
Margin="3,0.5,1,1" Fill="White" Stretch="Fill"/>
<Path Data="M6.5,0.99999996 C5.1209717,0.99999996 4,2.1209716 4,3.5 L4,4
0.99999994,4 0.99999994,7 12,7 12,4 9,4 9,3.5 C9,2.1209716
7.8790283,0.99999996 6.5,0.99999996 z M6.5,0 C8.2600098,-4.4703484E-08
9.7209473,1.3060302 9.9649658,3 L13,3 13,8 0,8 0,3 3.0350342,3
C3.2790527,1.3060302 4.7399902,-4.4703484E-08 6.5,0 z"
Height="8" Margin="4,0,10,0" VerticalAlignment="Top" Fill="#FF797774"
Stretch="Fill"/>
</Grid>
</DataTemplate>
<local:RibbonButtonIconTemplateSelector
x:Key="ribbonButtonIconTemplateSelector"
SmallTemplate="{StaticResource ribbonButtonSmallIconTemplate}"
LargeTemplate="{StaticResource ribbonButtonLargeIconTemplate}"/>
</syncfusion:RibbonWindow.Resources>
</Grid>

```

```

<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Clipboard">
<syncfusion:RibbonButton Label="Paste" SizeForm="Large"
IconTemplateSelector="{StaticResource ribbonButtonIconTemplateSelector}"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

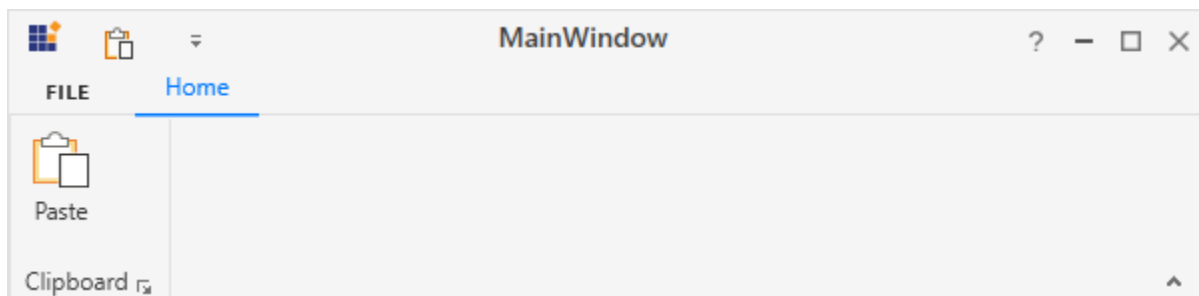
```

C#

```

public class RibbonButtonIconTemplateSelector : DataTemplateSelector
{
    public DataTemplate SmallTemplate { get; set; }
    public DataTemplate LargeTemplate { get; set; }
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        var item1 = (container as ContentPresenter);
        RibbonButton ribbonButton = (item1.TemplatedParent as RibbonButton);
        if (ribbonButton != null)
        {
            if (ribbonButton.SizeForm == SizeForm.Small || ribbonButton.SizeForm == SizeForm.ExtraSmall)
            {
                return SmallTemplate;
            }
            else if (ribbonButton.SizeForm == SizeForm.Large)
            {
                return LargeTemplate;
            }
        }
        return LargeTemplate;
    }
}

```



Note: [View sample in GitHub](#)

Setting icon template

The [IconTemplate](#) property provides support to set any type of image such as glyph, font or any custom content to the [RibbonButton](#). The [RibbonButton](#) will automatically resize the template content according to its [SizeForm](#). For simplified layout, the template content will be resized to 20 * 20 size which is the standard.

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.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:RibbonButton_IconTemp"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="800">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Clipboard">
<syncfusion:RibbonButton Label="Paste" SizeForm="Large">
<syncfusion:RibbonButton.IconTemplate>
<DataTemplate>
<Grid Margin="2">
<Path
Data="M17,0 L21,0 21,7 20,7 20,1.0000001 17,1.0000001 z M0,0 L4,0
4,1.0000001 1.0000001,1.0000001 1.0000001,23 12,23 12,24 0,24 z"
Margin="0,4,6,2" Fill="#FFED8733" Stretch="Fill" />
<Path
Data="M1.0000002,0.99999994 L1.0000002,17 13,17 13,0.99999994 z M0,0 L14,0
14,18 0,18 z"
Margin="13,11,0,0" Fill="#FF3C3B39" Stretch="Fill" />
<Path
Data="M16,0 L17,0 19,0 19,6 17,6 17,2 16,2 z M0,0 L2,0 3,0 3,2 2,2 2,20
11,20 11,22 0,22 z"
Margin="1,5,7,3" Fill="#FFF8DB8F" Stretch="Fill"/>
<Path
Data="M10.999956,12.5 L22.999956,12.5 22.999956,28.5 10.999956,28.5 z
M7.4999558,0 C9.1569382,0 10.499956,1.3439941 10.499956,3 L13.499956,3
13.499956,6.5 15,6.5 15,10.5 9.0000001,10.5 9.0000001,24.5 0,24.5 0,6.5
1.4999557,6.5 1.4999557,3 4.4999558,3 C4.4999558,1.3439941 5.8439499,0
7.4999558,0 z"
Margin="3,0.5,1,1" Fill="White" Stretch="Fill"/>
<Path
Data="M6.5,0.99999996 C5.1209717,0.99999996 4,2.1209716 4,3.5 L4,4
0.99999994,4 0.99999994,7 12,7 12,4 9,4 9,3.5 C9,2.1209716
7.8790283,0.99999996 6.5,0.99999996 z M6.5,0 C8.2600098,-4.4703484E-08
9.7209473,1.3060302 9.9649658,3 L13,3 13,8 0,8 0,3 3.0350342,3
C3.2790527,1.3060302 4.7399902,-4.4703484E-08 6.5,0 z"
Height="8" Margin="4,0,10,0" VerticalAlignment="Top" Fill="#FF797774"
Stretch="Fill"/>
</Grid>
</DataTemplate>
```

```

</syncfusion:RibbonButton.IconTemplate>
</syncfusion:RibbonButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

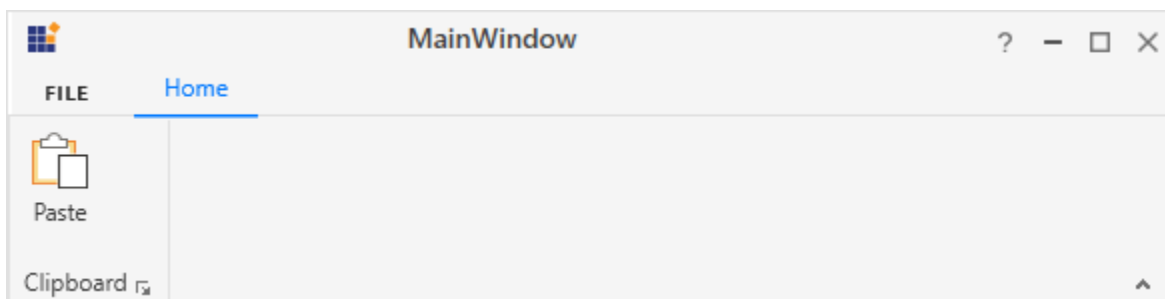
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
RibbonTab sendTab = new RibbonTab();
sendTab.Caption = "Send / Receive";
// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Clipboard";
// Creating items
RibbonButton pasteButton = new RibbonButton();
pasteButton.Label = "Paste";
pasteButton.SizeForm = SizeForm.Large;
DataTemplate iconDataTemplate = new DataTemplate();
FrameworkElementFactory gridElement = new
FrameworkElementFactory(typeof(Grid));
FrameworkElementFactory pathElement1 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement2 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement3 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement4 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement5 = new
FrameworkElementFactory(typeof(Path));
gridElement.SetValue(Grid.MarginProperty, new Thickness(2));
pathElement1.SetValue(Path.DataProperty, Geometry.Parse("M17,0 L21,0 21,7
20,7 20,1.0000001 17,1.0000001 z M0,0 L4,0 4,1.0000001 1.0000001,1.0000001
1.0000001,23 12,23 12,24 0,24 z"));
pathElement1.SetValue(Path.MarginProperty, new Thickness(0, 4, 6, 2));
pathElement1.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(237, 135, 51)));
pathElement1.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement2.SetValue(Path.DataProperty,
Geometry.Parse("M1.0000002,0.99999994 L1.0000002,17 13,17 13,0.99999994 z
M0,0 L14,0 14,18 0,18 z"));
pathElement2.SetValue(Path.MarginProperty, new Thickness(13, 11, 0, 0));
pathElement2.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(60, 59, 57)));
pathElement2.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement3.SetValue(Path.DataProperty, Geometry.Parse("M16,0 L17,0 19,0
19,6 17,6 17,2 16,2 z M0,0 L2,0 3,0 3,2 2,2 2,20 11,20 11,22 0,22 z"));
pathElement3.SetValue(Path.MarginProperty, new Thickness(1, 5, 7, 3));

```

```

pathElement3.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(248, 219, 143)));
pathElement3.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement4.SetValue(Path.DataProperty, Geometry.Parse("M10.999956,12.5
L22.999956,12.5 22.999956,28.5 10.999956,28.5 z M7.4999558,0 C9.1569382,0
10.499956,1.3439941 10.499956,3 L13.499956,3 13.499956,6.5 15,6.5 15,10.5
9.0000001,10.5 9.0000001,24.5 0,24.5 0,6.5 1.4999557,6.5 1.4999557,3
4.4999558,3 C4.4999558,1.3439941 5.8439499,0 7.4999558,0 z"));
pathElement4.SetValue(Path.MarginProperty, new Thickness(3, 0.5, 1, 1));
pathElement4.SetValue(Path.FillProperty, new SolidColorBrush(Colors.White));
pathElement4.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement5.SetValue(Path.DataProperty, Geometry.Parse("M6.5,0.99999996
C5.1209717,0.99999996 4,2.1209716 4,3.5 L4,4 0.99999994,4 0.99999994,7 12,7
12,4 9,4 9,3.5 C9,2.1209716 7.8790283,0.99999996 6.5,0.99999996 z M6.5,0
C8.2600098,-4.4703484E-08 9.7209473,1.3060302 9.9649658,3 L13,3 13,8 0,8 0,3
3.0350342,3 C3.2790527,1.3060302 4.7399902,-4.4703484E-08 6.5,0 z"));
pathElement5.SetValue(Path.MarginProperty, new Thickness(4, 0, 10, 0));
pathElement5.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(121, 119, 116)));
pathElement5.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement5.SetValue(Path.HeightProperty, (double)8);
pathElement5.SetValue(Path.VerticalAlignmentProperty,
VerticalAlignment.Top);
gridElement.AppendChild(pathElement1);
gridElement.AppendChild(pathElement2);
gridElement.AppendChild(pathElement3);
gridElement.AppendChild(pathElement4);
gridElement.AppendChild(pathElement5);
iconDataTemplate.VisualTree = gridElement;
pasteButton.IconTemplate = iconDataTemplate;
clipboardBar.Items.Add(pasteButton);
// Adding bars to the tabs
homeTab.Items.Add(clipboardBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
ribbon.Items.Add(sendTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Note: [View sample in GitHub](#)

Setting image path

The [RibbonButton](#) allows to set the image according to the different [SizeForm](#) values. To set the image to [RibbonButton](#), the following properties are used:

- [SmallIcon](#) - 16 * 16 size image to be displayed in normal layout for “ExtraSmall” and “Small” size form.
- [MediumIcon](#) - 20 * 20 size image to be displayed in simplified layout.
- [LargeIcon](#) - 32 * 32 size image to be displayed in normal layout for “Large” size form.

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.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:RibbonButton_IconTemp"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top" x:Name="ribbon"
EnableSimplifiedLayoutMode="True" LayoutMode="Simplified"
ShowCustomizeRibbon="True">
<syncfusion:RibbonTab Caption="Home" IsChecked="True">
<syncfusion:RibbonBar Header="Clipboard">
<syncfusion:RibbonButton Label="Paste"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Simplified"
SizeForm="Large" MediumIcon="/Resources/Paste20.png"
LargeIcon="/Resources/Paste32.png" />
<syncfusion:RibbonButton Label="Cut" SizeForm="Small"
MediumIcon="/Resources/Cut_20.png" SmallIcon="/Resources/Cut16.png" />
<syncfusion:RibbonButton Label="Copy" SizeForm="Small"
MediumIcon="/Resources/Copy_20.png" SmallIcon="/Resources/Copy16.png" />
<syncfusion:RibbonButton Label="Format Painter" SizeForm="Small"
MediumIcon="/Resources/FormatPainter20.png"
SmallIcon="/Resources/FormatPainter16.png" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Font" >
<syncfusion:RibbonButton Label="Bold" SmallIcon="/Resources/Bold16.png"
MediumIcon="/Resources/Bold_20.png" SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton Label="Italic" SmallIcon="/Resources/Italic16.png"
MediumIcon="/Resources/Italic_20.png" SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton Label="Underline"
SmallIcon="/Resources/Underline16.png"
MediumIcon="/Resources/Underline_20.png" SizeForm="ExtraSmall"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Send / Receive"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
RibbonTab sendTab = new RibbonTab();
sendTab.Caption = "Send / Receive";
// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Clipboard";
// Creating items
RibbonButton pasteButton = new RibbonButton();
pasteButton.Label = "Paste";
pasteButton.SizeForm = SizeForm.Large;
pasteButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Paste20.png",
UriKind.RelativeOrAbsolute));
pasteButton.LargeIcon = new BitmapImage(new Uri(@"Resources/Paste32.png",
UriKind.RelativeOrAbsolute));
RibbonButton cutButton = new RibbonButton();
cutButton.Label = "Cut";
cutButton.SizeForm = SizeForm.Small;
cutButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Cut_20.png",
UriKind.RelativeOrAbsolute));
cutButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Cut16.png",
UriKind.RelativeOrAbsolute));
RibbonButton copyButton = new RibbonButton();
copyButton.Label = "Copy";
copyButton.SizeForm = SizeForm.Small;
copyButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Copy_20.png",
UriKind.RelativeOrAbsolute));
copyButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Copy16.png",
UriKind.RelativeOrAbsolute));
RibbonButton formatButton = new RibbonButton();
formatButton.Label = "Format Painter";
formatButton.SizeForm = SizeForm.Small;
formatButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/FormatPainter20.png", UriKind.RelativeOrAbsolute));
formatButton.SmallIcon = new BitmapImage(new
Uri(@"Resources/FormatPainter16.png", UriKind.RelativeOrAbsolute));
// Adding items to bar
clipboardBar.Items.Add(pasteButton);
clipboardBar.Items.Add(cutButton);
clipboardBar.Items.Add(copyButton);
clipboardBar.Items.Add(formatButton);
// Creating new bar
RibbonBar fontBar = new RibbonBar();
fontBar.Header = "Font";
// Creating items
RibbonButton boldButton = new RibbonButton();
boldButton.Label = "Bold";
boldButton.SizeForm = SizeForm.ExtraSmall;
boldButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Bold16.png",
UriKind.RelativeOrAbsolute));
```

```

boldButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Bold_20.png",
UriKind.RelativeOrAbsolute));
RibbonButton italicButton = new RibbonButton();
italicButton.Label = "Italic";
italicButton.SizeForm = SizeForm.ExtraSmall;
italicButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Italic16.png",
UriKind.RelativeOrAbsolute));
italicButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/Italic_20.png", UriKind.RelativeOrAbsolute));
RibbonButton underlineButton = new RibbonButton();
underlineButton.Label = "Underline";
underlineButton.SizeForm = SizeForm.ExtraSmall;
underlineButton.SmallIcon = new BitmapImage(new
Uri(@"Resources/Underline16.png", UriKind.RelativeOrAbsolute));
underlineButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/Underline_20.png", UriKind.RelativeOrAbsolute));
fontBar.Items.Add(boldButton);
fontBar.Items.Add(italicButton);
fontBar.Items.Add(underlineButton);
// Adding bars to the tabs
homeTab.Items.Add(clipboardBar);
homeTab.Items.Add(fontBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
ribbon.Items.Add(sendTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Normal layout



Simplified layout

Setting vector image

The [VectorImage](#) property is of type `ObservableCollection<Path>` which allows the image to be set as path type. The [RibbonButton](#) will automatically resize the image according to its [SizeForm](#). For simplified layout, the image will be resized to 20 * 20 size which is the standard.

Note: The [IconTemplateSelector](#) and [IconTemplate](#) properties are the preferred options to set any type of image such as glyph, font or any custom content when compared to the [VectorImage](#) property.

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton IconTemp.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:RibbonButton_IconTemp"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" skin:SfSkinManager.VisualStyle="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top" x:Name="ribbon"
EnableSimplifiedLayoutMode="True" LayoutMode="Simplified"
ShowCustomizeRibbon="True">
<syncfusion:RibbonTab Caption="Home" IsChecked="True">
<syncfusion:RibbonBar Header="Clipboard">
<syncfusion:RibbonButton Label="Paste" SizeForm="Large"
IconType="VectorImage">
<syncfusion:RibbonButton.VectorImage>
<Path Data="M17,0 L21,0 21,7 20,7 20,1.0000001 17,1.0000001 z M0,0 L4,0
4,1.0000001 1.0000001,1.0000001 1.0000001,23 12,23 12,24 0,24 z"
Margin="0,4,6,2" Fill="#FFED8733" Stretch="Fill" />
<Path Data="M1.0000002,0.99999994 L1.0000002,17 13,17 13,0.99999994 z M0,0
L14,0 14,18 0,18 z"
Margin="13,11,0,0" Fill="#FF3C3B39" Stretch="Fill" />
<Path Data="M16,0 L17,0 19,0 19,6 17,6 17,2 16,2 z M0,0 L2,0 3,0 3,2 2,2
2,20 11,20 11,22 0,22 z"
Margin="1,5,7,3" Fill="#FFF8DB8F" Stretch="Fill"/>
<Path Data="M10.999956,12.5 L22.999956,12.5 22.999956,28.5 10.999956,28.5 z
M7.4999558,0 C9.1569382,0 10.499956,1.3439941 10.499956,3 L13.499956,3
13.499956,6.5 15,6.5 15,10.5 9.0000001,10.5 9.0000001,24.5 0,24.5 0,6.5
1.4999557,6.5 1.4999557,3 4.4999558,3 C4.4999558,1.3439941 5.8439499,0
7.4999558,0 z"
Margin="3,0.5,1,1" Fill="White" Stretch="Fill"/>
<Path Data="M6.5,0.99999996 C5.1209717,0.99999996 4,2.1209716 4,3.5 L4,4
0.99999994,4 0.99999994,7 12,7 12,4 9,4 9,3.5 C9,2.1209716
7.8790283,0.99999996 6.5,0.99999996 z M6.5,0 C8.2600098,-4.4703484E-08
9.7209473,1.3060302 9.9649658,3 L13,3 13,8 0,8 0,3 3.0350342,3
C3.2790527,1.3060302 4.7399902,-4.4703484E-08 6.5,0 z"
Height="8" Margin="4,0,10,0" VerticalAlignment="Top" Fill="#FF797774"
Stretch="Fill"/>
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton Label="Cut" SizeForm="Small" IconType="VectorImage"
>
<syncfusion:RibbonButton.VectorImage>
<Path Width="10" Height="8" Margin="3.747,0,1.805,4.614" Stretch="Fill"
Data="M0.4800034,0 L3.2370005,5.6329948 5.9950049,0 6.4480002,1.3919982
3.8000043,6.7859942 6.4040015,12.108999 5.4240053,12.385 3.2370005,7.9400011
1.0859987,12.314001 0,12.249991 2.675004,6.7859942 0.027000348,1.3919982 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"/>
<Path Width="13" Height="4" Margin="2,0,0,1" VerticalAlignment="Bottom"
Data="M2.0000019,1.0000033 C1.4480005,1.0000033 1.0000028,1.4489932
1.0000028,2.0000033 1.0000028,2.5509982 1.4480005,3.0000033
2.0000019,3.0000033 2.5519957,3.0000033 3.0000009,2.5509982
3.0000009,2.0000033 3.0000009,1.4489932 2.5519957,1.0000033

```

```

2.0000019,1.0000033 z M7.9999966,0.9999999 C7.4479966,0.99999993
6.9999966,1.449 6.9999966,2 6.9999966,2.5509999 7.4479966,3 7.9999966,3
8.5519962,3 8.9999962,2.5509999 8.9999962,2 8.9999962,1.449
8.5519962,0.99999993 7.9999966,0.9999999 z M2.0000019,3.2782542E-06
C3.1029978,3.3312692E-06 4,0.89700651 4,2.0000033 4,3.1030002
3.1029978,4.0000033 2.0000019,4.0000033 0.8969985,4.0000033 0,3.1030002
0,2.0000033 0,0.89700651 0.8969985,3.3312692E-06 2.0000019,3.2782542E-06 z
M7.9999966,0 C9.1029968,-3.7871359E-08 9.9999962,0.89699995 9.9999962,2
9.9999962,3.1029999 9.1029968,4 7.9999966,4 6.8969965,4 5.9999966,3.1029999
5.9999966,2 5.9999966,0.89699995 6.8969965,-3.7871359E-08 7.9999966,0 z"
Fill="#FF1D8BCC" Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton Label="Copy" SizeForm="Small"
IconType="VectorImage" >
<syncfusion:RibbonButton.VectorImage>
<Path Margin="3,1,0.5,0.5" Fill="White" Stretch="Fill"
Data="M5.5000009,2.500005 L10.500001,2.500005 14.500001,6.500005
14.500001,14.500005 5.5000009,14.500005 z M0,0 L4.0000037,0 4.0000037,12
0,12 z"/>
<Path Margin="2,0,0,0" Fill="#FF3A3939" Stretch="Fill"
Data="M9.0000026,11.999999 L13.000003,11.999999 13.000003,12.999999
9.0000026,12.999999 z M9.0000026,9.9999986 L13.000003,9.9999986
13.000003,10.999999 9.0000026,10.999999 z M12,4.7070035 L12,7.0000033
14.293,7.0000033 z M6.9999967,4.0000001 L6.9999967,15 14.999997,15
14.999997,8.0000033 11,8.0000033 11,4.0000001 z M5.9999967,2.9999999
L11.706997,2.9999999 15.999997,7.293 15.999997,16 5.9999967,16 z M0,0
L6.9999967,0 6.9999967,2 5.9999971,2 5.9999971,1 1,1 1,13 4.9999976,13
4.9999976,14 0,14 z" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton Label="Format Painter" SizeForm="Small"
IconType="VectorImage">
<syncfusion:RibbonButton.VectorImage>
<Path Data="M13.434632,4.827646E-05 C13.680801,0.0024180976
13.926775,0.092046673 14.121765,0.27108389 14.544701,0.66010981
14.565695,1.3190858 14.167659,1.7340507 L11.531004,4.480899
13.369993,6.3119885 11.726018,7.9489524 6.8310282,12.794981 0,5.9919826
3.8440161,5.3719827 6.5970273,2.8419837 11.564087,7.7877166
6.5979662,2.8409979 8.2409729,1.2050026 10.105034,3.0610356
12.670845,0.32509833 C12.8764,0.10675568 13.155641,-0.0026375318
13.434632,4.827646E-05 z"
Margin="2.048,0.498,0.501,0.705" Fill="White" Stretch="Fill" />
<Path Data="M2.3529817,2.4090486 L1.417989,3.3400479 5.8379548,7.7410448
6.7719474,6.8110455 z M7.5172076,0.99873667 C7.3818266,1.0037418
7.2480633,1.0597443 7.1480229,1.165737 L4.9139335,3.5478707
5.6360054,4.2659228 7.9211223,1.8858034 C8.0180495,1.782771
8.0700533,1.6487924 8.0661471,1.5067568 8.0611417,1.3647821
8.0010812,1.2347711 7.8960969,1.1377204 7.7895868,1.0397238
7.6525886,0.99373155 7.5172076,0.99873667 z M7.5527165,7.1653047E-05
C7.9184291,0.0035863224 8.2838595,0.13640766 8.5721467,0.40172305
8.8770893,0.68072825 9.0521443,1.0617281 9.0650842,1.4747729
9.0791228,1.8877565 8.9290931,2.2788277 8.6430719,2.5777922
L6.3439374,4.9723075 8.1899364,6.8110455 5.8379548,9.153044 0,3.3400479
2.3529817,0.99704962 4.205102,2.8418849 6.4200914,0.48174404
C6.7234051,0.15819254 7.1382422,-0.0039115331 7.5527165,7.1653047E-05 z"
Fill="#FF484644" Margin="7.936,0,0,4.845" Stretch="Fill" />

```

```

<Path Data="M0.77698034,0 L10.585,3.2109802 6.8309581,6.9279997 0,0.12499928
z"
Fill="#FFF8DB8F" Margin="2.049,6.365,4.368,0.705" Stretch="Fill" />
<Path Data="M2.3239305,4.4059882 L7.8809197,9.9390366 10.69988,7.1480067 z
M7.630995,1.3829954 L5.1189968,3.6919881 3.9397421,3.8823536
11.49895,6.3573499 12.063992,5.7979813 z M7.657995,0 L13.483991,5.7999814
7.9662354,11.262604 7.8789665,11.349017 7.8779948,11.349963 0,3.5039886
4.6639969,2.7519911 z"
Fill="#FFEE9243" Margin="1,2.648,2.518,0" Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Send / Receive"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
RibbonTab sendTab = new RibbonTab();
sendTab.Caption = "Send / Receive";
// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Clipboard";
// Creating items
RibbonButton pasteButton = new RibbonButton();
pasteButton.Label = "Paste";
pasteButton.SizeForm = SizeForm.Large;
pasteButton.IconType = IconType.VectorImage;
Path pastePath1 = new Path();
pastePath1.Data = Geometry.Parse("M17,0 L21,0 21,7 20,7 20,1.0000001
17,1.0000001 z M0,0 L4,0 4,1.0000001 1.0000001,1.0000001 1.0000001,23 12,23
12,24 0,24 z");
pastePath1.Fill = new SolidColorBrush(Color.FromRgb(237, 135, 51));
pastePath1.Margin = new Thickness(0, 4, 6, 2);
pastePath1.Stretch = Stretch.Fill;
Path pastePath2 = new Path();
pastePath2.Data = Geometry.Parse("M1.0000002,0.99999994 L1.0000002,17 13,17
13,0.99999994 z M0,0 L14,0 14,18 0,18 z");
pastePath2.Fill = new SolidColorBrush(Color.FromRgb(60, 59, 57));
pastePath2.Margin = new Thickness(13, 11, 0, 0);
pastePath2.Stretch = Stretch.Fill;
Path pastePath3 = new Path();

```

```

pastePath3.Data = Geometry.Parse("M16,0 L17,0 19,0 19,6 17,6 17,2 16,2 z
M0,0 L2,0 3,0 3,2 2,2 2,20 11,20 11,22 0,22 z");
pastePath3.Fill = new SolidColorBrush(Color.FromRgb(248, 219, 143));
pastePath3.Margin = new Thickness(1, 5, 7, 3);
pastePath3.Stretch = Stretch.Fill;
Path pastePath4 = new Path();
pastePath4.Data = Geometry.Parse("M10.999956,12.5 L22.999956,12.5
22.999956,28.5 10.999956,28.5 z M7.4999558,0 C9.1569382,0
10.499956,1.3439941 10.499956,3 L13.499956,3 13.499956,6.5 15,6.5 15,10.5
9.0000001,10.5 9.0000001,24.5 0,24.5 0,6.5 1.4999557,6.5 1.4999557,3
4.4999558,3 C4.4999558,1.3439941 5.8439499,0 7.4999558,0 z");
pastePath4.Fill = new SolidColorBrush(Colors.White);
pastePath4.Margin = new Thickness(3, 0.5, 1, 1);
pastePath4.Stretch = Stretch.Fill;
Path pastePath5 = new Path();
pastePath5.Data = Geometry.Parse("M6.5,0.99999996 C5.1209717,0.99999996
4,2.1209716 4,3.5 L4,4 0.99999994,4 0.99999994,7 12,7 12,4 9,4 9,3.5
C9,2.1209716 7.8790283,0.99999996 6.5,0.99999996 z M6.5,0 C8.2600098,-
4.4703484E-08 9.7209473,1.3060302 9.9649658,3 L13,3 13,8 0,8 0,3 3.0350342,3
C3.2790527,1.3060302 4.7399902,-4.4703484E-08 6.5,0 z");
pastePath5.Fill = new SolidColorBrush(Color.FromRgb(121, 119, 116));
pastePath5.Margin = new Thickness(4, 0, 10, 0);
pastePath5.Stretch = Stretch.Fill;
pastePath5.Height = 8;
pastePath5.VerticalAlignment = VerticalAlignment.Top;
pasteButton.VectorImage.Add(pastePath1);
pasteButton.VectorImage.Add(pastePath2);
pasteButton.VectorImage.Add(pastePath3);
pasteButton.VectorImage.Add(pastePath4);
pasteButton.VectorImage.Add(pastePath5);
RibbonButton cutButton = new RibbonButton();
cutButton.Label = "Cut";
cutButton.SizeForm = SizeForm.Small;
cutButton.IconType = IconType.VectorImage;
Path cutPath1 = new Path();
cutPath1.Data = Geometry.Parse("M0.4800034,0 L3.2370005,5.6329948
5.9950049,0 6.4480002,1.3919982 3.8000043,6.7859942 6.4040015,12.108999
5.4240053,12.385 3.2370005,7.9400011 1.0859987,12.314001 0,12.249991
2.675004,6.7859942 0.027000348,1.3919982 z");
cutPath1.Fill = new SolidColorBrush(Colors.Black);
cutPath1.Margin = new Thickness(3.747, 0, 1.805, 4.614);
cutPath1.Stretch = Stretch.Fill;
cutPath1.Height = 8;
cutPath1.Width = 10;
Path cutPath2 = new Path();
cutPath2.Data = Geometry.Parse("M2.0000019,1.0000033 C1.4480005,1.0000033
1.0000028,1.4489932 1.0000028,2.0000033 1.0000028,2.5509982
1.4480005,3.0000033 2.0000019,3.0000033 2.5519957,3.0000033
3.0000009,2.5509982 3.0000009,2.0000033 3.0000009,1.4489932
2.5519957,1.0000033 2.0000019,1.0000033 z M7.9999966,0.99999999
C7.4479966,0.99999993 6.9999966,1.449 6.9999966,2 6.9999966,2.5509999
7.4479966,3 7.9999966,3 8.5519962,3 8.9999962,2.5509999 8.9999962,2
8.9999962,1.449 8.5519962,0.99999993 7.9999966,0.99999999 z
M2.0000019,3.2782542E-06 C3.1029978,3.3312692E-06 4,0.89700651 4,2.0000033
4,3.1030002 3.1029978,4.0000033 2.0000019,4.0000033 0.8969985,4.0000033
0,3.1030002 0,2.0000033 0,0.89700651 0.8969985,3.3312692E-06
2.0000019,3.2782542E-06 z M7.9999966,0 C9.1029968,-3.7871359E-08

```

```

9.9999962,0.89699995 9.9999962,2 9.9999962,3.1029999 9.1029968,4 7.9999966,4
6.8969965,4 5.9999966,3.1029999 5.9999966,2 5.9999966,0.89699995 6.8969965,-
3.7871359E-08 7.9999966,0 z");
cutPath2.Fill = new SolidColorBrush(Color.FromRgb(29, 139, 204));
cutPath2.Margin = new Thickness(2, 0, 0, 1);
cutPath2.Stretch = Stretch.Fill;
cutPath2.VerticalAlignment = VerticalAlignment.Bottom;
cutPath2.Height = 4;
cutPath2.Width = 13;
cutButton.VectorImage.Add(cutPath1);
cutButton.VectorImage.Add(cutPath2);
RibbonButton copyButton = new RibbonButton();
copyButton.Label = "Copy";
copyButton.SizeForm = SizeForm.Small;
copyButton.IconType = IconType.VectorImage;
Path copyPath1 = new Path();
copyPath1.Data = Geometry.Parse("M5.5000009,2.500005 L10.500001,2.500005
14.500001,6.500005 14.500001,14.500005 5.500009,14.500005 z M0,0
L4.0000037,0 4.0000037,12 0,12 z");
copyPath1.Fill = new SolidColorBrush(Colors.White);
copyPath1.Margin = new Thickness(3, 1, 0.5, 0.5);
copyPath1.Stretch = Stretch.Fill;
Path copyPath2 = new Path();
copyPath2.Data = Geometry.Parse("M9.0000026,11.999999 L13.000003,11.999999
13.000003,12.999999 9.0000026,12.999999 z M9.0000026,9.9999986
L13.000003,9.9999986 13.000003,10.999999 9.0000026,10.999999 z M12,4.7070035
L12,7.0000033 14.293,7.0000033 z M6.9999967,4.0000001 L6.9999967,15
14.999997,15 14.999997,8.0000033 11,8.0000033 11,4.0000001 z
M5.9999967,2.9999999 L11.706997,2.9999999 15.999997,7.293 15.999997,16
5.9999967,16 z M0,0 L6.9999967,0 6.9999967,2 5.9999971,2 5.9999971,1 1,1
1,13 4.9999976,13 4.9999976,14 0,14 z");
copyPath2.Fill = new SolidColorBrush(Color.FromRgb(58, 57, 57));
copyPath2.Margin = new Thickness(2, 0, 0, 0);
copyPath2.Stretch = Stretch.Fill;
copyButton.VectorImage.Add(copyPath1);
copyButton.VectorImage.Add(copyPath2);
RibbonButton formatButton = new RibbonButton();
formatButton.Label = "Format Painter";
formatButton.SizeForm = SizeForm.Small;
formatButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/FormatPainter20.png", UriKind.RelativeOrAbsolute));
formatButton.SmallIcon = new BitmapImage(new
Uri(@"Resources/FormatPainter16.png", UriKind.RelativeOrAbsolute));
Path formatPath1 = new Path();
formatPath1.Data = Geometry.Parse("M13.434632,4.827646E-05
C13.680801,0.0024180976 13.926775,0.092046673 14.121765,0.27108389
14.544701,0.66010981 14.565695,1.3190858 14.167659,1.7340507
L11.531004,4.480899 13.369993,6.3119885 11.726018,7.9489524
6.8310282,12.794981 0,5.9919826 3.8440161,5.3719827 6.5970273,2.8419837
11.564087,7.7877166 6.5979662,2.8409979 8.2409729,1.2050026
10.105034,3.0610356 12.670845,0.32509833 C12.8764,0.10675568 13.155641,-
0.0026375318 13.434632,4.827646E-05 z");
formatPath1.Fill = new SolidColorBrush(Colors.White);
formatPath1.Margin = new Thickness(2.048, 0.498, 0.501, 0.705);
formatPath1.Stretch = Stretch.Fill;
Path formatPath2 = new Path();

```



```

formatPath2.Data = Geometry.Parse("M2.3529817,2.4090486 L1.417989,3.3400479
5.8379548,7.7410448 6.7719474,6.8110455 z M7.5172076,0.99873667
C7.3818266,1.0037418 7.2480633,1.0597443 7.1480229,1.165737
L4.9139335,3.5478707 5.6360054,4.2659228 7.9211223,1.8858034
C8.0180495,1.782771 8.0700533,1.6487924 8.0661471,1.5067568
8.0611417,1.3647821 8.0010812,1.2347711 7.8960969,1.1377204
7.7895868,1.0397238 7.6525886,0.99373155 7.5172076,0.99873667 z
M7.5527165,7.1653047E-05 C7.9184291,0.0035863224 8.2838595,0.13640766
8.5721467,0.40172305 8.8770893,0.68072825 9.0521443,1.0617281
9.0650842,1.4747729 9.0791228,1.8877565 8.9290931,2.2788277
8.6430719,2.5777922 L6.3439374,4.9723075 8.1899364,6.8110455
5.8379548,9.153044 0,3.3400479 2.3529817,0.99704962 4.205102,2.8418849
6.4200914,0.48174404 C6.7234051,0.15819254 7.1382422,-0.0039115331
7.5527165,7.1653047E-05 z");
formatPath2.Fill = new SolidColorBrush(Color.FromRgb(72, 70, 68));
formatPath2.Margin = new Thickness(7.936, 0, 0, 4.845);
formatPath2.Stretch = Stretch.Fill;
Path formatPath3 = new Path();
formatPath3.Data = Geometry.Parse("M0.77698034,0 L10.585,3.2109802
6.8309581,6.9279997 0,0.12499928 z");
formatPath3.Fill = new SolidColorBrush(Color.FromRgb(248, 219, 143));
formatPath3.Margin = new Thickness(2.049, 6.365, 4.368, 0.705);
formatPath3.Stretch = Stretch.Fill;
Path formatPath4 = new Path();
formatPath4.Data = Geometry.Parse("M2.3239305,4.4059882 L7.8809197,9.9390366
10.69988,7.1480067 z M7.630995,1.3829954 L5.1189968,3.6919881
3.9397421,3.8823536 11.49895,6.3573499 12.063992,5.7979813 z M7.657995,0
L13.483991,5.7999814 7.9662354,11.262604 7.8789665,11.349017
7.8779948,11.349963 0,3.5039886 4.6639969,2.7519911 z");
formatPath4.Fill = new SolidColorBrush(Color.FromRgb(238, 146, 67));
formatPath4.Margin = new Thickness(1, 2.648, 2.518, 0);
formatPath4.Stretch = Stretch.Fill;
formatButton.VectorImage.Add(formatPath1);
formatButton.VectorImage.Add(formatPath2);
formatButton.VectorImage.Add(formatPath3);
formatButton.VectorImage.Add(formatPath4);
// Adding items to bar
clipboardBar.Items.Add(pasteButton);
clipboardBar.Items.Add(cutButton);
clipboardBar.Items.Add(copyButton);
clipboardBar.Items.Add(formatButton);
// Creating new bar
RibbonBar fontBar = new RibbonBar();
fontBar.Header = "Font";
// Creating items
RibbonButton boldButton = new RibbonButton();
boldButton.Label = "Bold";
boldButton.SizeForm = SizeForm.ExtraSmall;
boldButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Bold16.png",
UriKind.RelativeOrAbsolute));
boldButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Bold_20.png",
UriKind.RelativeOrAbsolute));
RibbonButton italicButton = new RibbonButton();
italicButton.Label = "Italic";
italicButton.SizeForm = SizeForm.ExtraSmall;
italicButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Italic16.png",
UriKind.RelativeOrAbsolute));

```

```

italicButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/Italic_20.png", UriKind.RelativeOrAbsolute));
RibbonButton underlineButton = new RibbonButton();
underlineButton.Label = "Underline";
underlineButton.SizeForm = SizeForm.ExtraSmall;
underlineButton.SmallIcon = new BitmapImage(new
Uri(@"Resources/Underline16.png", UriKind.RelativeOrAbsolute));
underlineButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/Underline_20.png", UriKind.RelativeOrAbsolute));
fontBar.Items.Add(boldButton);
fontBar.Items.Add(italicButton);
fontBar.Items.Add(underlineButton);
// Adding bars to the tabs
homeTab.Items.Add(clipboardBar);
homeTab.Items.Add(fontBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
ribbon.Items.Add(sendTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Normal layout



Simplified layout

Add RibbonButton to the simplified layout

When the simplified layout is enabled, the [RibbonButton](#) can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```

<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.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:RibbonButton_IconTemp"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" skin:SfSkinManager.VisualStyle="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">

```

```

<Grid>
<syncfusion:Ribbon VerticalAlignment="Top" x:Name="ribbon"
EnableSimplifiedLayoutMode="True" LayoutMode="Simplified"
ShowCustomizeRibbon="True">
<syncfusion:RibbonTab Caption="Home" IsChecked="True">
<syncfusion:RibbonBar Header="Clipboard">
<syncfusion:RibbonButton Label="Paste" SizeForm="Large"
MediumIcon="/Resources/Paste20.png" LargeIcon="/Resources/Paste32.png" />
<syncfusion:RibbonButton Label="Cut" SizeForm="Small"
MediumIcon="/Resources/Cut_20.png" SmallIcon="/Resources/Cut16.png" />
<syncfusion:RibbonButton Label="Copy" SizeForm="Small"
MediumIcon="/Resources/Copy_20.png" SmallIcon="/Resources/Copy16.png" />
<syncfusion:RibbonButton Label="Format Painter" SizeForm="Small"
MediumIcon="/Resources/FormatPainter20.png"
SmallIcon="/Resources/FormatPainter16.png" />
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Font" >
<syncfusion:RibbonButton Label="Bold" SmallIcon="/Resources/Bold16.png"
MediumIcon="/Resources/Bold_20.png" SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton Label="Italic" SmallIcon="/Resources/Italic16.png"
MediumIcon="/Resources/Italic_20.png" SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton Label="Underline"
SmallIcon="/Resources/Underline16.png"
MediumIcon="/Resources/Underline_20.png" SizeForm="ExtraSmall"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Send / Receive"/>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
RibbonTab sendTab = new RibbonTab();
sendTab.Caption = "Send / Receive";
// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Clipboard";
// Creating items
RibbonButton pasteButton = new RibbonButton();
pasteButton.Label = "Paste";
pasteButton.SizeForm = SizeForm.Large;
pasteButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Paste20.png",
UriKind.RelativeOrAbsolute));

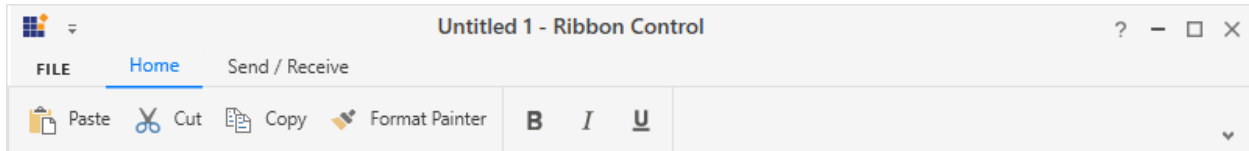
```

```
pasteButton.LargeIcon = new BitmapImage(new Uri(@"Resources/Paste32.png",
UriKind.RelativeOrAbsolute));
RibbonButton cutButton = new RibbonButton();
cutButton.Label = "Cut";
cutButton.SizeForm = SizeForm.Small;
cutButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Cut_20.png",
UriKind.RelativeOrAbsolute));
cutButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Cut16.png",
UriKind.RelativeOrAbsolute));
RibbonButton copyButton = new RibbonButton();
copyButton.Label = "Copy";
copyButton.SizeForm = SizeForm.Small;
copyButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Copy_20.png",
UriKind.RelativeOrAbsolute));
copyButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Copy16.png",
UriKind.RelativeOrAbsolute));
RibbonButton formatButton = new RibbonButton();
formatButton.Label = "Format Painter";
formatButton.SizeForm = SizeForm.Small;
formatButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/FormatPainter20.png", UriKind.RelativeOrAbsolute));
formatButton.SmallIcon = new BitmapImage(new
Uri(@"Resources/FormatPainter16.png", UriKind.RelativeOrAbsolute));
// Adding items to bar
clipboardBar.Items.Add(pasteButton);
clipboardBar.Items.Add(cutButton);
clipboardBar.Items.Add(copyButton);
clipboardBar.Items.Add(formatButton);
// Creating new bar
RibbonBar fontBar = new RibbonBar();
fontBar.Header = "Font";
// Creating items
RibbonButton boldButton = new RibbonButton();
boldButton.Label = "Bold";
boldButton.SizeForm = SizeForm.ExtraSmall;
boldButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Bold16.png",
UriKind.RelativeOrAbsolute));
boldButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Bold_20.png",
UriKind.RelativeOrAbsolute));
RibbonButton italicButton = new RibbonButton();
italicButton.Label = "Italic";
italicButton.SizeForm = SizeForm.ExtraSmall;
italicButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Italic16.png",
UriKind.RelativeOrAbsolute));
italicButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/Italic_20.png", UriKind.RelativeOrAbsolute));
RibbonButton underlineButton = new RibbonButton();
underlineButton.Label = "Underline";
underlineButton.SizeForm = SizeForm.ExtraSmall;
underlineButton.SmallIcon = new BitmapImage(new
Uri(@"Resources/Underline16.png", UriKind.RelativeOrAbsolute));
underlineButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/Underline_20.png", UriKind.RelativeOrAbsolute));
fontBar.Items.Add(boldButton);
fontBar.Items.Add(italicButton);
fontBar.Items.Add(underlineButton);
// Adding bars to the tabs
```

```

homeTab.Items.Add(clipboardBar);
homeTab.Items.Add(fontBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
ribbon.Items.Add(sendTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the **RibbonButton** can be ignored as it will be resized automatically to the standard width and height. If the **RibbonButton** is to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```

<syncfusion:RibbonButton Label="Copy"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified"
SizeForm="Small" MediumIcon="/Resources/Copy_20.png"
SmallIcon="/Resources/Copy16.png" >
<syncfusion:RibbonButton.Style >
<Style TargetType="syncfusion:RibbonButton" BasedOn="{StaticResource
SyncfusionRibbonButtonStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="48"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>
</syncfusion:RibbonButton.Style>
</syncfusion:RibbonButton>

```

RibbonDropDownButton in WPF Ribbon

DropDownButton appears like normal button that contains a drop arrow. It displays some items, while click on it. It accepts **DropDownMenuItem** as its children.

Add DropDownMenuItem

DropDownMenuItem are the items with **Header** property that is used to set header.

XML

```

<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" >
<syncfusion:RibbonBar Name="_ribbonBar1">
<syncfusion:DropDownButton Label="View"/>
<syncfusion:DropDownButton Label="Layout"/>

```

```
<syncfusion:DropDownButton Name="_dropDownButton" SizeForm = "Large"
Width="100" Label="File" >
<syncfusion:DropDownMenuItem Header="New"></syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem Header="Open"></syncfusion:DropDownMenuItem>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
```

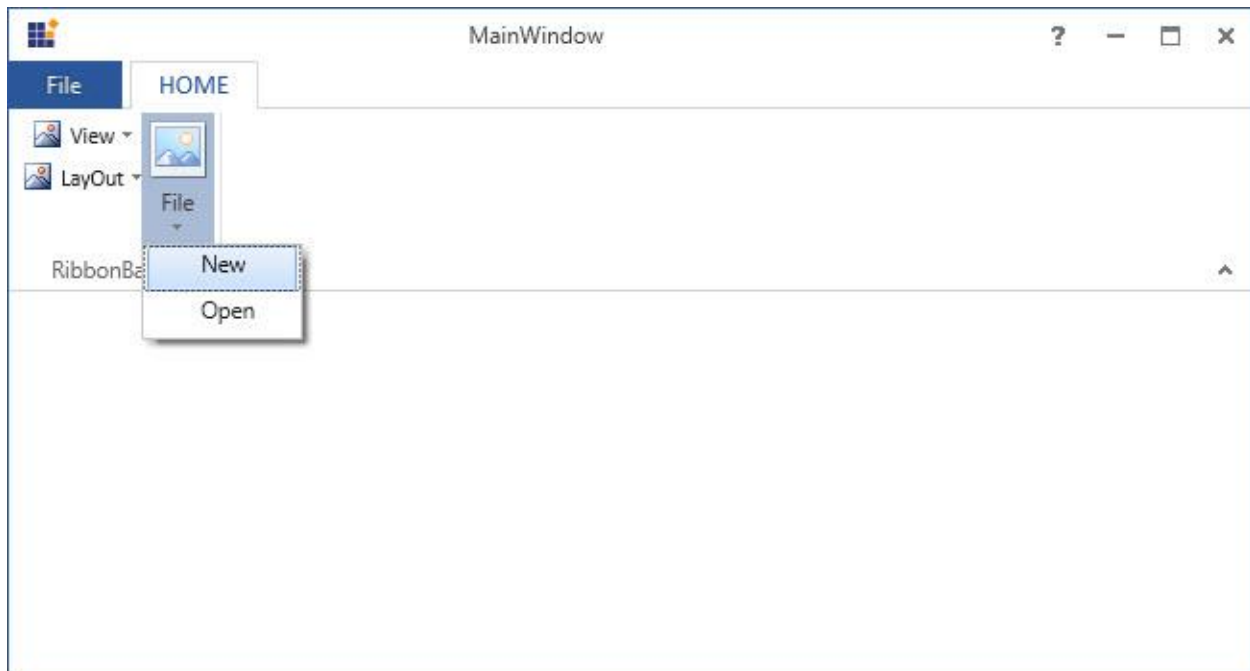
Create instance of `DropDownMenuItems` and add it to `DropDownButton` Items in code behind.

C#

```
DropDownMenuItem _dropDownMenuItem1 = new DropDownMenuItem() {Header="New"
};
DropDownMenuItem _dropDownMenuItem2 = new DropDownMenuItem() { Header =
"Open" };
_dropDownButton.Items.Add(_dropDownMenuItem1);
_dropDownButton.Items.Add(_dropDownMenuItem2);
```

VB.NET

```
Dim _dropDownMenuItem1 As New DropDownMenuItem() With {.Header="New"}
Dim _dropDownMenuItem2 As New DropDownMenuItem() With {.Header = "Open"}
_dropDownButton.Items.Add(_dropDownMenuItem1)
_dropDownButton.Items.Add(_dropDownMenuItem2)
```



Add DropDownMenuItem to DropDownMenuGroup

In `DropDownButton`, similar items can be grouped together and separated from other menu items using `DropDownMenuGroup`.

XML

```

<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="_ribbonBar">
<syncfusion:DropDownButton Label="View"/>
<syncfusion:DropDownButton Label="LayOut"/>
<syncfusion:DropDownButton Name="_dropDownButton" SizeForm = "Large"
Width="100" Label="File" >
<syncfusion:DropDownMenuGroup Name="_dropDownMenuGroup1" Header="Group1">
<syncfusion:DropDownMenuItem Header="New"/>
<syncfusion:DropDownMenuItem Header="Open"/>
</syncfusion:DropDownMenuGroup>
<syncfusion:DropDownMenuGroup Name="_dropDownMenuGroup2" Header="Group2">
<syncfusion:DropDownMenuItem Header="Cut"/>
<syncfusion:DropDownMenuItem Header="Copy"/>
</syncfusion:DropDownMenuGroup>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab IsChecked="False" Caption="DESIGN"/>
</syncfusion:Ribbon>

```

Create instance of `DropDownMenuItem` and add it to `DropDownMenuGroup` Items.

C#

```

DropDownMenuItem _dropDownMenuItem1 = new DropDownMenuItem() { Header =
"New" };
DropDownMenuItem _dropDownMenuItem2 = new DropDownMenuItem() { Header =
"Open" };
DropDownMenuItem _dropDownMenuItem3 = new DropDownMenuItem() { Header =
"Cut" };
DropDownMenuItem _dropDownMenuItem4 = new DropDownMenuItem() { Header =
"Copy" };
_dropDownMenuGroup1.Items.Add(_dropDownMenuItem1);
_dropDownMenuGroup1.Items.Add(_dropDownMenuItem2);
_dropDownMenuGroup2.Items.Add(_dropDownMenuItem3);
_dropDownMenuGroup2.Items.Add(_dropDownMenuItem4);

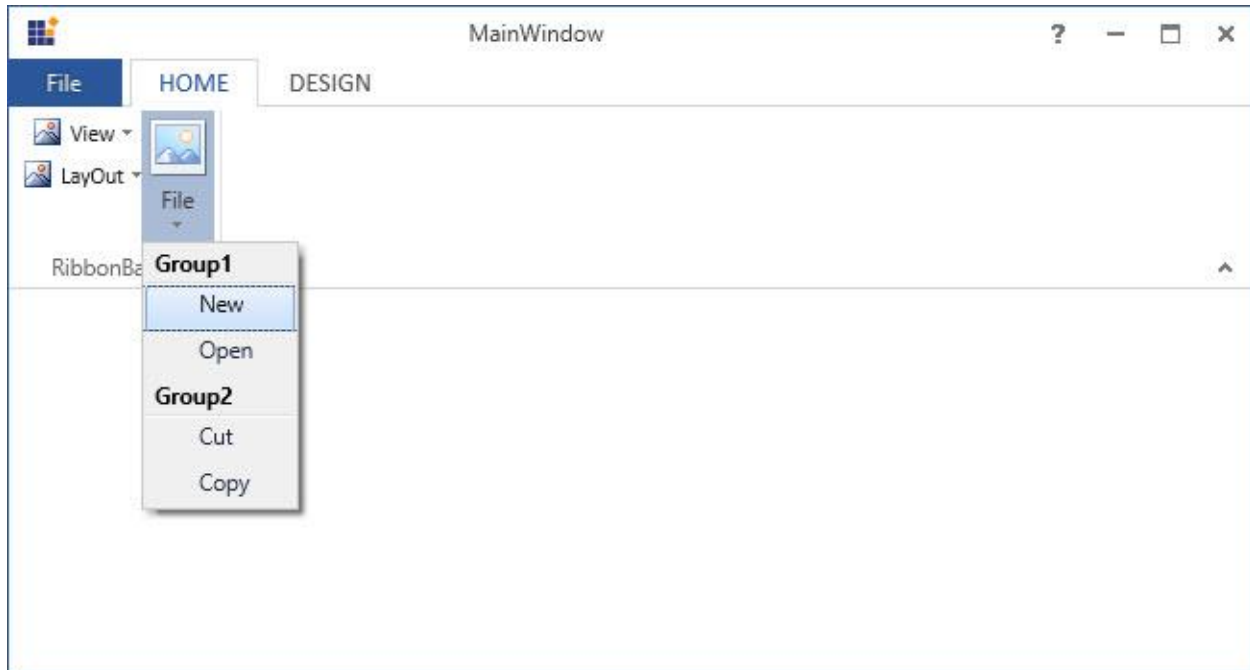
```

VB.NET

```

Dim _dropDownMenuItem1 As New DropDownMenuItem() With {.Header = "New"}
Dim _dropDownMenuItem2 As New DropDownMenuItem() With {.Header = "Open"}
Dim _dropDownMenuItem3 As New DropDownMenuItem() With {.Header = "Cut"}
Dim _dropDownMenuItem4 As New DropDownMenuItem() With {.Header = "Copy"}
_dropDownMenuGroup1.Items.Add(_dropDownMenuItem1)
_dropDownMenuGroup1.Items.Add(_dropDownMenuItem2)
_dropDownMenuGroup2.Items.Add(_dropDownMenuItem3)
_dropDownMenuGroup2.Items.Add(_dropDownMenuItem4)

```



Set various sizes for DropDownButton

[DropDownButton](#) supports three types of size modes and it can be set using the [SizeForm](#) property. The different [SizeForm](#) available are as follows:

- **ExtraSmall** - Displays only the image in 16 * 16 size.
- **Small** - Displays the label and the image in 16 * 16 size.
- **Large** - Displays the label and the image in 32 * 32 size.

XML

```
<syncfusion:RibbonBar Name="_ribbonBar2">
<syncfusion:DropDownButton SizeForm = "Large" Label="Large" />
<syncfusion:DropDownButton SizeForm = "Small" Label="Small" />
<syncfusion:DropDownButton SizeForm = "ExtraSmall" Label="ExtraSmall" />
</syncfusion:RibbonBar>
```

C#

```
DropDownButton _dropDownButton1 = new DropDownButton() { Label = "Large",
SizeForm = SizeForm.Large };
DropDownButton _dropDownButton2 = new DropDownButton() { Label = "Small",
SizeForm = SizeForm.Small };
DropDownButton _dropDownButton3 = new DropDownButton() { Label =
"ExtraSmall", SizeForm = SizeForm.ExtraSmall };
_ribbonBar2.Items.Add(_dropDownButton1);
_ribbonBar2.Items.Add(_dropDownButton2);
_ribbonBar2.Items.Add(_dropDownButton3);
```

VB.NET

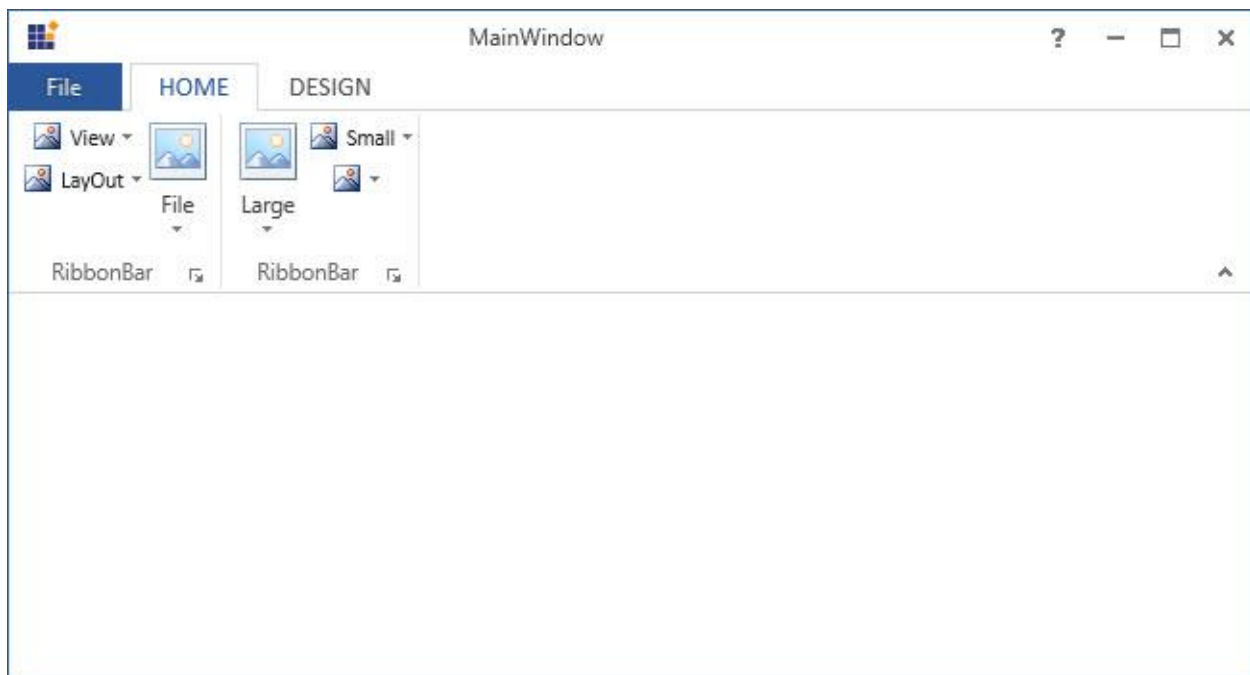
```
Dim _dropDownButton1 As New DropDownButton() With {
.Label = "Large",
```



```

.SizeForm = SizeForm.Large
}
Dim _dropDownButton2 As New DropDownButton() With {
.Label = "Small",
.SizeForm = SizeForm.Small
}
Dim _dropDownButton3 As New DropDownButton() With {
.Label = "ExtraSmall",
.SizeForm = SizeForm.ExtraSmall
}
_ribbonBar2.Items.Add(_dropDownButton1)
_ribbonBar2.Items.Add(_dropDownButton2)
_ribbonBar2.Items.Add(_dropDownButton3)

```



Note: When **simplified** layout is set, [DropDownButton](#) displays the image in 20 * 20 size irrespective of the size form. Also, the text in the **Large** size form will appear to the right of the image.

Setting image to DropDownButton

The [DropDownButton](#) allows to display any type of image such as glyph, font or any custom content using the [IconTemplateSelector](#) and [IconTemplate](#) property, which are the preferred options. It allows to display a normal image or vector image using the [IconType](#) enumeration property. The default value of the [IconType](#) property is "Icon". The [IconType](#) enumeration has the following values:

- **Icon** - Gets the detail of the icon path from the [SmallIcon](#), [MediumIcon](#) or [LargeIcon](#) properties and sets it to the DropDownButton.
- **VectorImage** - Gets the details of the icon's path data from the [VectorImage](#) property and sets it to the DropDownButton.

Note: The [DropDownButton](#) loads icon in the following priority order,

- [IconTemplateSelector](#)
- [IconTemplate](#)
- [VectorImage](#)
- [LargeIcon](#)
- [MediumIcon](#)
- [SmallIcon](#)

Setting icon template selector

The [IconTemplateSelector](#) property provides support to specify a different data template based on the value of the [SizeForm](#) or [LayoutMode](#) properties. For simplified layout, the template content will be resized to 20 * 20 size which is the standard.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpfx"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<syncfusion:RibbonWindow.Resources>
<DataTemplate x:Key="dropDownButtonSmallIconTemplate">
<Grid Width="16" Height="16">
<Path Margin="0.5,4.5,0.5,0.5" Data="M0,0 L27,0 27,23 0,23 z" Fill="White"
Stretch="Fill" />
<Path Height="4" Margin="0.5,0.5,0.5,0" VerticalAlignment="Top" Data="M0,0
L27,0 27,4 0,4 z" Fill="#FFC8C6C4" Stretch="Fill" />
<Path Data="M9,8 L9,14 17,14 17,8 z M8,0 L9,0 9,7 17,7 17,0 18,0 18,7 26,7
26,8 18,8 18,14 26,14 26,15 18,15 18,22 17,22 17,15 9,15 9,22 8,22 8,15 0,15
0,14 8,14&#xa;8,8 0,8 0,7 8,7 z"
Margin="1,5,1,1" Fill="#FF797774" Stretch="Fill" />
<Path Data="M0.99999994,5.00000001 L0.99999994,27 27,27 27,5.00000001 z
M0.99999994,1 L0.99999994,4.00000002 27,4.00000002 27,1 z M0,0 L28,0
28,4.00000002 28,5.00000001 28,28 0,28 0,5.00000001 0,4.00000002 z"
Fill="#FF3A3A38" Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="dropDownButtonLargeIconTemplate">
<Grid Margin="2">
<Path Margin="0.5,4.5,0.5,0.5" Data="M0,0 L27,0 27,23 0,23 z" Fill="White"
Stretch="Fill" />
<Path Height="4" Margin="0.5,0.5,0.5,0" VerticalAlignment="Top" Data="M0,0
L27,0 27,4 0,4 z" Fill="#FFC8C6C4" Stretch="Fill" />
<Path Data="M9,8 L9,14 17,14 17,8 z M8,0 L9,0 9,7 17,7 17,0 18,0 18,7 26,7
26,8 18,8 18,14 26,14 26,15 18,15 18,22 17,22 17,15 9,15 9,22 8,22 8,15 0,15
0,14 8,14&#xa;8,8 0,8 0,7 8,7 z"
Margin="1,5,1,1" Fill="#FF797774" Stretch="Fill" />
<Path Data="M0.99999994,5.00000001 L0.99999994,27 27,27 27,5.00000001 z
M0.99999994,1 L0.99999994,4.00000002 27,4.00000002 27,1 z M0,0 L28,0
28,4.00000002 28,5.00000001 28,28 0,28 0,5.00000001 0,4.00000002 z"
Fill="#FF3A3A38" Stretch="Fill" />
</Grid>
</DataTemplate>
```

```

Fill="#FF3A3A38" Stretch="Fill" />
</Grid>
</DataTemplate>
<local:DropDownButtonIconTemplateSelector
x:Key="dropDownButtonIconTemplateSelector"
SmallTemplate="{StaticResource dropDownButtonSmallIconTemplate}"
LargeTemplate="{StaticResource dropDownButtonLargeIconTemplate}"/>
</syncfusion:RibbonWindow.Resources>
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Tables">
<syncfusion:DropDownButton Label="Tables" SizeForm="Large"
IconTemplateSelector="{StaticResource dropDownButtonIconTemplateSelector}"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

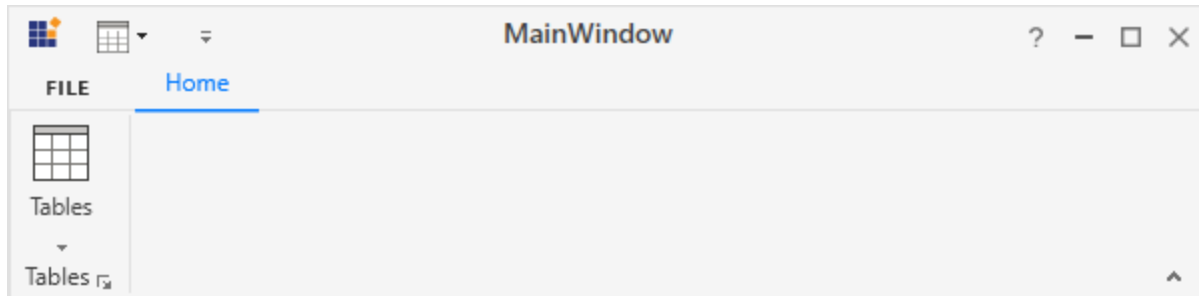
```

C#

```

public class DropDownButtonIconTemplateSelector : DataTemplateSelector
{
    public DataTemplate SmallTemplate { get; set; }
    public DataTemplate LargeTemplate { get; set; }
    public override DataTemplate SelectTemplate(object item, DependencyObject
container)
    {
        var item1 = (container as ContentPresenter);
        DropDownButton dropDownButton = (item1.TemplatedParent as DropDownButton);
        if (dropDownButton != null)
        {
            if (dropDownButton.SizeForm == SizeForm.Small || dropDownButton.SizeForm ==
SizeForm.ExtraSmall)
            {
                return SmallTemplate;
            }
            else if (dropDownButton.SizeForm == SizeForm.Large)
            {
                return LargeTemplate;
            }
        }
        return LargeTemplate;
    }
}

```



Note: [View sample in GitHub](#)

Setting icon template

The [IconTemplate](#) property provides support to set any type of image such as glyph, font or any custom content to the [DropDownButton](#). The [DropDownButton](#) will automatically resize the template content according to its [SizeForm](#). For simplified layout, the template content will be resized to 20 * 20 size which is the standard.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Tables">
<syncfusion:DropDownButton Label="Tables" SizeForm="Large">
<syncfusion:DropDownButton.IconTemplate>
<DataTemplate>
<Grid MaxHeight="32" MaxWidth="32">
<Path
Margin="0.5,4.5,0.5,0.5" Data="M0,0 L27,0 27,23 0,23 z" Fill="White"
Stretch="Fill" />
<Path
Height="4" Margin="0.5,0.5,0.5,0" VerticalAlignment="Top" Data="M0,0 L27,0
27,4 0,4 z" Fill="#FFC8C6C4" Stretch="Fill" />
<Path
Data="M9,8 L9,14 17,14 17,8 z M8,0 L9,0 9,7 17,7 17,0 18,0 18,7 26,7 26,8
18,8 18,14 26,14 26,15 18,15 18,22 17,22 17,15 9,15 9,22 8,22 8,15 0,15 0,14
8,14&#xa;8,8 0,8 0,7 8,7 z"
Margin="1,5,1,1" Fill="#FF797774" Stretch="Fill" />
<Path
Data="M0.999999994,5.00000001 L0.999999994,27 27,27 27,5.00000001 z
M0.999999994,1 L0.999999994,4.00000002 27,4.00000002 27,1 z M0,0 L28,0
28,4.00000002 28,5.00000001 28,28 0,28 0,5.00000001 0,4.00000002 z"
Fill="#FF3A3A38" Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
```

```

</DataTemplate>
</syncfusion:DropDownButton.IconTemplate>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

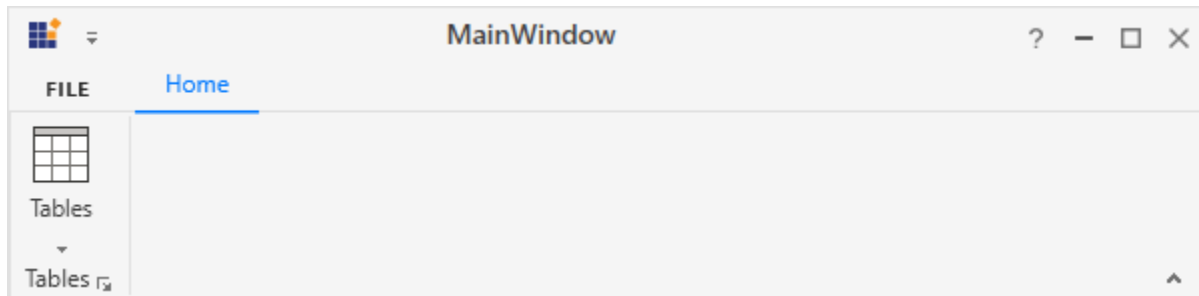
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar tablesBar = new RibbonBar();
tablesBar.Header = "Tables";
// Creating items
DropDownButton tablesButton = new DropDownButton();
tablesButton.Label = "Tables";
tablesButton.SizeForm = SizeForm.Large;
DataTemplate iconDataTemplate = new DataTemplate();
FrameworkElementFactory gridElement = new
FrameworkElementFactory(typeof(Grid));
FrameworkElementFactory pathElement1 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement2 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement3 = new
FrameworkElementFactory(typeof(Path));
gridElement.SetValue(Grid.MarginProperty, new Thickness(2));
pathElement1.SetValue(Path.DataProperty, Geometry.Parse("M0,0 L27,0 27,23
0,23 z"));
pathElement1.SetValue(Path.MarginProperty, new Thickness(0.5, 4.5, 0.5,
0.5));
pathElement1.SetValue(Path.FillProperty, new SolidColorBrush(Colors.White));
pathElement1.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement2.SetValue(Path.DataProperty, Geometry.Parse("M0,0 L27,0 27,4 0,4
z"));
pathElement2.SetValue(Path.MarginProperty, new Thickness(0.5, 0.5, 0.5, 0));
pathElement2.SetValue(Path.HeightProperty, (double)4);
pathElement2.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(200, 198, 196)));
pathElement2.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement3.SetValue(Path.DataProperty,
Geometry.Parse("M0.99999994,5.0000001 L0.99999994,27 27,27 27,5.0000001 z
M0.99999994,1 L0.99999994,4.0000002 27,4.0000002 27,1 z M0,0 L28,0
28,4.0000002 28,5.0000001 28,28 0,28 0,5.0000001 0,4.0000002 z"));
pathElement3.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(58, 58, 56)));
pathElement3.SetValue(Path.StretchProperty, Stretch.Fill);
gridElement.AppendChild(pathElement1);
gridElement.AppendChild(pathElement2);

```

```

gridElement.AppendChild(pathElement3);
iconDataTemplate.VisualTree = gridElement;
tablesButton.IconTemplate = iconDataTemplate;
tablesBar.Items.Add(tablesButton);
// Adding bars to the tabs
homeTab.Items.Add(tablesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Note: [View sample in GitHub](#)

Setting image path

The [DropDownButton](#) allows to set the image according to the different [SizeForm](#) values. To set the image to DropDownButton, the following properties are used:

- [SmallIcon](#) - 16 * 16 size image to be displayed in normal layout for “ExtraSmall” and “Small” size form.
- [MediumIcon](#) - 20 * 20 size image to be displayed in simplified layout.
- [LargeIcon](#) - 32 * 32 size image to be displayed in normal layout for “Large” size form.

XML

```

<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStyle="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top"
EnableSimplifiedLayoutMode="True">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Tables">
<syncfusion:DropDownButton
Label="Table"
LargeIcon="Resources/Table_32.png"
MediumIcon="Resources/Table20.png"

```

```

SizeForm="Large">
</syncfusion:DropDownButton>
<syncfusion:DropDownButton
Label="Shapes"
SmallIcon="Resources/Insert Shapes16.png"
MediumIcon="Resources/Insert Shapes20.png"
SizeForm="Small">
</syncfusion:DropDownButton>
<syncfusion:DropDownButton
Label="Chart"
SmallIcon="Resources/Base chart16.png"
MediumIcon="Resources/Base chart20.png"
SizeForm="Small">
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar tablesBar = new RibbonBar();
tablesBar.Header = "Tables";
// Creating items
DropDownButton tableButton = new DropDownButton();
tableButton.Label = "Table";
tableButton.SizeForm = SizeForm.Large;
tableButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Table20.png",
UriKind.RelativeOrAbsolute));
tableButton.LargeIcon = new BitmapImage(new Uri(@"Resources/Table_32.png",
UriKind.RelativeOrAbsolute));
DropDownButton shapesButton = new DropDownButton();
shapesButton.Label = "Shapes";
shapesButton.SizeForm = SizeForm.Small;
shapesButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Insert
Shapes20.png", UriKind.RelativeOrAbsolute));
shapesButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Insert
Shapes16.png", UriKind.RelativeOrAbsolute));
DropDownButton chartButton = new DropDownButton();
chartButton.Label = "Chart";
chartButton.SizeForm = SizeForm.Small;
chartButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Base
chart20.png", UriKind.RelativeOrAbsolute));
chartButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Base
chart16.png", UriKind.RelativeOrAbsolute));
// Adding items to bar
tablesBar.Items.Add(tableButton);

```

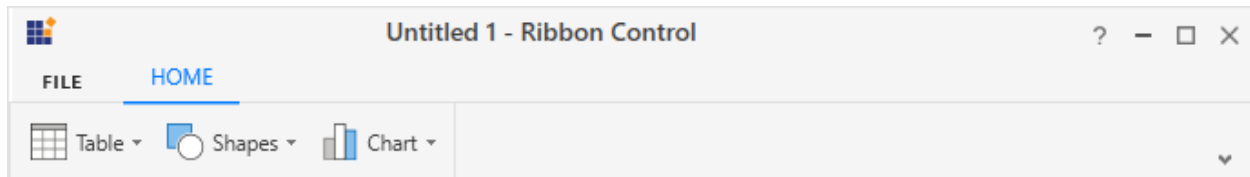
```

tablesBar.Items.Add(shapesButton);
tablesBar.Items.Add(chartButton);
// Adding bars to the tabs
homeTab.Items.Add(tablesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Normal layout



Simplified layout

Setting vector image

The [VectorImage](#) property is of type `ObservableCollection<Path>` which allows the image to be set as path type. The `DropDownButton` will automatically resize the image according to its [SizeForm](#). For simplified layout, the image will be resized to 20 * 20 size which is the standard.

Note: The [IconTemplateSelector](#) and [IconTemplate](#) properties are the preferred options to set any type of image such as glyph, font or any custom content when compared to the [VectorImage](#) property.

XML

```

<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStyle="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top"
EnableSimplifiedLayoutMode="True">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Tables">
<syncfusion:DropDownButton
IconType="VectorImage"
Label="Table"

```



```

SizeForm="Large">
<syncfusion:DropDownButton.VectorImage>
<Path
Margin="0.5,4.5,0.5,0.5" Data="M0,0 L27,0 27,23 0,23 z"
Fill="White" Stretch="Fill" />
<Path
Height="4" Margin="0.5,0.5,0.5,0"
VerticalAlignment="Top" Data="M0,0 L27,0 27,4 0,4 z"
Fill="#FFC8C6C4" Stretch="Fill" />
<Path
Margin="1,5,1,1"
Data="M9,8 L9,14 17,14 17,8 z M8,0 L9,0 9,7 17,7 17,0 18,0 18,7 26,7 26,8
18,8 18,14 26,14 26,15 18,15 18,22 17,22 17,15 9,15 9,22 8,22 8,15 0,15 0,14
8,14&#xa;8,8 0,8 0,7 8,7 z"
Fill="#FF797774" Stretch="Fill" />
<Path
Data="M0.999999994,5.00000001 L0.999999994,27 27,27 27,5.00000001 z
M0.999999994,1 L0.999999994,4.00000002 27,4.00000002 27,1 z M0,0 L28,0
28,4.00000002 28,5.00000001 28,28 0,28 0,5.00000001 0,4.00000002 z"
Fill="#FF3A3A38" Stretch="Fill" />
</syncfusion:DropDownButton.VectorImage>
</syncfusion:DropDownButton>
<syncfusion:DropDownButton
IconType="VectorImage"
Label="Shapes"
SizeForm="Small">
<syncfusion:DropDownButton.VectorImage>
<Path
Margin="0.5,0.5,9.5,9.5"
Data="M0,0 L18,0 18,6.5250005 C17.833,6.5180005 17.668999,6.5000008
17.5,6.5000008 11.434998,6.5000008 6.5,11.435001 6.5,17.5 6.5,17.669002
6.5179977,17.833 6.5249977,18 L0,18 z"
Fill="#FF83BEEC" Stretch="Fill" />
<Path
Margin="8.5,8.5,0.5,0.5"
Data="M9.5000001,0 C14.746998,0 19,4.253001 19,9.4999988 19,14.747001
14.746998,19.000001 9.5000001,19.000001 4.2539979,19.000001 8.1026528E-
08,14.747001 0,9.4999988 8.1026528E-08,4.253001 4.2539979,0 9.5000001,0 z"
Fill="White" Stretch="Fill" />
<Path
Margin="8,8,0,0"
Data="M10,0.99999995 C5.0369988,1.0000001 1.0000001,5.0370014
0.99999998,9.9999989 1.0000001,14.963001 5.0369988,19.000001 10,19.000001
14.962998,19.000001 19,14.963001 19,9.9999989 19,5.0370014
14.962998,1.0000001 10,0.99999995 z M10,0 C15.514,0 20,4.4860004
20,9.9999989 20,15.514 15.514,20.000001 10,20.000001 4.4860002,20.000001
9.5927689E-08,15.514 0,9.9999989 9.5927689E-08,4.4860004 4.4860002,0 10,0 z"
Fill="#FF3A3A38" Stretch="Fill" />
<Path
Margin="0,0,9,9"
Data="M0,0 L19,0 19,7.0510015 C18.669998,7.0210008 18.337997,7.0000009
18,7.0000009 L18,1.0000002 0.99999988,1.0000002 0.99999988,18 7,18
C7,18.338001 7.0209999,18.670002 7.0509987,19 L0,19 z"
Fill="#FF0063B1" Stretch="Fill" />
</syncfusion:DropDownButton.VectorImage>
</syncfusion:DropDownButton>
<syncfusion:DropDownButton

```

```

IconType="VectorImage"
Label="Chart"
SizeForm="Small">
<syncfusion:DropDownButton.VectorImage>
<Path
Width="8" Margin="0.5,11.5,0,0.5" HorizontalAlignment="Left"
Data="M0,0 L7.9999996,0 7.9999996,16 0,16 z"
Fill="#FFC8C6C4" Stretch="Fill" />
<Path
Width="9" Margin="0,11,0,0" HorizontalAlignment="Left"
Data="M0.99999667,0.99999994 L0.99999667,16 8.0000032,16
8.0000032,0.99999994 z M0,0 L9.0000004,0 9.0000004,17 3.2610174,17
2.5740174,17 0,17 z"
Fill="#FF797774" Stretch="Fill" />
<Path
Width="8" Margin="0,5.5,0.5,0.5" HorizontalAlignment="Right"
Data="M0,0 L8.0000001,0 8.0000001,22 0,22 z"
Fill="#FF83BEEC" Stretch="Fill" />
<Path
Width="9" Margin="0,5,0,0" HorizontalAlignment="Right"
Data="M1.0000001,1.0000001 L1.0000001,22 8,22 8,1.0000001 z M0,0 L9,0 9,23
0,23 z"
Fill="#FF0063B1" Stretch="Fill" />
<Path
Margin="8.5,0.5" Data="M0,0 L8,0 8,27 0,27 z"
Fill="White" Stretch="Fill" />
<Path
Margin="8,0"
Data="M0.99999994,0.99999994 L0.99999994,27 7.9999999,27
7.9999999,0.99999994 z M0,0 L8.9999999,0 8.9999999,28 2.355,28 1.7929999,28
0,28 z"
Fill="#FF3A3A38" Stretch="Fill" />
</syncfusion:DropDownButton.VectorImage>
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar tablesBar = new RibbonBar();
tablesBar.Header = "Tables";
// Creating items
DropDownButton tableButton = new DropDownButton();
tableButton.Label = "Table";
tableButton.SizeForm = SizeForm.Large;

```

```

tableButton.IconType = IconType.VectorImage;
Path tablePath1 = new Path();
tablePath1.Data = Geometry.Parse("M0,0 L27,0 27,23 0,23 z");
tablePath1.Margin = new Thickness(0.5, 4.5, 0.5, 0.5);
tablePath1.Fill = new SolidColorBrush(Colors.White);
Path tablePath2 = new Path();
tablePath2.Data = Geometry.Parse("M0,0 L27,0 27,4 0,4 z");
tablePath2.Margin = new Thickness(0.5, 0.5, 0.5, 0);
tablePath2.Fill = new SolidColorBrush(Color.FromRgb(200, 198, 196));
tablePath2.Height = 4;
tablePath2.VerticalAlignment = VerticalAlignment.Top;
Path tablePath3 = new Path();
tablePath3.Data = Geometry.Parse("M9,8 L9,14 17,14 17,8 z M8,0 L9,0 9,7 17,7 17,0 18,0 18,7 26,7 26,8 18,8 18,14 26,14 26,15 18,15 18,22 17,22 17,15 9,15 9,22 8,22 8,15 0,15 0,14 8,14&#xa;8,8 0,8 0,7 8,7 z");
tablePath3.Margin = new Thickness(1, 5, 1, 1);
tablePath3.Fill = new SolidColorBrush(Color.FromRgb(121, 119, 116));
Path tablePath4 = new Path();
tablePath4.Data = Geometry.Parse("M0.99999994,5.0000001 L0.99999994,27 27,27 27,5.0000001 z M0.99999994,1 L0.99999994,4.0000002 27,4.0000002 27,1 z M0,0 L28,0 28,4.0000002 28,5.0000001 28,28 0,28 0,5.0000001 0,4.0000002 z");
tablePath4.Fill = new SolidColorBrush(Color.FromRgb(58, 58, 56));
tableButton.VectorImage.Add(tablePath1);
tableButton.VectorImage.Add(tablePath2);
tableButton.VectorImage.Add(tablePath3);
tableButton.VectorImage.Add(tablePath4);
DropDownButton shapesButton = new DropDownButton();
shapesButton.Label = "Shapes";
shapesButton.SizeForm = SizeForm.Small;
shapesButton.IconType = IconType.VectorImage;
Path shapePath1 = new Path();
shapePath1.Data = Geometry.Parse("M0,0 L18,0 18,6.5250005 C17.833,6.5180005 17.668999,6.5000008 17.5,6.5000008 11.434998,6.5000008 6.5,11.435001 6.5,17.5 6.5,17.669002 6.5179977,17.833 6.5249977,18 L0,18 z");
shapePath1.Margin = new Thickness(0.5, 0.5, 9.5, 9.5);
shapePath1.Fill = new SolidColorBrush(Color.FromRgb(131, 190, 236));
Path shapePath2 = new Path();
shapePath2.Data = Geometry.Parse("M9.5000001,0 C14.746998,0 19,4.253001 19,9.4999988 19,14.747001 14.746998,19.000001 9.5000001,19.000001 4.2539979,19.000001 8.1026528E-08,14.747001 0,9.4999988 8.1026528E-08,4.253001 4.2539979,0 9.5000001,0 z");
shapePath2.Margin = new Thickness(8.5, 8.5, 0.5, 0.5);
shapePath2.Fill = new SolidColorBrush(Colors.White);
Path shapePath3 = new Path();
shapePath3.Data = Geometry.Parse("M10,0.99999995 C5.0369988,1.0000001 1.0000001,5.0370014 0.99999998,9.9999989 1.0000001,14.963001 5.0369988,19.000001 10,19.000001 14.962998,19.000001 19,14.963001 19,9.9999989 19,5.0370014 14.962998,1.0000001 10,0.99999995 z M10,0 C15.514,0 20,4.4860004 20,9.9999989 20,15.514 15.514,20.000001 10,20.000001 4.4860002,20.000001 9.5927689E-08,15.514 0,9.9999989 9.5927689E-08,4.4860004 4.4860002,0 10,0 z");
shapePath3.Margin = new Thickness(8, 8, 0, 0);
shapePath3.Fill = new SolidColorBrush(Color.FromRgb(58, 58, 56));
Path shapePath4 = new Path();
shapePath4.Data = Geometry.Parse("M0,0 L19,0 19,7.0510015 C18.669998,7.0210008 18.337997,7.0000009 18,7.0000009 L18,1.0000002

```

```

0.99999988,1.0000002 0.99999988,18 7,18 C7,18.338001 7.0209999,18.670002
7.0509987,19 L0,19 z");
shapePath4.Margin = new Thickness(0, 0, 9, 9);
shapePath4.Fill = new SolidColorBrush(Color.FromRgb(0, 99, 177));
shapesButton.VectorImage.Add(shapePath1);
shapesButton.VectorImage.Add(shapePath2);
shapesButton.VectorImage.Add(shapePath3);
shapesButton.VectorImage.Add(shapePath4);
DropDownButton chartButton = new DropDownButton();
chartButton.Label = "Chart";
chartButton.SizeForm = SizeForm.Small;
chartButton.IconType = IconType.VectorImage;
Path chartPath1 = new Path();
chartPath1.Data = Geometry.Parse("M0,0 L7.9999996,0 7.9999996,16 0,16 z");
chartPath1.Margin = new Thickness(0.5, 11.5, 0, 0.5);
chartPath1.Fill = new SolidColorBrush(Color.FromRgb(200, 198, 196));
chartPath1.HorizontalAlignment = HorizontalAlignment.Left;
chartPath1.Width = 8;
Path chartPath2 = new Path();
chartPath2.Data = Geometry.Parse("M0.99999667,0.99999994 L0.99999667,16
8.0000032,16 8.0000032,0.99999994 z M0,0 L9.0000004,0 9.0000004,17
3.2610174,17 2.5740174,17 0,17 z");
chartPath2.Margin = new Thickness(0, 11, 0, 0);
chartPath2.Fill = new SolidColorBrush(Color.FromRgb(121, 119, 116));
chartPath2.HorizontalAlignment = HorizontalAlignment.Left;
chartPath2.Width = 9;
Path chartPath3 = new Path();
chartPath3.Data = Geometry.Parse("M0,0 L8.0000001,0 8.0000001,22 0,22 z");
chartPath3.Margin = new Thickness(0, 5.5, 0.5, 0.5);
chartPath3.Fill = new SolidColorBrush(Color.FromRgb(131, 190, 236));
chartPath3.HorizontalAlignment = HorizontalAlignment.Right;
chartPath3.Width = 8;
Path chartPath4 = new Path();
chartPath4.Data = Geometry.Parse("M1.0000001,1.0000001 L1.0000001,22 8,22
8,1.0000001 z M0,0 L9,0 9,23 0,23 z");
chartPath4.Margin = new Thickness(0, 5, 0, 0);
chartPath4.Fill = new SolidColorBrush(Color.FromRgb(0, 99, 177));
chartPath4.HorizontalAlignment = HorizontalAlignment.Right;
chartPath4.Width = 9;
Path chartPath5 = new Path();
chartPath5.Data = Geometry.Parse("M0,0 L8,0 8,27 0,27 z");
chartPath5.Margin = new Thickness(8.5, 0.5, 8.5, 0.5);
chartPath5.Fill = new SolidColorBrush(Colors.White);
Path chartPath6 = new Path();
chartPath6.Data = Geometry.Parse("M0.99999994,0.99999994 L0.99999994,27
7.9999999,27 7.9999999,0.99999994 z M0,0 L8.9999999,0 8.9999999,28 2.355,28
1.7929999,28 0,28 z");
chartPath6.Margin = new Thickness(8, 0, 8, 0);
chartPath6.Fill = new SolidColorBrush(Color.FromRgb(58, 58, 56));
chartButton.VectorImage.Add(chartPath1);
chartButton.VectorImage.Add(chartPath2);
chartButton.VectorImage.Add(chartPath3);
chartButton.VectorImage.Add(chartPath4);
chartButton.VectorImage.Add(chartPath5);
chartButton.VectorImage.Add(chartPath6);
// Adding items to bar
tablesBar.Items.Add(tableButton);

```

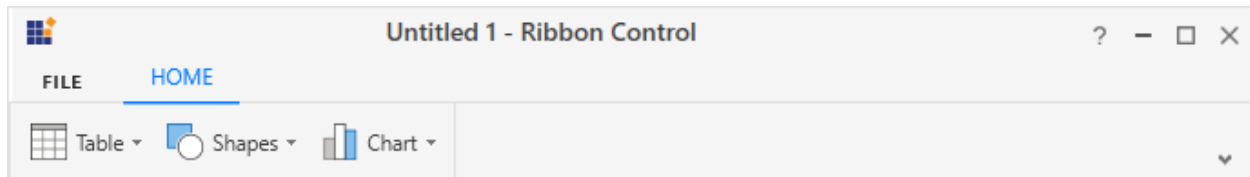
```

tablesBar.Items.Add(shapesButton);
tablesBar.Items.Add(chartButton);
// Adding bars to the tabs
homeTab.Items.Add(tablesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Normal layout



Simplified layout

Add DropDownButton to the simplified layout

When the simplified layout is enabled, the [DropDownButton](#) can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```

<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStyle="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top"
EnableSimplifiedLayoutMode="True">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Tables">
<syncfusion:DropDownButton
Label="Table"
LargeIcon="Resources/Table_32.png"
MediumIcon="Resources/Table20.png"
SizeForm="Large">
</syncfusion:DropDownButton>

```

```

<syncfusion:DropDownButton
Label="Shapes"
SmallIcon="Resources/Insert Shapes16.png"
MediumIcon="Resources/Insert Shapes20.png"
SizeForm="Small">
</syncfusion:DropDownButton>
<syncfusion:DropDownButton
Label="Chart"
SmallIcon="Resources/Base chart16.png"
MediumIcon="Resources/Base chart20.png"
SizeForm="Small">
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

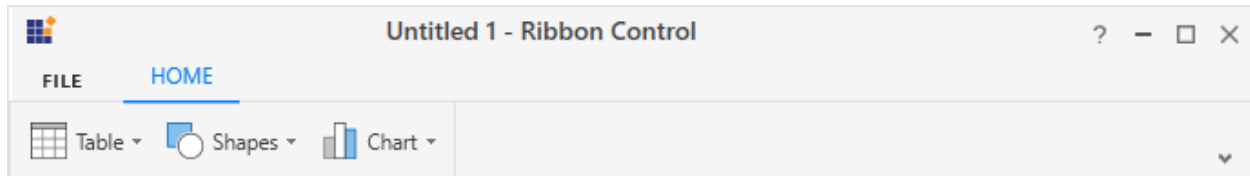
C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar tablesBar = new RibbonBar();
tablesBar.Header = "Tables";
// Creating items
DropDownButton tableButton = new DropDownButton();
tableButton.Label = "Table";
tableButton.SizeForm = SizeForm.Large;
tableButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Table20.png",
UriKind.RelativeOrAbsolute));
tableButton.LargeIcon = new BitmapImage(new Uri(@"Resources/Table_32.png",
UriKind.RelativeOrAbsolute));
DropDownButton shapesButton = new DropDownButton();
shapesButton.Label = "Shapes";
shapesButton.SizeForm = SizeForm.Small;
shapesButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Insert
Shapes20.png", UriKind.RelativeOrAbsolute));
shapesButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Insert
Shapes16.png", UriKind.RelativeOrAbsolute));
DropDownButton chartButton = new DropDownButton();
chartButton.Label = "Chart";
chartButton.SizeForm = SizeForm.Small;
chartButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Base
chart20.png", UriKind.RelativeOrAbsolute));
chartButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Base
chart16.png", UriKind.RelativeOrAbsolute));
// Adding items to bar
tablesBar.Items.Add(tableButton);
tablesBar.Items.Add(shapesButton);
tablesBar.Items.Add(chartButton);

```

```
// Adding bars to the tabs
homeTab.Items.Add(tablesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
```



When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the **DropDownButton** can be ignored as it will be resized automatically to the standard width and height. If the **DropDownButton** is to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```
<syncfusion:DropDownButton Label="Copy"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified"
SizeForm="Small" MediumIcon="/Resources/Copy_20.png"
SmallIcon="/Resources/Copy16.png" >
<syncfusion:DropDownButton.Style>
<Style TargetType="syncfusion:DropDownButton" BasedOn="{StaticResource
SyncfusionRibbonDropDownButtonStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="48"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>
</syncfusion:DropDownButton.Style>
</syncfusion:DropDownButton >
```

RibbonSplitButton in WPF Ribbon

SplitButton can perform like both normal **Button** as well as **DropDownButton**. It allow to click the button directly by clicking the upper part of the button and also it display list of items while click on the arrow.

Add DropDownMenuItem

DropDownMenuItem are the items with **Header** property that is used to set header.

XML

```
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Name="_ribbonTab1" Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="_ribbonBar1" Header="RibbonBar1">
<syncfusion:RibbonButton Label="Cut"/>
<syncfusion:RibbonButton Label="Copy"/>
</syncfusion:RibbonBar>
```

```
<syncfusion:RibbonBar Name="_ribbonBar2" Header="RibbonBar2">
<syncfusion:SplitButton Label="Chart">
<syncfusion:DropDownMenuItem Header="PieChart"/>
<syncfusion:DropDownMenuItem Header="ColumnChart"/>
</syncfusion:SplitButton>
<syncfusion:SplitButton Label="Table"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="EDIT" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
```

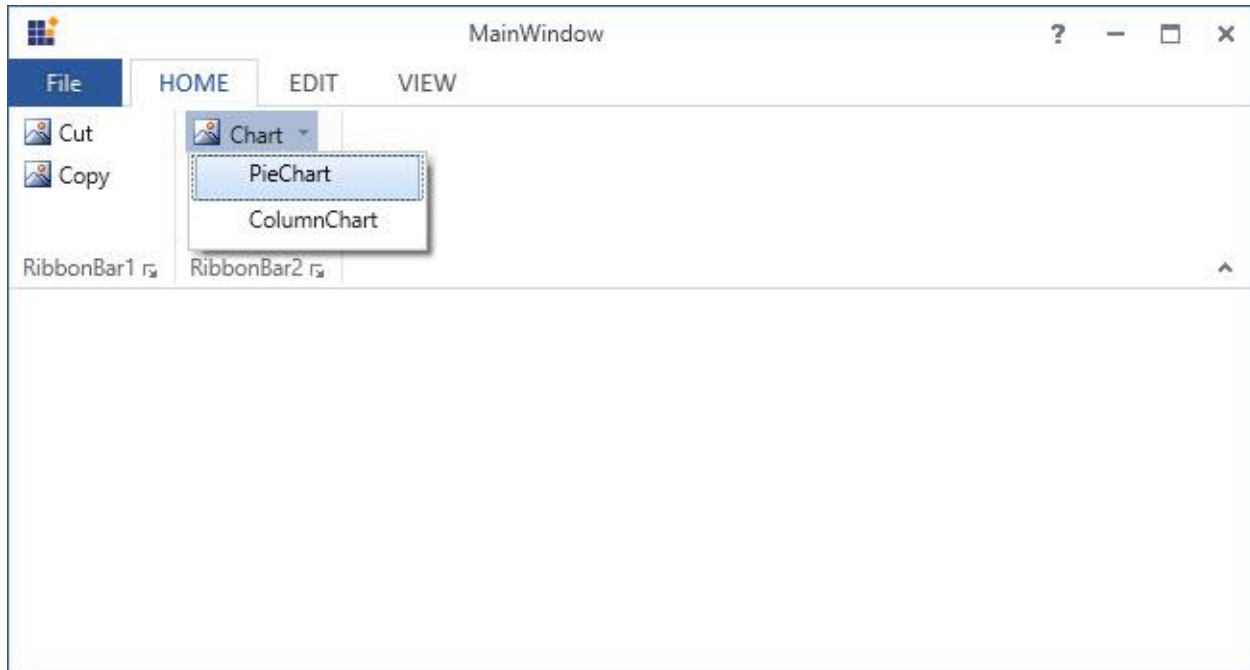
Create instance of `DropDownMenuItem` and add it to `SplitButton` in code behind.

C#

```
SplitButton _splitbutton = new SplitButton() { Label = "Chart" };
DropDownMenuItem _dropDownMenuItem1 = new DropDownMenuItem() { Header =
"PieChart" };
DropDownMenuItem _dropDownMenuItem2 = new DropDownMenuItem() { Header =
"ColumnChart" };
_splitbutton.Items.Add(_dropDownMenuItem1);
_splitbutton.Items.Add(_dropDownMenuItem2);
RibbonBar2.Items.Add( _splitbutton);
```

VB.NET

```
Dim _splitbutton As New SplitButton() With {.Label = "Chart"}
Dim _dropDownMenuItem1 As New DropDownMenuItem() With {.Header = "PieChart"}
Dim _dropDownMenuItem2 As New DropDownMenuItem() With {.Header =
"ColumnChart"}
_splitbutton.Items.Add(_dropDownMenuItem1)
_splitbutton.Items.Add(_dropDownMenuItem2)
RibbonBar2.Items.Add( _splitbutton)
```

Set various sizes for SplitButton

[SplitButton](#) supports three types of size modes and it can be set using the [SizeForm](#) property. The different [SizeForm](#) available are as follows:

- **ExtraSmall** - Displays only the image in 16 * 16 size.
- **Small** - Displays the label and the image in 16 * 16 size.
- **Large** - Displays the label and the image in 32 * 32 size.

XML

```
<syncfusion:SplitButton SizeForm="Large" Label="Large"/>
<syncfusion:SplitButton SizeForm="Small" Label="Small"/>
<syncfusion:SplitButton SizeForm="ExtraSmall" Label="ExtraSmall"/>
```

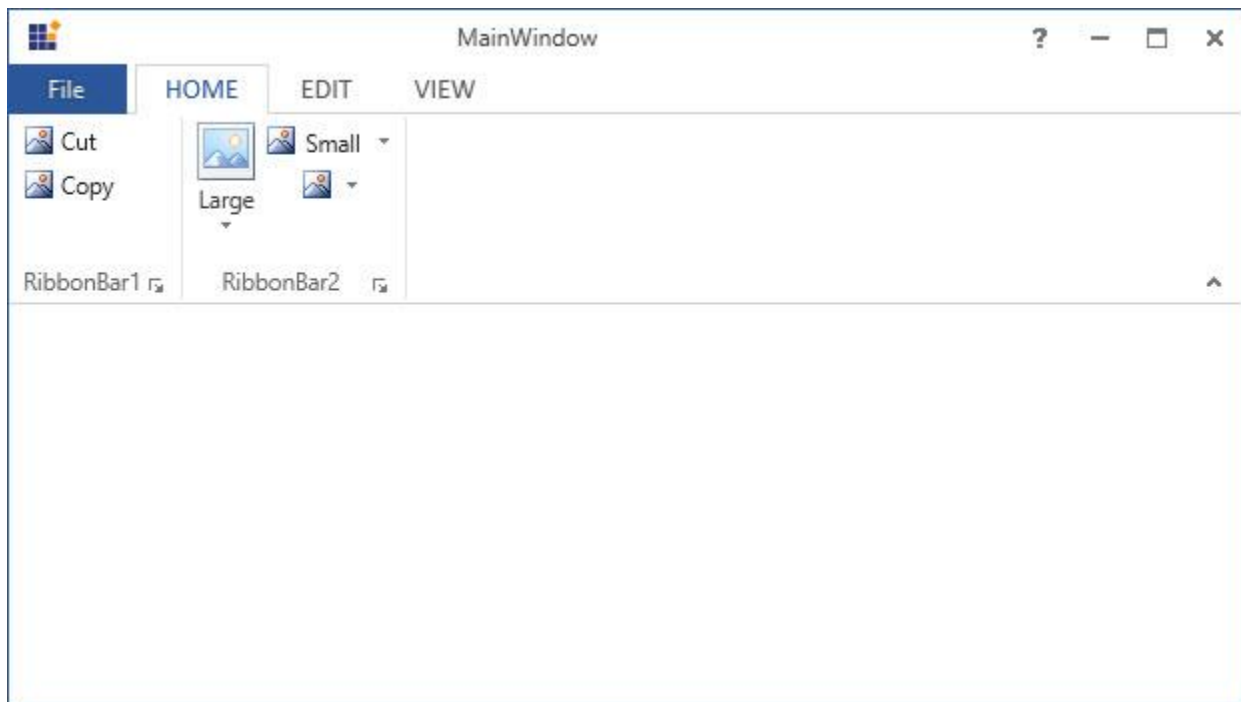
C#

```
SplitButton _splitbutton1 = new SplitButton() {Label = "Large", SizeForm=
SizeForm.Large};
SplitButton _splitbutton2 = new SplitButton() {Label = "Small", SizeForm=
SizeForm.Small};
SplitButton _splitbutton3 = new SplitButton() {Label = "ExtraSmall", SizeForm=
SizeForm.ExtraSmall};
_ribbonBar2.Items.Add(_splitbutton1);
_ribbonBar2.Items.Add(_splitbutton2);
_ribbonBar2.Items.Add(_splitbutton3);
```

VB.NET

```
Dim _splitbutton1 As New SplitButton() With {
.Label = "Large",
.SizeForm= SizeForm.Large
}
```

```
Dim _splitbutton2 As New SplitButton() With {  
    .Label = "Small",  
    .SizeForm= SizeForm.Small  
}  
Dim _splitbutton3 As New SplitButton() With {  
    .Label = "ExtraSmall",  
    .SizeForm= SizeForm.ExtraSmall  
}  
_ribbonBar2.Items.Add(_splitbutton1)  
_ribbonBar2.Items.Add(_splitbutton2)  
_ribbonBar2.Items.Add(_splitbutton3)
```



Note: When **simplified** layout is set, [SplitButton](#) displays the image in 20 * 20 size irrespective of the size form. Also, the text in the **Large** size form will appear to the right of the image.

Setting image to SplitButton

The [SplitButton](#) allows to display any type of image such as glyph, font or any custom content using the [IconTemplateSelector](#) and [IconTemplate](#) property, which are the preferred options. It allows to display a normal image or vector image using the [IconType](#) enumeration property. The default value of the [IconType](#) property is "Icon". The [IconType](#) enumeration has the following values:

- **Icon** - Gets the detail of the icon path from the [SmallIcon](#), [MediumIcon](#) or [LargeIcon](#) properties and sets it to the SplitButton.
- **VectorImage** - Gets the details of the icon path data from the [VectorImage](#) property and sets it to the SplitButton.

Note: The [SplitButton](#) loads icon in the following priority order,

- [IconTemplateSelector](#)

- [IconTemplate](#)
- [VectorImage](#)
- [LargeIcon](#)
- [MediumIcon](#)
- [SmallIcon](#)

Setting icon template selector

The [IconTemplateSelector](#) property provides support to specify a different data template based on the value of the [SizeForm](#) or [LayoutMode](#) properties. For simplified layout, the template content will be resized to 20 * 20 size which is the standard.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<syncfusion:RibbonWindow.Resources>
<DataTemplate x:Key="splitButtonSmallIconTemplate">
<Grid Width="16" Height="16">
<Path Data="M7,0C10.86,0,14,3.556,14,7.9259996L14,20.344C14,20.896
13.552,21.344 13,21.344 12.448,21.344 12,20.896
12,20.344L12,7.9259996C12,4.659 9.757,2 7,2 4.243,2 2,4.659 2,7.9259996L2,17
2,21 2,27.0569996C2,28.68 3.346,30 5,30 6.654,30 8,28.68 8,27.0569996L8,22
8,17 8,8.875C8,8.3930054 7.5510254,8 7,8 6.4489746,8 6,8.3930054
6,8.875L6,23C6,23.552002 5.552002,24 5,24 4.447998,24 4,23.552002
4,23L4,8.875C4,7.289978 5.3460083,6 7,6 8.6539917,6 10,7.289978
10,8.875L10,17 10,22 10,27.0569996C10,29.783001 7.757,32 5,32 2.243,32
0,29.783001 0,27.0569996L0,21 0,17 0,7.9259996C0,3.556,3.14,0,7,0z"
Stretch="Uniform" Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" 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>
</DataTemplate>
<DataTemplate x:Key="splitButtonLargeIconTemplate">
<Grid Margin="2">
<Path Data="M7,0C10.86,0,14,3.556,14,7.9259996L14,20.344C14,20.896
13.552,21.344 13,21.344 12.448,21.344 12,20.896
12,20.344L12,7.9259996C12,4.659 9.757,2 7,2 4.243,2 2,4.659 2,7.9259996L2,17
```

```

2,21 2,27.056999C2,28.68 3.346,30 5,30 6.654,30 8,28.68 8,27.056999L8,22
8,17 8,8.875C8,8.3930054 7.5510254,8 7,8 6.4489746,8 6,8.3930054
6,8.875L6,23C6,23.552002 5.552002,24 5,24 4.447998,24 4,23.552002
4,23L4,8.875C4,7.289978 5.3460083,6 7,6 8.6539917,6 10,7.289978
10,8.875L10,17 10,22 10,27.056999C10,29.783001 7.757,32 5,32 2.243,32
0,29.783001 0,27.056999L0,21 0,17 0,7.9259996C0,3.556,3.14,0,7,0z"
Stretch="Uniform" Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" 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>
</DataTemplate>
<local:SplitButtonIconTemplateSelector
x:Key="splitButtonIconTemplateSelector"
SmallTemplate="{StaticResource splitButtonSmallIconTemplate}"
LargeTemplate="{StaticResource splitButtonLargeIconTemplate}"/>
</syncfusion:RibbonWindow.Resources>
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Include">
<syncfusion:SplitButton Label="Attach" SizeForm="Large"
IconTemplateSelector="{StaticResource splitButtonIconTemplateSelector}"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

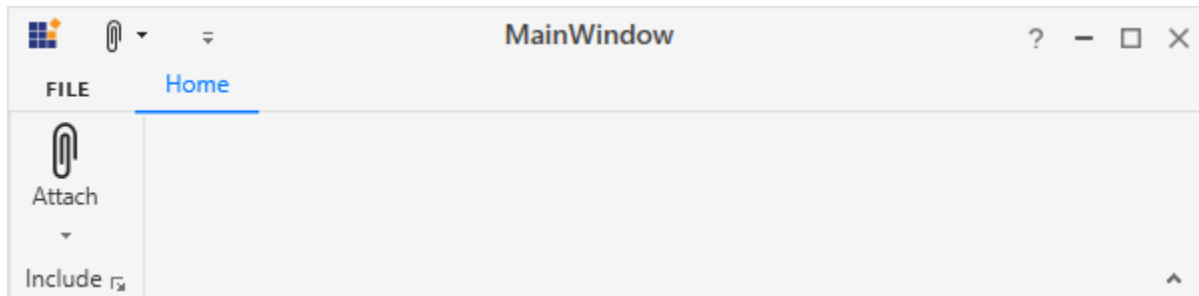
public class SplitButtonIconTemplateSelector : DataTemplateSelector
{
    public DataTemplate SmallTemplate { get; set; }
    public DataTemplate LargeTemplate { get; set; }
    public override DataTemplate SelectTemplate(object item, DependencyObject
container)
    {
        var item1 = (container as ContentPresenter);
        SplitButton splitButton = (item1.TemplatedParent as SplitButton);
        if (splitButton != null)
        {
            if (splitButton.SizeForm == SizeForm.Small || splitButton.SizeForm ==
SizeForm.ExtraSmall)
            {

```

```

return SmallTemplate;
}
else if (splitButton.SizeForm == SizeForm.Large)
{
return LargeTemplate;
}
}
return LargeTemplate;
}
}

```



Note: [View sample in GitHub](#)

Setting icon template

The [IconTemplate](#) property provides support to set any type of image such as glyph, font or any custom content to the [SplitButton](#). The [SplitButton](#) will automatically resize the template content according to its [SizeForm](#). For simplified layout, the template content will be resized to 20 * 20 size which is the standard.

XML

```

<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Include">
<syncfusion:SplitButton Label="Attach" SizeForm="Large">
<syncfusion:SplitButton.IconTemplate>
<DataTemplate>
<Grid Margin="2">
<Path Data="M7,0C10.86,0,14,3.556,14,7.9259996L14,20.344C14,20.896
13.552,21.344 13,21.344 12.448,21.344 12,20.896
12,20.344L12,7.9259996C12,4.659 9.757,2 7,2 4.243,2 2,4.659 2,7.9259996L2,17
2,21 2,27.0569996C2,28.68 3.346,30 5,30 6.654,30 8,28.68 8,27.0569996L8,22
8,17 8,8.875C8,8.3930054 7.5510254,8 7,8 6.4489746,8 6,8.3930054
6,8.875L6,23C6,23.552002 5.552002,24 5,24 4.447998,24 4,23.552002

```

```

4,23L4,8.875C4,7.289978 5.3460083,6 7,6 8.6539917,6 10,7.289978
10,8.875L10,17 10,22 10,27.056999C10,29.783001 7.757,32 5,32 2.243,32
0,29.783001 0,27.056999L0,21 0,17 0,7.9259996C0,3.556,3.14,0,7,0z"
Stretch="Uniform" Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" 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>
</DataTemplate>
</syncfusion:SplitButton.IconTemplate>
</syncfusion:SplitButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

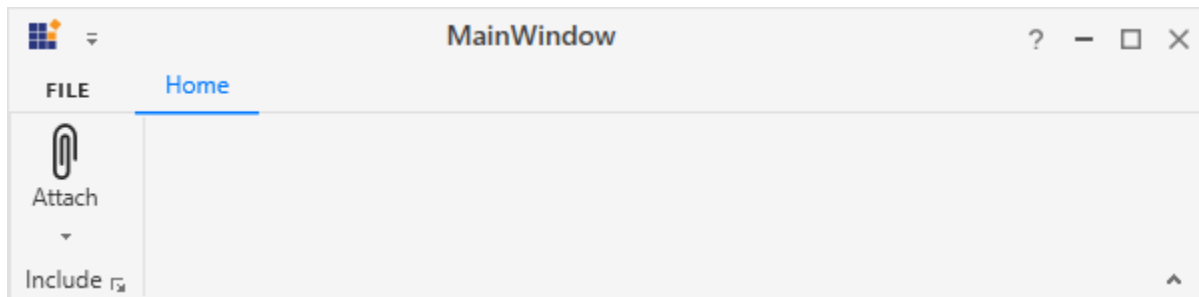
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar includeBar = new RibbonBar();
includeBar.Header = "Include";
// Creating items
SplitButton attachButton = new SplitButton();
attachButton.Label = "Attach";
attachButton.SizeForm = SizeForm.Large;
DataTemplate iconDataTemplate = new DataTemplate();
FrameworkElementFactory gridElement = new
FrameworkElementFactory(typeof(Grid));
FrameworkElementFactory pathElement1 = new
FrameworkElementFactory(typeof(Path));
gridElement.SetValue(Grid.MarginProperty, new Thickness(2));
pathElement1.SetValue(Path.DataProperty,
Geometry.Parse("M7,0C10.86,0,14,3.556,14,7.9259996L14,20.344C14,20.896
13.552,21.344 13,21.344 12.448,21.344 12,20.896
12,20.344L12,7.9259996C12,4.659 9.757,2 7,2 4.243,2 2,4.659 2,7.9259996L2,17
2,21 2,27.056999C2,28.68 3.346,30 5,30 6.654,30 8,28.68 8,27.056999L8,22
8,17 8,8.875C8,8.3930054 7.5510254,8 7,8 6.4489746,8 6,8.3930054
6,8.875L6,23C6,23.552002 5.552002,24 5,24 4.447998,24 4,23.552002
4,23L4,8.875C4,7.289978 5.3460083,6 7,6 8.6539917,6 10,7.289978
10,8.875L10,17 10,22 10,27.056999C10,29.783001 7.757,32 5,32 2.243,32
0,29.783001 0,27.056999L0,21 0,17 0,7.9259996C0,3.556,3.14,0,7,0z"));

```

```

pathElement1.SetValue(Path.FillProperty, new SolidColorBrush(Colors.Black));
pathElement1.SetValue(Path.StretchProperty, Stretch.Uniform);
gridElement.AppendChild(pathElement1);
iconDataTemplate.VisualTree = gridElement;
attachButton.IconTemplate = iconDataTemplate;
includeBar.Items.Add(attachButton);
// Adding bars to the tabs
homeTab.Items.Add(includeBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Note: [View sample in GitHub](#)

Setting image path

The [SplitButton](#) allows to set the image according to the different [SizeForm](#) values. To set the image to SplitButton, the following properties are used:

- [SmallIcon](#) - 16 * 16 size image to be displayed in normal layout for “ExtraSmall” and “Small” size form.
- [MediumIcon](#) - 20 * 20 size image to be displayed in simplified layout.
- [LargeIcon](#) - 32 * 32 size image to be displayed in normal layout for “Large” size form.

XML

```

<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStyle="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top"
EnableSimplifiedLayoutMode="True">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Tables">
<syncfusion:SplitButton
Label="Table"

```

```

LargeIcon="Resources/Table_32.png"
MediumIcon="Resources/Table20.png"
SizeForm="Large">
</syncfusion:SplitButton>
<syncfusion:SplitButton
Label="Shapes"
SmallIcon="Resources/Insert Shapes16.png"
MediumIcon="Resources/Insert Shapes20.png"
SizeForm="Small">
</syncfusion:SplitButton>
<syncfusion:SplitButton
Label="Chart"
SmallIcon="Resources/Base chart16.png"
MediumIcon="Resources/Base chart20.png"
SizeForm="Small">
</syncfusion:SplitButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar tablesBar = new RibbonBar();
tablesBar.Header = "Tables";
// Creating items
SplitButton tableButton = new SplitButton();
tableButton.Label = "Table";
tableButton.SizeForm = SizeForm.Large;
tableButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Table20.png",
UriKind.RelativeOrAbsolute));
tableButton.LargeIcon = new BitmapImage(new Uri(@"Resources/Table_32.png",
UriKind.RelativeOrAbsolute));
SplitButton shapesButton = new SplitButton();
shapesButton.Label = "Shapes";
shapesButton.SizeForm = SizeForm.Small;
shapesButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Insert
Shapes20.png", UriKind.RelativeOrAbsolute));
shapesButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Insert
Shapes16.png", UriKind.RelativeOrAbsolute));
SplitButton chartButton = new SplitButton();
chartButton.Label = "Chart";
chartButton.SizeForm = SizeForm.Small;
chartButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Base
chart20.png", UriKind.RelativeOrAbsolute));
chartButton.SmallIcon = new BitmapImage(new Uri(@"Resources/Base
chart16.png", UriKind.RelativeOrAbsolute));

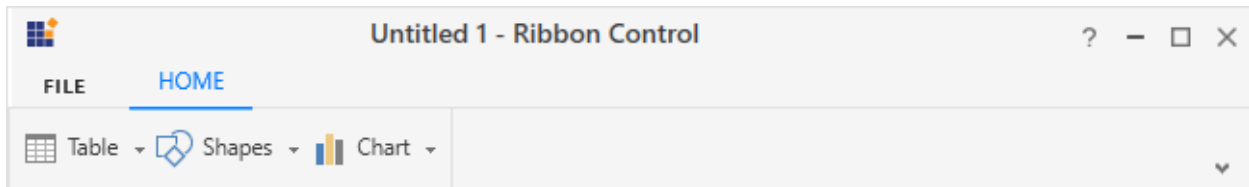
```



```
// Adding items to bar
tablesBar.Items.Add(tableButton);
tablesBar.Items.Add(shapesButton);
tablesBar.Items.Add(chartButton);
// Adding bars to the tabs
homeTab.Items.Add(tablesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
```



Normal layout



Simplified layout

Setting vector image

The [VectorImage](#) property is of type `ObservableCollection<Path>` which allows the image to be set as path type. The SplitButton will automatically resize the image according to its [SizeForm](#). For simplified layout, the image will be resized to 20 * 20 size which is the standard.

Note: The [IconTemplateSelector](#) and [IconTemplate](#) properties are the preferred options to set any type of image such as glyph, font or any custom content when compared to the [VectorImage](#) property.

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStyle="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top"
EnableSimplifiedLayoutMode="True">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Tables">
```

```

<syncfusion:SplitButton
IconType="VectorImage"
Label="Table"
SizeForm="Large">
<syncfusion:SplitButton.VectorImage>
<Path
Margin="0.5,4.5,0.5,0.5"
Data="M0,0 L27,0 27,23 0,23 z"
Fill="White"
Stretch="Fill" />
<Path
Height="4"
Margin="0.5,0.5,0.5,0"
VerticalAlignment="Top"
Data="M0,0 L27,0 27,4 0,4 z"
Fill="#FFC8C6C4"
Stretch="Fill" />
<Path
Margin="1,5,1,1"
Data="M9,8 L9,14 17,14 17,8 z M8,0 L9,0 9,7 17,7 17,0 18,0 18,7 26,7 26,8
18,8 18,14 26,14 26,15 18,15 18,22 17,22 17,15 9,15 9,22 8,22 8,15 0,15 0,14
8,14&#xa;8,8 0,8 0,7 8,7 z"
Fill="#FF797774"
Stretch="Fill" />
<Path
Data="M0.99999994,5.0000001 L0.99999994,27 27,27 27,5.0000001 z
M0.99999994,1 L0.99999994,4.0000002 27,4.0000002 27,1 z M0,0 L28,0
28,4.0000002 28,5.0000001 28,28 0,28 0,5.0000001 0,4.0000002 z"
Fill="#FF3A3A38"
Stretch="Fill" />
</syncfusion:SplitButton.VectorImage>
</syncfusion:SplitButton>
<syncfusion:SplitButton
IconType="VectorImage"
Label="Shapes"
SizeForm="Small">
<syncfusion:SplitButton.VectorImage>
<Path
Margin="0.5,0.5,9.5,9.5"
Data="M0,0 L18,0 18,6.5250005 C17.833,6.5180005 17.668999,6.5000008
17.5,6.5000008 11.434998,6.5000008 6.5,11.435001 6.5,17.5 6.5,17.669002
6.5179977,17.833 6.5249977,18 L0,18 z"
Fill="#FF83BEEC"
Stretch="Fill" />
<Path
Margin="8.5,8.5,0.5,0.5"
Data="M9.5000001,0 C14.746998,0 19,4.253001 19,9.4999988 19,14.747001
14.746998,19.000001 9.5000001,19.000001 4.2539979,19.000001 8.1026528E-
08,14.747001 0,9.4999988 8.1026528E-08,4.253001 4.2539979,0 9.5000001,0 z"
Fill="White"
Stretch="Fill" />
<Path
Margin="8,8,0,0"
Data="M10,0.99999995 C5.0369988,1.0000001 1.0000001,5.0370014
0.99999998,9.9999989 1.0000001,14.963001 5.0369988,19.000001 10,19.000001
14.962998,19.000001 19,14.963001 19,9.9999989 19,5.0370014
14.962998,1.0000001 10,0.99999995 z M10,0 C15.514,0 20,4.4860004

```

```

20,9.9999989 20,15.514 15.514,20.000001 10,20.000001 4.4860002,20.000001
9.5927689E-08,15.514 0,9.9999989 9.5927689E-08,4.4860004 4.4860002,0 10,0 z"
Fill="#FF3A3A38"
Stretch="Fill" />
<Path
Margin="0,0,9,9"
Data="M0,0 L19,0 19,7.0510015 C18.669998,7.0210008 18.337997,7.0000009
18,7.0000009 L18,1.0000002 0.99999988,1.0000002 0.99999988,18 7,18
C7,18.338001 7.0209999,18.670002 7.0509987,19 L0,19 z"
Fill="#FF0063B1"
Stretch="Fill" />
</syncfusion:SplitButton.VectorImage>
</syncfusion:SplitButton>
<syncfusion:SplitButton
IconType="VectorImage"
Label="Chart"
SizeForm="Small">
<syncfusion:SplitButton.VectorImage>
<Path
Width="8"
Margin="0.5,11.5,0,0.5"
HorizontalAlignment="Left"
Data="M0,0 L7.9999996,0 7.9999996,16 0,16 z"
Fill="#FFC8C6C4"
Stretch="Fill" />
<Path
Width="9"
Margin="0,11,0,0"
HorizontalAlignment="Left"
Data="M0.99999667,0.99999994 L0.99999667,16 8.0000032,16
8.0000032,0.99999994 z M0,0 L9.0000004,0 9.0000004,17 3.2610174,17
2.5740174,17 0,17 z"
Fill="#FF797774"
Stretch="Fill" />
<Path
Width="8"
Margin="0,5.5,0.5,0.5"
HorizontalAlignment="Right"
Data="M0,0 L8.0000001,0 8.0000001,22 0,22 z"
Fill="#FF83BEEC"
Stretch="Fill" />
<Path
Width="9"
Margin="0,5,0,0"
HorizontalAlignment="Right"
Data="M1.0000001,1.0000001 L1.0000001,22 8,22 8,1.0000001 z M0,0 L9,0 9,23
0,23 z"
Fill="#FF0063B1"
Stretch="Fill" />
<Path
Margin="8.5,0.5"
Data="M0,0 L8,0 8,27 0,27 z"
Fill="White"
Stretch="Fill" />
<Path
Margin="8,0"

```

```

Data="M0.99999994,0.99999994 L0.99999994,27 7.9999999,27
7.9999999,0.99999994 z M0,0 L8.9999999,0 8.9999999,28 2.355,28 1.7929999,28
0,28 z"
Fill="#FF3A3A38"
Stretch="Fill" />
</syncfusion:SplitButton.VectorImage>
</syncfusion:SplitButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar tablesBar = new RibbonBar();
tablesBar.Header = "Tables";
// Creating items
SplitButton tableButton = new SplitButton();
tableButton.Label = "Table";
tableButton.SizeForm = SizeForm.Large;
tableButton.IconType = IconType.VectorImage;
Path tablePath1 = new Path();
tablePath1.Data = Geometry.Parse("M0,0 L27,0 27,23 0,23 z");
tablePath1.Margin = new Thickness(0.5, 4.5, 0.5, 0.5);
tablePath1.Fill = new SolidColorBrush(Colors.White);
Path tablePath2 = new Path();
tablePath2.Data = Geometry.Parse("M0,0 L27,0 27,4 0,4 z");
tablePath2.Margin = new Thickness(0.5, 0.5, 0.5, 0);
tablePath2.Fill = new SolidColorBrush(Color.FromRgb(200, 198, 196));
tablePath2.Height = 4;
tablePath2.VerticalAlignment = VerticalAlignment.Top;
Path tablePath3 = new Path();
tablePath3.Data = Geometry.Parse("M9,8 L9,14 17,14 17,8 z M8,0 L9,0 9,7 17,7
17,0 18,0 18,7 26,7 26,8 18,8 18,14 26,14 26,15 18,15 18,22 17,22 17,15 9,15
9,22 8,22 8,15 0,15 0,14 8,14&#xa;8,8 0,8 0,7 8,7 z");
tablePath3.Margin = new Thickness(1, 5, 1, 1);
tablePath3.Fill = new SolidColorBrush(Color.FromRgb(121, 119, 116));
Path tablePath4 = new Path();
tablePath4.Data = Geometry.Parse("M0.99999994,5.0000001 L0.99999994,27 27,27
27,5.0000001 z M0.99999994,1 L0.99999994,4.0000002 27,4.0000002 27,1 z M0,0
L28,0 28,4.0000002 28,5.0000001 28,28 0,28 0,5.0000001 0,4.0000002 z");
tablePath4.Fill = new SolidColorBrush(Color.FromRgb(58, 58, 56));
tableButton.VectorImage.Add(tablePath1);
tableButton.VectorImage.Add(tablePath2);
tableButton.VectorImage.Add(tablePath3);
tableButton.VectorImage.Add(tablePath4);
SplitButton shapesButton = new SplitButton();

```

```

shapesButton.Label = "Shapes";
shapesButton.SizeForm = SizeForm.Small;
shapesButton.IconType = IconType.VectorImage;
Path shapePath1 = new Path();
shapePath1.Data = Geometry.Parse("M0,0 L18,0 18,6.5250005 C17.833,6.5180005 17.668999,6.5000008 17.5,6.5000008 11.434998,6.5000008 6.5,11.435001 6.5,17.5 6.5,17.669002 6.5179977,17.833 6.5249977,18 L0,18 z");
shapePath1.Margin = new Thickness(0.5, 0.5, 9.5, 9.5);
shapePath1.Fill = new SolidColorBrush(Color.FromRgb(131, 190, 236));
Path shapePath2 = new Path();
shapePath2.Data = Geometry.Parse("M9.5000001,0 C14.746998,0 19,4.253001 19,9.4999988 19,14.747001 14.746998,19.000001 9.5000001,19.000001 4.2539979,19.000001 8.1026528E-08,14.747001 0,9.4999988 8.1026528E-08,4.253001 4.2539979,0 9.5000001,0 z");
shapePath2.Margin = new Thickness(8.5, 8.5, 0.5, 0.5);
shapePath2.Fill = new SolidColorBrush(Colors.White);
Path shapePath3 = new Path();
shapePath3.Data = Geometry.Parse("M10,0.99999995 C5.0369988,1.0000001 1.0000001,5.0370014 0.99999998,9.9999989 1.0000001,14.963001 5.0369988,19.000001 10,19.000001 14.962998,19.000001 19,14.963001 19,9.9999989 19,5.0370014 14.962998,1.0000001 10,0.99999995 z M10,0 C15.514,0 20,4.4860004 20,9.9999989 20,15.514 15.514,20.000001 10,20.000001 4.4860002,20.000001 9.5927689E-08,15.514 0,9.9999989 9.5927689E-08,4.4860004 4.4860002,0 10,0 z");
shapePath3.Margin = new Thickness(8, 8, 0, 0);
shapePath3.Fill = new SolidColorBrush(Color.FromRgb(58, 58, 56));
Path shapePath4 = new Path();
shapePath4.Data = Geometry.Parse("M0,0 L19,0 19,7.0510015 C18.669998,7.0210008 18.337997,7.0000009 18,7.0000009 L18,1.0000002 0.9999988,1.0000002 0.9999988,18 7,18 C7,18.338001 7.0209999,18.670002 7.0509987,19 L0,19 z");
shapePath4.Margin = new Thickness(0, 0, 9, 9);
shapePath4.Fill = new SolidColorBrush(Color.FromRgb(0, 99, 177));
shapesButton.VectorImage.Add(shapePath1);
shapesButton.VectorImage.Add(shapePath2);
shapesButton.VectorImage.Add(shapePath3);
shapesButton.VectorImage.Add(shapePath4);
SplitButton chartButton = new SplitButton();
chartButton.Label = "Chart";
chartButton.SizeForm = SizeForm.Small;
chartButton.IconType = IconType.VectorImage;
Path chartPath1 = new Path();
chartPath1.Data = Geometry.Parse("M0,0 L7.9999996,0 7.9999996,16 0,16 z");
chartPath1.Margin = new Thickness(0.5, 11.5, 0, 0.5);
chartPath1.Fill = new SolidColorBrush(Color.FromRgb(200, 198, 196));
chartPath1.HorizontalAlignment = HorizontalAlignment.Left;
chartPath1.Width = 8;
Path chartPath2 = new Path();
chartPath2.Data = Geometry.Parse("M0.99999667,0.99999994 L0.99999667,16 8.0000032,16 8.0000032,0.99999994 z M0,0 L9.0000004,0 9.0000004,17 3.2610174,17 2.5740174,17 0,17 z");
chartPath2.Margin = new Thickness(0, 11, 0, 0);
chartPath2.Fill = new SolidColorBrush(Color.FromRgb(121, 119, 116));
chartPath2.HorizontalAlignment = HorizontalAlignment.Left;
chartPath2.Width = 9;
Path chartPath3 = new Path();
chartPath3.Data = Geometry.Parse("M0,0 L8.0000001,0 8.0000001,22 0,22 z");

```

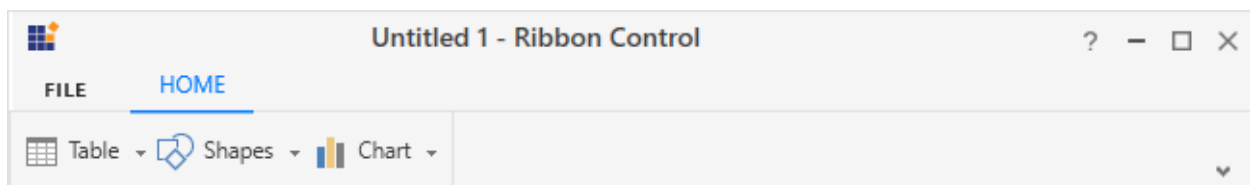
```

chartPath3.Margin = new Thickness(0, 5.5, 0.5, 0.5);
chartPath3.Fill = new SolidColorBrush(Color.FromRgb(131, 190, 236));
chartPath3.HorizontalAlignment = HorizontalAlignment.Right;
chartPath3.Width = 8;
Path chartPath4 = new Path();
chartPath4.Data = Geometry.Parse("M1.0000001,1.0000001 L1.0000001,22 8,22 8,1.0000001 z M0,0 L9,0 9,23 0,23 z");
chartPath4.Margin = new Thickness(0, 5, 0, 0);
chartPath4.Fill = new SolidColorBrush(Color.FromRgb(0, 99, 177));
chartPath4.HorizontalAlignment = HorizontalAlignment.Right;
chartPath4.Width = 9;
Path chartPath5 = new Path();
chartPath5.Data = Geometry.Parse("M0,0 L8,0 8,27 0,27 z");
chartPath5.Margin = new Thickness(8.5, 0.5, 8.5, 0.5);
chartPath5.Fill = new SolidColorBrush(Colors.White);
Path chartPath6 = new Path();
chartPath6.Data = Geometry.Parse("M0.99999994,0.99999994 L0.99999994,27 7.9999999,27 7.9999999,0.99999994 z M0,0 L8.9999999,0 8.9999999,28 2.355,28 1.7929999,28 0,28 z");
chartPath6.Margin = new Thickness(8, 0, 8, 0);
chartPath6.Fill = new SolidColorBrush(Color.FromRgb(58, 58, 56));
chartButton.VectorImage.Add(chartPath1);
chartButton.VectorImage.Add(chartPath2);
chartButton.VectorImage.Add(chartPath3);
chartButton.VectorImage.Add(chartPath4);
chartButton.VectorImage.Add(chartPath5);
chartButton.VectorImage.Add(chartPath6);
// Adding items to bar
tablesBar.Items.Add(tableButton);
tablesBar.Items.Add(shapesButton);
tablesBar.Items.Add(chartButton);
// Adding bars to the tabs
homeTab.Items.Add(tablesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Normal layout



*Simplified layout**Add SplitButton to the simplified layout*

When the simplified layout is enabled, the [SplitButton](#) can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

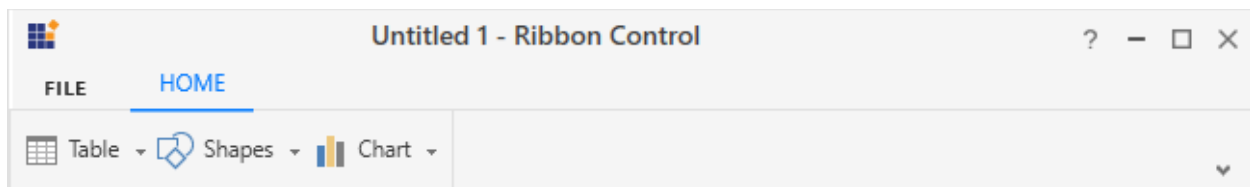
XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top"
EnableSimplifiedLayoutMode="True">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Tables">
<syncfusion:SplitButton
Label="Table"
MediumIcon="Resources/Table20.png"
SizeForm="Large">
</syncfusion:SplitButton>
<syncfusion:SplitButton
Label="Shapes"
MediumIcon="Resources/Insert Shapes20.png"
SizeForm="Small">
</syncfusion:SplitButton>
<syncfusion:SplitButton
Label="Chart"
MediumIcon="Resources/Base chart20.png"
SizeForm="Small">
</syncfusion:SplitButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar tablesBar = new RibbonBar();
tablesBar.Header = "Tables";
```

```
// Creating items
SplitButton tableButton = new SplitButton();
tableButton.Label = "Table";
tableButton.SizeForm = SizeForm.Large;
tableButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Table20.png",
UriKind.RelativeOrAbsolute));
SplitButton shapesButton = new SplitButton();
shapesButton.Label = "Shapes";
shapesButton.SizeForm = SizeForm.Small;
shapesButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Insert
Shapes20.png", UriKind.RelativeOrAbsolute));
SplitButton chartButton = new SplitButton();
chartButton.Label = "Chart";
chartButton.SizeForm = SizeForm.Small;
chartButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Base
chart20.png", UriKind.RelativeOrAbsolute));
// Adding items to bar
tablesBar.Items.Add(tableButton);
tablesBar.Items.Add(shapesButton);
tablesBar.Items.Add(chartButton);
// Adding bars to the tabs
homeTab.Items.Add(tablesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
```



When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the SplitButton can be ignored as it will be resized automatically to the standard width and height. If the SplitButton is to be shown in both normal and simplified layout, **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```
<syncfusion:SplitButton Label="Copy"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified"
SizeForm="Small" MediumIcon="/Resources/Copy_20.png"
SmallIcon="/Resources/Copy16.png" >
<syncfusion:SplitButton.Style>
<Style TargetType="syncfusion:SplitButton" BasedOn="{StaticResource
SyncfusionSplitButtonStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="48"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
```



```

</Style>
</syncfusion:SplitButton.Style>
</syncfusion:SplitButton >

```

RibbonComboBox in WPF Ribbon

RibbonComboBox control is used to display the list of items, as ComboBox.

Add ComboBoxItems

ComboBoxItems are the items with **Header** property that is used to set header.

XML

```

<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Name="_ribbonTab1" Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="_ribbonBar1" Header="RibbonBar1">
<syncfusion:RibbonButton Label="Cut"/>
<syncfusion:RibbonButton Label="Copy"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="_ribbonBar2" Width="150" Header="RibbonBar2">
<syncfusion:ButtonPanel>
<syncfusion:RibbonComboBox SelectedItem="Arial" Width="80">
<syncfusion:RibbonComboBoxItem>Arial</syncfusion:RibbonComboBoxItem>
<syncfusion:RibbonComboBoxItem>Tahoma</syncfusion:RibbonComboBoxItem>
<syncfusion:RibbonComboBoxItem>Calibri</syncfusion:RibbonComboBoxItem>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox SelectedItem="12" Width="50">
<syncfusion:RibbonComboBoxItem>11</syncfusion:RibbonComboBoxItem>
<syncfusion:RibbonComboBoxItem>12</syncfusion:RibbonComboBoxItem>
<syncfusion:RibbonComboBoxItem>13</syncfusion:RibbonComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="EDIT" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>

```

Create instance for **RibbonComboBox** and add it to RibbonBar Items in Code behind

C#

```

ButtonPanel _buttonPanel = new ButtonPanel();
RibbonComboBox _ribbonComboBox1 = new RibbonComboBox() { Width=80};
RibbonComboBox _ribbonComboBox2 = new RibbonComboBox() { Width=50};
RibbonComboBoxItem comboBoxItem1 = new RibbonComboBoxItem() { Content =
"Arial" };
RibbonComboBoxItem comboBoxItem2 = new RibbonComboBoxItem() { Content =
"Calibri" };
RibbonComboBoxItem comboBoxItem3 = new RibbonComboBoxItem() { Content =
"Tahoma" };
RibbonComboBoxItem comboBoxItem4 = new RibbonComboBoxItem() { Content = "11"
};
RibbonComboBoxItem comboBoxItem5 = new RibbonComboBoxItem() { Content = "12"
};

```

```

RibbonComboBoxItem comboBoxItem6 = new RibbonComboBoxItem() { Content = "13"
};
_ribbonComboBox.Items.Add(comboBoxItem1);
_ribbonComboBox.Items.Add(comboBoxItem2);
_ribbonComboBox.Items.Add(comboBoxItem3);
_RibbonComboBox1.Items.Add(comboBoxItem4);
_ribbonComboBox1.Items.Add(comboBoxItem5);
_ribbonComboBox1.Items.Add(comboBoxItem6);
_buttonPanel.Children.Add(_ribbonComboBox1);
_buttonPanel.Children.Add(_ribbonComboBox2);
_ribbonBar2.Items.Add(_buttonPanel);

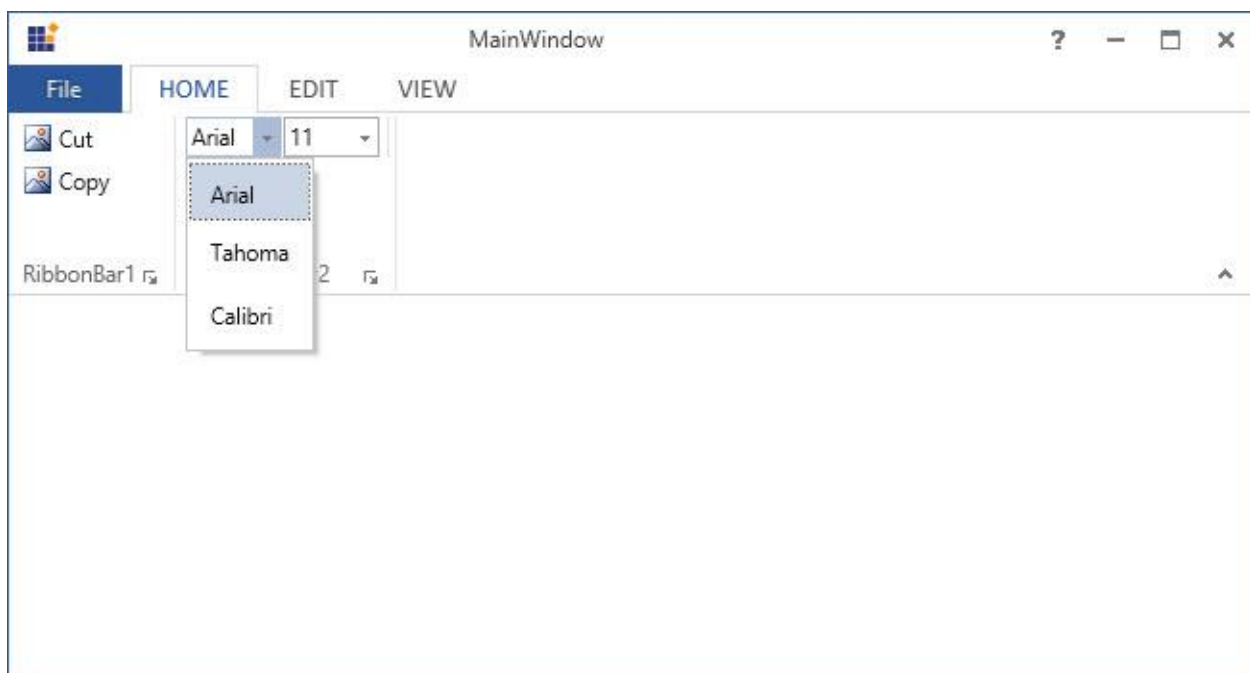
```

VB.NET

```

Dim _buttonPanel As New ButtonPanel()
Dim _ribbonComboBox1 As New RibbonComboBox() With {.Width=80}
Dim _ribbonComboBox2 As New RibbonComboBox() With {.Width=50}
Dim comboBoxItem1 As New RibbonComboBoxItem() With {.Content = "Arial"}
Dim comboBoxItem2 As New RibbonComboBoxItem() With {.Content = "Calibri"}
Dim comboBoxItem3 As New RibbonComboBoxItem() With {.Content = "Tahoma"}
Dim comboBoxItem4 As New RibbonComboBoxItem() With {.Content = "11"}
Dim comboBoxItem5 As New RibbonComboBoxItem() With {.Content = "12"}
Dim comboBoxItem6 As New RibbonComboBoxItem() With {.Content = "13"}
_ribbonComboBox.Items.Add(comboBoxItem1)
_ribbonComboBox.Items.Add(comboBoxItem2)
_ribbonComboBox.Items.Add(comboBoxItem3)
_RibbonComboBox1.Items.Add(comboBoxItem4)
_ribbonComboBox1.Items.Add(comboBoxItem5)
_ribbonComboBox1.Items.Add(comboBoxItem6)
_buttonPanel.Children.Add(_ribbonComboBox1)
_buttonPanel.Children.Add(_ribbonComboBox2)
_ribbonBar2.Items.Add(_buttonPanel)

```



Add ComboBox to the simplified layout

When the simplified layout is enabled, the RibbonComboBox can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top"
EnableSimplifiedLayoutMode="True" LayoutMode="Simplified">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Font">
<syncfusion:RibbonComboBox Width="80">
<syncfusion:RibbonComboBoxItem>Arial</syncfusion:RibbonComboBoxItem>
<syncfusion:RibbonComboBoxItem
IsSelected="True">Tahoma</syncfusion:RibbonComboBoxItem>
<syncfusion:RibbonComboBoxItem>Calibri</syncfusion:RibbonComboBoxItem>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox Width="50">
<syncfusion:RibbonComboBoxItem>11</syncfusion:RibbonComboBoxItem>
<syncfusion:RibbonComboBoxItem
IsSelected="True">12</syncfusion:RibbonComboBoxItem>
<syncfusion:RibbonComboBoxItem>13</syncfusion:RibbonComboBoxItem>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar fontBar = new RibbonBar();
fontBar.Header = "Font";
// Creating items
RibbonComboBox ribbonComboBox1 = new RibbonComboBox() { Width = 80 };
RibbonComboBox ribbonComboBox2 = new RibbonComboBox() { Width = 50 };
```

```

RibbonComboBoxItem comboBoxItem1 = new RibbonComboBoxItem() { Content =
"Arial" };
RibbonComboBoxItem comboBoxItem2 = new RibbonComboBoxItem() { Content =
"Calibri" };
RibbonComboBoxItem comboBoxItem3 = new RibbonComboBoxItem() { Content =
"Tahoma", IsSelected=true };
RibbonComboBoxItem comboBoxItem4 = new RibbonComboBoxItem() { Content = "11"
};
RibbonComboBoxItem comboBoxItem5 = new RibbonComboBoxItem() { Content =
"12", IsSelected = true };
RibbonComboBoxItem comboBoxItem6 = new RibbonComboBoxItem() { Content = "13"
};
ribbonComboBox1.Items.Add(comboBoxItem1);
ribbonComboBox1.Items.Add(comboBoxItem2);
ribbonComboBox1.Items.Add(comboBoxItem3);
ribbonComboBox2.Items.Add(comboBoxItem4);
ribbonComboBox2.Items.Add(comboBoxItem5);
ribbonComboBox2.Items.Add(comboBoxItem6);
// Adding items to bar
fontBar.Items.Add(ribbonComboBox1);
fontBar.Items.Add(ribbonComboBox2);
// Adding bars to the tabs
homeTab.Items.Add(fontBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the **RibbonComboBox** can be ignored as it will be resized automatically to the standard width and height. If the **RibbonComboBox** is to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```

<syncfusion:RibbonComboBox
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified" >
<syncfusion:RibbonComboBox.Style>
<Style TargetType="syncfusion:RibbonComboBox" BasedOn="{StaticResource
SyncfusionRibbonComboBoxStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="25"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>

```

```
</syncfusion:RibbonComboBox.Style>
</syncfusion:RibbonComboBox >
```

Autosize width in RibbonComboBox

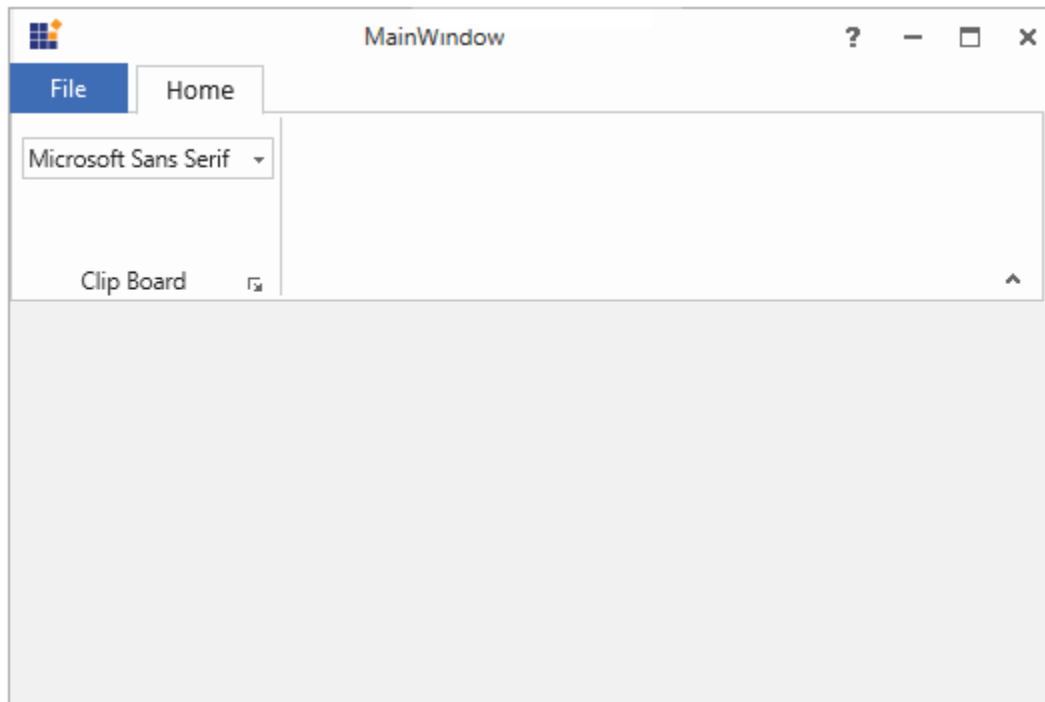
RibbonComboBox can be auto sized by not providing width to the RibbonComboBox and the width will be based on the selected item text.

If the width is given, then the width is set to the RibbonComboBox instead of the selected item text.

The following code example demonstrates how to set auto-width to the ComboBoxAdv control.

XML

```
<syncfusion:Ribbon x:Name="ribbon" HorizontalAlignment="Left" Height="94"
VerticalAlignment="Top" Width="517">
  <syncfusion:RibbonTab Background="Transparent" Caption="Home"
FocusVisualStyle="{x:Null}" Focusable="False" MinWidth="23">
    <syncfusion:RibbonBar Focusable="False" Header="Clip Board">
      <syncfusion:RibbonComboBox SelectedItem="Arial" Margin="0,10,0,0">
        <syncfusion:RibbonComboBoxItem>Arial</syncfusion:RibbonComboBoxItem>
        <syncfusion:RibbonComboBoxItem>Tahoma</syncfusion:RibbonComboBoxItem>
        <syncfusion:RibbonComboBoxItem>Microsoft Sans
Serif</syncfusion:RibbonComboBoxItem>
      </syncfusion:RibbonComboBox>
    </syncfusion:RibbonBar>
  </syncfusion:RibbonTab>
</syncfusion:Ribbon>
```



RibbonGallery in WPF Ribbon

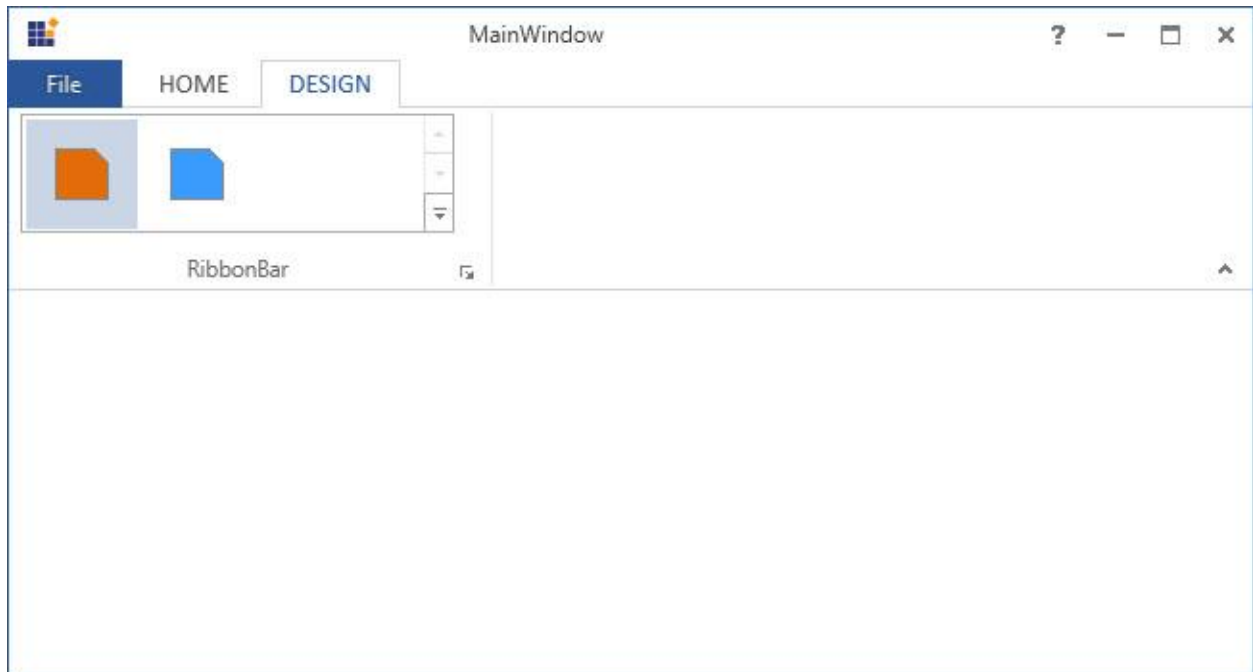
RibbonGallery provides `VisualMode` property, that helps to display items in two different ways.

Ribbon mode

To display items as a normal gallery control in the ribbon, set `VisualMode` property as `InRibbon mode`

XML

```
<syncfusion:RibbonGallery Width="230" VisualMode="InRibbon"  
Label="RibbonGallery" LargeIcon="Word.png"/>
```

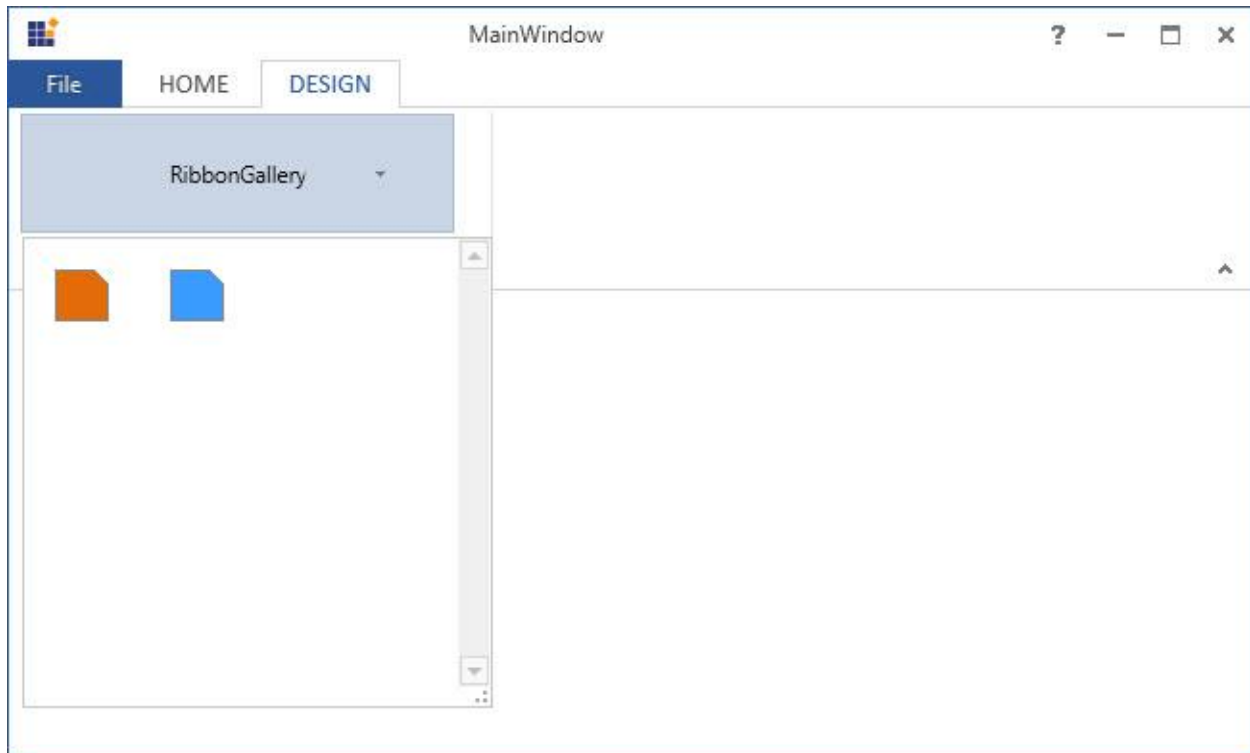


DropDown mode

To display items as DropDown in the ribbon, set `VisualMode` property as `DropDown mode`

XML

```
<syncfusion:RibbonGallery VisualMode="DropDown" Label="RibbonGallery"  
LargeIcon="Word.png" >
```



Ribbon gallery item

Ribbon control provides `RibbonGalleryItem` that add as items in RibbonGallery.

XML

```
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
  <syncfusion:RibbonTab Caption="HOME" IsChecked="False"/>
  <syncfusion:RibbonTab IsChecked="True" Caption="DESIGN">
    <syncfusion:RibbonBar Header="RibbonBar">
      <syncfusion:RibbonGallery Name="_ribbonGallery" Width="230"
VisualMode="InRibbon" Label="RibbonGallery" LargeIcon="Word.png" >
        <syncfusion:RibbonGalleryItem>
          <Image Source="OrangeLarge.png"/>
        </syncfusion:RibbonGalleryItem>
        <syncfusion:RibbonGalleryItem >
          <Image Source="BlueLarge.png"/>
        </syncfusion:RibbonGalleryItem>
      </syncfusion:RibbonGallery>
    </syncfusion:RibbonBar>
  </syncfusion:RibbonTab>
</syncfusion:Ribbon>
```

Add `RibbonGalleryItem` in code behind.

C#

```
Image _image1 = new Image() { Source =new BitmapImage(new
Uri(@"OrangeLarge.png", UriKind.RelativeOrAbsolute)) };
Image _image2 = new Image() { Source = new BitmapImage(new
Uri(@"BlueLarge.png", UriKind.RelativeOrAbsolute)) };
```

```

RibbonGalleryItem _ribbonGalleryItem1 = new RibbonGalleryItem()
{Content=_image1};
RibbonGalleryItem _ribbonGalleryItem2 = new RibbonGalleryItem() { Content =
_image2 };
_ribbonGallery.Items.Add(_ribbonGalleryItem1);
_ribbonGallery.Items.Add(_ribbonGalleryItem2);

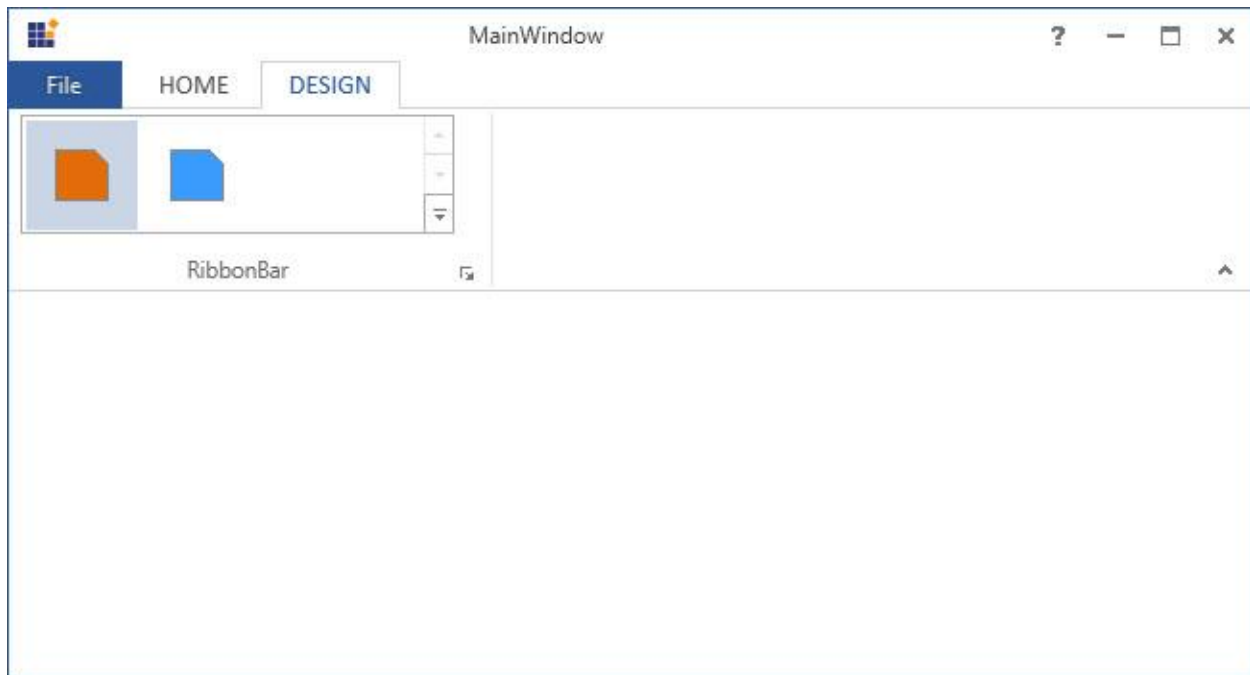
```

VB.NET

```

Dim _image1 As New Image() With {.Source = New BitmapImage(New
Uri("OrangeLarge.png", UriKind.RelativeOrAbsolute))}
Dim _image2 As New Image() With {.Source = New BitmapImage(New
Uri("BlueLarge.png", UriKind.RelativeOrAbsolute))}
Dim _ribbonGalleryItem1 As New RibbonGalleryItem() With {.Content=_image1}
Dim _ribbonGalleryItem2 As New RibbonGalleryItem() With {.Content = _image2}
_ribbonGallery.Items.Add(_ribbonGalleryItem1)
_ribbonGallery.Items.Add(_ribbonGalleryItem2)

```



Ribbon gallery group

Ribbon Gallery Group is a collection of Ribbon Gallery Items. The items are grouped in the Ribbon Gallery control based on some classifications.

Gallery filter

GalleryGroupFilters are used to view particular group. The `FilterIndexes` property is used to specify the indexes of the filters.

XML

```

<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
  <syncfusion:RibbonTab Caption="HOME" IsChecked="False"/>
  <syncfusion:RibbonTab Caption="DESIGN" IsChecked="True">
    <syncfusion:RibbonBar Width="250" Header="RibbonBar">

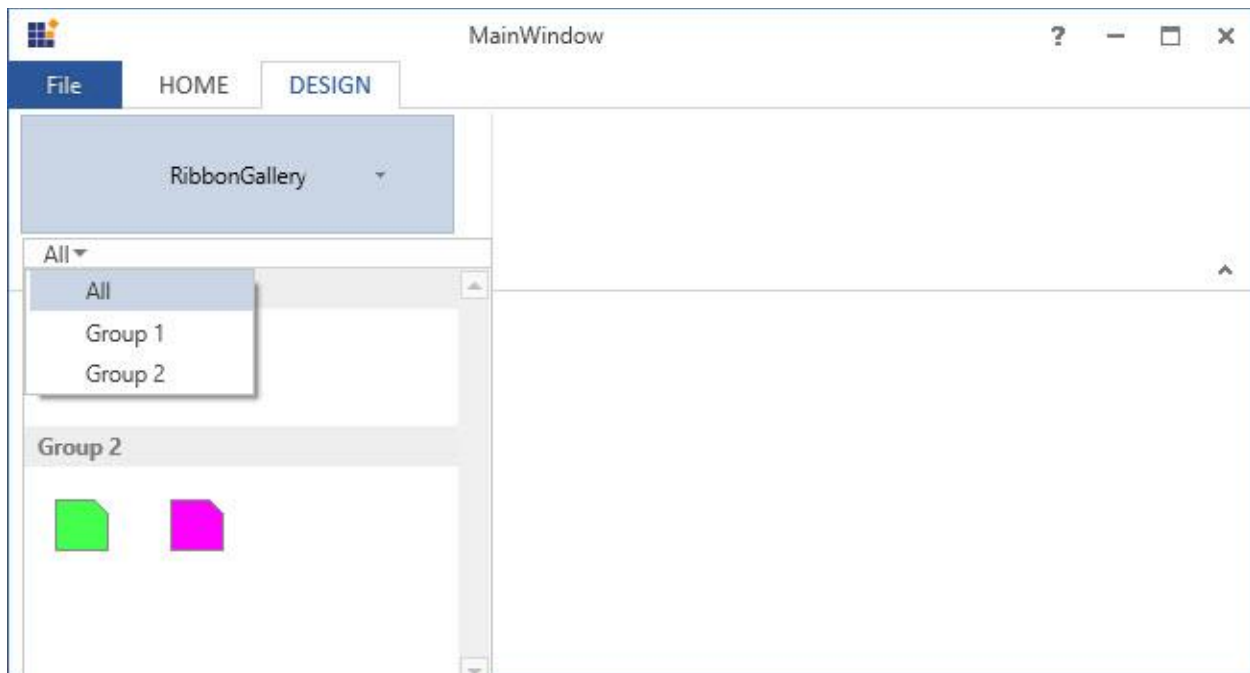
```



```

<syncfusion:RibbonGallery Name="_ribbonGallery" Width="230"
VisualMode="DropDown" Label="RibbonGallery">
<syncfusion:RibbonGallery.GalleryFilters>
<syncfusion:RibbonGalleryFilter Label="All"/>
<syncfusion:RibbonGalleryFilter Label="Group 1"/>
<syncfusion:RibbonGalleryFilter Label="Group 2"/>
</syncfusion:RibbonGallery.GalleryFilters>
<syncfusion:RibbonGallery.GalleryGroups>
<syncfusion:RibbonGalleryGroup Name="_ribbonGalleryGroup1" Label="Group 1"
syncfusion:RibbonGallery.FilterIndexes="0, 1">
<syncfusion:RibbonGalleryItem Name="_ribbonGalleryItem1" Margin="5">
<Image Source="OrangeLarge.png" Stretch="None" />
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem Name="_ribbonGalleryItem2" Margin="5" >
<Image Source="BlueLarge.png" Stretch="None"/>
</syncfusion:RibbonGalleryItem>
</syncfusion:RibbonGalleryGroup>
<syncfusion:RibbonGalleryGroup Name="_ribbonGalleryGroup2" Label="Group 2"
syncfusion:RibbonGallery.FilterIndexes="0, 2">
<syncfusion:RibbonGalleryItem Name="_RibbonGalleryItem3" Margin="5">
<Image Source="GreenLarge.png" Stretch="None"/>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem Name="_RibbonGalleryItem4" Margin="5" >
<Image Source="PinkLarge.png" Stretch="None" />
</syncfusion:RibbonGalleryItem>
</syncfusion:RibbonGalleryGroup>
</syncfusion:RibbonGallery.GalleryGroups>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>

```



Add custom menu items

In the expanded Gallery items (in both the Visual Mode), can add custom menu items to the bottom of the Ribbon Gallery control, using the `MenuItem` property of `RibbonGallery`.

XML

```
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
  <syncfusion:RibbonTab Caption="HOME" IsChecked="True"/>
  <syncfusion:RibbonTab IsChecked="False" Caption="DESIGN">
    <syncfusion:RibbonBar Width="250" Header="RibbonBar">
      <syncfusion:RibbonGallery Name="_ribbonGallery" Width="230"
VisualMode="InRibbon" Label="RibbonGallery" LargeIcon="Word.png" >
        <syncfusion:RibbonGalleryItem Margin="5">
          <Image Source="OrangeLarge.png"/>
        </syncfusion:RibbonGalleryItem>
        <syncfusion:RibbonGalleryItem Margin="5">
          <Image Source="PinkLarge.png"/>
        </syncfusion:RibbonGalleryItem>
        <syncfusion:RibbonGallery.MenuItems>
          <syncfusion:RibbonButton SizeForm = "Small" Label="Menu Item-1"/>
          <syncfusion:RibbonButton SizeForm = "Small" Label="Menu Item-2"/>
          <syncfusion:RibbonButton SizeForm = "Small" Label="Menu Item-3"/>
        </syncfusion:RibbonGallery.MenuItems>
      </syncfusion:RibbonGallery>
    </syncfusion:RibbonBar>
  </syncfusion:RibbonTab>
</syncfusion:Ribbon>
```

Custom Menu items of `RibbonGallery` added by creating instance of `RibbonButton` and add it to `MenuItems` property of `RibbonGallery` in code behind. `SplitButton` or `MenuButton` can also added instead of `RibbonButton`.

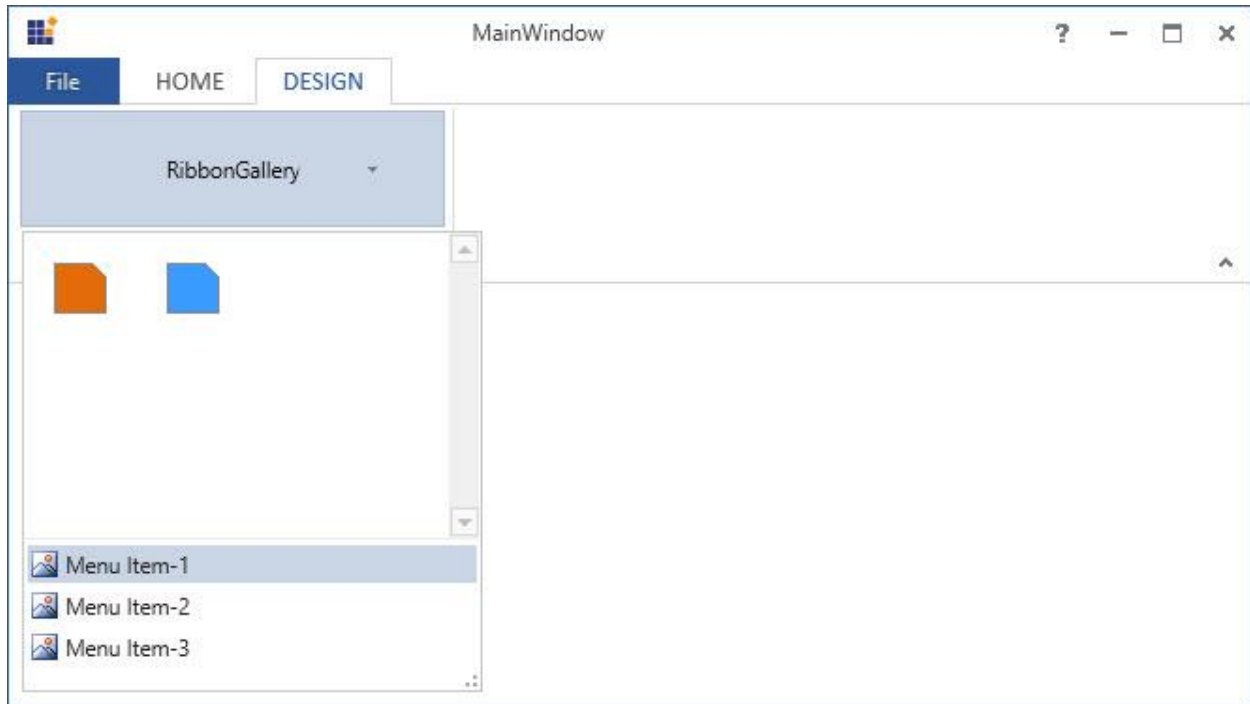
C#

```
RibbonButton _ribbonButton1 = new RibbonButton() { SizeForm =
SizeForm.Small, Label = "Menu Item-1" };
RibbonButton _ribbonButton2 = new RibbonButton() { SizeForm =
SizeForm.Small, Label = "Menu Item-2" };
RibbonButton _ribbonButton3 = new RibbonButton() { SizeForm =
SizeForm.Small, Label = "Menu Item-3" };
_ribbonGallery.MenuItems.Add(_ribbonButton1);
_ribbonGallery.MenuItems.Add(_ribbonButton2);
_ribbonGallery.MenuItems.Add(_ribbonButton3);
```

VB.NET

```
Dim _ribbonButton1 As New RibbonButton() With {
    .SizeForm = SizeForm.Small,
    .Label = "Menu Item-1"
}
Dim _ribbonButton2 As New RibbonButton() With {
    .SizeForm = SizeForm.Small,
    .Label = "Menu Item-2"
}
Dim _ribbonButton3 As New RibbonButton() With {
```

```
.SizeForm = SizeForm.Small,
.Label = "Menu Item-3"
}
_ribbonGallery.MenuItems.Add(_ribbonButton1)
_ribbonGallery.MenuItems.Add(_ribbonButton2)
_ribbonGallery.MenuItems.Add(_ribbonButton3)
```



Add custom context menu

The context menu is a type of menu that appears when a right-click operation is performed on the target. The ribbon gallery and its items allow to add a custom context menu with user-defined menu items set to the [ContextMenu](#) property. This will override the built-in context menu of the ribbon gallery and its items.

XML

```
<!-- A custom context menu with user-defined menu items -->
<syncfusion:RibbonWindow.Resources>
<syncfusion:RibbonContextMenu x:Key="galleryContextMenu"
ItemsSource="{Binding}">
<syncfusion:RibbonMenuItem Header="Apply Style" Command="{Binding
DataContext.ApplyStyleCommand}" IconBarEnabled="True"/>
<syncfusion:RibbonMenuItem Header="Remove from Style Gallery"
Command="{Binding DataContext.RemoveItemCommand}" IconBarEnabled="True"/>
<Separator />
<syncfusion:RibbonMenuItem Header="Minimize Ribbon" Command="{Binding
DataContext.MinimizeRibbonCommand}" IconBarEnabled="True" />
</syncfusion:RibbonContextMenu>
</syncfusion:RibbonWindow.Resources>
<!-- Replace the custom context menu with the built-in context menu of the
ribbon gallery -->
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
```

```

<syncfusion:RibbonTab Caption="HOME" IsChecked="False"/>
<syncfusion:RibbonTab IsChecked="True" Caption="DESIGN">
<syncfusion:RibbonBar Header="RibbonBar">
<syncfusion:RibbonGallery x:Name="gallery" ContextMenu="{DynamicResource
galleryContextMenu}"
Width="380" SizeForm="Large" Height="67" ItemHeight="60" ItemWidth="60"
MenuIconBarEnabled="True">
<syncfusion:RibbonGalleryItem Margin="2" CheckOnClick="False">
<StackPanel>
<TextBlock TextAlignment="Center" Margin="0,9,0,0" Text="AaBbCc"/>
<TextBlock TextAlignment="Center" Margin="0,11,0,0" Text="Normal"/>
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem Margin="2" CheckOnClick="False">
<StackPanel>
<TextBlock TextAlignment="Center" Margin="0,9,0,0" Text="AaBbCc"/>
<TextBlock TextAlignment="Center" Margin="0,11,0,0" Text="No Spa.."/>
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem Margin="2" CheckOnClick="False">
<StackPanel>
<TextBlock FontSize="18" Foreground="DarkBlue" TextAlignment="Center"
Margin="0,5,0,0" Text="AaBbCc"/>
<TextBlock TextAlignment="Center" Margin="0,6,0,0" Text="Heading 1"/>
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem Margin="2" CheckOnClick="False">
<StackPanel>
<TextBlock FontSize="16" Foreground="DarkBlue" TextAlignment="Center"
Margin="0,6,0,0" Text="AaBbCc"/>
<TextBlock TextAlignment="Center" Margin="0,8,0,0" Text="Heading 2"/>
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem Margin="2" CheckOnClick="False">
<StackPanel>
<TextBlock FontSize="14" Foreground="DarkBlue" TextAlignment="Center"
Margin="0,7,0,0" Text="AaBbCc"/>
<TextBlock TextAlignment="Center" Margin="0,9,0,0" Text="Heading 3"/>
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem Margin="2" CheckOnClick="False">
<StackPanel>
<TextBlock FontSize="15" Foreground="DarkBlue" TextAlignment="Center"
Margin="0,7,0,0" Text="AaBbCc"/>
<TextBlock TextAlignment="Center" Margin="0,9,0,0" Text="Heading 4"/>
</StackPanel>
</syncfusion:RibbonGalleryItem>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>

```

C#

```

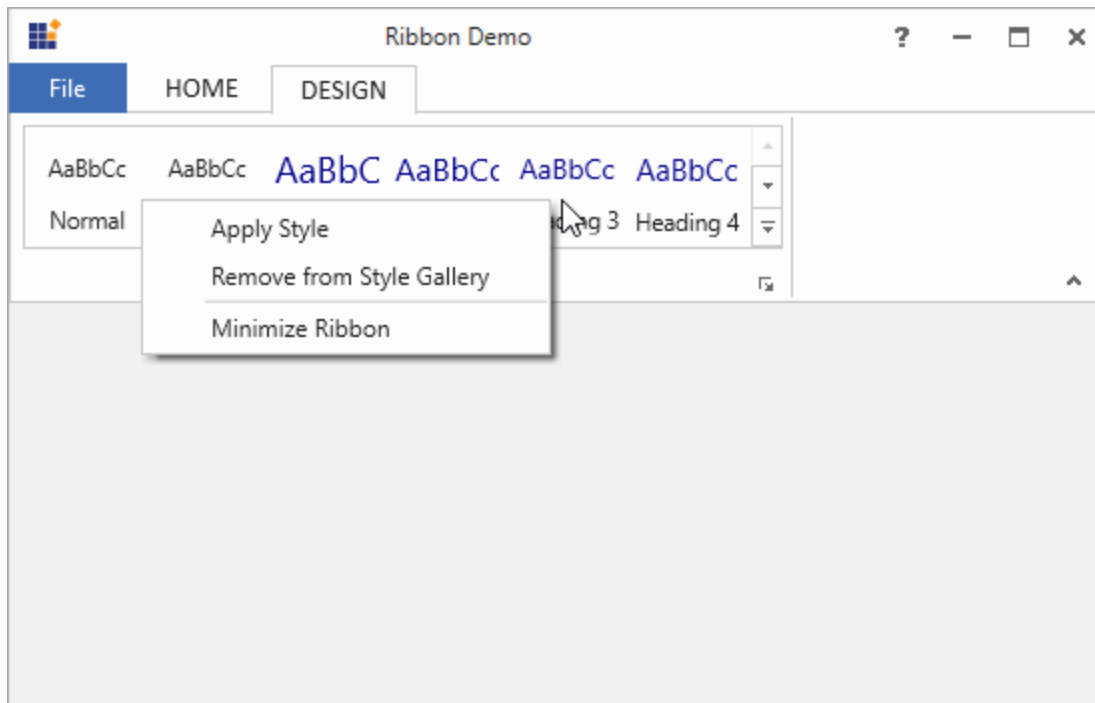
/// <summary>
/// Command for "ApplyStyle" option in the context menu.

```

```
/// </summary>
public DelegateCommand ApplyStyleCommand
{
    get
    {
        return new DelegateCommand(ApplyStyleCommandAction);
    }
}
/// <summary>
/// Action that occurs when "ApplyStyle" menu item is selected from the
/// context menu.
/// </summary>
private void ApplyStyleCommandAction(object param)
{
    SelectionForeground = new SolidColorBrush(Color.FromRgb(0, 0, 0));
    SelectionFontSize = 14;
    foreach (Paragraph paragraph in paragraphs)
    {
        paragraph.LineHeight = 20;
        paragraph.Foreground = new SolidColorBrush(Color.FromRgb(0, 0, 0));
        paragraph.FontSize = 14;
    }
}
/// <summary>
/// Command for "RemoveItem" option in the context menu.
/// </summary>
public DelegateCommand RemoveItemCommand
{
    get
    {
        return new DelegateCommand(RemoveItemCommandAction);
    }
}
/// <summary>
/// Action that occurs when "RemoveItem" is selected from the context menu.
/// </summary>
private void RemoveItemCommandAction(object param)
{
    if (gallery.Items.Count > 0)
        gallery.Items.RemoveAt(0);
}
/// <summary>
/// Command for "MinimizeRibbon" option in the context menu.
/// </summary>
public DelegateCommand MinimizeRibbonCommand
{
    get
    {
        return new DelegateCommand(MinimizeRibbonCommandAction);
    }
}
/// <summary>
/// Action that occurs when "MinimizeRibbon" item is selected from the
/// context menu.
/// </summary>
private void MinimizeRibbonCommandAction(object param)
{

```

```
this.Ribbon.RibbonState = RibbonState.Hide;
}
/// <summary>
/// A class that defines the interface for the command.
/// </summary>
public class DelegateCommand : ICommand
{
    private readonly Action<object> _execute;
    /// <summary>
    /// Raises when changes occur and specifies whether or not the command
    /// should be executed.
    /// </summary>
    public event EventHandler CanExecuteChanged;
    /// <summary>
    /// Constructor of the Delegate command
    /// </summary>
    /// <param name="execute"></param>
    public DelegateCommand(Action<object> execute)
    {
        _execute = execute;
    }
    /// <summary>
    /// Defines the method that determines whether the command can execute in
    /// its current state.
    /// </summary>
    public bool CanExecute(object parameter)
    {
        return true;
    }
    /// <summary>
    /// Defines the method to be called when the command is invoked.
    /// </summary>
    public void Execute(object parameter)
    {
        _execute(parameter);
    }
    /// <summary>
    /// Defines the method to be called when changes occur that affect the
    /// command.
    /// </summary>
    public void RaiseCanExecuteChanged()
    {
        if (CanExecuteChanged != null)
        {
            CanExecuteChanged(this, EventArgs.Empty);
        }
    }
}
```



Set various sizes for RibbonGallery

The [RibbonGallery](#) supports three types of size modes in its DropDown [VisualMode](#) and it can be set using the [SizeForm](#) property. The different [SizeForm](#) available are as follows:

- **ExtraSmall** - Displays only the image in 16 * 16 size.
- **Small** - Displays the label and the image in 16 * 16 size.
- **Large** - Displays the label and the image in 32 * 32 size.

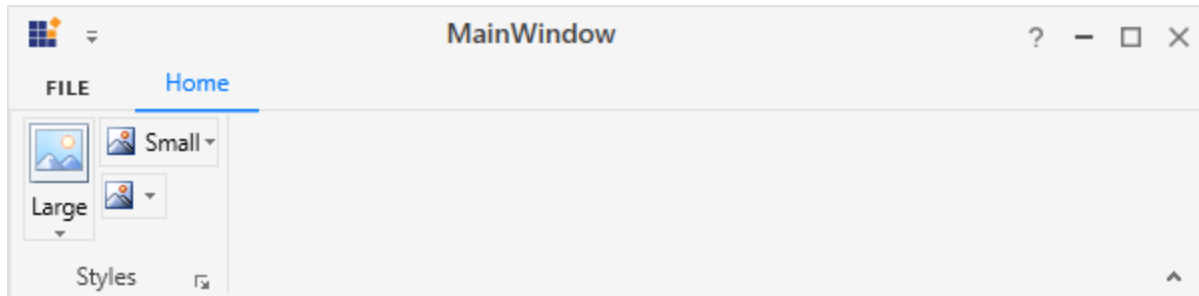
XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Styles">
<syncfusion:RibbonGallery VisualMode="DropDown" Label="Styles"
SizeForm="Large"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</Grid>
</syncfusion:RibbonWindow>
```

```
<syncfusion:RibbonGallery VisualMode="DropDown" Label="Change Indent"
SizeForm="Small" Height="25"/>
<syncfusion:RibbonGallery VisualMode="DropDown" Label="Change Font"
SizeForm="ExtraSmall" Height="25" Width="35" HorizontalAlignment="Left"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar stylesBar = new RibbonBar();
stylesBar.Header = "Styles";
// Creating items
RibbonGallery stylesGallery = new RibbonGallery();
stylesGallery.Label = "Styles";
stylesGallery.SizeForm = SizeForm.Large;
stylesGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
RibbonGallery indentGallery = new RibbonGallery();
indentGallery.Label = "Change Indent";
indentGallery.SizeForm = SizeForm.Small;
indentGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
indentGallery.Height = 25;
RibbonGallery fontGallery = new RibbonGallery();
fontGallery.Label = "Change Font";
fontGallery.SizeForm = SizeForm.ExtraSmall;
fontGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
indentGallery.Height = 25;
indentGallery.Width = 35;
indentGallery.HorizontalAlignment = HorizontalAlignment.Left;
// Adding items to bar
stylesBar.Items.Add(stylesGallery);
stylesBar.Items.Add(indentGallery);
stylesBar.Items.Add(fontGallery);
// Adding bars to the tabs
homeTab.Items.Add(stylesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
```

Note: When **simplified** layout is set, [RibbonGallery](#) displays the image in 20 * 20 size irrespective of the size form. Also, the text in the **Large** size form will appear to the right of the image.

Setting image to RibbonGallery

The [RibbonGallery](#) allows to display any type of image such as glyph, font or any custom content using the [IconTemplateSelector](#) and [IconTemplate](#) property, which are the preferred options. It allows to display a normal image or vector image using the [IconType](#) enumeration property. The default value of the [IconType](#) property is "Icon". The [IconType](#) enumeration has the following values:

- **Icon** - Gets the detail of the icon path from the [SmallIcon](#), [MediumIcon](#) or [LargeIcon](#) properties and sets it to the DropDownButton.
- **VectorImage** - Gets the details of the icon's path data from the [VectorImage](#) property and sets it to the DropDownButton.

Note: The [RibbonGallery](#) loads icon in the following priority order,

- [IconTemplateSelector](#)
- [IconTemplate](#)
- [VectorImage](#)
- [LargeIcon](#)
- [MediumIcon](#)
- [SmallIcon](#)

Setting icon template selector

The [IconTemplateSelector](#) property provides support to specify a different data template based on the value of the [SizeForm](#) or [LayoutMode](#) properties. For simplified layout, the template content will be resized to 20 * 20 size which is the standard.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<syncfusion:RibbonWindow.Resources>
```

```

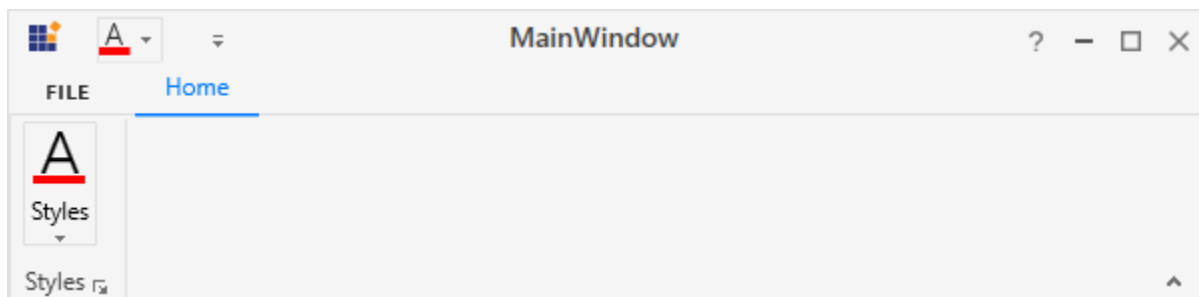
<DataTemplate x:Key="ribbonGallerySmallIconTemplate">
<Grid Width="14" Height="16">
<Path
Height="4"
VerticalAlignment="Bottom"
Data="M0,0 L16,0 16,4 0,4 z"
Fill="#FFFE0000"
Stretch="Fill" />
<Path
Margin="3.344,0,3.352,5"
Data="M4.6480023,0.95898432 C4.6079937,1.0870056 4.5689923,1.2149963
4.533012,1.34198 4.4980089,1.4909973 4.4510118,1.6419983 4.394005,1.7949829
L2.5330156,6.8809814 6.787006,6.8809814 4.9330055,1.7799988
C4.8510047,1.5699768 4.7659832,1.2969971 4.6790081,0.95898432 z M4.0779959,0
L5.209006,0 9.304,11 8.3170024,11 7.2039977,8.0019836 2.1150171,8.0029907
1.0100081,11 0,11 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="ribbonGalleryLargeIconTemplate">
<Grid Height="26" Width="26" >
<Path
Height="4"
VerticalAlignment="Bottom"
Data="M0,0 L16,0 16,4 0,4 z"
Fill="#FFFE0000"
Stretch="Fill" />
<Path
Margin="3.344,0,3.352,5"
Data="M4.6480023,0.95898432 C4.6079937,1.0870056 4.5689923,1.2149963
4.533012,1.34198 4.4980089,1.4909973 4.4510118,1.6419983 4.394005,1.7949829
L2.5330156,6.8809814 6.787006,6.8809814 4.9330055,1.7799988
C4.8510047,1.5699768 4.7659832,1.2969971 4.6790081,0.95898432 z M4.0779959,0
L5.209006,0 9.304,11 8.3170024,11 7.2039977,8.0019836 2.1150171,8.0029907
1.0100081,11 0,11 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</Grid>
</DataTemplate>
<local:RibbonGalleryIconTemplateSelector
x:Key="ribbonGalleryIconTemplateSelector"
SmallTemplate="{StaticResource ribbonGallerySmallIconTemplate}"
LargeTemplate="{StaticResource ribbonGalleryLargeIconTemplate}"/>
</syncfusion:RibbonWindow.Resources>
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonGallery VisualMode="DropDown" Label="Font Color"
SizeForm="ExtraSmall" IconTemplateSelector="{StaticResource
ribbonGalleryIconTemplateSelector}"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
</syncfusion:RibbonTab Caption="Home">

```

```
<syncfusion:RibbonBar Header="Styles">
<syncfusion:RibbonGallery VisualMode="DropDown" Label="Font Color"
SizeForm="Large" IconTemplateSelector="{StaticResource
ribbonGalleryIconTemplateSelector}"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
public class RibbonGalleryIconTemplateSelector : DataTemplateSelector
{
    public DataTemplate SmallTemplate { get; set; }
    public DataTemplate LargeTemplate { get; set; }
    public override DataTemplate SelectTemplate(object item, DependencyObject
container)
    {
        var item1 = (container as ContentPresenter);
        RibbonGallery ribbonGallery = (item1.TemplatedParent as RibbonGallery);
        if (ribbonGallery != null)
        {
            if (ribbonGallery.SizeForm == SizeForm.Small || ribbonGallery.SizeForm ==
SizeForm.ExtraSmall)
            {
                return SmallTemplate;
            }
            else if (ribbonGallery.SizeForm == SizeForm.Large)
            {
                return LargeTemplate;
            }
        }
        return LargeTemplate;
    }
}
```



Note: [View sample in GitHub](#)

Setting icon template

The [IconTemplate](#) property provides support to set any type of image such as glyph, font or any custom content to the [RibbonGallery](#). The [RibbonGallery](#) will automatically resize the template content according to its [SizeForm](#). For simplified layout, the template content will be resized to 20 * 20 size which is the standard.

XML

```

<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Styles">
<syncfusion:RibbonGallery VisualMode="DropDown" Label="Font Color"
SizeForm="Large">
<syncfusion:RibbonGallery.IconTemplate>
<DataTemplate>
<Grid Height="26" Width="26" >
<Path
Height="4" VerticalAlignment="Bottom" Data="M0,0 L16,0 16,4 0,4 z"
Fill="#FFFE0000" Stretch="Fill" />
<Path
Margin="3.344,0,3.352,5"
Data="M4.6480023,0.95898432 C4.6079937,1.0870056 4.5689923,1.2149963
4.533012,1.34198 4.4980089,1.4909973 4.4510118,1.6419983 4.394005,1.7949829
L2.5330156,6.8809814 6.787006,6.8809814 4.9330055,1.7799988
C4.8510047,1.5699768 4.7659832,1.2969971 4.6790081,0.95898432 z M4.0779959,0
L5.209006,0 9.304,11 8.3170024,11 7.2039977,8.0019836 2.1150171,8.0029907
1.0100081,11 0,11 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground)}" Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:RibbonGallery.IconTemplate>
</syncfusion:RibbonGallery>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

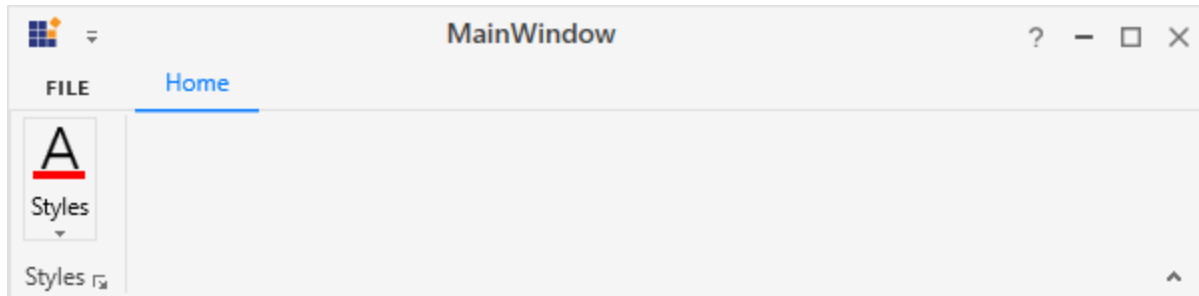
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;

```

```

// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Styles";
// Creating items
RibbonGallery ribbonGallery = new RibbonGallery();
ribbonGallery.Label = "Font Color";
ribbonGallery.SizeForm = SizeForm.Large;
ribbonGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
DataTemplate iconDataTemplate = new DataTemplate();
FrameworkElementFactory gridElement = new
FrameworkElementFactory(typeof(Grid));
FrameworkElementFactory pathElement1 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement2 = new
FrameworkElementFactory(typeof(Path));
gridElement.SetValue(Grid.HeightProperty, (double)26);
gridElement.SetValue(Grid.WidthProperty, (double)26);
pathElement1.SetValue(Path.DataProperty, Geometry.Parse("M0,0 L16,0 16,4 0,4
z"));
pathElement1.SetValue(Path.HeightProperty, (double)4);
pathElement1.SetValue(Path.FillProperty, new SolidColorBrush(Colors.Red));
pathElement1.SetValue(Path.VerticalAlignmentProperty,
VerticalAlignment.Bottom );
pathElement1.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement2.SetValue(Path.DataProperty,
Geometry.Parse("M4.6480023,0.95898432 C4.6079937,1.0870056
4.5689923,1.2149963 4.533012,1.34198 4.4980089,1.4909973 4.4510118,1.6419983
4.394005,1.7949829 L2.5330156,6.8809814 6.787006,6.8809814
4.9330055,1.7799988 C4.8510047,1.5699768 4.7659832,1.2969971
4.6790081,0.95898432 z M4.0779959,0 L5.209006,0 9.304,11 8.3170024,11
7.2039977,8.0019836 2.1150171,8.0029907 1.0100081,11 0,11 z"));
pathElement2.SetValue(Path.MarginProperty, new Thickness(3.344, 0, 3.352,
5));
pathElement2.SetValue(Path.FillProperty, new SolidColorBrush(Colors.Black));
pathElement2.SetValue(Path.StretchProperty, Stretch.Fill);
gridElement.AppendChild(pathElement1);
gridElement.AppendChild(pathElement2);
iconDataTemplate.VisualTree = gridElement;
ribbonGallery.IconTemplate = iconDataTemplate;
clipboardBar.Items.Add(ribbonGallery);
// Adding bars to the tabs
homeTab.Items.Add(clipboardBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Note: [View sample in GitHub](#)

Setting image path

The [RibbonGallery](#) allows to set the image according to the different [SizeForm](#) values. To set the image to [RibbonGallery](#), the following properties are used:

- [SmallIcon](#) - 16 * 16 size image to be displayed in normal layout for “ExtraSmall” and “Small” size form.
- [MediumIcon](#) - 20 * 20 size image to be displayed in simplified layout.
- [LargeIcon](#) - 32 * 32 size image to be displayed in normal layout for “Large” size form.

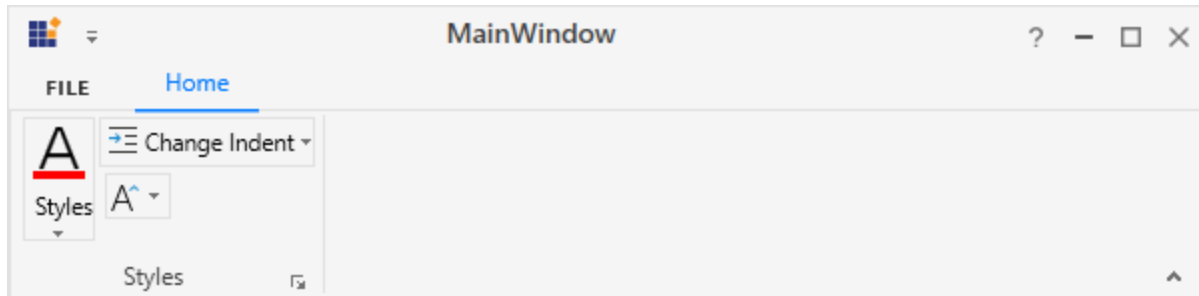
XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Styles">
<syncfusion:RibbonGallery MediumIcon="/Resources/Styles20.png"
LargeIcon="/Resources/Styles32.png"
VisualMode="DropDown" Label="Styles" SizeForm="Large"/>
<syncfusion:RibbonGallery MediumIcon="/Resources/Indent20.png"
SmallIcon="/Resources/Indent16.png"
VisualMode="DropDown" Label="Change Indent" SizeForm="Small" Height="25"/>
<syncfusion:RibbonGallery MediumIcon="/Resources/Font20.png"
SmallIcon="/Resources/Font16.png"
VisualMode="DropDown" Label="Change Font" SizeForm="ExtraSmall" Height="25"
Width="35" HorizontalAlignment="Left"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
```

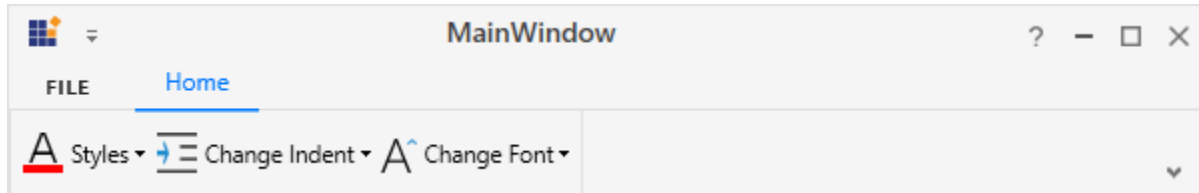
```
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar stylesBar = new RibbonBar();
stylesBar.Header = "Styles";
// Creating items
RibbonGallery stylesGallery = new RibbonGallery();
stylesGallery.Label = "Styles";
stylesGallery.SizeForm = SizeForm.Large;
stylesGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
stylesGallery.MediumIcon = new BitmapImage(new
Uri(@"Resources/Styles20.png", UriKind.RelativeOrAbsolute));
stylesGallery.LargeIcon = new BitmapImage(new
Uri(@"Resources/Styles32.png", UriKind.RelativeOrAbsolute));
RibbonGallery indentGallery = new RibbonGallery();
indentGallery.Label = "Change Indent";
indentGallery.SizeForm = SizeForm.Small;
indentGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
indentGallery.Height = 25;
indentGallery.MediumIcon = new BitmapImage(new
Uri(@"Resources/Indent20.png", UriKind.RelativeOrAbsolute));
indentGallery.SmallIcon = new BitmapImage(new
Uri(@"Resources/Indent16.png", UriKind.RelativeOrAbsolute));
RibbonGallery fontGallery = new RibbonGallery();
fontGallery.Label = "Change Font";
fontGallery.SizeForm = SizeForm.ExtraSmall;
fontGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
indentGallery.Height = 25;
indentGallery.Width = 35;
indentGallery.HorizontalAlignment = HorizontalAlignment.Left;
fontGallery.MediumIcon = new BitmapImage(new Uri(@"Resources/Font20.png",
UriKind.RelativeOrAbsolute));
fontGallery.SmallIcon = new BitmapImage(new Uri(@"Resources/Font16.png",
UriKind.RelativeOrAbsolute));
// Adding items to bar
stylesBar.Items.Add(stylesGallery);
stylesBar.Items.Add(indentGallery);
stylesBar.Items.Add(fontGallery);
// Adding bars to the tabs
homeTab.Items.Add(stylesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
```



Normal layout



Simplified layout

Setting vector image

The [VectorImage](#) property is of the type `ObservableCollection<Path>` which allows the image to be set as path type. The [RibbonGallery](#) will automatically resize the image according to its [SizeForm](#). For simplified layout, the image will be resized to 20 * 20 size which is the standard.

Note: The [IconTemplateSelector](#) and [IconTemplate](#) properties are the preferred options to set any type of image such as glyph, font or any custom content when compared to the [VectorImage](#) property.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="450" Width="600">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Styles">
<syncfusion:RibbonGallery VisualMode="DropDown" IconType="VectorImage"
Label="Styles" SizeForm="Large">
<syncfusion:RibbonGallery.VectorImage>
<Path
Height="4" VerticalAlignment="Bottom" Data="M0,0 L16,0 16,4 0,4 z"
Fill="#FFFE0000" Stretch="Fill" />
<Path
```



```

Margin="3.344,0,3.352,5"
Data="M4.6480023,0.95898432 C4.6079937,1.0870056 4.5689923,1.2149963
4.533012,1.34198 4.4980089,1.4909973 4.4510118,1.6419983 4.394005,1.7949829
L2.5330156,6.8809814 6.787006,6.8809814 4.9330055,1.7799988
C4.8510047,1.5699768 4.7659832,1.2969971 4.6790081,0.95898432 z M4.0779959,0
L5.209006,0 9.304,11 8.3170024,11 7.2039977,8.0019836 2.1150171,8.0029907
1.0100081,11 0,11 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground)}" Stretch="Fill" />
</syncfusion:RibbonGallery.VectorImage>
</syncfusion:RibbonGallery>
<syncfusion:RibbonGallery VisualMode="DropDown" IconType="VectorImage"
Label="Change Indent" SizeForm="Small" Height="25">
<syncfusion:RibbonGallery.VectorImage>
<Path Data="M0,12 L14,12 14,13 0,13 z M8,8 L14,8 14,9 8,9 z M8,4 L14,4 14,5
8,5 z M0,0 L14,0 14,1 0,1 z" Fill="#FF3A3A38" Stretch="Fill" />
<Path Data="M3.7870274,0 L6.1870296,2.5269938 3.7140274,4.9499879
3.0150268,4.2359896 4.2667925,3.008994 0,3.008994 0,2.0089941
4.3160441,2.0089941 3.0630267,0.68899834 z"
Fill="#FF1E8BCD" HorizontalAlignment="Left" Margin="0,3.991,0,4.059"
Stretch="Fill" Width="6.187"/>
</syncfusion:RibbonGallery.VectorImage>
</syncfusion:RibbonGallery>
<syncfusion:RibbonGallery VisualMode="DropDown" IconType="VectorImage"
Label="Change Font" SizeForm="ExtraSmall" Height="25" Width="35"
HorizontalAlignment="Left">
<syncfusion:RibbonGallery.VectorImage>
<Path
Width="5.005" Height="3.002" Margin="0,3,0,0" HorizontalAlignment="Right"
VerticalAlignment="Top"
Data="M2.5189795,0 L5.0050001,2.2590036 4.3330035,3.0000011
2.5149817,1.3470006 0.66799865,3.002 0,2.2570047 z"
Fill="#FF3094D0" Stretch="Fill" />
<Path
Data="M4.4360077,0.8710022 C4.3949921,0.98799133 4.3580048,1.1040039
4.321994,1.2200012 4.2859833,1.3549957 4.2420075,1.4919891
4.1850006,1.6309967 L2.3469865,6.2549896 6.5470016,6.2549896
4.7169835,1.6179962 C4.6359894,1.427002 4.5530121,1.1779938
4.4660065,0.8710022 z M4.0639984,0 L4.7779882,0 8.8869998,10 8.0589959,10
6.8140003,6.8899994 2.0889907,6.8899994 0.84298758,10 0,10 z"
Margin="2,3,3,0" Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground)}"
Stretch="Fill" />
</syncfusion:RibbonGallery.VectorImage>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;

```

```

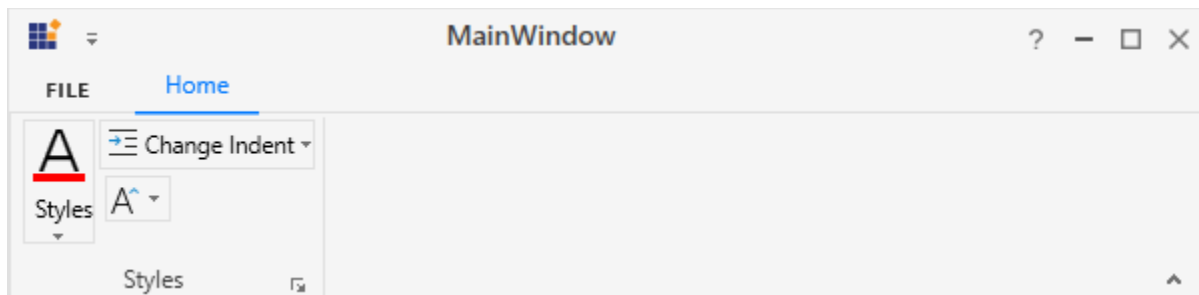
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar stylesBar = new RibbonBar();
stylesBar.Header = "Styles";
// Creating items
RibbonGallery styleGallery = new RibbonGallery();
styleGallery.Label = "Styles";
styleGallery.SizeForm = SizeForm.Large;
styleGallery.IconType = IconType.VectorImage;
styleGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
Path stylePath1 = new Path();
stylePath1.Data = Geometry.Parse("M0,0 L16,0 16,4 0,4 z");
stylePath1.Height = 4;
stylePath1.VerticalAlignment = VerticalAlignment.Bottom;
stylePath1.Fill = new SolidColorBrush(Color.FromRgb(254, 0, 0));
Path stylePath2 = new Path();
stylePath2.Data = Geometry.Parse("M4.6480023,0.95898432 C4.6079937,1.0870056 4.5689923,1.2149963 4.533012,1.34198 4.4980089,1.4909973 4.4510118,1.6419983 4.394005,1.7949829 L2.5330156,6.8809814 6.787006,6.8809814 4.9330055,1.7799988 C4.8510047,1.5699768 4.7659832,1.2969971 4.6790081,0.95898432 z M4.0779959,0 L5.209006,0 9.304,11 8.3170024,11 7.2039977,8.0019836 2.1150171,8.0029907 1.0100081,11 0,11 z");
stylePath2.Margin = new Thickness(3.344, 0, 3.352, 5);
stylePath2.Fill = new SolidColorBrush(Colors.Black);
styleGallery.VectorImage.Add(stylePath1);
styleGallery.VectorImage.Add(stylePath2);
RibbonGallery indentGallery = new RibbonGallery();
indentGallery.Label = "Change Indent";
indentGallery.SizeForm = SizeForm.Small;
indentGallery.IconType = IconType.VectorImage;
indentGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
indentGallery.Height = 25;
Path indentPath1 = new Path();
indentPath1.Data = Geometry.Parse("M0,12 L14,12 14,13 0,13 z M8,8 L14,8 14,9 8,9 z M8,4 L14,4 14,5 8,5 z M0,0 L14,0 14,1 0,1 z");
indentPath1.Fill = new SolidColorBrush(Color.FromRgb(58, 58, 56));
Path indentPath2 = new Path();
indentPath2.Data = Geometry.Parse("M3.7870274,0 L6.1870296,2.5269938 3.7140274,4.9499879 3.0150268,4.2359896 4.2667925,3.008994 0,3.008994 0,2.0089941 4.3160441,2.0089941 3.0630267,0.68899834 z");
indentPath2.Margin = new Thickness(0, 3.991, 0, 4.059);
indentPath2.Fill = new SolidColorBrush(Color.FromRgb(30, 139, 205));
indentPath2.Width = 6.187;
indentPath2.HorizontalAlignment = HorizontalAlignment.Left;
indentGallery.VectorImage.Add(indentPath1);
indentGallery.VectorImage.Add(indentPath2);
RibbonGallery fontGallery = new RibbonGallery();
fontGallery.Label = "Change Font";
fontGallery.SizeForm = SizeForm.ExtraSmall;
fontGallery.IconType = IconType.VectorImage;
fontGallery.VisualMode = RibbonGalleryVisualMode.DropDown;
fontGallery.HorizontalAlignment = HorizontalAlignment.Left;
fontGallery.Height = 25;
fontGallery.Width = 35;

```

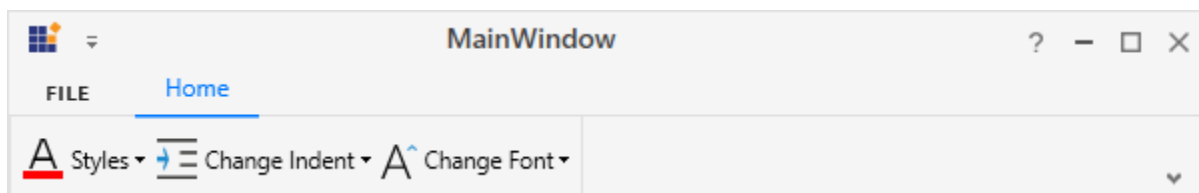
```

Path fontPath1 = new Path();
fontPath1.Data = Geometry.Parse("M2.5189795,0 L5.0050001,2.2590036
4.3330035,3.0000011 2.5149817,1.3470006 0.66799865,3.002 0,2.2570047 z");
fontPath1.Margin = new Thickness(0, 3, 0, 0);
fontPath1.Fill = new SolidColorBrush(Color.FromRgb(48, 148, 208));
fontPath1.HorizontalAlignment = HorizontalAlignment.Right;
fontPath1.VerticalAlignment = VerticalAlignment.Top;
fontPath1.Width = 5;
fontPath1.Height = 3;
Path fontPath2 = new Path();
fontPath2.Data = Geometry.Parse("M4.4360077,0.8710022 C4.3949921,0.98799133
4.3580048,1.1040039 4.321994,1.2200012 4.2859833,1.3549957
4.2420075,1.4919891 4.1850006,1.6309967 L2.3469865,6.2549896
6.5470016,6.2549896 4.7169835,1.6179962 C4.6359894,1.427002
4.5530121,1.1779938 4.4660065,0.8710022 z M4.0639984,0 L4.7779882,0
8.8869998,10 8.0589959,10 6.8140003,6.8899994 2.0889907,6.8899994
0.84298758,10 0,10 z");
fontPath2.Margin = new Thickness(2, 3, 3, 0);
fontPath2.Fill = new SolidColorBrush(Colors.Black);
fontGallery.VectorImage.Add(fontPath1);
fontGallery.VectorImage.Add(fontPath2);
// Adding items to bar
stylesBar.Items.Add(styleGallery);
stylesBar.Items.Add(indentGallery);
stylesBar.Items.Add(fontGallery);
// Adding bars to the tabs
homeTab.Items.Add(stylesBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Normal layout



Simplified layout

Add Gallery to the simplified layout

Add Gallery in InRibbon visual mode

When the simplified layout is enabled, the RibbonGallery can be added and displayed in a single line in [InRibbon](#) VisualMode as shown below. To know more about the simplified layout, refer [here](#).

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Options">
<syncfusion:RibbonGallery VisualMode="InRibbon" Label="Styles"
MediumIcon="/Resources/Finalmark20.png" >
<syncfusion:RibbonGalleryItem CheckOnClick="True">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,11,0,0"
Text="Normal"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem CheckOnClick="True">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,11,0,0"
Text="No Spa.."
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem CheckOnClick="True">
<StackPanel>
<TextBlock
Margin="0,5,0,0"
FontSize="18"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
```

```

<TextBlock
Margin="0,6,0,0"
Text="Heading 1"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Add Gallery in DropDown visual mode

When the simplified layout is enabled, the RibbonGallery can be added and displayed in a single line in [DropDown](#) VisualMode as shown below. To know more about the simplified layout, refer [here](#).

XML

```

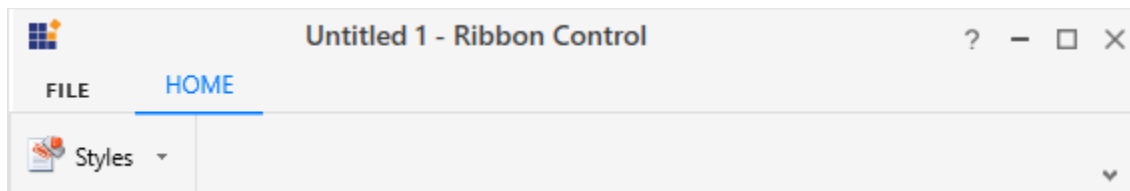
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Options">
<syncfusion:RibbonGallery VisualMode="DropDown" Label="Styles"
MediumIcon="/Resources/Finalmark20.png" Width="85">
<syncfusion:RibbonGalleryItem CheckOnClick="True">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,11,0,0"
Text="Normal"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem CheckOnClick="True">

```

```

<StackPanel>
  <TextBlock
    Margin="0,9,0,0"
    Text="AaBbCc"
    TextAlignment="Center" />
  <TextBlock
    Margin="0,11,0,0"
    Text="No Spa.."
    TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem CheckOnClick="True">
  <StackPanel>
    <TextBlock
      Margin="0,5,0,0"
      FontSize="18"
      Foreground="DarkBlue"
      Text="AaBbCc"
      TextAlignment="Center" />
    <TextBlock
      Margin="0,6,0,0"
      Text="Heading 1"
      TextAlignment="Center" />
  </StackPanel>
</syncfusion:RibbonGalleryItem>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Arranging between layouts

When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the RibbonGallery can be ignored as it will be resized automatically to the standard width and height. If the RibbonGallery is to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```

<syncfusion:RibbonGallery
  syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified" >
  <syncfusion:RibbonGalleryItem Content="Item 1"/>
  <syncfusion:RibbonGalleryItem Content="Item 2"/>
  <syncfusion:RibbonGallery.Style>
    <Style TargetType="syncfusion:RibbonGallery" BasedOn="{StaticResource
      SyncfusionRibbonGalleryStyle}">
    <Style.Triggers>

```

```
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="48"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>
</syncfusion:RibbonGallery.Style>
</syncfusion:RibbonGallery >
```

RibbonTextBox in WPF Ribbon

RibbonTextBox control provide similar set of functionalities like normal TextBox control in Ribbon Instance.

Add TextBox to the RibbonBar

XML

```
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Name="_ribbonTab1" Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="_ribbonBar2" Header="RibbonBar1">
<syncfusion:RibbonButton Label="Cut"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="_ribbonBar2" Width="150" Header="RibbonBar2">
<syncfusion:RibbonTextBox Width="140"
Text="RibbonTextBox"></syncfusion:RibbonTextBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="EDIT" IsChecked="False"/>
</syncfusion:Ribbon>
```

Create instance of RibbonTextBox and add it to RibbonBar through code behind.

C#

```
RibbonTextBox _ribbonTextBox = new RibbonTextBox() {Text = "RibbonTextBox"};
ribbonBar2.Items.Add( _ribbonTextBox);
```

VB.NET

```
Dim _ribbonTextBox As New RibbonTextBox() With {.Text = "RibbonTextBox"}
_ribbonBar2.Items.Add( _ribbonTextBox)
```

![Adding text box to the ribbon](RibbonTextBoximages/RibbonTextBoximg1.jpg)

Add TextBox to the simplified layout

When the simplified layout is enabled, the RibbonTextBox can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Font">
<syncfusion:RibbonTextBox Width="140"
Text="RibbonTextBox"></syncfusion:RibbonTextBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar fontBar = new RibbonBar();
fontBar.Header = "Font";
// Creating items
RibbonTextBox ribbonTextBox = new RibbonTextBox() { Text = "RibbonTextBox"
};
// Adding items to bar
fontBar.Items.Add(ribbonTextBox);
// Adding bars to the tabs
homeTab.Items.Add(fontBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```

![RibbonTextBox during simplified layout](RibbonTextBoximages/RibbonTextBoxSimplified.png)

When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the RibbonTextBox can be ignored as it will be resized automatically to the standard width and height. If the RibbonTextBox is to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```

<syncfusion:RibbonTextBox Text="Enter text"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified" >

```



```
<syncfusion:RibbonTextBox.Style>
<Style TargetType="syncfusion:RibbonTextBox" BasedOn="{StaticResource
SyncfusionRibbonTextBoxStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="25"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>
</syncfusion:RibbonTextBox.Style>
</syncfusion:RibbonTextBox >
```

RibbonCheckBox in WPF Ribbon

[RibbonCheckBox](#) is used to select or unselect options. It provides similar set of functionalities like normal CheckBox control in [Ribbon](#).

The following code example illustrates how to use [RibbonCheckBox](#) control in [Ribbon](#) instance.

XML

```
<syncfusion:Ribbon Name="ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Name="ribbonTab1" Caption="HOME" >
<syncfusion:RibbonBar Name="ribbonBar1" Header="RibbonBar1">
<syncfusion:RibbonButton Label="Cut"/>
<syncfusion:RibbonButton Label="Copy"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="ribbonBar2" Width="150" Header="RibbonBar2">
<syncfusion:RibbonCheckBox Width="140" Content="SelectAll"
IsChecked="True"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="EDIT" IsChecked="False"/>
</syncfusion:Ribbon>
```

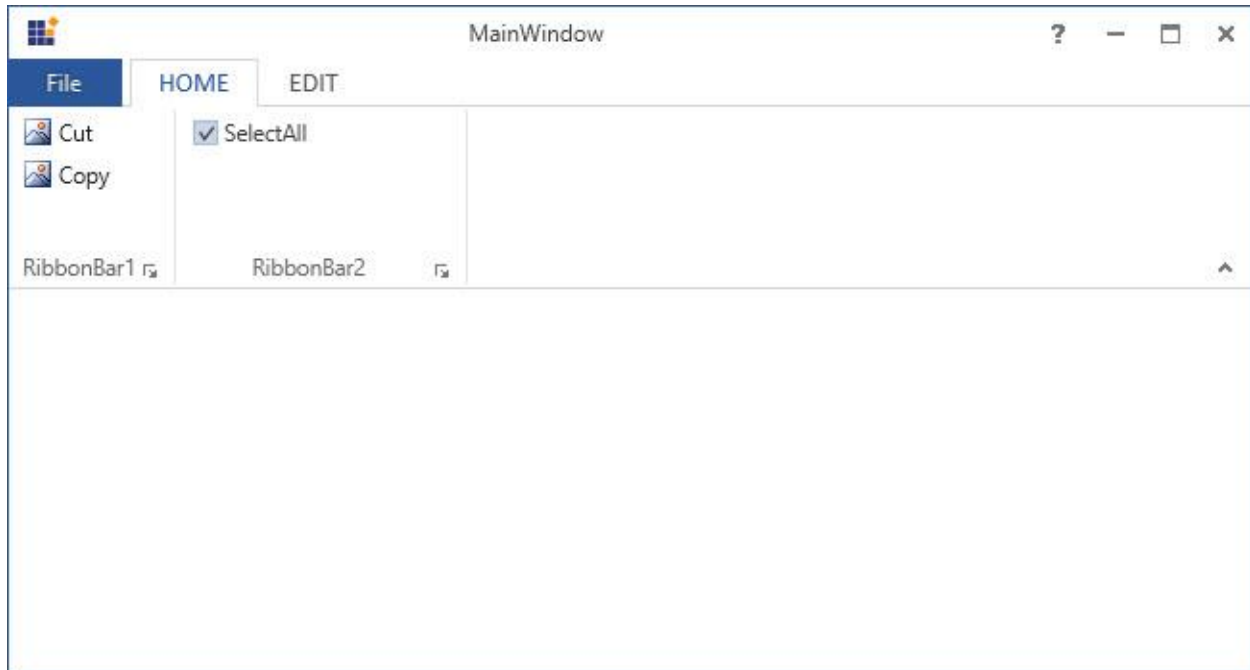
Create instance of [RibbonCheckBox](#) and add it to [RibbonBar](#) through code behind.

C#

```
RibbonCheckBox ribbonCheckBox = new RibbonCheckBox() { Content = "SelectAll",
IsChecked=true };
ribbonBar2.Items.Add(ribbonCheckBox);
```

VB.NET

```
Dim ribbonCheckBox As New RibbonCheckBox() With {
.Content = "SelectAll",
.IsChecked=True
}
ribbonBar2.Items.Add(ribbonCheckBox)
```



Add CheckBox to the simplified layout

When the simplified layout is enabled, the `RibbonCheckBox` can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Options">
<syncfusion:RibbonCheckBox Width="140" Content="SelectAll"
IsChecked="True"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

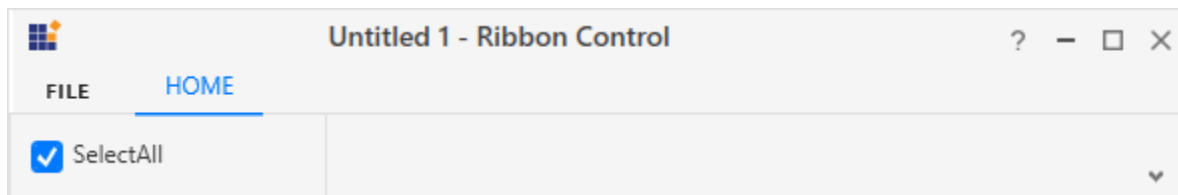
C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
```

```

ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar optionsBar = new RibbonBar();
optionsBar.Header = "Options";
// Creating items
RibbonCheckBox ribbonCheckBox = new RibbonCheckBox() { Content =
"SelectAll", IsChecked = true };
// Adding items to bar
optionsBar.Items.Add(ribbonCheckBox);
// Adding bars to the tabs
homeTab.Items.Add(optionsBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the **RibbonCheckBox** can be ignored as it will be resized automatically to the standard width and height. If the **RibbonCheckBox** is to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```

<syncfusion:RibbonCheckBox Content="SelectAll"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified" >
<syncfusion:RibbonCheckBox.Style>
<Style TargetType="syncfusion:RibbonCheckBox" BasedOn="{StaticResource
SyncfusionRibbonCheckBoxStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="25"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>
</syncfusion:RibbonCheckBox.Style>
</syncfusion:RibbonCheckBox >

```

See Also

[How to bind the RibbonCheckBox in WPF Ribbon control with MVVM pattern?](#).

RibbonRadioButton in WPF Ribbon

RibbonRadioButton control is used to select a option like normal RadioButton .

Add RadioButton to the RibbonBar

XML

```
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Name="_ribbonTab1" Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Name="_ribbonBar1" Header="RibbonBar1">
<syncfusion:RibbonButton Label="Cut"/>
<syncfusion:RibbonButton Label="Copy"/>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="_ribbonBar2" Width="150" Header="RibbonBar2">
<syncfusion:RibbonRadioButton GroupName="Group1" Width="140"
Content="ReadOnly" IsChecked="True"></syncfusion:RibbonRadioButton>
<syncfusion:RibbonRadioButton GroupName="Group1" Width="140"
Content="WriteOnly" ></syncfusion:RibbonRadioButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="EDIT" IsChecked="False"/>
</syncfusion:Ribbon>
```

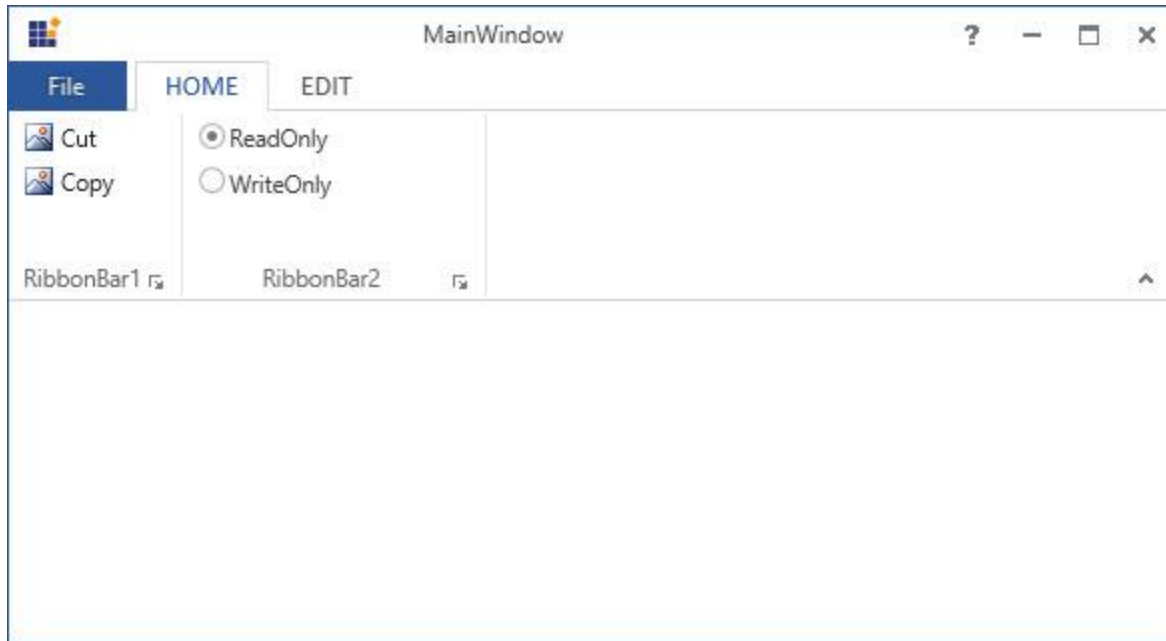
Create instance of RibbonRadioButton and add it to RibbonBar through code behind.

C#

```
RibbonRadioButton _ribbonRadioButton1 = new RibbonRadioButton() {Content =
"ReadOnly", IsChecked = true , GroupName = "Group1"};
RibbonRadioButton _ribbonRadioButton2 = new RibbonRadioButton() {Content =
"WriteOnly", GroupName = "Group1"};
_ribbonBar2.Items.Add(_ribbonRadioButton1);
_ribbonBar2.Items.Add(_ribbonRadioButton2);
```

VB.NET

```
Dim _ribbonRadioButton1 As New RibbonRadioButton() With {
.Content = "ReadOnly",
.IsChecked = True,
.GroupName = "Group1"
}
Dim _ribbonRadioButton2 As New RibbonRadioButton() With {
.Content = "WriteOnly",
.GroupName = "Group1"
}
_ribbonBar2.Items.Add(_ribbonRadioButton1)
_ribbonBar2.Items.Add(_ribbonRadioButton2)
```



Add RadioButton to the simplified layout

When the simplified layout is enabled, the RibbonRadioButton can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Options">
<syncfusion:RibbonRadioButton GroupName="Mode" Content="Read only"
IsChecked="True"></syncfusion:RibbonRadioButton>
<syncfusion:RibbonRadioButton GroupName="Mode" Content="Write only"
></syncfusion:RibbonRadioButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
```

```

ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar optionsBar = new RibbonBar();
optionsBar.Header = "Options";
// Creating items
RibbonRadioButton ribbonRadioButton = new RibbonRadioButton() { Content =
"Read only", IsChecked = true, GroupName="Mode" };
RibbonRadioButton ribbonRadioButton2 = new RibbonRadioButton() { Content =
"Write only", GroupName="Mode" };
// Adding items to bar
optionsBar.Items.Add(ribbonRadioButton);
optionsBar.Items.Add(ribbonRadioButton2);
// Adding bars to the tabs
homeTab.Items.Add(optionsBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the **RibbonRadioButton** can be ignored as it will be resized automatically to the standard width and height. If the **RibbonRadioButton** is to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```

<syncfusion:RibbonRadioButton Content="SelectAll"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified" >
<syncfusion:RibbonRadioButton.Style >
<Style TargetType="syncfusion:RibbonRadioButton" BasedOn="{StaticResource
SyncfusionRibbonRadioButtonStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="25"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>
</syncfusion:RibbonRadioButton.Style>
</syncfusion:RibbonRadioButton >

```

RibbonListBox in WPF Ribbon

RibbonListBox control is used to display a list of items in a Ribbon. It accepts any type of content as RibbonListBox items but **ListBoxItem** by default.

Add ListBox to the RibbonBar

XML

```
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
  <syncfusion:RibbonTab Name="_ribbonTab1" Caption="HOME" IsChecked="True">
    <syncfusion:RibbonBar Name="_ribbonBar1" Header="RibbonBar1">
      <syncfusion:RibbonButton Label="Cut"/>
      <syncfusion:RibbonButton Label="Copy"/>
    </syncfusion:RibbonBar>
    <syncfusion:RibbonBar Name="_ribbonBar2" Width="150" Header="RibbonBar2">
      <syncfusion:RibbonListBox Width="140" >
        <ListBoxItem Content="Office2003Theme"/>
        <ListBoxItem Content="Office2007Theme"/>
        <ListBoxItem Content="Office2010Theme"/>
      </syncfusion:RibbonListBox>
    </syncfusion:RibbonBar>
  </syncfusion:RibbonTab>
  <syncfusion:RibbonTab Caption="EDIT" IsChecked="False"/>
</syncfusion:Ribbon>
```

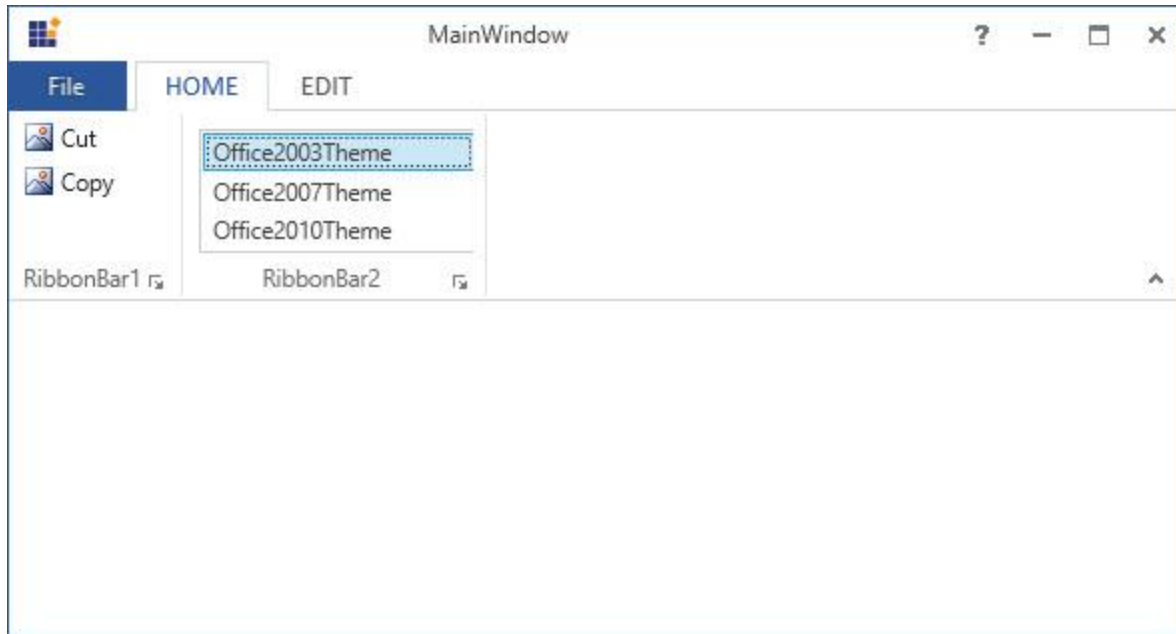
Create instance of RibbonListBox and add it to RibbonBar through code behind.

C#

```
RibbonListBox _ribbonListBox = new RibbonListBox();
ListBoxItem item1 = new ListBoxItem() {Content = "Office2003Theme"};
ListBoxItem item2 = new ListBoxItem() {Content = "Office2007Theme"};
ListBoxItem item3 = new ListBoxItem() {Content = "Office2010Theme"};
_ribbonListBox.Items.Add(item1);
_ribbonListBox.Items.Add(item2);
_ribbonListBox.Items.Add(item3);
ribbonBar2.Items.Add( ribbonListBox);
```

VB.NET

```
Dim _ribbonListBox As New RibbonListBox()
Dim item1 As New ListBoxItem() With {.Content = "Office2003Theme"}
Dim item2 As New ListBoxItem() With {.Content = "Office2007Theme"}
Dim item3 As New ListBoxItem() With {.Content = "Office2010Theme"}
_ribbonListBox.Items.Add(item1)
_ribbonListBox.Items.Add(item2)
_ribbonListBox.Items.Add(item3)
ribbonBar2.Items.Add( ribbonListBox)
```



Add ListBox to the simplified layout

When the simplified layout is enabled, the RibbonListBox can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStyle="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True">
<syncfusion:RibbonBar Header="Options">
<syncfusion:RibbonListBox Width="100" >
<ListBoxItem Content="Create New"/>
<ListBoxItem Content="Modify"/>
<ListBoxItem Content="Delete"/>
</syncfusion:RibbonListBox>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
Ribbon ribbon = new Ribbon();
```



```

ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar optionsBar = new RibbonBar();
optionsBar.Header = "Options";
// Creating items
RibbonListBox ribbonListBox = new RibbonListBox();
ListBoxItem item1 = new ListBoxItem() { Content = "Create New" };
ListBoxItem item2 = new ListBoxItem() { Content = "Modify" };
ListBoxItem item3 = new ListBoxItem() { Content = "Delete" };
ribbonListBox.Items.Add(item1);
ribbonListBox.Items.Add(item2);
ribbonListBox.Items.Add(item3);
// Adding items to bar
optionsBar.Items.Add(ribbonListBox);
// Adding bars to the tabs
homeTab.Items.Add(optionsBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the **RibbonListBox** can be ignored as it will be resized automatically to the standard width and height. If the **RibbonListBox** is to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```

<syncfusion:RibbonListBox
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified" >
<ListBoxItem Content="Item 1"/>
<ListBoxItem Content="Item 2"/>
<syncfusion:RibbonListBox.Style >
<Style TargetType="syncfusion:RibbonListBox" BasedOn="{StaticResource
SyncfusionRibbonListBoxStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="48"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>

```

```

</Style>
</syncfusion:RibbonListBox.Style>
</syncfusion:RibbonListBox>

```

RibbonStatusBar in WPF Ribbon

The [RibbonStatusBar](#) control is added to [RibbonWindow](#) to display the current status of the application or document similar in Microsoft Office.

XML

```

<syncfusion:RibbonWindow x:Class="Ribbonstatusbar_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:Ribbonstatusbar_sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="Office2016Colorful"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<syncfusion:RibbonWindow.StatusBar>
<syncfusion:RibbonStatusBar Background="DarkBlue">
<WrapPanel>
<TextBlock Text="Ready" Margin="10,0,0,0" Foreground="AntiqueWhite" />
<TextBlock Text="Page No 1" Margin="20,0,0,0" Foreground="AntiqueWhite" />
</WrapPanel>
</syncfusion:RibbonStatusBar>
</syncfusion:RibbonWindow.StatusBar>
<Grid>
<syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

Create an instance of the StatusBar and assign it to [RibbonStatusBar](#) property of [RibbonWindow](#) through code behind.

C#

```

//Initialize the Ribbon control.
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
//Creating new tabs.
RibbonTab ribbonTab = new RibbonTab();
ribbonTab.Caption = "Home";
ribbonTab.IsChecked = true;
RibbonTab viewTab = new RibbonTab();
viewTab.Caption = "View";
viewTab.IsChecked = false;
//Adding tabs to the Ribbon.
ribbon.Items.Add(ribbonTab);

```

```

ribbon.Items.Add(viewTab);
grid.Children.Add(ribbon);
//Initialize the ribbon status bar.
RibbonStatusBar ribbonStatusBar = new RibbonStatusBar();
ribbonStatusBar.Background = Brushes.DarkBlue;
//Initializing wrap panel.
WrapPanel wrapPanel = new WrapPanel();
double left = 10, top = 0, right = 0, bottom = 0;
double left1 = 20, top1 = 0, right1 = 0, bottom1 = 0;
//Initialize the TextBlock and add the values.
TextBlock textBlock1 = new TextBlock { Text = "Ready", Foreground =
Brushes.AntiqueWhite };
TextBlock textBlock2 = new TextBlock { Text = "Page No 1", Foreground =
Brushes.AntiqueWhite };
textBlock1.Margin = new Thickness(left, top, right, bottom);
textBlock2.Margin = new Thickness(left1, top1, right1, bottom1);
//Add textblock to the wrappanel.
wrapPanel.Children.Add(textBlock1);
wrapPanel.Children.Add(textBlock2);
//Add the wrappanel to the statusbar.
ribbonStatusBar.Items.Add(wrapPanel);
//Add statusbar to the ribbonwindow.
this.StatusBar = ribbonStatusBar;

```

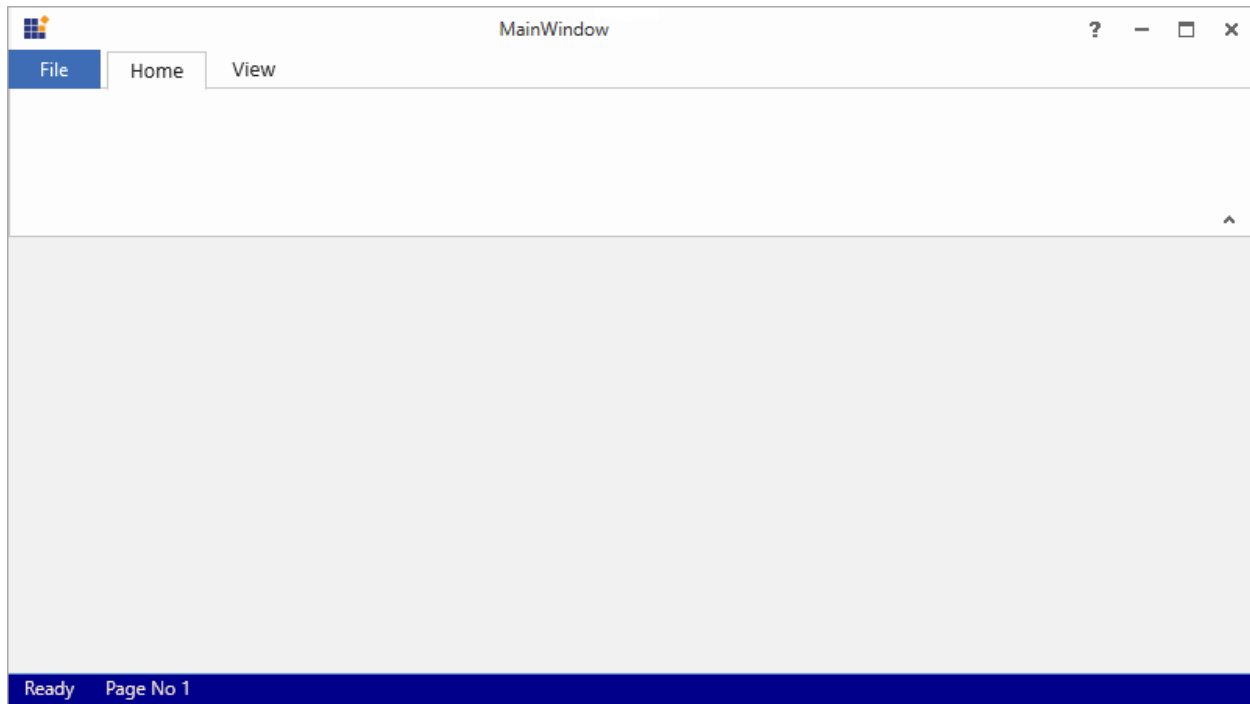
VB.NET

```

'Initialize the Ribbon control.
Dim ribbon As New Ribbon()
ribbon.VerticalAlignment = VerticalAlignment.Top
'Creating new tabs.
Dim ribbonTab As New RibbonTab()
ribbonTab.Caption = "Home"
ribbonTab.IsChecked = True
Dim viewTab As New RibbonTab()
viewTab.Caption = "View"
viewTab.IsChecked = False
'Adding tabs to the Ribbon.
ribbon.Items.Add(ribbonTab)
ribbon.Items.Add(viewTab)
grid.Children.Add(ribbon)
'Initialize the ribbon status bar.
Dim ribbonStatusBar As New RibbonStatusBar()
ribbonStatusBar.Background = Brushes.DarkBlue
'Initializing wrap panel.
Dim wrapPanel As New WrapPanel()
Dim left As Double = 10, top As Double = 0, right As Double = 0, bottom As
Double = 0
Dim left1 As Double = 20, top1 As Double = 0, right1 As Double = 0, bottom1
As Double = 0
'Initialize the TextBlock and add the values.
Dim textBlock1 As TextBlock = New TextBlock With {.Text = "Ready",
.Foreground = Brushes.AntiqueWhite}
Dim textBlock2 As TextBlock = New TextBlock With {.Text = "Page No 1",
.Foreground = Brushes.AntiqueWhite}
textBlock1.Margin = New Thickness(left, top, right, bottom)
textBlock2.Margin = New Thickness(left1, top1, right1, bottom1)

```

```
'Add textblock to the wrappanel.
wrapPanel.Children.Add(textBlock1)
wrapPanel.Children.Add(textBlock2)
'Add the wrappanel to the statusbar.
ribbonStatusBar.Items.Add(wrapPanel)
'Add statusbar to the ribbonwindow.
Me.StatusBar = ribbonStatusBar
```



SimpleMenuButton in WPF Ribbon

SimpleMenuButton provides functionality similar to other menu button and it doesn't carries a sub menu. It can be used with **ApplicationMenu** and it can also be placed inside the **RibbonBar** in Ribbon control.

XML

```
<syncfusion:RibbonWindow x:Class="TemplateSupport.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:TemplateSupport"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skinManager="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d"
skinManager:SfSkinManager.VisualStyle="MaterialLight"
Height="450" Width="800">
<Grid>
<syncfusion:Ribbon Name="_Ribbon1" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Name="_RibbonTab1" Caption="HOME" IsChecked="False">
<syncfusion:RibbonBar Name="_RibbonBar1" Header="RibbonBar1">
```

```

<syncfusion:SimpleMenuButton Icon="/Resources/Copy16.png" Label="Open"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="EDIT" IsChecked="False"/>
<syncfusion:Ribbon.ApplicationMenu>
<syncfusion:ApplicationMenu Name="_ApplicationMenu" Width="38" Height="38"
syncfusion:Ribbon.KeyTip="F" IsPopupOpen="False"
ApplicationButtonImage="/Resources/App.ico">
<syncfusion:SimpleMenuButton Label="File"/>
<syncfusion:SimpleMenuButton Label="Open"/>
</syncfusion:ApplicationMenu>
</syncfusion:Ribbon.ApplicationMenu>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

Create an instance of SimpleMenuButton and add it to ApplicationMenu through code behind.

C#

```

using Syncfusion.SfSkinManager;
using Syncfusion.Windows.Tools.Controls;
using System;
using System.Windows;
using System.Windows.Media.Imaging;
namespace TemplateSupport
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : RibbonWindow
    {
        public MainWindow()
        {
            InitializeComponent();
            Ribbon ribbon = new Ribbon();
            ribbon.VerticalAlignment = VerticalAlignment.Top;
            // Creating new tabs
            RibbonTab homeTab = new RibbonTab();
            homeTab.Caption = "Home";
            homeTab.IsChecked = true;
            RibbonTab editTab = new RibbonTab();
            editTab.Caption = "Edit";
            // Creating new bar
            RibbonBar ribbonBar = new RibbonBar();
            ribbonBar.Header = "Clipboard";
            // Creating items
            ApplicationMenu _ApplicationMenu = new ApplicationMenu();
            ribbon.ApplicationMenu = _ApplicationMenu;
            SimpleMenuButton _SimpleMenuButton = new SimpleMenuButton() { Label =
            "File", Width = 100 };
            SimpleMenuButton _SimpleMenuButton1 = new SimpleMenuButton() { Label =
            "Open", Width = 100 };
            SimpleMenuButton _SimpleMenuButton2 = new SimpleMenuButton() { Label =
            "Menu", Width = 100 };
            _ApplicationMenu.Items.Add( _SimpleMenuButton);

```

```

_ApplicationMenu.Items.Add(_SimpleMenuButton1);
ribbonBar.Items.Add(_SimpleMenuButton2);
homeTab.Items.Add(ribbonBar);
ribbon.Items.Add(homeTab);
ribbon.Items.Add(editTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
}
}
}

```

VB.NET

```

Dim _SimpleMenuButton As New SimpleMenuButton() With {
    .Label = "File",
    .Width = 100
}
Dim _SimpleMenuButton1 As New SimpleMenuButton() With {
    .Label = "Open",
    .Width = 100
}
_ApplicationMenu.Items.Add(_SimpleMenuButton)
_ApplicationMenu.Items.Add(_SimpleMenuButton1)

```

Setting image to SimpleMenuButton

The [SimpleMenuButton](#) allows to display any type of image such as glyph, font or any custom content using [IconTemplate](#) property. It also allows to display a normal image using [Icon](#) property.

Note: The [SimpleMenuButton](#) loads icon in the following priority order,

- [IconTemplate](#)
- [Icon](#)

Setting icon template

The [IconTemplate](#) property provides support to set any type of image such as glyph, font or any custom content to the [SimpleMenuButton](#). The [SimpleMenuButton](#) will automatically resize the template content.

XML

```

<syncfusion:RibbonWindow x:Class="TemplateSupport.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:TemplateSupport"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skinManager="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d"
skinManager:SfSkinManager.VisualStyle="MaterialLight"
Height="450" Width="800">
<Grid>

```

```

<syncfusion:Ribbon Name="_Ribbon1" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
  <syncfusion:RibbonTab Caption="HOME" IsChecked="False">
    <syncfusion:RibbonBar Header="RibbonBar1">
      <syncfusion:SimpleMenuButton Label="Menu">
        <syncfusion:SimpleMenuButton.IconTemplate>
          <DataTemplate >
            <Grid>
              <Path
                Margin="3,1,0.5,0.5"
                Data="M5.5000009,2.500005 L10.500001,2.500005 14.500001,6.500005
                14.500001,14.500005 5.500009,14.500005 z M0,0 L4.0000037,0 4.0000037,12
                0,12 z"
                Fill="White"
                Stretch="Fill" />
              <Path
                Margin="2,0,0,0"
                Data="M9.0000026,11.999999 L13.000003,11.999999 13.000003,12.999999
                9.0000026,12.999999 z M9.0000026,9.9999986 L13.000003,9.9999986
                13.000003,10.999999 9.0000026,10.999999 z M12,4.7070035 L12,7.0000033
                14.293,7.0000033 z M6.9999967,4.0000001 L6.9999967,15 14.999997,15
                14.999997,8.0000033 11,8.0000033 11,4.0000001 z M5.9999967,2.9999999
                L11.706997,2.9999999 15.999997,7.293 15.999997,16 5.9999967,16 z M0,0
                L6.9999967,0 6.9999967,2 5.9999971,2 5.9999971,1 1,1 1,13 4.9999976,13
                4.9999976,14 0,14 z"
                Fill="#FF3A3939"
                Stretch="Fill" />
            </Grid>
          </DataTemplate>
        </syncfusion:SimpleMenuButton.IconTemplate>
      </syncfusion:SimpleMenuButton>
    </syncfusion:RibbonBar>
  </syncfusion:RibbonTab>
  <syncfusion:RibbonTab Caption="EDIT" IsChecked="False"/>
  <syncfusion:Ribbon.ApplicationMenu>
    <syncfusion:ApplicationMenu Name="_ApplicationMenu" Width="38" Height="38"
    syncfusion:Ribbon.KeyTip="F" IsPopupOpen="False">
      <syncfusion:SimpleMenuButton Label="File">
        <syncfusion:SimpleMenuButton.IconTemplate>
          <DataTemplate>
            <Grid>
              <Path
                Width="13"
                Height="16"
                Margin="0.5"
                Data="M0,0 L11,0 11,15 0,15 z"
                Fill="White"
                Stretch="Fill" />
              <Path
                Margin="1"
                Data="M1,1 L1,15 11,15 11,1 z M0,0 L12,0 12,4.158 12,5.0689998 12,16 0,16 z"
                Fill="#FF3A3A38"
                Stretch="Fill" />
              <Path
                Margin="3"
                Data="M0,8.9999991 L5.9999999,8.9999991 5.9999999,9.999999 0,9.999999 z
                M3.2782552E-06,5.9999998 L6.0000033,5.9999998 6.0000033,6.9999996

```

```

3.2782552E-06,6.9999996 z M3.2782552E-06,2.9999995 L6.0000033,2.9999995
6.0000033,3.9999995 3.2782552E-06,3.9999995 z M3.4272668E-06,0 L6.0000033,0
6.0000033,0.99999952 3.4272668E-06,0.99999952 z"
Fill="#FF797774"
Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:SimpleMenuButton.IconTemplate>
</syncfusion:SimpleMenuButton>
<syncfusion:SimpleMenuButton Label="Open" >
<syncfusion:SimpleMenuButton.IconTemplate>
<DataTemplate >
<Grid>
<Path
Margin="3,1,0.5,0.5"
Data="M5.5000009,2.500005 L10.500001,2.500005 14.500001,6.500005
14.500001,14.500005 5.5000009,14.500005 z M0,0 L4.0000037,0 4.0000037,12
0,12 z"
Fill="White"
Stretch="Fill" />
<Path
Margin="2,0,0,0"
Data="M9.0000026,11.999999 L13.000003,11.999999 13.000003,12.999999
9.0000026,12.999999 z M9.0000026,9.9999986 L13.000003,9.9999986
13.000003,10.999999 9.0000026,10.999999 z M12,4.7070035 L12,7.0000033
14.293,7.0000033 z M6.9999967,4.0000001 L6.9999967,15 14.999997,15
14.999997,8.0000033 11,8.0000033 11,4.0000001 z M5.9999967,2.9999999
L11.706997,2.9999999 15.999997,7.293 15.999997,16 5.9999967,16 z M0,0
L6.9999967,0 6.9999967,2 5.9999971,2 5.9999971,1 1,1 1,13 4.9999976,13
4.9999976,14 0,14 z"
Fill="#FF3A3939"
Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:SimpleMenuButton.IconTemplate>
</syncfusion:SimpleMenuButton>
</syncfusion:ApplicationMenu>
</syncfusion:Ribbon.ApplicationMenu>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

using Syncfusion.SfSkinManager;
using Syncfusion.Windows.Tools.Controls;
using System;
using System.Windows;
using System.Drawing;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
namespace TemplateSupport
{
    /// <summary>

```



```

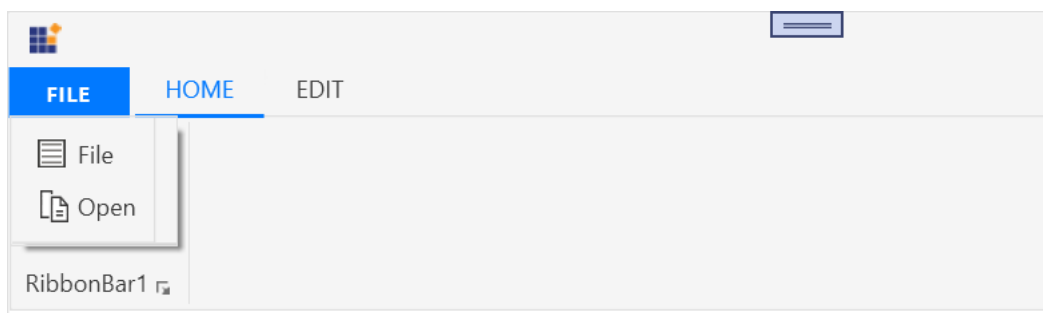
/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : RibbonWindow
{
    public MainWindow()
    {
        InitializeComponent();
        Ribbon ribbon = new Ribbon();
        ribbon.VerticalAlignment = VerticalAlignment.Top;
        // Creating new tabs
        RibbonTab homeTab = new RibbonTab();
        homeTab.Caption = "Home";
        homeTab.IsChecked = true;
        RibbonTab editTab = new RibbonTab();
        editTab.Caption = "Edit";
        // Creating new bar
        RibbonBar ribbonBar = new RibbonBar();
        ribbonBar.Header = "Clipboard";
        // Creating items
        ApplicationMenu _ApplicationMenu = new ApplicationMenu();
        ribbon.ApplicationMenu = _ApplicationMenu;
        DataTemplate iconTemplate1 = new DataTemplate();
        FrameworkElementFactory gridElement = new
        FrameworkElementFactory(typeof(Grid));
        FrameworkElementFactory pathElement1 = new
        FrameworkElementFactory(typeof(Path));
        FrameworkElementFactory pathElement2 = new
        FrameworkElementFactory(typeof(Path));
        pathElement1.SetValue(Path.DataProperty, Geometry.Parse("M5.5000009,2.500005
        L10.500001,2.500005 14.500001,6.500005 14.500001,14.500005
        5.5000009,14.500005 z M0,0 L4.0000037,0 4.0000037,12 0,12 z"));
        pathElement1.SetValue(Path.MarginProperty, new Thickness(3, 1, 0.5, 0.5));
        pathElement1.SetValue(Path.FillProperty, Brushes.White);
        pathElement1.SetValue(Path.StretchProperty, Stretch.Fill);
        pathElement2.SetValue(Path.DataProperty,
        Geometry.Parse("M9.0000026,11.999999 L13.000003,11.999999
        13.000003,12.999999 9.0000026,12.999999 z M9.0000026,9.9999986
        L13.000003,9.9999986 13.000003,10.999999 9.0000026,10.999999 z M12,4.7070035
        L12,7.0000033 14.293,7.0000033 z M6.9999967,4.0000001 L6.9999967,15
        14.999997,15 14.999997,8.0000033 11,8.0000033 11,4.0000001 z
        M5.9999967,2.9999999 L11.706997,2.9999999 15.999997,7.293 15.999997,16
        5.9999967,16 z M0,0 L6.9999967,0 6.9999967,2 5.9999971,2 5.9999971,1 1,1
        1,13 4.9999976,13 4.9999976,14 0,14 z"));
        pathElement2.SetValue(Path.MarginProperty, new Thickness(2, 0, 0, 0));
        pathElement2.SetValue(Path.FillProperty, new
        SolidColorBrush(Color.FromRgb(58, 57, 57)));
        pathElement2.SetValue(Path.StretchProperty, Stretch.Fill);
        gridElement.AppendChild(pathElement1);
        gridElement.AppendChild(pathElement2);
        iconTemplate1.VisualTree = gridElement;
        DataTemplate iconTemplate2 = new DataTemplate();
        FrameworkElementFactory grid2 = new FrameworkElementFactory(typeof(Grid));
        FrameworkElementFactory pathData1 = new
        FrameworkElementFactory(typeof(Path));
        FrameworkElementFactory pathData2 = new
        FrameworkElementFactory(typeof(Path));

```

```

FrameworkElementFactory pathData3 = new
FrameworkElementFactory(typeof(Path));
pathData1.SetValue(Path.DataProperty, Geometry.Parse("M0,0 L11,0 11,15 0,15
z"));
pathData1.SetValue(Path.MarginProperty, new Thickness(0.5));
pathData1.SetValue(Path.FillProperty, Brushes.White);
pathData1.SetValue(Path.StretchProperty, Stretch.Fill);
pathData2.SetValue(Path.DataProperty, Geometry.Parse("M1,1 L1,15 11,15 11,1
z M0,0 L12,0 12,4.158 12,5.0689998 12,16 0,16 z"));
pathData2.SetValue(Path.MarginProperty, new Thickness(1));
pathData2.SetValue(Path.FillProperty, new SolidColorBrush(Color.FromRgb(58,
58, 56)));
pathData2.SetValue(Path.StretchProperty, Stretch.Fill);
pathData3.SetValue(Path.DataProperty, Geometry.Parse("M0,8.9999991
L5.9999999,8.9999991 5.9999999,9.999999 0,9.999999 z M3.2782552E-
06,5.9999998 L6.0000033,5.9999998 6.0000033,6.9999996 3.2782552E-
06,6.9999996 z M3.2782552E-06,2.9999995 L6.0000033,2.9999995
6.0000033,3.9999995 3.2782552E-06,3.9999995 z M3.4272668E-06,0 L6.0000033,0
6.0000033,0.99999952 3.4272668E-06,0.99999952 z"));
pathData3.SetValue(Path.MarginProperty, new Thickness(3));
pathData3.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(121,119,116)));
pathData3.SetValue(Path.StretchProperty, Stretch.Fill);
grid2.AppendChild(pathData1);
grid2.AppendChild(pathData2);
grid2.AppendChild(pathData3);
iconTemplate2.VisualTree = grid2;
SimpleMenuButton _SimpleMenuButton = new SimpleMenuButton() { Label =
"File", Width = 100,IconTemplate=iconTemplate2 };
SimpleMenuButton _SimpleMenuButton1 = new SimpleMenuButton() { Label =
"Open", Width = 100,IconTemplate=iconTemplatel };
SimpleMenuButton _SimpleMenuButton2 = new SimpleMenuButton() { Label =
"Menu", Width = 100,IconTemplate=iconTemplatel };
_ApplicationMenu.Items.Add(_SimpleMenuButton);
_ApplicationMenu.Items.Add(_SimpleMenuButton1);
ribbonBar.Items.Add(_SimpleMenuButton2);
homeTab.Items.Add(ribbonBar);
ribbon.Items.Add(homeTab);
ribbon.Items.Add(editTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
}
}
}

```



Setting image path

The [Icon](#) property is used to set the image path directly to the `SimpleMenuButton`.

XML

```
<syncfusion:RibbonWindow x:Class="TemplateSupport.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:TemplateSupport"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skinManager="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d"
skinManager:SfSkinManager.VisualStudio="MaterialLight"
Height="450" Width="800">
<Grid>
<syncfusion:Ribbon Name="_Ribbon1" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Name="_RibbonTab1" Caption="HOME" IsChecked="False">
<syncfusion:RibbonBar Name="_RibbonBar1" Header="RibbonBar1">
<syncfusion:SimpleMenuButton Icon="/Resources/Copy16.png" Label="Open"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="EDIT" IsChecked="False"/>
<syncfusion:Ribbon.ApplicationMenu>
<syncfusion:ApplicationMenu Name="_ApplicationMenu" Width="38" Height="38"
syncfusion:Ribbon.KeyTip="F" IsPopupOpen="False"
ApplicationButtonImage="/Resources/App.ico">
<syncfusion:SimpleMenuButton Label="Open" Icon="/Resources/Open32.png"/>
<syncfusion:SimpleMenuButton Label="Save" Icon="/Resources/Save16.png"/>
</syncfusion:ApplicationMenu>
</syncfusion:Ribbon.ApplicationMenu>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
using Syncfusion.SfSkinManager;
using Syncfusion.Windows.Tools.Controls;
using System;
using System.Windows;
using System.Windows.Media.Imaging;
namespace TemplateSupport
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : RibbonWindow
    {
        public MainWindow()
        {
            InitializeComponent();
            Ribbon ribbon = new Ribbon();
        }
    }
}
```

```

ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
RibbonTab editTab = new RibbonTab();
editTab.Caption = "Edit";
// Creating new bar
RibbonBar ribbonBar = new RibbonBar();
ribbonBar.Header = "Clipboard";
// Creating items
ApplicationMenu _ApplicationMenu = new ApplicationMenu();
ribbon.ApplicationMenu = _ApplicationMenu;
SimpleMenuButton _SimpleMenuButton = new SimpleMenuButton() { Label =
"Open", Width = 100, Icon = new BitmapImage(new
Uri(@"Resources/Open32.png", UriKind.RelativeOrAbsolute)) };
SimpleMenuButton _SimpleMenuButton1 = new SimpleMenuButton() { Label =
"Save", Width = 100, Icon = new BitmapImage(new
Uri(@"Resources/Save16.png", UriKind.RelativeOrAbsolute)) };
SimpleMenuButton _SimpleMenuButton2 = new SimpleMenuButton() { Label =
"Menu", Width = 100, Icon = new BitmapImage(new
Uri(@"Resources/Copy16.png", UriKind.RelativeOrAbsolute)) };
_ApplicationMenu.Items.Add(_SimpleMenuButton);
_ApplicationMenu.Items.Add(_SimpleMenuButton1);
ribbonBar.Items.Add(_SimpleMenuButton2);
homeTab.Items.Add(ribbonBar);
ribbon.Items.Add(homeTab);
ribbon.Items.Add(editTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
}
}
}

```



RibbonMenuItem in WPF Ribbon

RibbonMenuItem used as entity in menus like ApplicationMenu, DropDownButton, SplitButton, context menu, and so on.

RibbonMenuItem header

The header property used to set the name of the MenuItem. The same has been explained below:

XML

```

<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Name="_ribbonTab1" Caption="HOME" >

```

```
<syncfusion:RibbonBar Name="_ribbonBar1">
<syncfusion:RibbonMenuItem Header="NEW"
Width="100"></syncfusion:RibbonMenuItem>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
```

Create instance of RibbonMenuItem and add it to RibbonBar through code behind.

C#

```
RibbonMenuItem _ribbonMenuItem = new RibbonMenuItem() {Header = "NEW", Width
= 100};
ribbonBar1.Items.Add( _ribbonMenuItem);
```

VB.NET

```
Dim _ribbonMenuItem As New RibbonMenuItem() With {
.Header = "NEW",
.Width = 100
}
ribbonBar1.Items.Add( _ribbonMenuItem)
```



RibbonMenuItem icon

The Icon property used to set the Icon for that RibbonMenuItem. The same has been explained in the below code example:

XML

```
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
<syncfusion:RibbonTab Name="_ribbonTab1" Caption="HOME" >
```

```

<syncfusion:RibbonBar Name="_ribbonBar1">
<syncfusion:RibbonMenuItem Header="NEW">
<syncfusion:RibbonMenuItem.Icon>
<Image Source="SampleImages\options.png"/>
</syncfusion:RibbonMenuItem.Icon>
</syncfusion:RibbonMenuItem>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Name="_ribbonTab2" Caption="View">
<syncfusion:RibbonBar Name="_ribbonBar2">
<syncfusion:RibbonMenuItem Header="View" IconBarEnabled="True">
<syncfusion:RibbonMenuItem.Icon>
<Image Source="SampleImages\sharing.png"/>
</syncfusion:RibbonMenuItem.Icon>
</syncfusion:RibbonMenuItem>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>

```

Create instance of RibbonMenuItem and add the icon for RibbonBar through code behind.

C#

```

RibbonMenuItem _ribbonMenuItem = new RibbonMenuItem() { Header = "NEW1",
Width = 100, Icon = new Image { Source = new BitmapImage(new
Uri(@"SampleImages\sharing.png", UriKind.RelativeOrAbsolute)) } };

```

VB.NET

```

RibbonMenuItem _ribbonMenuItem = new RibbonMenuItem() { Header = "NEW1",
Width = 100, Icon = new Image { Source = new BitmapImage(new
Uri(@"SampleImages\sharing.png", UriKind.RelativeOrAbsolute)) } };

```



Add MenuItem to the simplified layout

When the simplified layout is enabled, the RibbonMenuItem can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

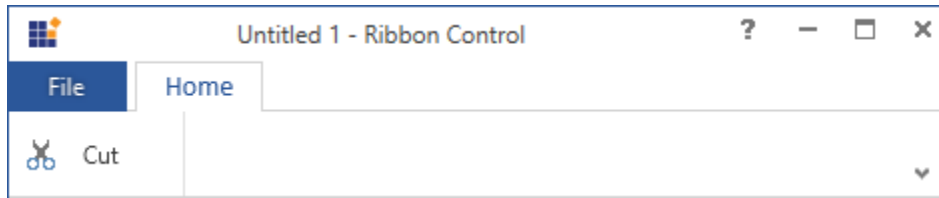
XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.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:RibbonButton_IconTemp"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
syncfusion:SkinStorage.VisualStudio="Office2013"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:RibbonTab Caption="Home" IsChecked="True">
<syncfusion:RibbonBar Header="Clipboard">
<syncfusion:RibbonMenuItem Header="Cut" IconBarEnabled="True">
<syncfusion:RibbonMenuItem.Icon>
<Image Source="/Resources/Cut16.png"/>
</syncfusion:RibbonMenuItem.Icon>
</syncfusion:RibbonMenuItem>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Clipboard";
// Creating items
RibbonMenuItem ribbonMenuItem = new RibbonMenuItem();
ribbonMenuItem.Header = "Cut";
ribbonMenuItem.IconBarEnabled = true;
ribbonMenuItem.Icon = new Image { Source = new BitmapImage(new
Uri(@"Resources\Cut16.png", UriKind.RelativeOrAbsolute)) };
// Adding items to the bar
clipboardBar.Items.Add(ribbonMenuItem);
homeTab.Items.Add(clipboardBar);
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
```

```
SkinStorage.SetVisualStyle(this, "Office2013");
```



When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the RibbonMenuItem can be ignored as it will be resized automatically to the standard width and height. If the RibbonMenuItem is to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```
<syncfusion:RibbonMenuItem Header="Menu item"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified" >
<syncfusion:RibbonMenuItem.Style>
<Style TargetType="syncfusion:RibbonMenuItem" BasedOn="{StaticResource
Office2013RibbonMenuItemStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="48"/>
<Setter Property="Width" Value="100"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>
</syncfusion:RibbonMenuItem.Style>
</syncfusion:RibbonMenuItem >
```

IconSizeChanged event

The event occurs when the IconSize property of the RibbonMenuItem get changed.

XML

```
<syncfusion:RibbonMenuItem Header="View" IconBarEnabled="True"
IconSizeChanged="RibbonMenuItem_IconSizeChanged"/>
```

C#

```
RibbonMenuItem menuItem = new RibbonMenuItem();
menuItem.IconSizeChanged += RibbonMenuItem_IconSizeChanged;
```

VB.NET

```
Private menuItem As RibbonMenuItem = New RibbonMenuItem()
menuItem.IconSizeChanged += RibbonMenuItem_IconSizeChanged
```

To handle the IconSizeChanged event, refer the following code:

C#


```
private void RibbonMenuItem_IconSizeChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    // Insert code to do some operations when the IconSize property is changed
}
```

VB.NET

```
Private Sub RibbonMenuItem_IconSizeChanged(ByVal d As DependencyObject,
ByVal e As DependencyPropertyChangedEventArgs)
    'Insert code to do some operations when the IconSize property is changed
End Sub
```

FlowDirectionChanged event

The event occurs when the FlowDirection property of the RibbonMenuItem get changed.

XML

```
<syncfusion:RibbonMenuItem Header="View"
FlowDirectionChanged="RibbonMenuItem_FlowDirectionChanged"/>
```

C#

```
RibbonMenuItem menuItem = new RibbonMenuItem();
menuItem.FlowDirectionChanged += RibbonMenuItem_FlowDirectionChanged;
```

VB.NET

```
Private menuItem As RibbonMenuItem = New RibbonMenuItem()
menuItem.FlowDirectionChanged += RibbonMenuItem_FlowDirectionChanged
```

To handle the FlowDirectionChanged event, refer the following code:

C#

```
private void RibbonMenuItem_FlowDirectionChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    // Insert code to do some operations when the FlowDirection property is
    changed
}
```

VB.NET

```
Private Sub RibbonMenuItem_IconSizeChanged(ByVal d As DependencyObject,
ByVal e As DependencyPropertyChangedEventArgs)
    'Insert code to do some operations when the IconSize property is changed
End Sub
```

RibbonTabPanelItem in WPF Ribbon

RibbonTabPanelItem is used to display items below application Close button and above the **RibbonBar** content area. It is usually aligned in the right side of the Ribbon and we can place desire items like emoji's, help button etc., in this Tab panel.

XML

```
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
  <syncfusion:RibbonTab Name="_ribbonTab" Caption="HOME" IsChecked="True">
    <syncfusion:RibbonBar Name="_ribbonBar">
      <syncfusion:RibbonMenuItem Header="NEW"
Width="100"></syncfusion:RibbonMenuItem>
    </syncfusion:RibbonBar>
  </syncfusion:RibbonTab>
  <syncfusion:Ribbon.TabPanelItem>
    <syncfusion:RibbonButton SizeForm="Small" Label="Help"/>
  </syncfusion:Ribbon.TabPanelItem>
</syncfusion:Ribbon>
```

Create instance of **RibbonButton** and assign it to **TabPanelItem** property of **Ribbon** through code behind.

C#

```
RibbonButton _ribbonButton = new RibbonButton() { Label="Help"};
ribbon.TabPanelItem = _ribbonButton;
```

VB.NET

```
Dim _ribbonButton As New RibbonButton() With {.Label="Help"}
ribbon.TabPanelItem = _ribbonButton
```



Note:

Ribbon control supports TabPanelItem for both normal and simplified layout, where the simplified layout is designed to display the most commonly used Ribbon commands in a single line interface, allowing more screen space for compact viewing of the content. To know more about the simplified layout, refer [here](#).

RibbonSeparator in WPF Ribbon

RibbonSeparator used to separate the similar set of Ribbon elements in Ribbon. It can separate RibbonButtons according to their purpose.

*Add Separator to the RibbonBar***XML**

```
<syncfusion:Ribbon Name="_ribbon" HorizontalAlignment="Stretch"
VerticalAlignment="Top">
  <syncfusion:RibbonTab Caption="HOME" IsChecked="True">
    <syncfusion:RibbonBar>
      <syncfusion:RibbonButton Label="New"/>
      <syncfusion:RibbonButton Label="Open"/>
      <syncfusion:RibbonButton Label="Save"/>
      <syncfusion:RibbonSeparator/>
      <syncfusion:RibbonButton Label="Cut"/>
      <syncfusion:RibbonButton Label="Copy"/>
      <syncfusion:RibbonButton Label="Paste"/>
    </syncfusion:RibbonBar>
  </syncfusion:RibbonTab>
  <syncfusion:RibbonTab IsChecked="False"
Caption="DESIGN"></syncfusion:RibbonTab>
</syncfusion:Ribbon>
```

RibbonSeparator instance can be added as item to the group of other items through code behind.

C#

```
RibbonSeparator _ribbonSeparator = new RibbonSeparator();
RibbonButton _ribbonButton1 = new RibbonButton() { Label = "New" };
RibbonButton _ribbonButton2 = new RibbonButton() { Label = "Open" };
RibbonButton _ribbonButton3 = new RibbonButton() { Label = "Save" };
RibbonButton _ribbonButton4 = new RibbonButton() { Label = "Cut" };
RibbonButton _ribbonButton5 = new RibbonButton() { Label = "Copy" };
RibbonButton _ribbonButton6 = new RibbonButton() { Label = "Paste" };
_ribbonBar.Items.Add(_ribbonButton1);
_ribbonBar.Items.Add(_ribbonButton2);
_ribbonBar.Items.Add(_ribbonButton3);
_ribbonBar.Items.Add(_ribbonSeparator);
_ribbonBar.Items.Add(_ribbonButton4);
_ribbonBar.Items.Add(_ribbonButton5);
_ribbonBar.Items.Add(_ribbonButton6);
```

VB.NET

```
Dim _ribbonSeparator As New RibbonSeparator()
Dim _ribbonButton1 As New RibbonButton() With {.Label = "New"}
Dim _ribbonButton2 As New RibbonButton() With {.Label = "Open"}
Dim _ribbonButton3 As New RibbonButton() With {.Label = "Save"}
```

```

Dim _ribbonButton4 As New RibbonButton() With {.Label = "Cut"}
Dim _ribbonButton5 As New RibbonButton() With {.Label = "Copy"}
Dim _ribbonButton6 As New RibbonButton() With {.Label = "Paste"}
_ribbonBar.Items.Add(_ribbonButton1)
_ribbonBar.Items.Add(_ribbonButton2)
_ribbonBar.Items.Add(_ribbonButton3)
_ribbonBar.Items.Add(_ribbonSeparator)
_ribbonBar.Items.Add(_ribbonButton4)
_ribbonBar.Items.Add(_ribbonButton5)
_ribbonBar.Items.Add(_ribbonButton6)

```



Add Separator to the simplified layout

When the simplified layout is enabled, the RibbonSeparator can be added and displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```

<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.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:RibbonButton_IconTemp"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" skin:SfSkinManager.VisualStyle="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:RibbonTab Caption="Home" IsChecked="True">
<syncfusion:RibbonBar Header="Clipboard">

```

```
<syncfusion:RibbonButton Label="Paste" SizeForm="Large"
MediumIcon="/Resources/Paste20.png" />
<syncfusion:RibbonSeparator/>
<syncfusion:RibbonButton Label="Cut" SizeForm="Small"
MediumIcon="/Resources/Cut_20.png" />
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Clipboard";
// Creating items
RibbonButton pasteButton = new RibbonButton();
pasteButton.Label = "Paste";
pasteButton.SizeForm = SizeForm.Large;
pasteButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Paste20.png",
UriKind.RelativeOrAbsolute));
RibbonSeparator ribbonSeparator = new RibbonSeparator();
RibbonButton cutButton = new RibbonButton();
cutButton.Label = "Cut";
cutButton.SizeForm = SizeForm.Small;
cutButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Cut_20.png",
UriKind.RelativeOrAbsolute));
// Adding items to the bar
clipboardBar.Items.Add(pasteButton);
clipboardBar.Items.Add(ribbonSeparator);
clipboardBar.Items.Add(cutButton);
// Adding bar to the tab
homeTab.Items.Add(clipboardBar);
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);
```



When arranging in simplified layout alone, the **Margin**, **Width** and **Height** values of the **RibbonSeparator** can be ignored as it will be resized automatically to the standard width and height. If the

RibbonSeparator is to be shown in both normal and simplified layout, the **Margin, Width and Height** properties can be set for normal layout alone using triggers.

XML

```
<syncfusion:RibbonSeparator
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified" >
<syncfusion:RibbonSeparator.Style>
<Style TargetType="syncfusion:RibbonSeparator" BasedOn="{StaticResource
SyncfusionRibbonSeparatorStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="48"/>
<Setter Property="Width" Value="1"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>
</syncfusion:RibbonSeparator.Style>
</syncfusion:RibbonSeparator >
```

RibbonItemHost in WPF Ribbon

WPF Ribbon allows adding any control within the RibbonBar but it does not provide support to add that control in QAT or customize using QAT customization dialog. However, the [RibbonItemHost](#) allows hosting any control, such as Combobox, Textbox, Radio button, Checkbox, and more to be displayed on the [RibbonBar](#) and also allows to add in [QuickAccessToolBar](#) or customize using QAT customization dialog.

Adding custom items to the Ribbon

The [RibbonItemHost ContentTemplate](#) property can be used to host any control.

In the below example, MS CheckBox is set to the [ContentTemplate](#) property of the [RibbonItemHost](#) within the [RibbonBar](#).

XML

```
<syncfusion:RibbonWindow x:Class="Ribbon.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:Ribbon"
mc:Ignorable="d"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.Theme="{skin:SkinManagerExtension ThemeName=FluentLight}"
WindowStartupLocation="CenterScreen"
Title="Custom items demo" Height="450" Width="650">
<syncfusion:RibbonWindow.DataContext>
<local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
<Grid>
<syncfusion:Ribbon x:Name="mainRibbon">
<syncfusion:RibbonTab Caption="Design">
```

```

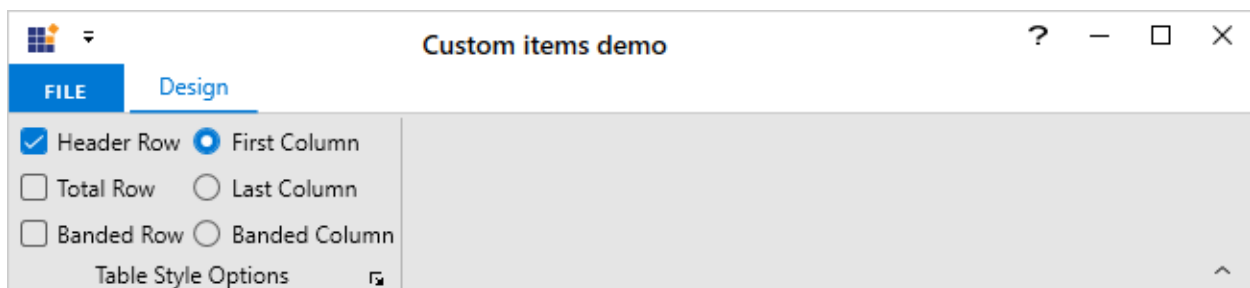
<syncfusion:RibbonBar Header="Table Style Options">
  <syncfusion:RibbonItemHost>
    <syncfusion:RibbonItemHost.ContentTemplate>
      <DataTemplate>
        <CheckBox Content="Header Row" Height="22"/>
      </DataTemplate>
    </syncfusion:RibbonItemHost.ContentTemplate>
  </syncfusion:RibbonItemHost>
  <syncfusion:RibbonItemHost>
    <syncfusion:RibbonItemHost.ContentTemplate>
      <DataTemplate>
        <CheckBox Content="Total Row" Height="22"/>
      </DataTemplate>
    </syncfusion:RibbonItemHost.ContentTemplate>
  </syncfusion:RibbonItemHost>
  <syncfusion:RibbonItemHost>
    <syncfusion:RibbonItemHost.ContentTemplate>
      <DataTemplate>
        <CheckBox Content="Banded Row" Height="22"/>
      </DataTemplate>
    </syncfusion:RibbonItemHost.ContentTemplate>
  </syncfusion:RibbonItemHost>
  <syncfusion:RibbonItemHost>
    <syncfusion:RibbonItemHost.ContentTemplate>
      <DataTemplate>
        <RadioButton Margin="4,0,0,0" Content="First Column" Height="22"/>
      </DataTemplate>
    </syncfusion:RibbonItemHost.ContentTemplate>
  </syncfusion:RibbonItemHost>
  <syncfusion:RibbonItemHost>
    <syncfusion:RibbonItemHost.ContentTemplate>
      <DataTemplate>
        <RadioButton Margin="4,0,0,0" Content="Last Column" Height="22"/>
      </DataTemplate>
    </syncfusion:RibbonItemHost.ContentTemplate>
  </syncfusion:RibbonItemHost>
  <syncfusion:RibbonItemHost>
    <syncfusion:RibbonItemHost.ContentTemplate>
      <DataTemplate>
        <RadioButton Margin="4,0,0,0" Content="Banded Column" Height="22"/>
      </DataTemplate>
    </syncfusion:RibbonItemHost.ContentTemplate>
  </syncfusion:RibbonItemHost>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>
  <syncfusion:BackStageCommandButton Header="Save" >
    <syncfusion:BackStageCommandButton.IconTemplate>
      <DataTemplate>
        <Path Width="12" Height="12" HorizontalAlignment="Center"
          VerticalAlignment="Center"
          Data="M5.0000019,11 L5.0000019,15 11.000002,15 11.000002,11 z M4.0000019,1
            L4.0000019,6 12.000002,6 12.000002,1 z M1,1 L1,13.174 2.7160001,15

```

```

4.0000019,15 4.0000019,10 12.000002,10 12.000002,15 15,15 15,1 13.000002,1
13.000002,7 3.0000019,7 3.0000019,1 z M0,0 L3.0000019,0 13.000002,0 16,0
16,16 12.000002,16 4.0000019,16 2.2840004,16 0,13.57 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" Stretch="Uniform" />
</DataTemplate>
</syncfusion:BackStageCommandButton.IconTemplate>
</syncfusion:BackStageCommandButton>
<syncfusion:BackStageCommandButton Header="Close" >
<syncfusion:BackStageCommandButton.IconTemplate>
<DataTemplate>
<Grid Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center" SnapsToDevicePixels="true">
<Path
Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M1.4139423,0L7.0029922,5.5845888 12.592018,0 14.006015,1.4149939
8.4180527,6.9985202 14.006,12.582007 12.591996,13.997001 7.0030056,8.4124444
1.4140122,13.997001 1.5026823E-05,12.582007 5.5879484,6.9985092 0,1.4149939z
"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
SnapsToDevicePixels="True" Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:BackStageCommandButton.IconTemplate>
</syncfusion:BackStageCommandButton>
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Adding the custom item to the QAT

The control hosted in [RibbonItemHost](#) can be added to the [QuickAccessToolBar](#) using ribbon context menu or through QAT customization dialog. In the QAT customization dialog, all the ribbon items are displayed using its [Label](#) and [IconTemplate](#) properties. Similarly, the [RibbonItemHost](#) also allows us to set the label and icon using its [Label](#) and [IconTemplate](#) properties respectively.

- [Label](#) - Used to display the text of [RibbonItemHost](#) inside the QAT customization dialog.
- [IconTemplate](#) - Gets or sets the template that is used to display the icon of [RibbonItemHost](#) inside the QAT customization dialog.

In the below example, the [Label](#) and [IconTemplate](#) are set for the [RibbonItemHost](#) controls which will be used when the items are displayed in the QAT customization dialog.

XML

```
<syncfusion:RibbonWindow x:Class="Ribbon.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:Ribbon"
mc:Ignorable="d"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.Theme="{skin:SkinManagerExtension ThemeName=FluentLight}"
WindowStartupLocation="CenterScreen"
Title="Custom items demo" Height="450" Width="650">
<syncfusion:RibbonWindow.DataContext>
<local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
<syncfusion:RibbonWindow.Resources>
<DataTemplate x:Key="OnePage">
<Grid>
<Path
Width="13"
Height="16"
Margin="0.5"
Data="M0,0 L11,0 11,15 0,15 z"
Fill="White"
Stretch="Fill" />
<Path
Margin="1"
Data="M1,1 L1,15 11,15 11,1 z M0,0 L12,0 12,4.158 12,5.0689998 12,16 0,16 z"
Fill="#FF3A3A38"
Stretch="Fill" />
<Path
Margin="3"
Data="M0,8.9999991 L5.9999999,8.9999991 5.9999999,9.999999 0,9.999999 z
M3.2782552E-06,5.9999998 L6.0000033,5.9999998 6.0000033,6.9999996
3.2782552E-06,6.9999996 z M3.2782552E-06,2.9999995 L6.0000033,2.9999995
6.0000033,3.9999995 3.2782552E-06,3.9999995 z M3.4272668E-06,0 L6.0000033,0
6.0000033,0.99999952 3.4272668E-06,0.99999952 z"
Fill="#FF797774"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="AlignLeft">
<Path
x:Name="Align_Left1"
Width="14"
Height="13"
Margin="2,0"
HorizontalAlignment="Left"
VerticalAlignment="Bottom"
Data="M0,12 L10,12 10,13 0,13 z M0,8 L14,8 14,9 0,9 z M0,4 L10,4 10,5 0,5 z
M0,0 L14,0 14,1 0,1 z"
/></DataTemplate>
</syncfusion:RibbonWindow.Resources>
</syncfusion:RibbonWindow.DataContext>
</local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
</syncfusion:RibbonWindow>
```

```

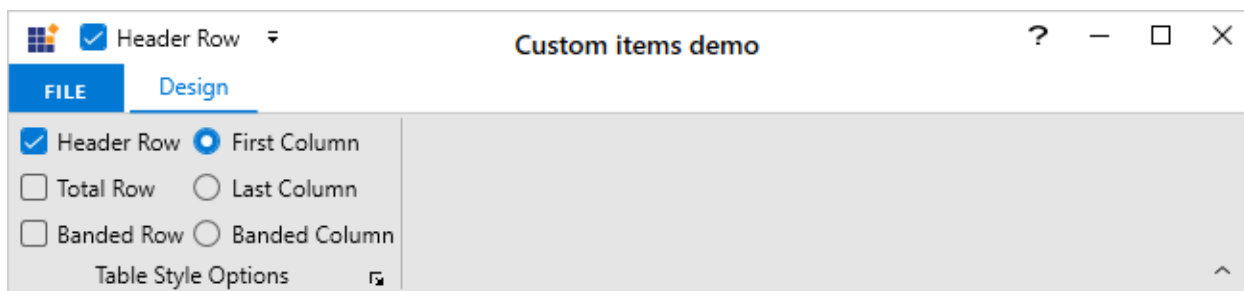
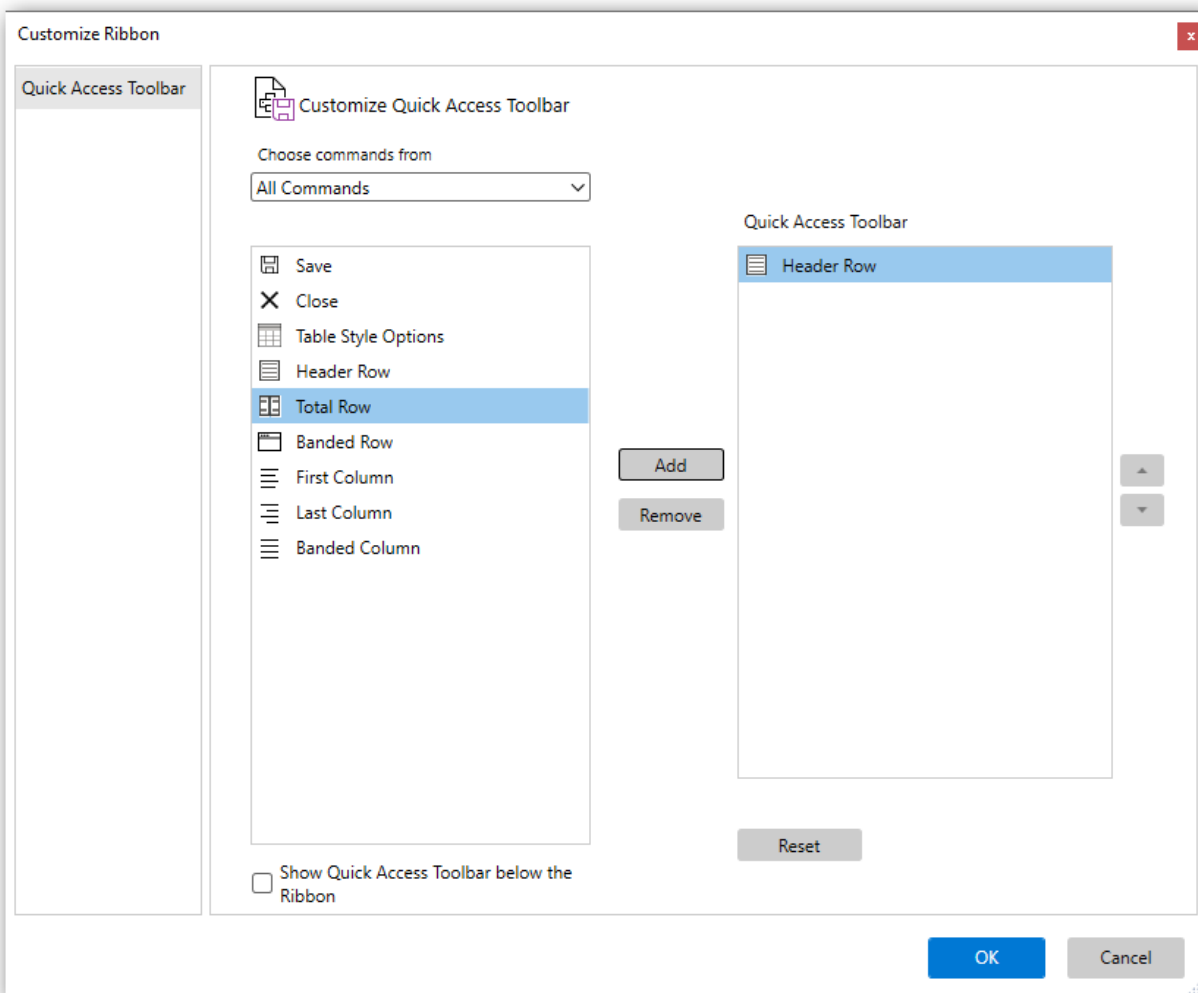
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
Stretch="Fill" />
</DataTemplate>
<DataTemplate x:Key="Tables">
<Grid MaxHeight="32" MaxWidth="32">
<Path
Margin="0.5,4.5,0.5,0.5"
Data="M0,0 L27,0 27,23 0,23 z"
Fill="White"
Stretch="Fill" />
<Path
Height="4"
Margin="0.5,0.5,0.5,0"
VerticalAlignment="Top"
Data="M0,0 L27,0 27,4 0,4 z"
Fill="#FFC8C6C4"
Stretch="Fill" />
<Path
Margin="1,5,1,1"
Data="M9,8 L9,14 17,14 17,8 z M8,0 L9,0 9,7 17,7 17,0 18,0 18,7 26,7 26,8
18,8 18,14 26,14 26,15 18,15 18,22 17,22 17,15 9,15 9,22 8,22 8,15 0,15 0,14
8,14&#xa;8,8 0,8 0,7 8,7 z"
Fill="#FF797774"
Stretch="Fill" />
<Path
Data="M0.99999994,5.0000001 L0.99999994,27 27,27 27,5.0000001 z
M0.99999994,1 L0.99999994,4.0000002 27,4.0000002 27,1 z M0,0 L28,0
28,4.0000002 28,5.0000001 28,28 0,28 0,5.0000001 0,4.0000002 z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:RibbonWindow.Resources>
<Grid>
<syncfusion:Ribbon x:Name="mainRibbon">
<syncfusion:RibbonTab Caption="Design">
<syncfusion:RibbonBar Header="Table Style Options"
IconTemplate="{StaticResource Tables}">
<syncfusion:RibbonItemHost Label="Header Row" IconTemplate="{StaticResource
OnePage}">
<syncfusion:RibbonItemHost.ContentTemplate>
<DataTemplate>
<CheckBox Content="Header Row" Height="22"/>
</DataTemplate>
</syncfusion:RibbonItemHost.ContentTemplate>
</syncfusion:RibbonItemHost>
<syncfusion:RibbonItemHost Label="First Column"
IconTemplate="{StaticResource AlignLeft}">
<syncfusion:RibbonItemHost.ContentTemplate>
<DataTemplate>
<RadioButton Margin="4,0,0,0" Content="First Column" Height="22"/>
</DataTemplate>
</syncfusion:RibbonItemHost.ContentTemplate>
</syncfusion:RibbonItemHost>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>

```

```

<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>
<syncfusion:BackStageCommandButton Header="Save" >
<syncfusion:BackStageCommandButton.IconTemplate>
<DataTemplate>
<Path Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M5.0000019,11 L5.0000019,15 11.000002,15 11.000002,11 z M4.0000019,1
L4.0000019,6 12.000002,6 12.000002,1 z M1,1 L1,13.174 2.7160001,15
4.0000019,15 4.0000019,10 12.000002,10 12.000002,15 15,15 15,1 13.000002,1
13.000002,7 3.0000019,7 3.0000019,1 z M0,0 L3.0000019,0 13.000002,0 16,0
16,16 12.000002,16 4.0000019,16 2.2840004,16 0,13.57 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" Stretch="Uniform" />
</DataTemplate>
</syncfusion:BackStageCommandButton.IconTemplate>
</syncfusion:BackStageCommandButton>
<syncfusion:BackStageCommandButton Header="Close" >
<syncfusion:BackStageCommandButton.IconTemplate>
<DataTemplate>
<Grid Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center" SnapsToDevicePixels="true">
<Path
Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M1.4139423,0L7.0029922,5.5845888 12.592018,0 14.006015,1.4149939
8.4180527,6.9985202 14.006,12.582007 12.591996,13.997001 7.0030056,8.4124444
1.4140122,13.997001 1.5026823E-05,12.582007 5.5879484,6.9985092 0,1.4149939z
"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
SnapsToDevicePixels="True" Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:BackStageCommandButton.IconTemplate>
</syncfusion:BackStageCommandButton>
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```



Note: [View sample in GitHub](#)

Quick Access ToolBar in WPF Ribbon

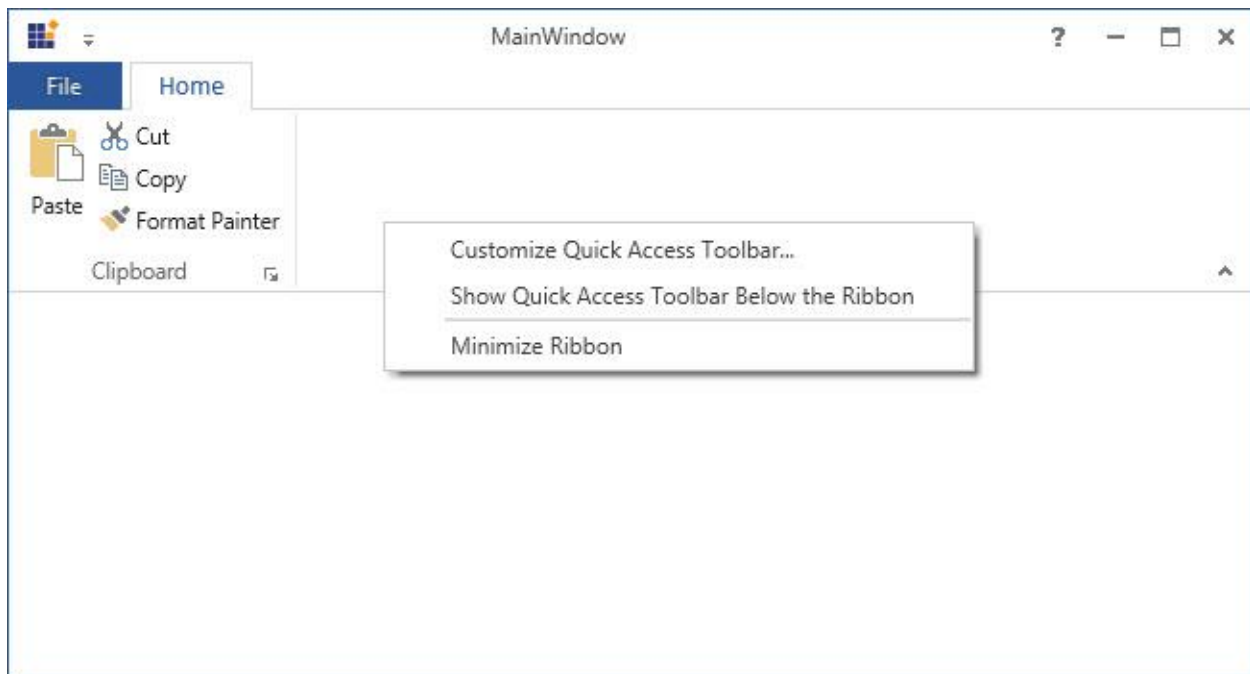
RibbonContextMenu allows to customize the Ribbon with the Right Click and it displays different functions for Ribbon and RibbonItems.

XML

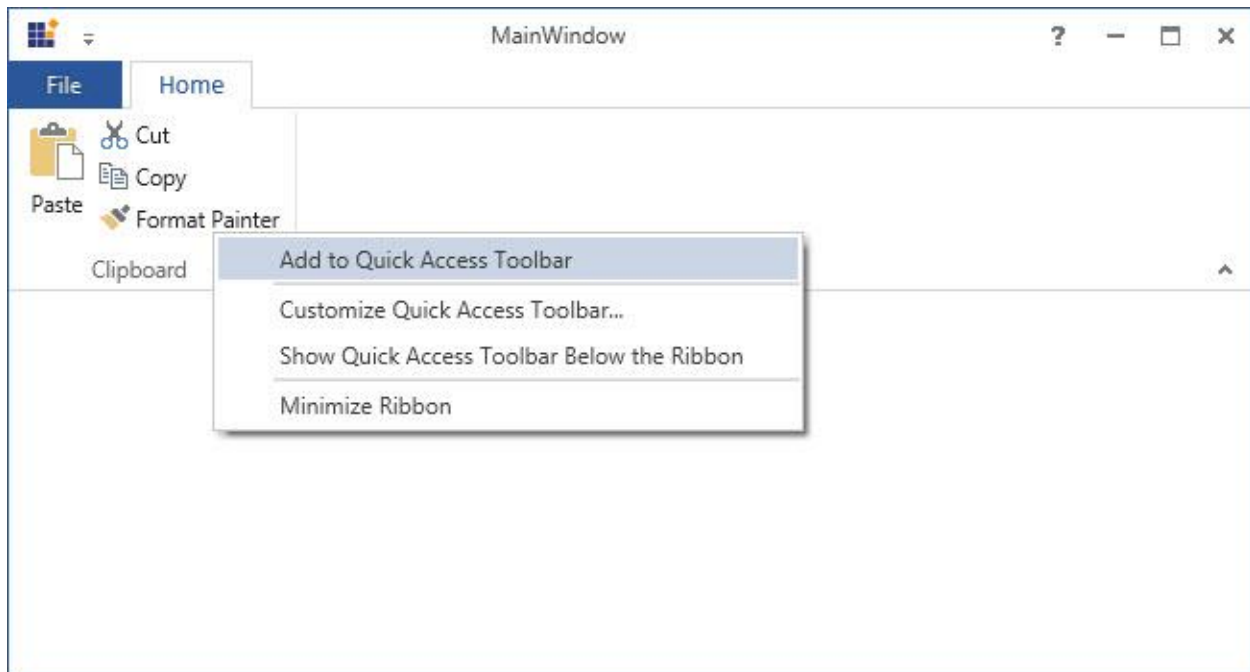
```
<syncfusion:Ribbon x:Name="Ribbon" VerticalAlignment="Top">
  <syncfusion:Ribbon.QuickAccessToolBar>
    <syncfusion:QuickAccessToolBar
      syncfusion:WindowChrome.IsHitTestVisibleInChrome="True"/>
  </syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon>
```

```
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home" >
  <syncfusion:RibbonBar Header="Clipboard" >
    <syncfusion:RibbonButton Label="Paste" SizeForm="Large"
    LargeIcon="/Resources/Paste32.png" />
    <syncfusion:RibbonButton Label="Cut" SizeForm="Small"
    SmallIcon="/Resources/Cut16.png" />
    <syncfusion:RibbonButton Label="Copy" SizeForm="Small"
    SmallIcon="/Resources/Copy16.png" />
    <syncfusion:RibbonButton Label="Format Painter" SizeForm="Small"
    SmallIcon="/Resources/FormatPainter16.png" />
  </syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
```

The below RibbonContextMenu gets generated when the user right click on the ribbon



It displays along with "Add to Quick Access Toolbar" function while clicking at the RibbonItem as like in the below screenshot

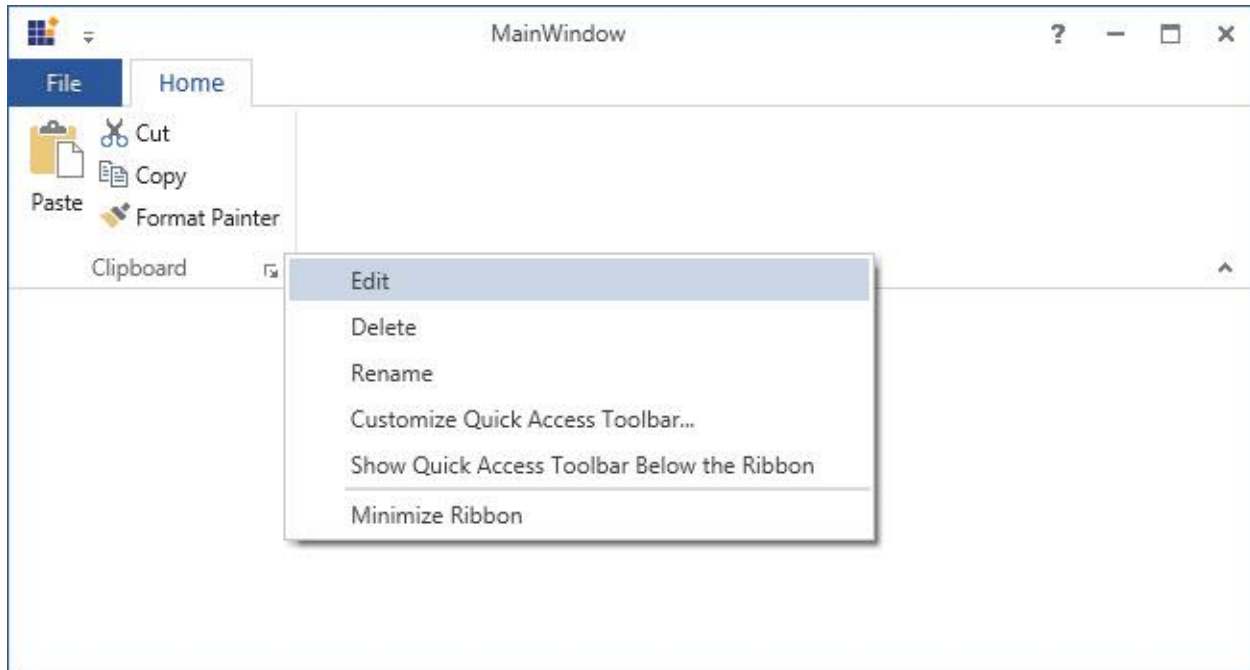


Adding custom item to the ContextMenu

Ribbon Context Menu supports display of custom items. To add the custom item, set an attached property called `CustomContextMenuItems` of the `RibbonContextMenu`

XML

```
<syncfusion:Ribbon x:Name="Ribbon" VerticalAlignment="Top"
syncfusion:RibbonContextMenu.IsCustomContextMenuItemsOnTop="True" >
  <syncfusion:RibbonContextMenu.CustomContextMenuItems >
    <syncfusion:RibbonMenuItem Header="Edit" IsCheckable="True" />
    <syncfusion:RibbonMenuItem Header="Delete" IsCheckable="True" />
    <syncfusion:RibbonMenuItem Header="Rename" IsCheckable="True" />
  </syncfusion:RibbonContextMenu.CustomContextMenuItems>
</syncfusion:Ribbon>
```



How to disable the RibbonContextMenu

In order to disable the ContextMenu handle the `RibbonContextMenuOpening` event

XML

```
<syncfusion:Ribbon x:Name="Ribbon" VerticalAlignment="Top"
RibbonContextMenuOpening="Ribbon_ContextMenuOpening">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar
syncfusion:WindowChrome.IsHitTestVisibleInChrome="True"/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="Home" >
<syncfusion:RibbonBar Header="Clipboard" >
<syncfusion:RibbonButton Label="Paste" SizeForm="Large"
LargeIcon="/Resources/Paste32.png" />
<syncfusion:RibbonButton Label="Cut" SizeForm="Small"
SmallIcon="/Resources/Cut16.png" />
<syncfusion:RibbonButton Label="Copy" SizeForm="Small"
SmallIcon="/Resources/Copy16.png" />
<syncfusion:RibbonButton Label="Format Painter" SizeForm="Small"
SmallIcon="/Resources/FormatPainter16.png" />
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
```

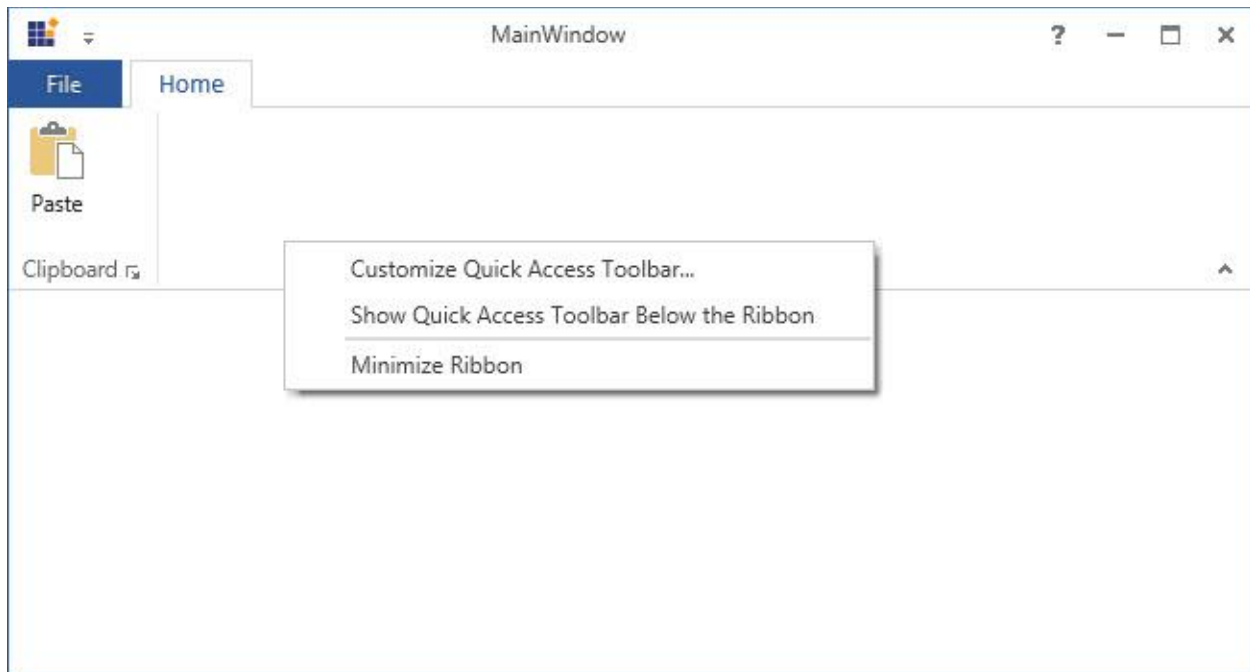
C#

```
private void Ribbon_ContextMenuOpening(object sender, ContextMenuEventArgs
e)
{
e.Handled = true;
}
```

VB.NET

```
Private Sub Ribbon_ContextMenuOpening(ByVal sender As Object, ByVal e As ContextMenuEventArgs)  
    e.Handled = True  
End Sub
```

The following snapshot gets generated before handling the RibbonContextMenu event



After the event is handled, the output gets display as follows



Add items to QuickAccessToolBar (QAT)

Quick Access Toolbar is used to group the frequently used commands above or under the Ribbon, and it allows to add or remove commands to it. It is placed next to the ApplicationMenu to provide end users with the easy accessibility.

Add default QAT items

Use the following code to add items to the QuickAccessToolBar

XML

```
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:RibbonButton Label="Undo" SmallIcon="/Resources/Undo16.png"
SizeForm="ExtraSmall"
ToolTip="Undo" syncfusion:RibbonCommandManager.SynchronizedItem="Undo" />
<syncfusion:RibbonButton Label="Redo" SmallIcon="/Resources/Redo16.png"
SizeForm="ExtraSmall"
ToolTip="Redo" syncfusion:RibbonCommandManager.SynchronizedItem="Redo"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
```

C#

```
RibbonButton RibbonButton = new RibbonButton();
RibbonButton.SmallIcon = new BitmapImage(new Uri("/Resources/Redo16.png",
UriKind.Relative));
RibbonButton.SizeForm = SizeForm.ExtraSmall;
QuickAccessToolBar QAT=new QuickAccessToolBar();
QAT.Items.Add(RibbonButton);
Ribbon.QuickAccessToolBar = QAT;
```

VB.NET

```
Dim RibbonButton As New RibbonButton()
RibbonButton.SmallIcon = New BitmapImage(New Uri("/Resources/Redo16.png",
UriKind.Relative))
RibbonButton.SizeForm = SizeForm.ExtraSmall
Dim QAT As New QuickAccessToolBar()
QAT.Items.Add(RibbonButton)
Ribbon.QuickAccessToolBar = QAT
```



Add items to QAT Menu items

Ribbon also supports to add the items to QAT Menu items. To add the items to the Drop Down Menu of the QuickAccessToolBar, use the attached property, `QATMenuItems` of the Quick Access ToolBar . `QATMenuItems` can be added to the QAT by making the Selection.

XML

```
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Save"
syncfusion:RibbonCommandManager.SynchronizedItem="Save" />
<syncfusion:RibbonButton Label="Quick Print"
syncfusion:RibbonCommandManager.SynchronizedItem="Quick Print"/>
<syncfusion:RibbonButton Label="Print Preview"
syncfusion:RibbonCommandManager.SynchronizedItem="Print Preview"/>
<syncfusion:RibbonButton Label="Undo"
syncfusion:RibbonCommandManager.SynchronizedItem="Undo" />
<syncfusion:RibbonButton Label="Redo"
syncfusion:RibbonCommandManager.SynchronizedItem="Redo" />
<syncfusion:RibbonButton Label="Paste"
syncfusion:RibbonCommandManager.SynchronizedItem="Paste"/>
</syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Undo" SmallIcon="/Resources/Undo16.png"
SizeForm="ExtraSmall"
ToolTip="Undo" syncfusion:RibbonCommandManager.SynchronizedItem="Undo" />
<syncfusion:RibbonButton Label="Redo" SmallIcon="/Resources/Redo16.png"
SizeForm="ExtraSmall"
ToolTip="Redo" syncfusion:RibbonCommandManager.SynchronizedItem="Redo"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
```

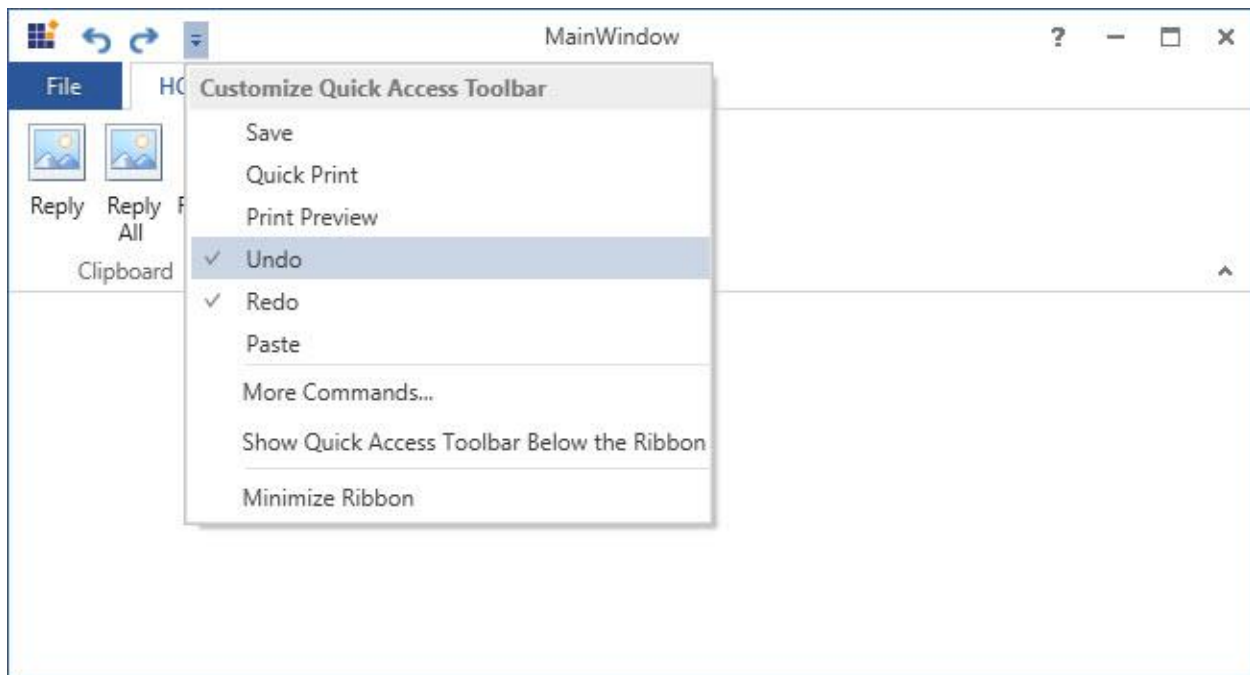
C#

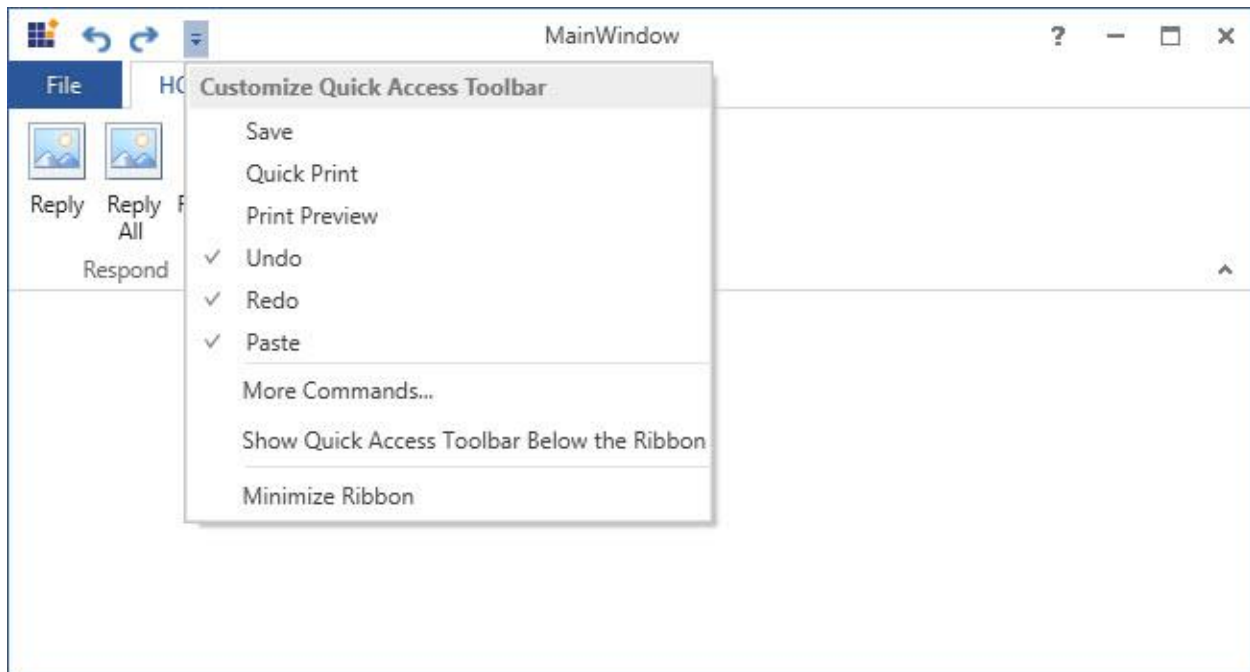
```
RibbonButton pasteRibbonButton = new RibbonButton() { Label = "Paste",  
SmallIcon = new BitmapImage(new Uri("/Resources/Paste32.png",  
UriKind.Relative)) };  
this.Ribbon.QuickAccessToolBar.QATMenuItems.Add(pasteRibbonButton);
```

VB.NET

```
Dim pasteRibbonButton As New RibbonButton() With {  
.Label = "Paste",  
.SmallIcon = New BitmapImage(New Uri("/Resources/Paste32.png",  
UriKind.Relative))  
}  
Me.Ribbon.QuickAccessToolBar.QATMenuItems.Add(pasteRibbonButton)
```

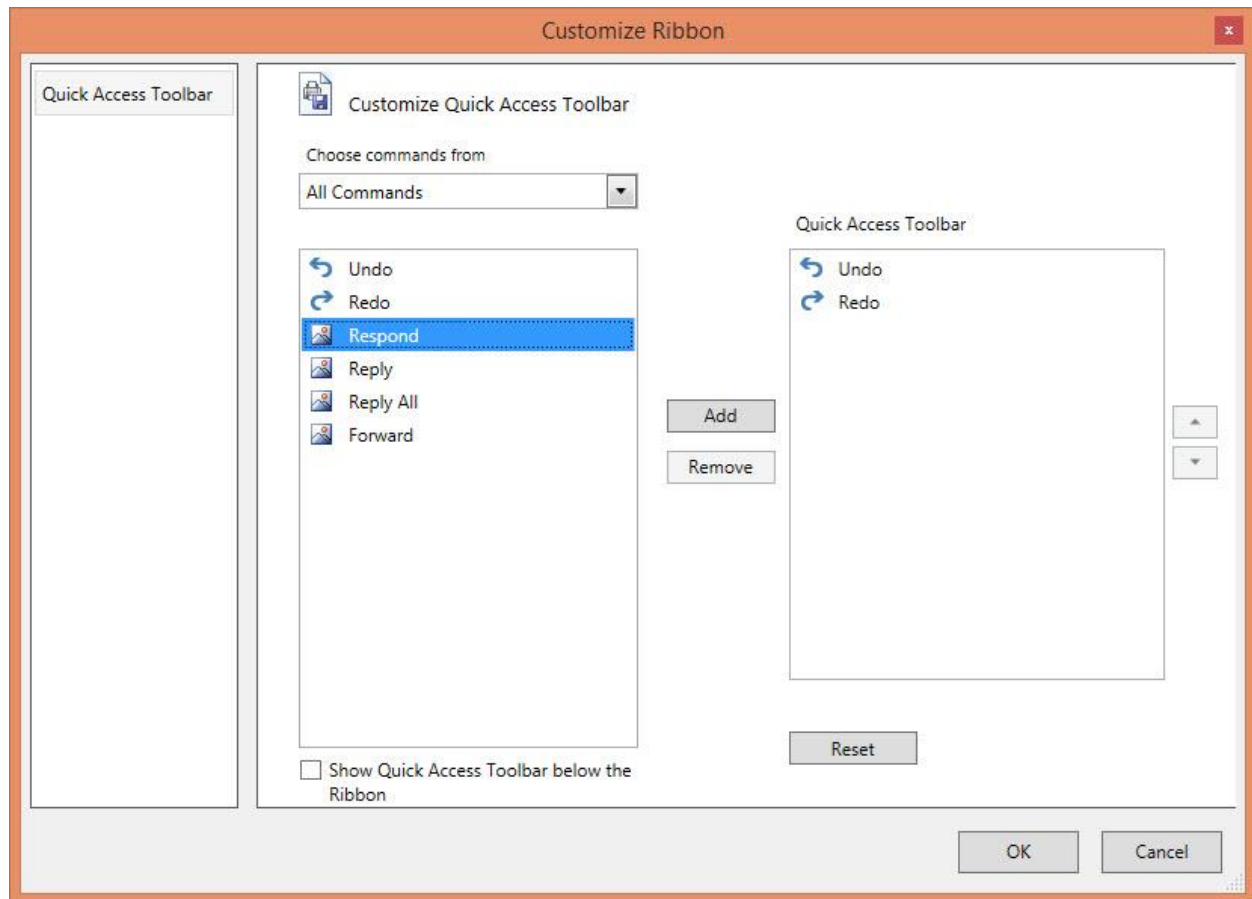
The “Paste” QATMenuItem has been selected and it is displayed as one of the items in the QAT.

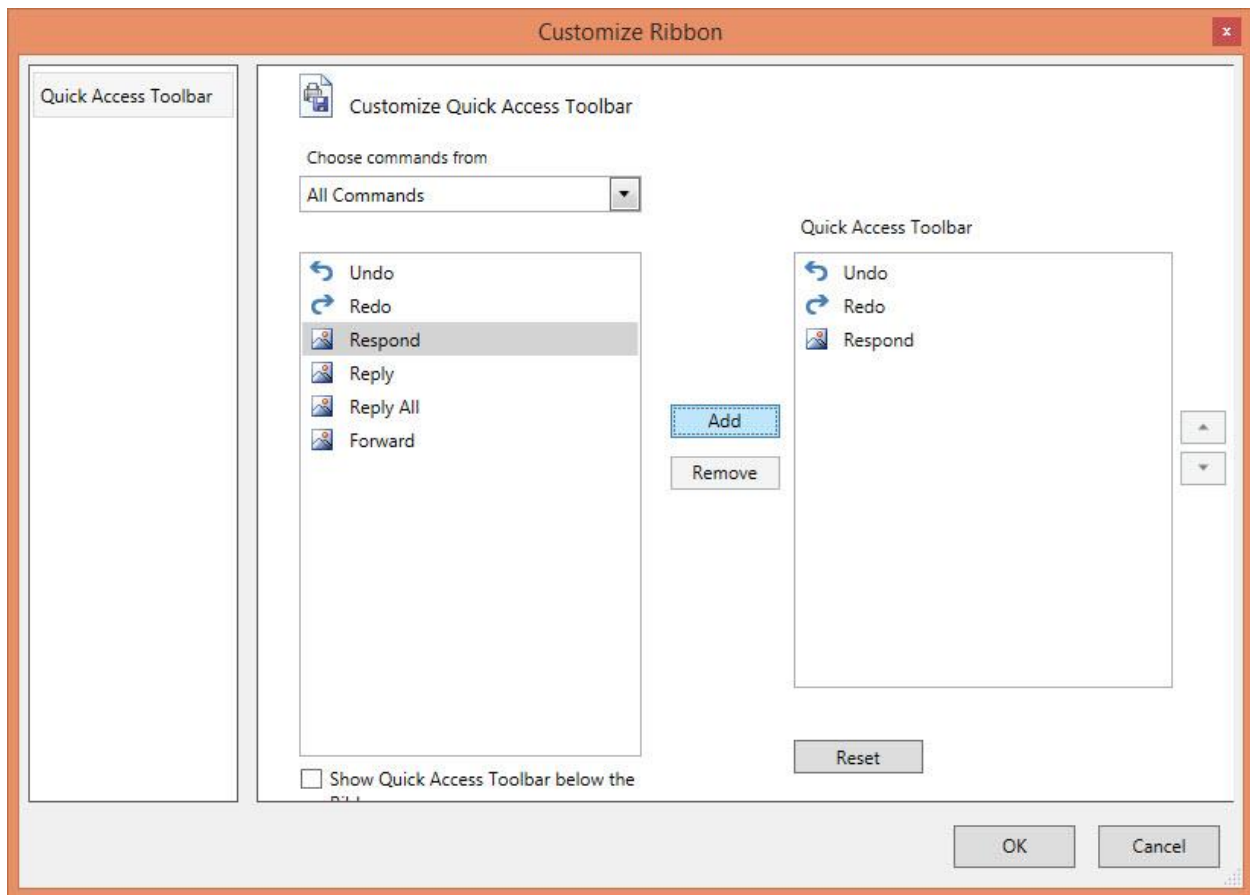




Add items to QAT customize window

To open QAT Customize Window, select **MoreCommands** option from the ContextMenu of the QAT. In the QAT Customized Window, the list of Commands is available. The Commands can be filtered only from the Particular tab by using **Choose commands from** option. Then, select the Command to add to the QuickAccessToolBar and add commands to the right Pane of the Quick Access ToolBar Dialog by clicking Add Button. Finally click OK.





Finally, the Items gets displayed in the QAT



Add custom QAT items

Ribbon control provides an option to add items to the QAT that will not be present in the Ribbon. These items can be defined in the [CustomQATItems](#) collection of the **QuickAccessToolBar** and can be accessed from the QAT Window in the “**Commands Not in the Ribbon**” and “**All Commands**” section.

XML

```
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar QATDropDownVisiblity="Visible">
<syncfusion:QuickAccessToolBar.CustomQATItems>
<syncfusion:RibbonButton Label="Undo" SmallIcon="/Resources/Undo_01.png" />
<syncfusion:RibbonButton Label="Redo" SmallIcon="/Resources/Redo_01.png" />
<syncfusion:RibbonButton Label="New" Command="ApplicationCommands.Open"
SmallIcon="/Resources/Document-01.png"/>
</syncfusion:QuickAccessToolBar.CustomQATItems>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
```

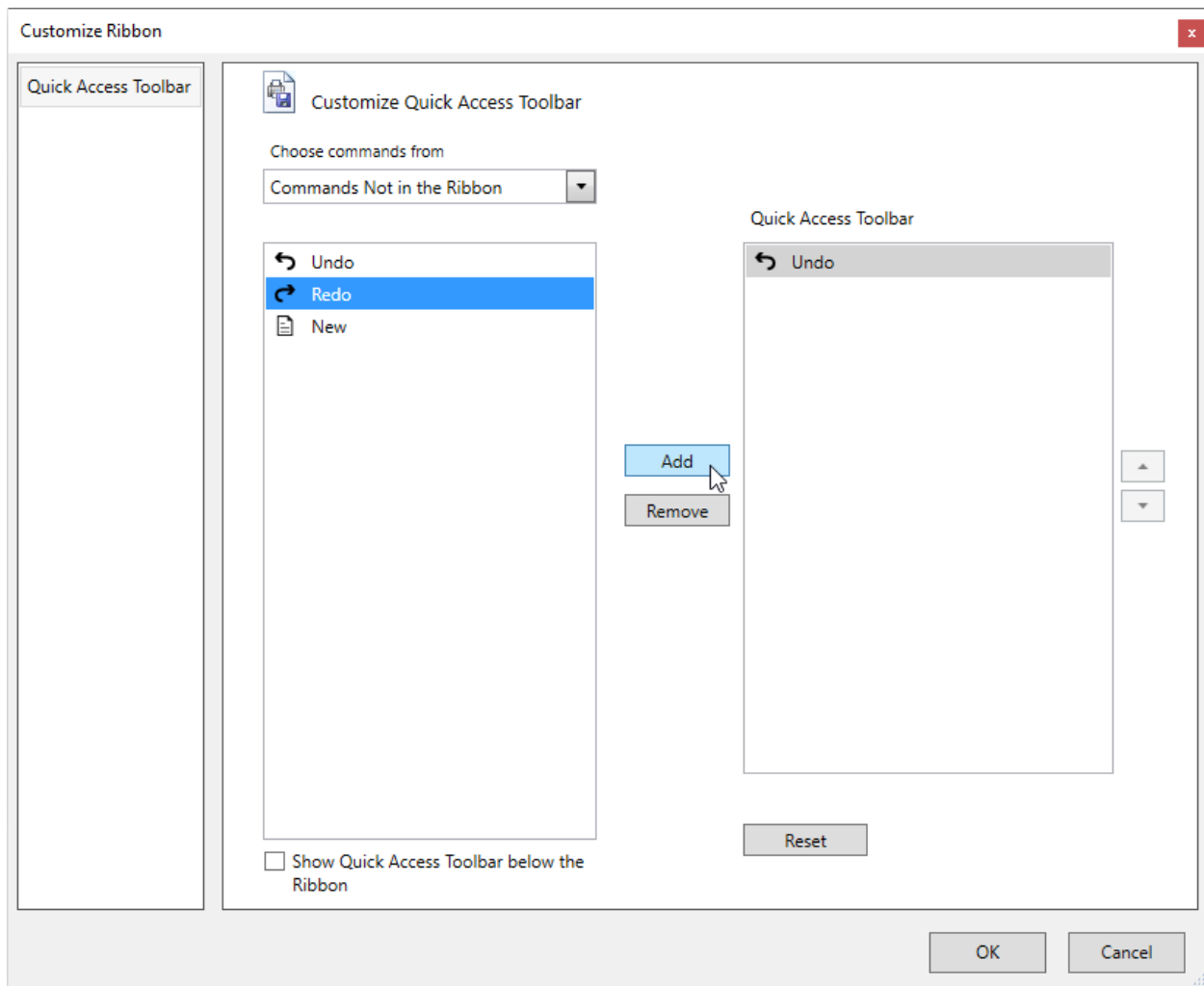
C#

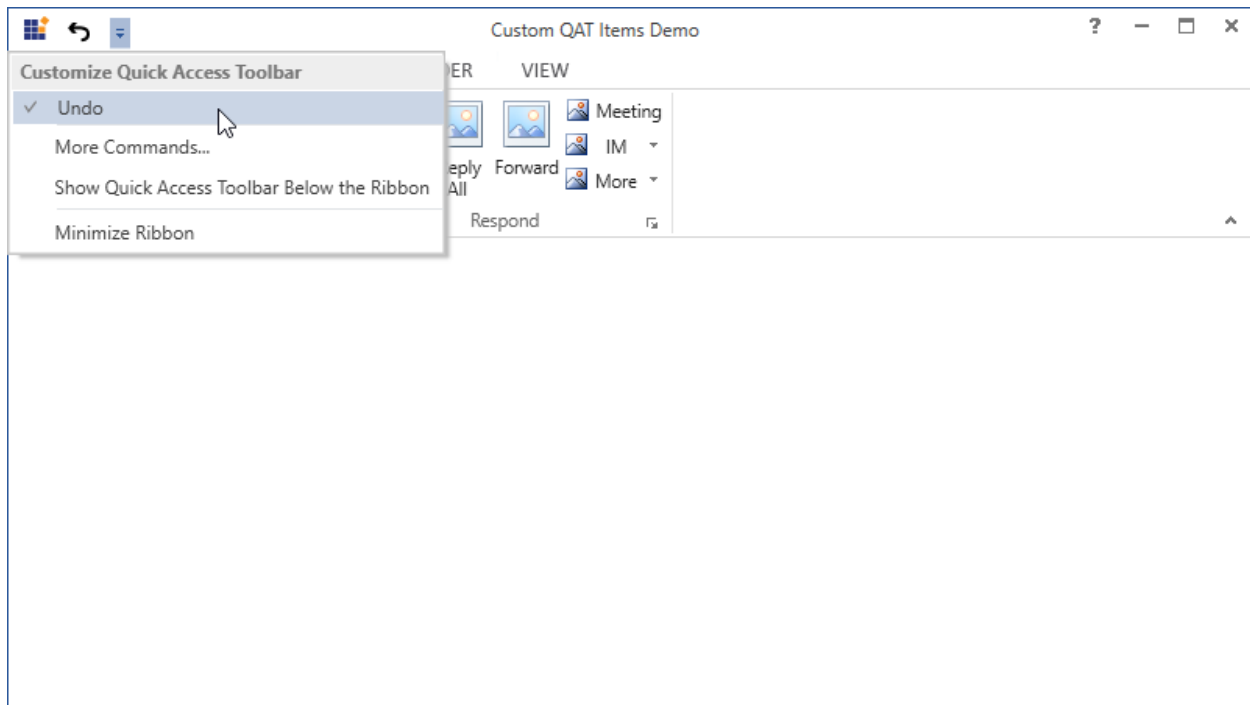
```
RibbonButton undoRibbonButton = new RibbonButton() { Label = "Undo",
SmallIcon = new BitmapImage(new Uri("/Resources/Undo_01.png",
UriKind.Relative))};
RibbonButton redoRibbonButton = new RibbonButton() { Label = "Redo",
SmallIcon = new BitmapImage(new Uri("/Resources/Redo_01.png",
UriKind.Relative))};
RibbonButton newRibbonButton = new RibbonButton() { Label = "New", SmallIcon
= new BitmapImage(new Uri("/Resources/Document-01.png", UriKind.Relative))};
this.Ribbon.QuickAccessToolBar.CustomQATItems.Add(undoRibbonButton);
this.Ribbon.QuickAccessToolBar.CustomQATItems.Add(redoRibbonButton);
this.Ribbon.QuickAccessToolBar.CustomQATItems.Add(newRibbonButton);
```

VB.NET

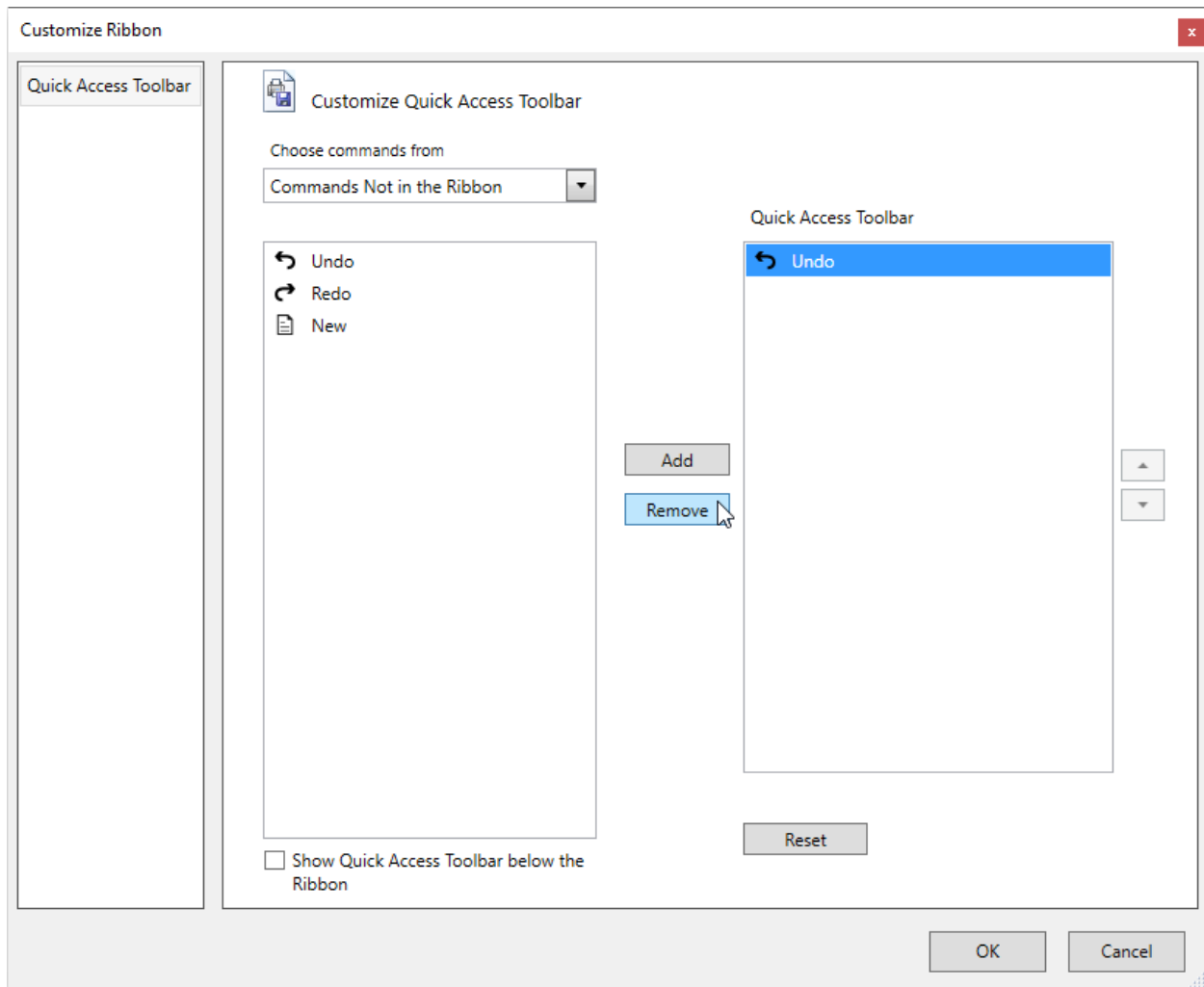
```
Dim undoRibbonButton As New RibbonButton() With {
.Label = "Undo",
.SmallIcon = New BitmapImage(New Uri("/Resources/Undo_01.png",
UriKind.Relative))
}
Dim redoRibbonButton As New RibbonButton() With {
.Label = "Redo",
.SmallIcon = New BitmapImage(New Uri("/Resources/Redo_01.png",
UriKind.Relative))
}
Dim newRibbonButton As New RibbonButton() With {
.Label = "New",
.SmallIcon = New BitmapImage(New Uri("/Resources/Document-01.png",
UriKind.Relative))
}
Me.Ribbon.QuickAccessToolBar.CustomQATItems.Add(undoRibbonButton)
Me.Ribbon.QuickAccessToolBar.CustomQATItems.Add(redoRibbonButton)
Me.Ribbon.QuickAccessToolBar.CustomQATItems.Add(newRibbonButton)
```

Once the [CustomQATItems](#) are added to the Quick Access Toolbar using the **Add** button, it will be visible in the Ribbon QAT and in the QAT Dropdown menu. The visibility of this item in the Ribbon QAT can be toggled by checking/unchecking from the QAT Dropdown menu.



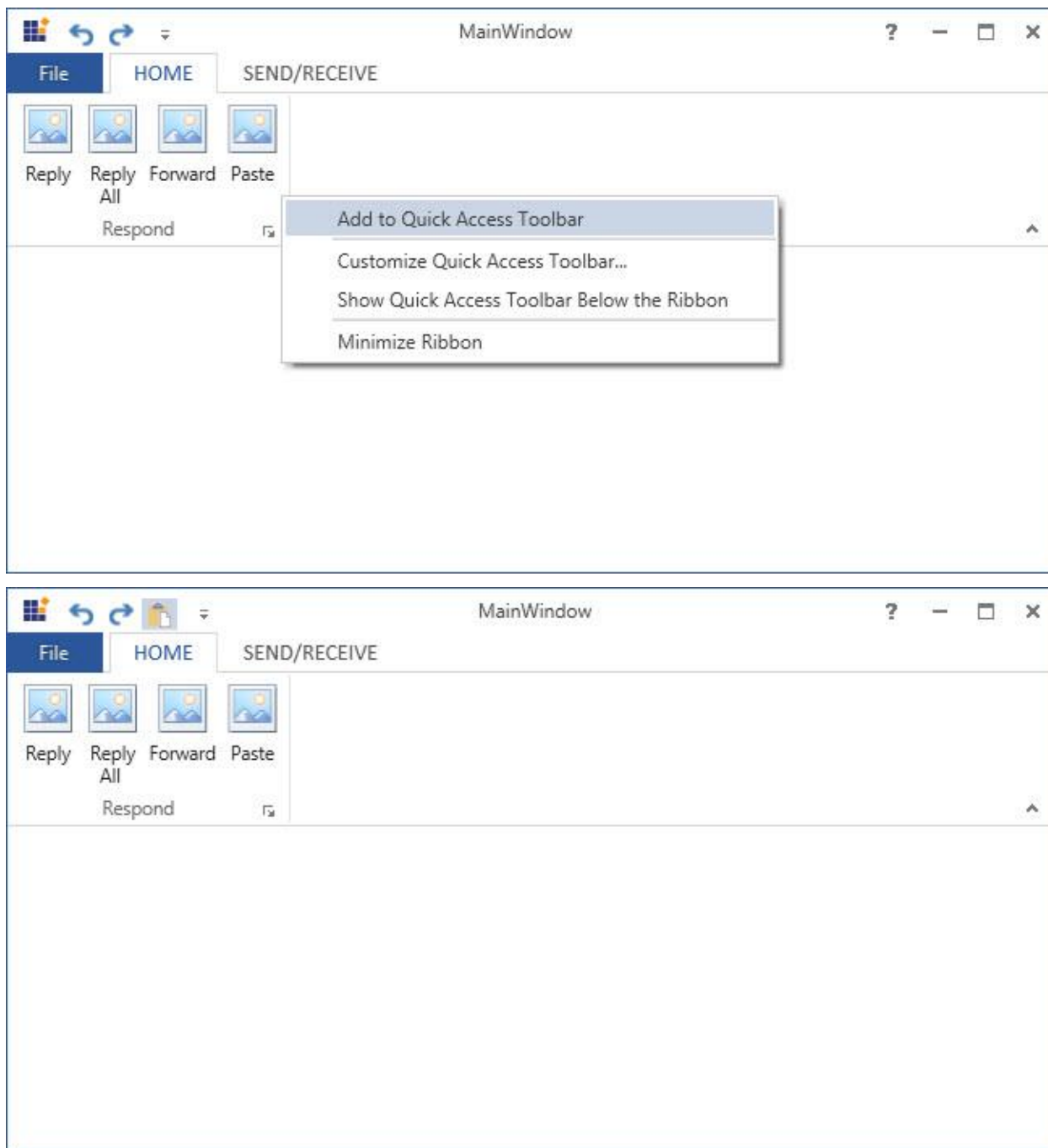


The added [CustomQATItems](#) in the QAT Dropdown menu can be removed through the QAT Window using the **Remove** button.



Add items from Ribbon context menu

QAT can also be customized by adding the items from the Ribbon ContextMenu. Select Add to Quick Access Toolbar by right-clicking the Ribbon item required to add to the QAT. Then, the respective item gets added as one of the items in the QAT.



Add custom items using RibbonItemHost

The control hosted in [RibbonItemHost](#) can be added to the [QuickAccessToolBar](#) using ribbon context menu or through QAT customization dialog. In the QAT customization dialog, all the ribbon items are displayed using its [Label](#) and [IconTemplate](#) properties. Similarly, the [RibbonItemHost](#) also allows us to set the label and icon using its [Label](#) and [IconTemplate](#) properties respectively.

- [Label](#) - Used to display the text of [RibbonItemHost](#) inside the QAT customization dialog.
- [IconTemplate](#) - Gets or sets the template that is used to display the icon of [RibbonItemHost](#) inside the QAT customization dialog.

In the below example, the [Label](#) and [IconTemplate](#) are set for the [RibbonItemHost](#) controls which will be used when the items are displayed in the QAT customization dialog.

XML

```
<syncfusion:RibbonWindow x:Class="Ribbon.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:Ribbon"
mc:Ignorable="d"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.Theme="{skin:SkinManagerExtension ThemeName=FluentLight}"
WindowStartupLocation="CenterScreen"
Title="Custom items demo" Height="450" Width="650">
<syncfusion:RibbonWindow.DataContext>
<local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
<syncfusion:RibbonWindow.Resources>
<DataTemplate x:Key="OnePage">
<Grid>
<Path
Width="13"
Height="16"
Margin="0.5"
Data="M0,0 L11,0 11,15 0,15 z"
Fill="White"
Stretch="Fill" />
<Path
Margin="1"
Data="M1,1 L1,15 11,15 11,1 z M0,0 L12,0 12,4.158 12,5.0689998 12,16 0,16 z"
Fill="#FF3A3A38"
Stretch="Fill" />
<Path
Margin="3"
Data="M0,8.9999991 L5.9999999,8.9999991 5.9999999,9.999999 0,9.999999 z
M3.2782552E-06,5.9999998 L6.0000033,5.9999998 6.0000033,6.9999996
3.2782552E-06,6.9999996 z M3.2782552E-06,2.9999995 L6.0000033,2.9999995
6.0000033,3.9999995 3.2782552E-06,3.9999995 z M3.4272668E-06,0 L6.0000033,0
6.0000033,0.99999952 3.4272668E-06,0.99999952 z"
Fill="#FF797774"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="AlignLeft">
<Path
x:Name="Align_Left1"
Width="14"
Height="13"
Margin="2,0"
HorizontalAlignment="Left"
VerticalAlignment="Bottom"
Data="M0,12 L10,12 10,13 0,13 z M0,8 L14,8 14,9 0,9 z M0,4 L10,4 10,5 0,5 z
M0,0 L14,0 14,1 0,1 z"
/></DataTemplate>
</syncfusion:RibbonWindow.Resources>
</syncfusion:RibbonWindow.DataContext>
</local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
</syncfusion:RibbonWindow>
```

```

Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
Stretch="Fill" />
</DataTemplate>
<DataTemplate x:Key="Tables">
<Grid MaxHeight="32" MaxWidth="32">
<Path
Margin="0.5,4.5,0.5,0.5"
Data="M0,0 L27,0 27,23 0,23 z"
Fill="White"
Stretch="Fill" />
<Path
Height="4"
Margin="0.5,0.5,0.5,0"
VerticalAlignment="Top"
Data="M0,0 L27,0 27,4 0,4 z"
Fill="#FFC8C6C4"
Stretch="Fill" />
<Path
Margin="1,5,1,1"
Data="M9,8 L9,14 17,14 17,8 z M8,0 L9,0 9,7 17,7 17,0 18,0 18,7 26,7 26,8
18,8 18,14 26,14 26,15 18,15 18,22 17,22 17,15 9,15 9,22 8,22 8,15 0,15 0,14
8,14&#xa;8,8 0,8 0,7 8,7 z"
Fill="#FF797774"
Stretch="Fill" />
<Path
Data="M0.99999994,5.0000001 L0.99999994,27 27,27 27,5.0000001 z
M0.99999994,1 L0.99999994,4.0000002 27,4.0000002 27,1 z M0,0 L28,0
28,4.0000002 28,5.0000001 28,28 0,28 0,5.0000001 0,4.0000002 z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:RibbonWindow.Resources>
<Grid>
<syncfusion:Ribbon x:Name="mainRibbon">
<syncfusion:RibbonTab Caption="Design">
<syncfusion:RibbonBar Header="Table Style Options"
IconTemplate="{StaticResource Tables}">
<syncfusion:RibbonItemHost Label="Header Row" IconTemplate="{StaticResource
OnePage}">
<syncfusion:RibbonItemHost.ContentTemplate>
<DataTemplate>
<CheckBox Content="Header Row" Height="22"/>
</DataTemplate>
</syncfusion:RibbonItemHost.ContentTemplate>
</syncfusion:RibbonItemHost>
<syncfusion:RibbonItemHost Label="First Column"
IconTemplate="{StaticResource AlignLeft}">
<syncfusion:RibbonItemHost.ContentTemplate>
<DataTemplate>
<RadioButton Margin="4,0,0,0" Content="First Column" Height="22"/>
</DataTemplate>
</syncfusion:RibbonItemHost.ContentTemplate>
</syncfusion:RibbonItemHost>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>

```

```

<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar/>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage>
<syncfusion:BackStageCommandButton Header="Save" >
<syncfusion:BackStageCommandButton.IconTemplate>
<DataTemplate>
<Path Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M5.0000019,11 L5.0000019,15 11.000002,15 11.000002,11 z M4.0000019,1
L4.0000019,6 12.000002,6 12.000002,1 z M1,1 L1,13.174 2.7160001,15
4.0000019,15 4.0000019,10 12.000002,10 12.000002,15 15,15 15,1 13.000002,1
13.000002,7 3.0000019,7 3.0000019,1 z M0,0 L3.0000019,0 13.000002,0 16,0
16,16 12.000002,16 4.0000019,16 2.2840004,16 0,13.57 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" Stretch="Uniform" />
</DataTemplate>
</syncfusion:BackStageCommandButton.IconTemplate>
</syncfusion:BackStageCommandButton>
<syncfusion:BackStageCommandButton Header="Close" >
<syncfusion:BackStageCommandButton.IconTemplate>
<DataTemplate>
<Grid Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center" SnapsToDevicePixels="true">
<Path
Width="12" Height="12" HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M1.4139423,0L7.0029922,5.5845888 12.592018,0 14.006015,1.4149939
8.4180527,6.9985202 14.006,12.582007 12.591996,13.997001 7.0030056,8.4124444
1.4140122,13.997001 1.5026823E-05,12.582007 5.5879484,6.9985092 0,1.4149939z
"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
SnapsToDevicePixels="True" Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:BackStageCommandButton.IconTemplate>
</syncfusion:BackStageCommandButton>
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

![WPF Ribbon Custom Items in MoreCommands Window](RibbonCustomItem_images/wpff-ribbon-custom-items-in-morecommands.png)

![Adding WPF Ribbon Custom Items in RibbonBar](RibbonCustomItem_images/wpff-ribbon-custom-items-with-ribbonbar.png)

To know more about the [RibbonItemHost](#), refer [here](#).

Add custom RibbonTab and RibbonBar

The following section illustrates how to customize Ribbon at the run time

QAT customized window

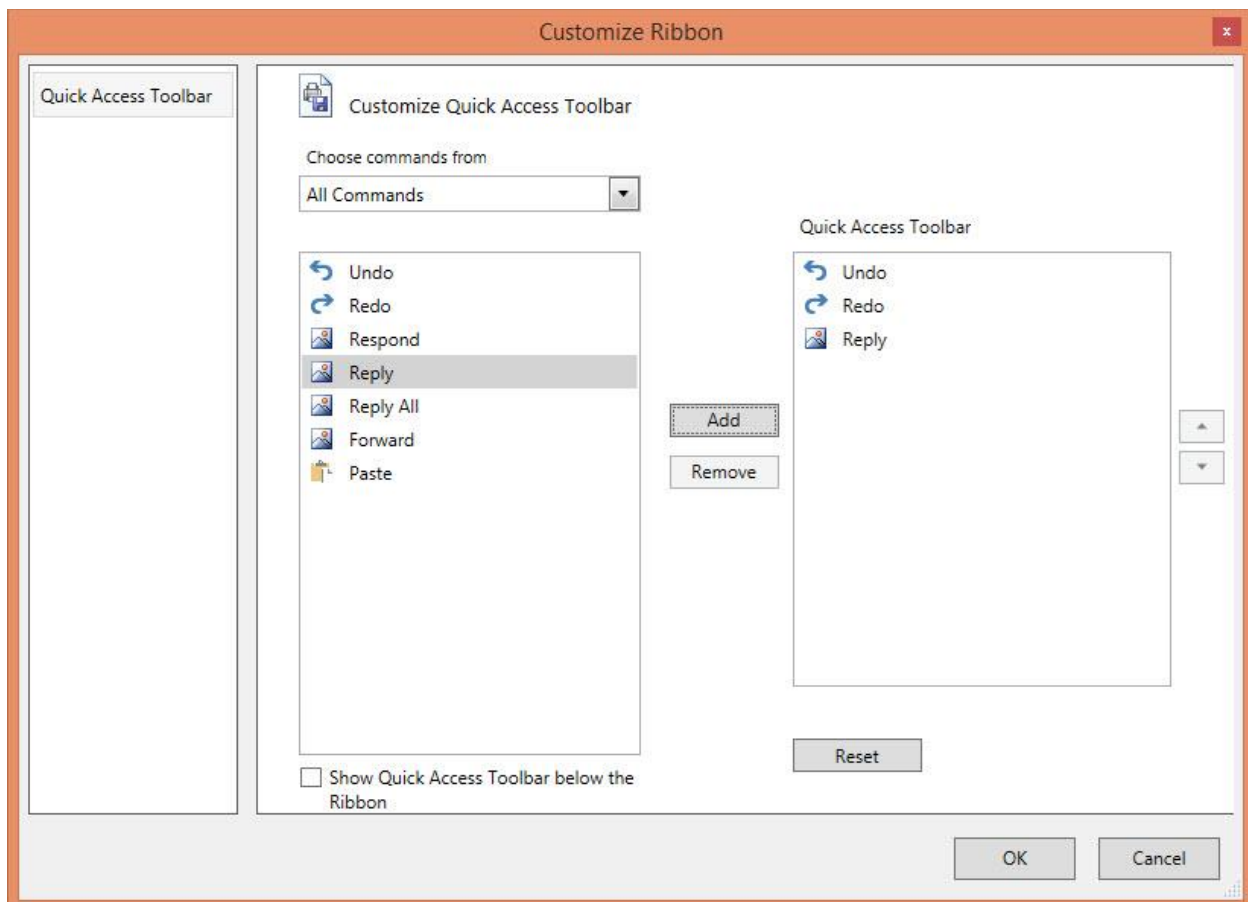
This topic illustrates in detail about the QAT topic.

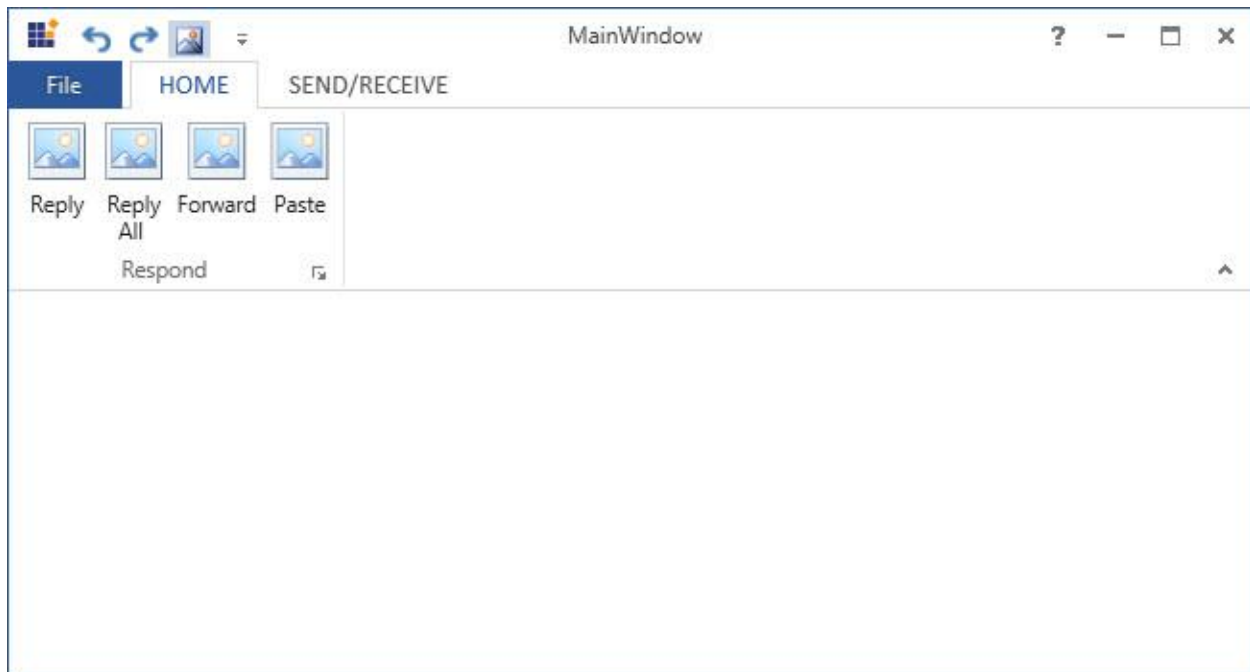
First, select MoreCommands option from the QAT to open the QAT Customized Window.

QAT can be Customized further by using the following options

1. Add

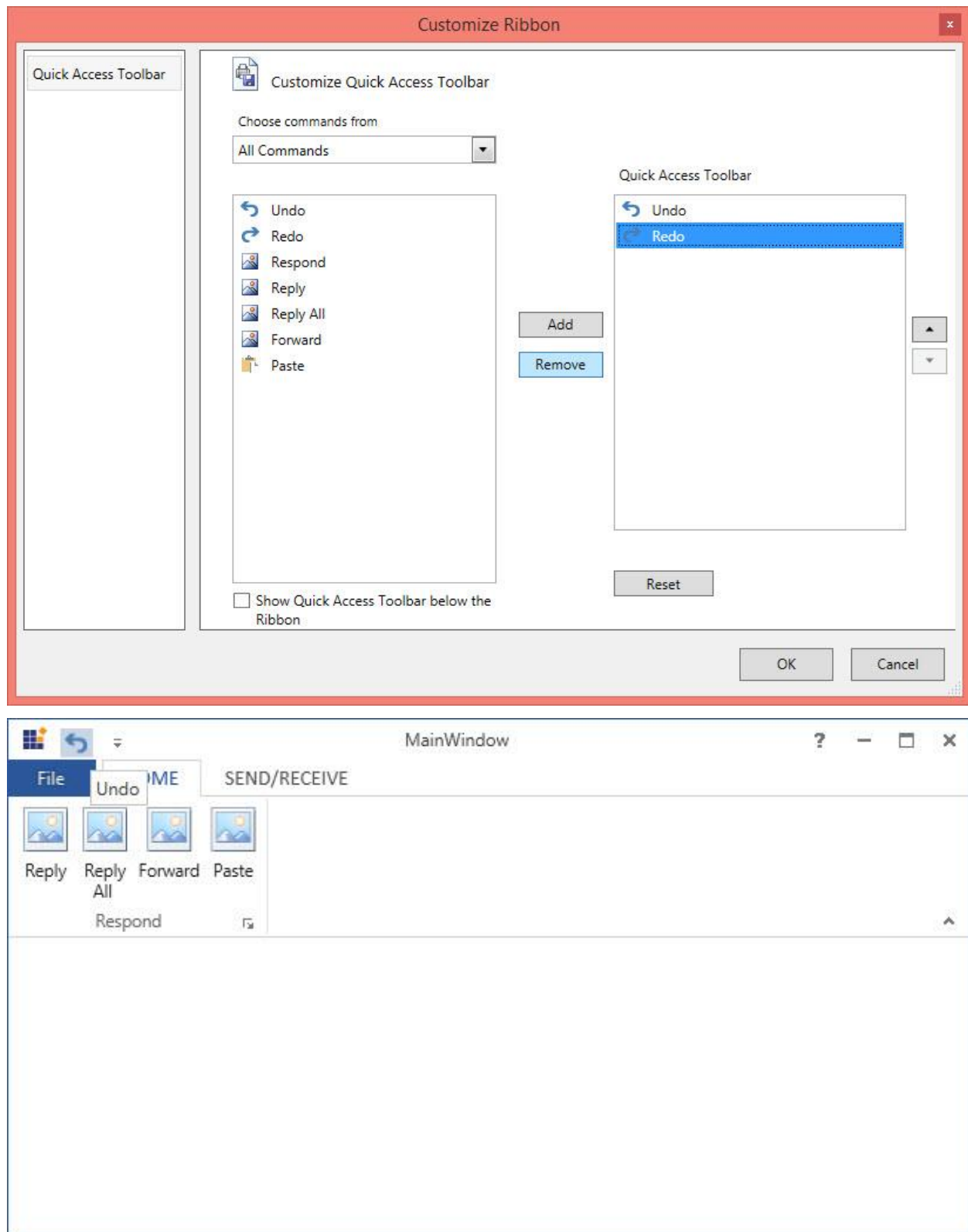
In the QAT Customized window, the list of Commands is available. The Commands can be filtered only from the Particular tab by using **Choose commands from** option. Then, select the commands to be added to the QAT and add the command to Right Pane of the QAT Dialog by clicking **Add** Button. Finally, Click OK. Now, the selected command gets added in the QAT.





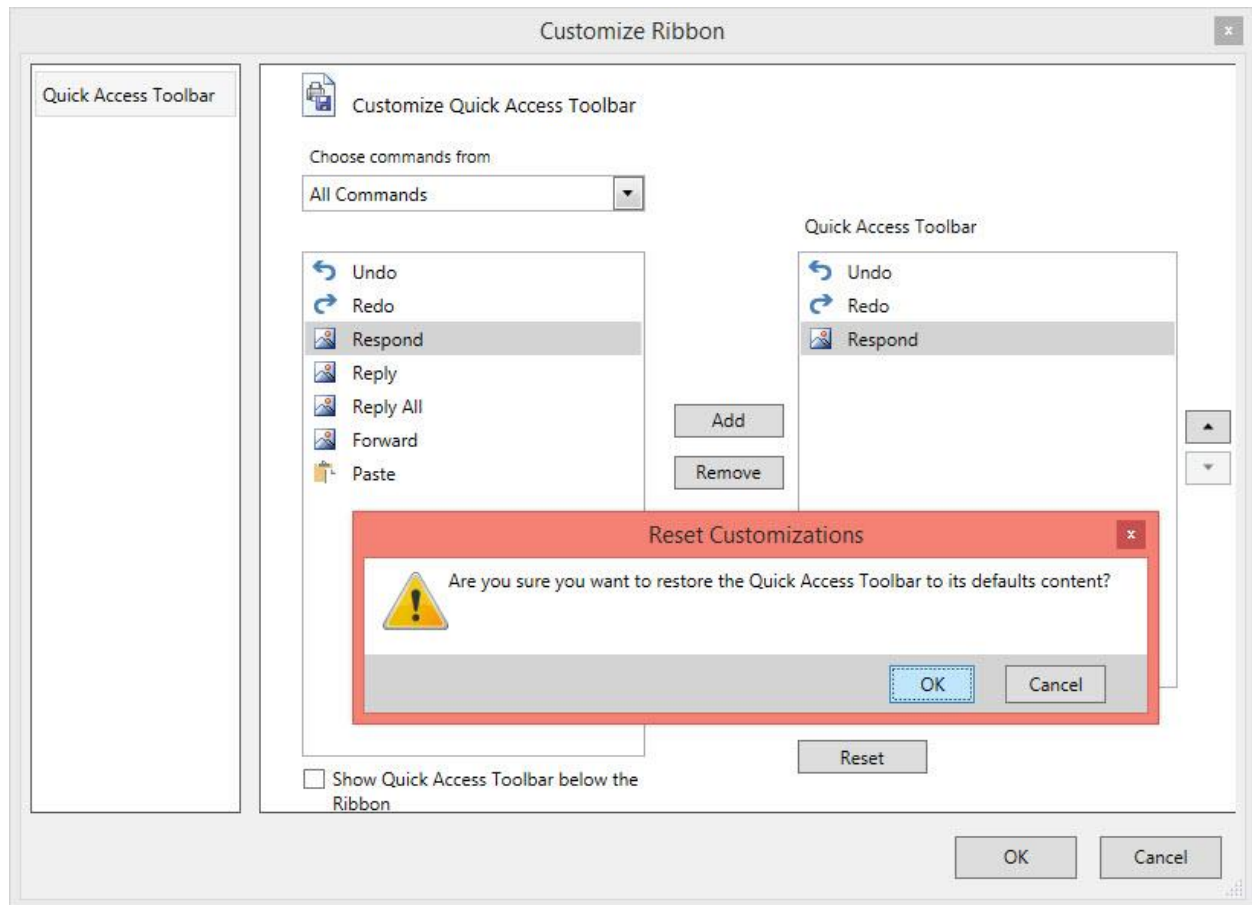
2. Remove

QAT items can be removed at the Run time. In the QAT Customize window, select the Command to be deleted from the QAT Dialog, and then click Remove Button. In the following screenshot, Redo command gets deleted and the output is changed according to it.



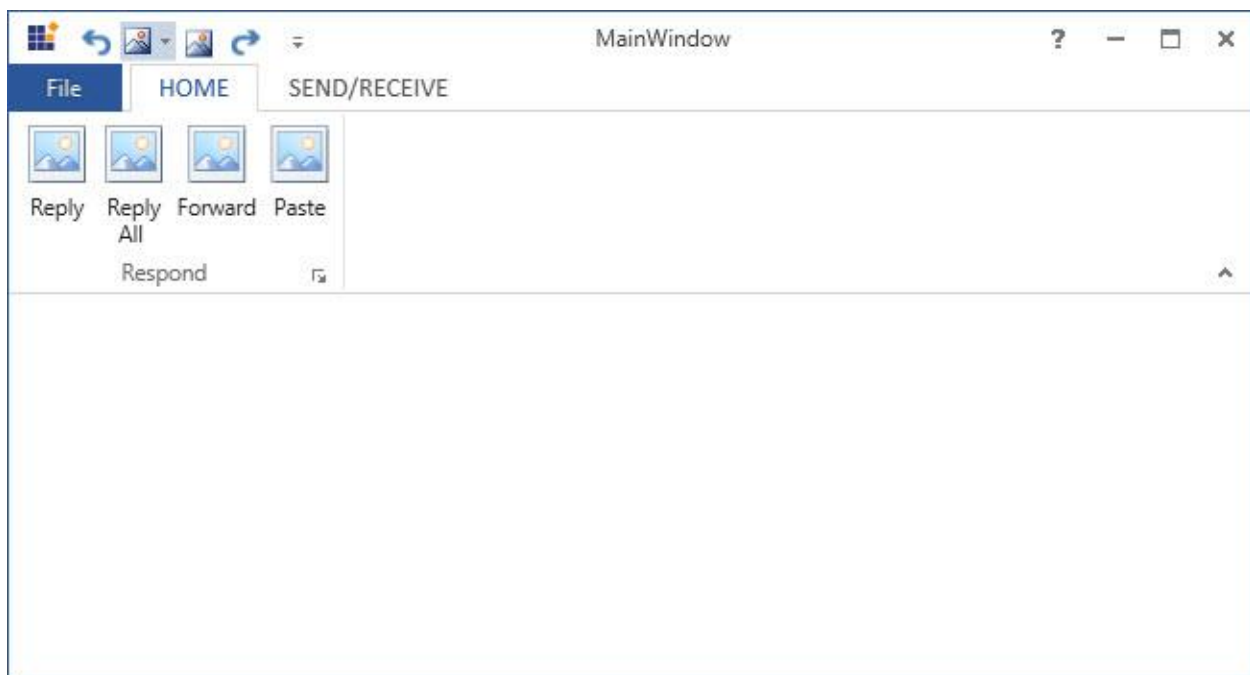
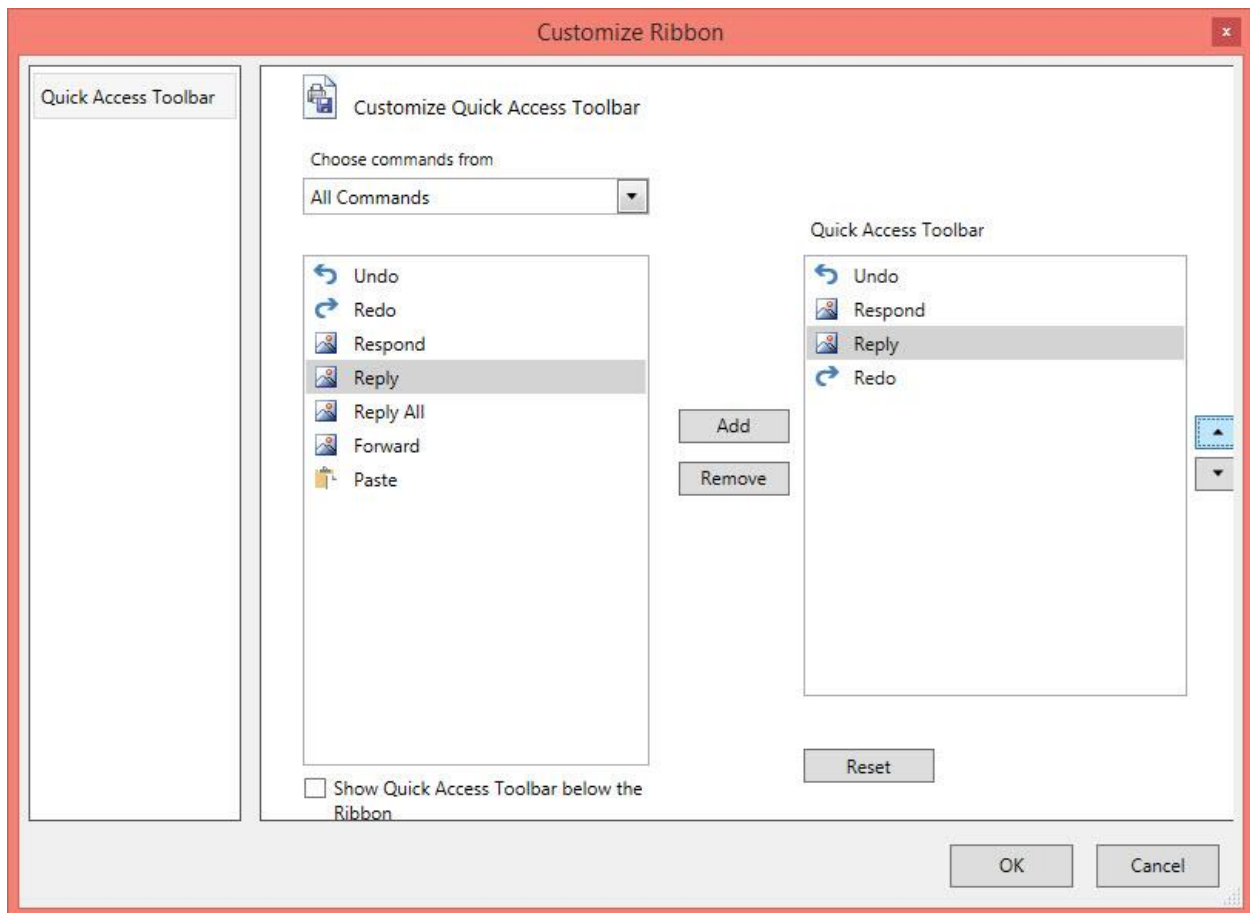
3. Reset

To restore the QAT to its default content, use this **Reset** option.



4. Reorder

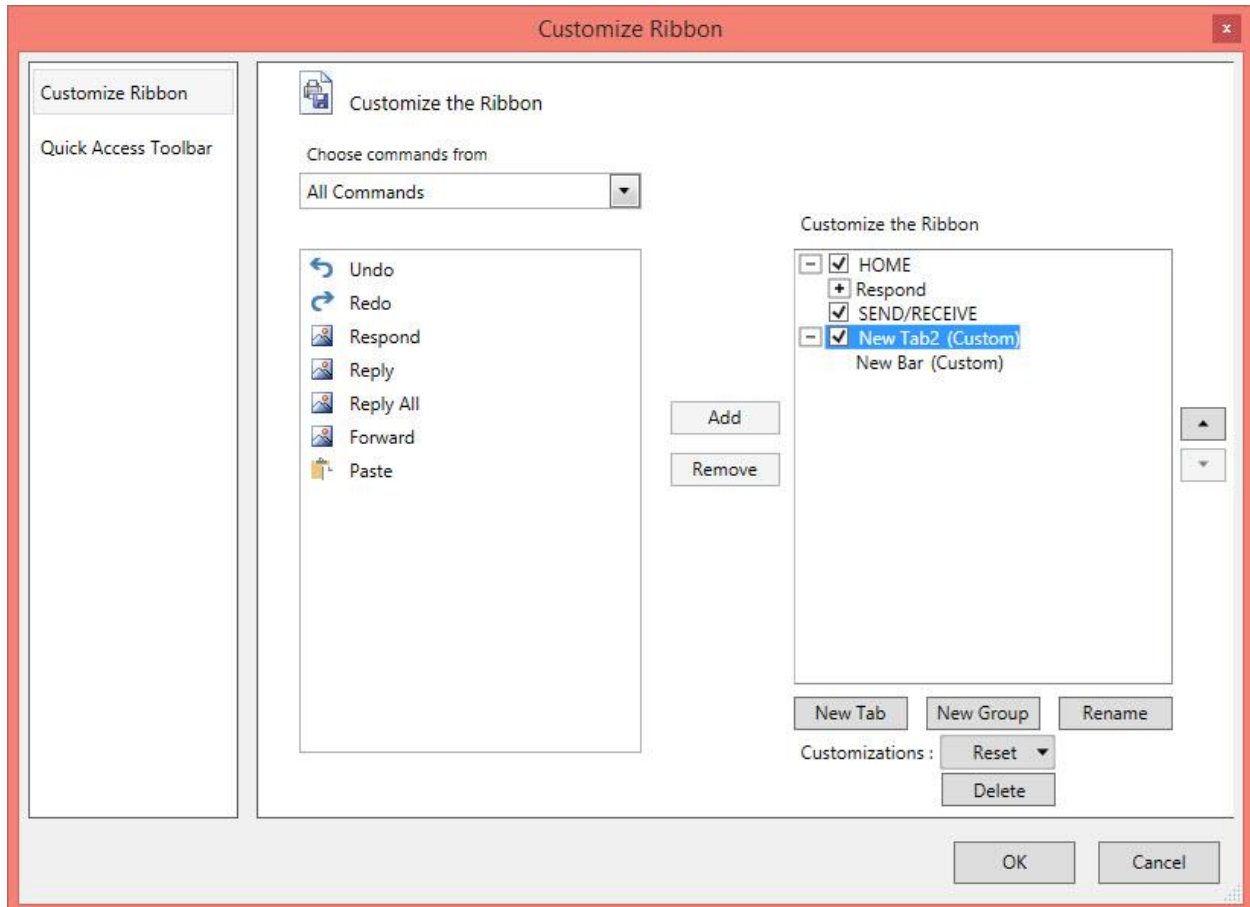
QAT also supports to reorder the items by clicking the Up and down arrow at the left of the QAT Dialog.



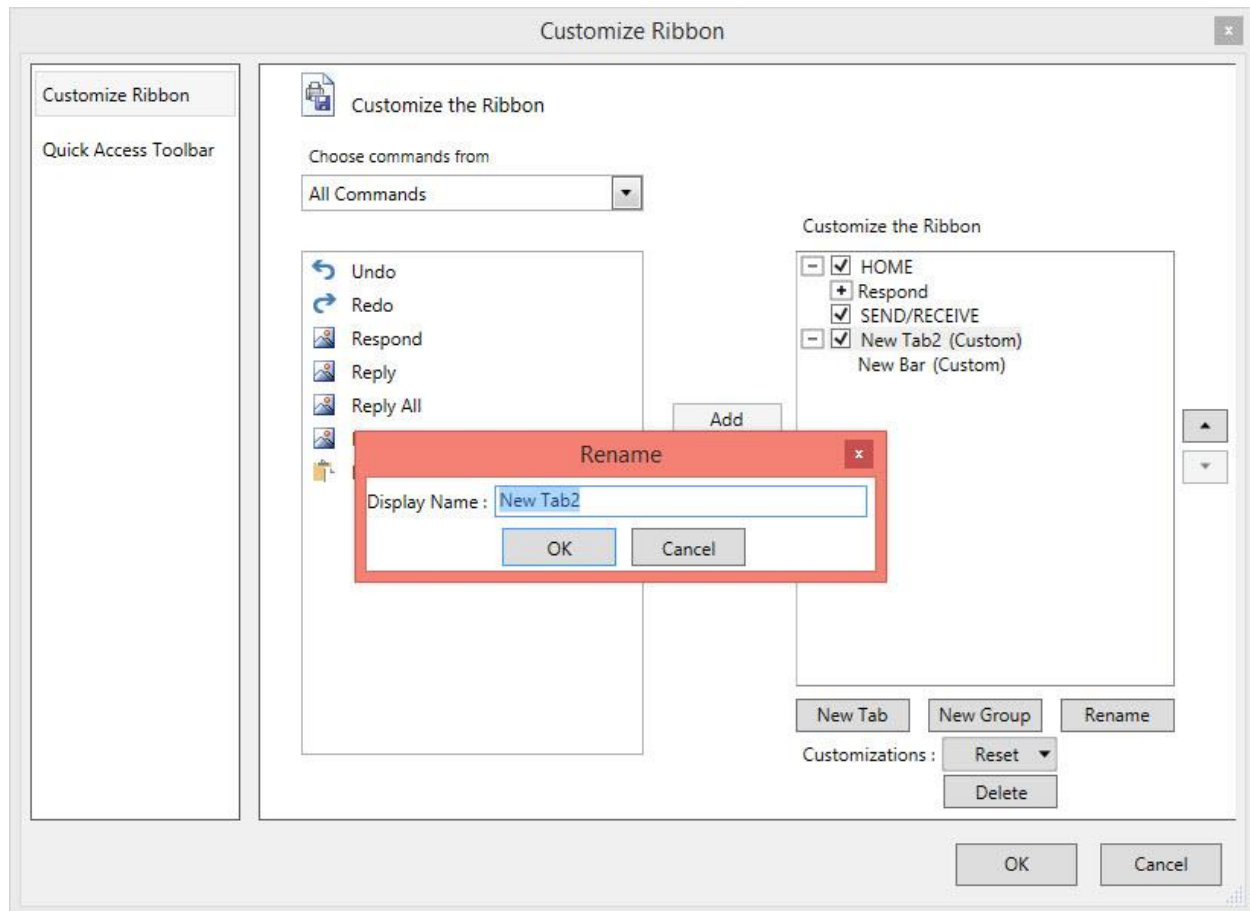
Customize Ribbon with RibbonTab and RibbonBar

QAT can also support to customize Ribbon Tab and RibbonBar while running. To customize the Ribbon, enable the `ShowCustomizeRibbon` property of the Ribbon to `true` and follow the steps.

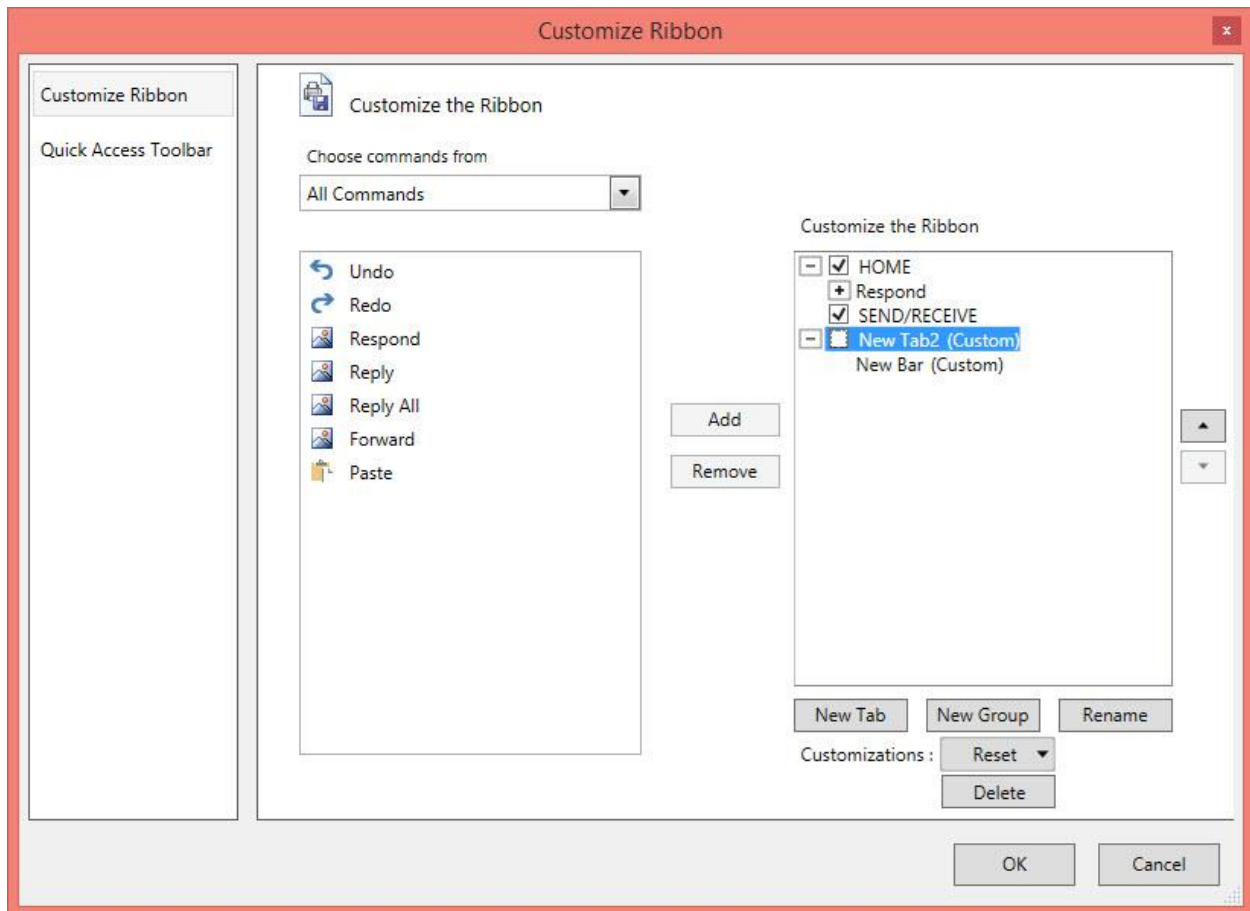
1. Click the Quick Access Toolbar drop-down button and click the **MoreCommands** menu.
2. Then, select the **Customize Ribbon** options from the left-side column.
3. You can add **RibbonTab** and **RibbonBar** by selecting New Tab and New Group Options Respectively.



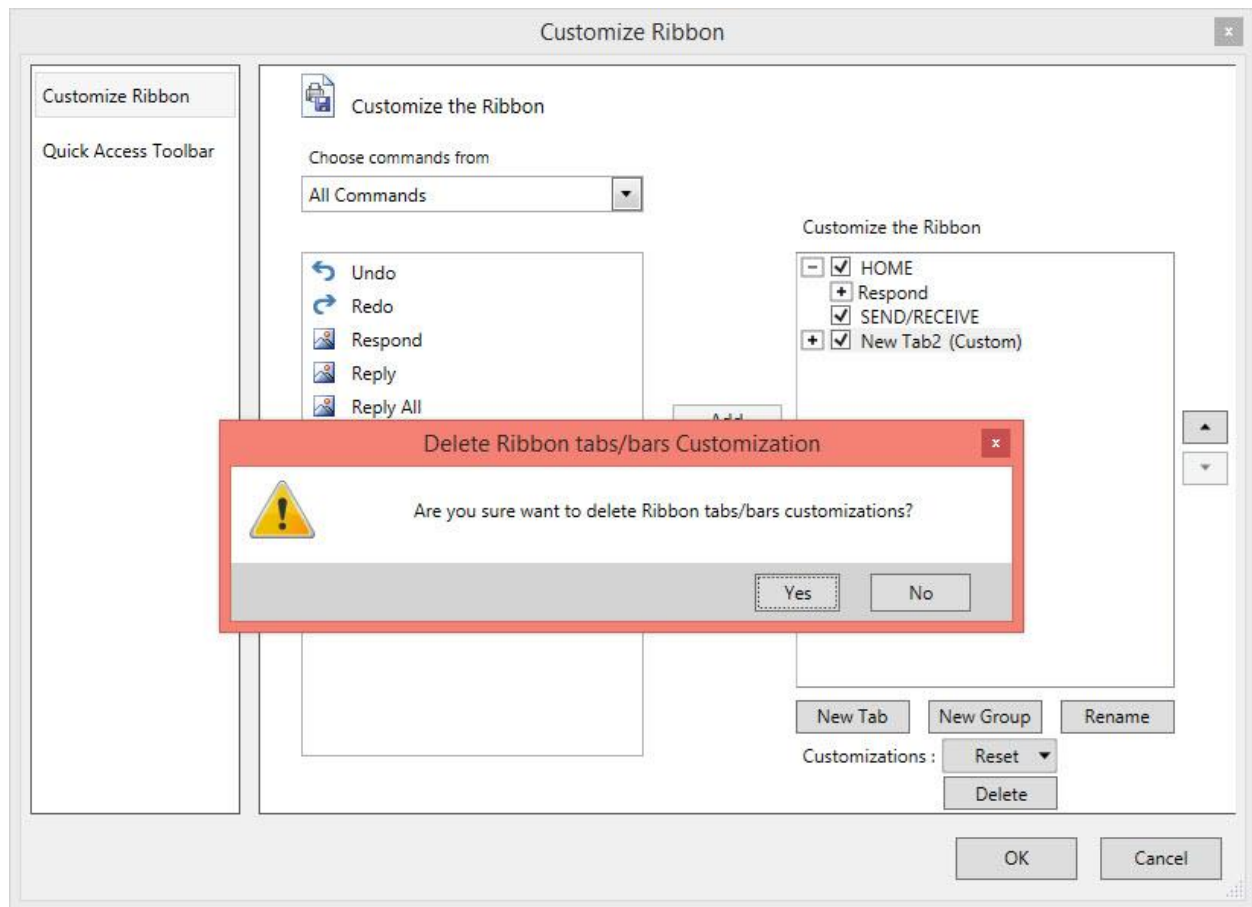
4. To move the tab up and down, select the tab to be moved and click the button at the right-side corner accordingly. Similarly, the group can be moved within the tab.
5. By using the **Rename** option, rename any particular Tab/Group present in the Ribbon by selecting it.



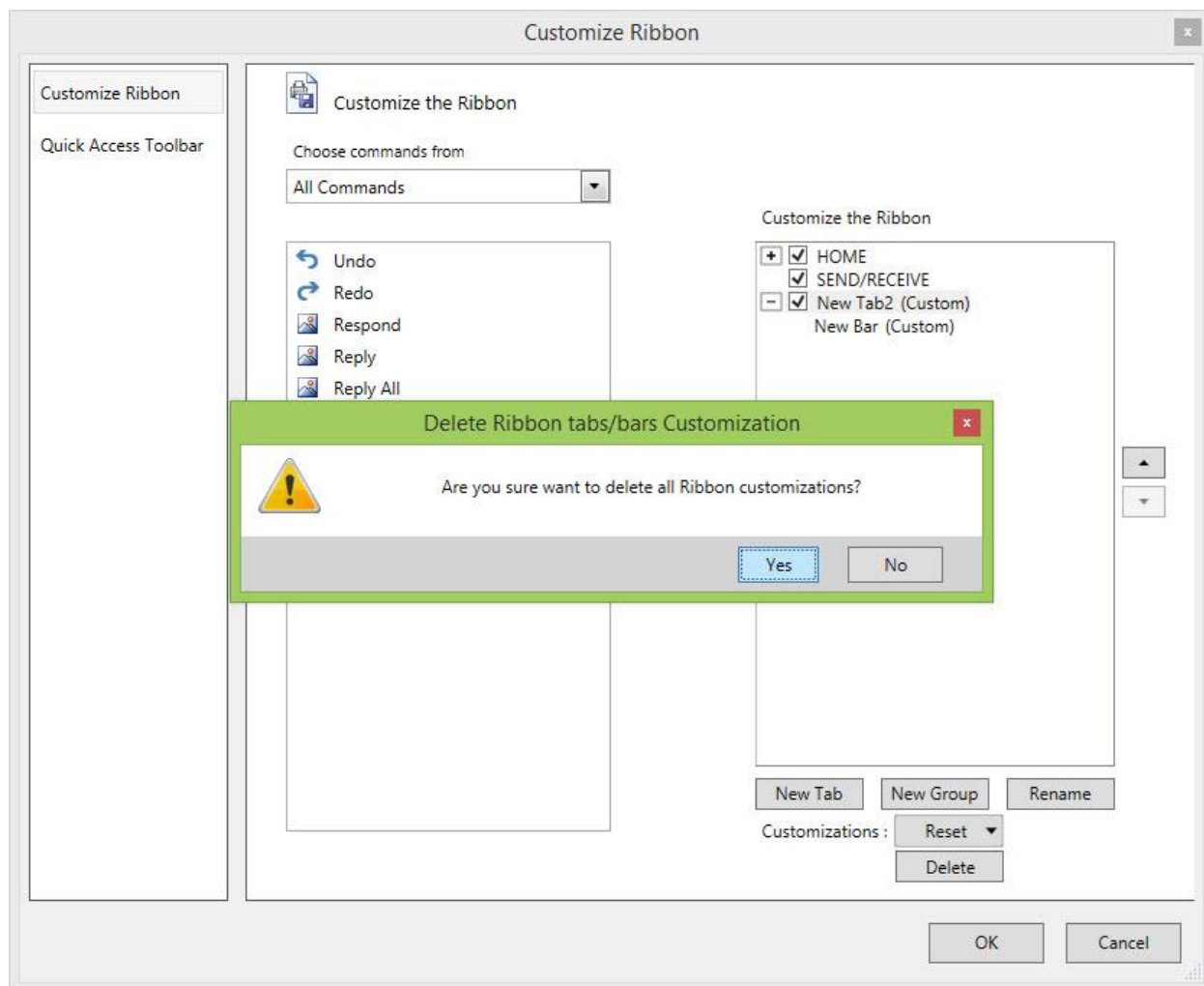
6. The visibility of any particular Tab can be changed by enabling the checkbox of the corresponding selection.



7. By using **Delete** option, the Custom RibbonBar and RibbonTab can be deleted. But, this option does not work in the case of original content.



8. To delete all the Ribbon Customization, use **Reset All Customization** from the Reset Drop Down Menu.

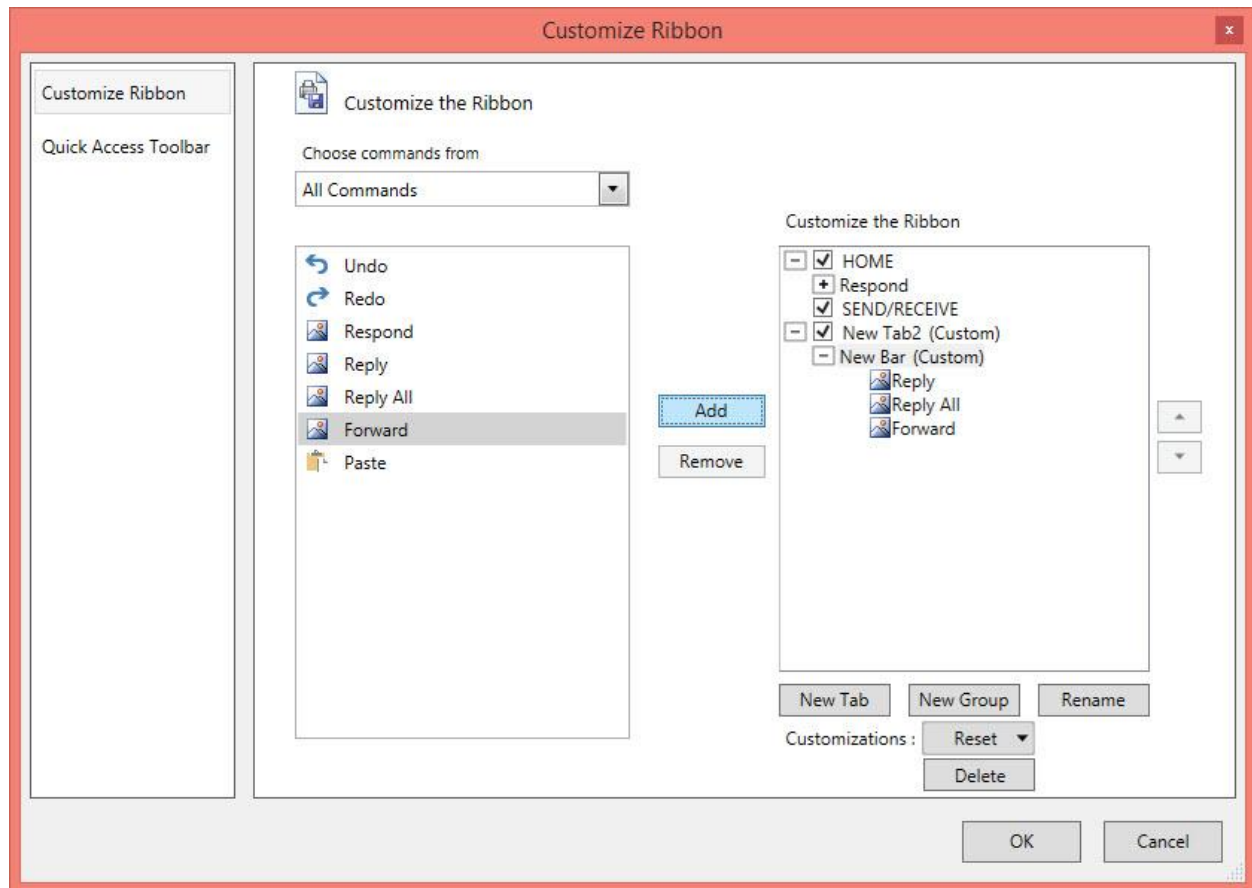


After customizing the RibbonTab and RibbonBar, RibbonItems can be customized. Customizing Ribbon Item concepts are explained as follows.

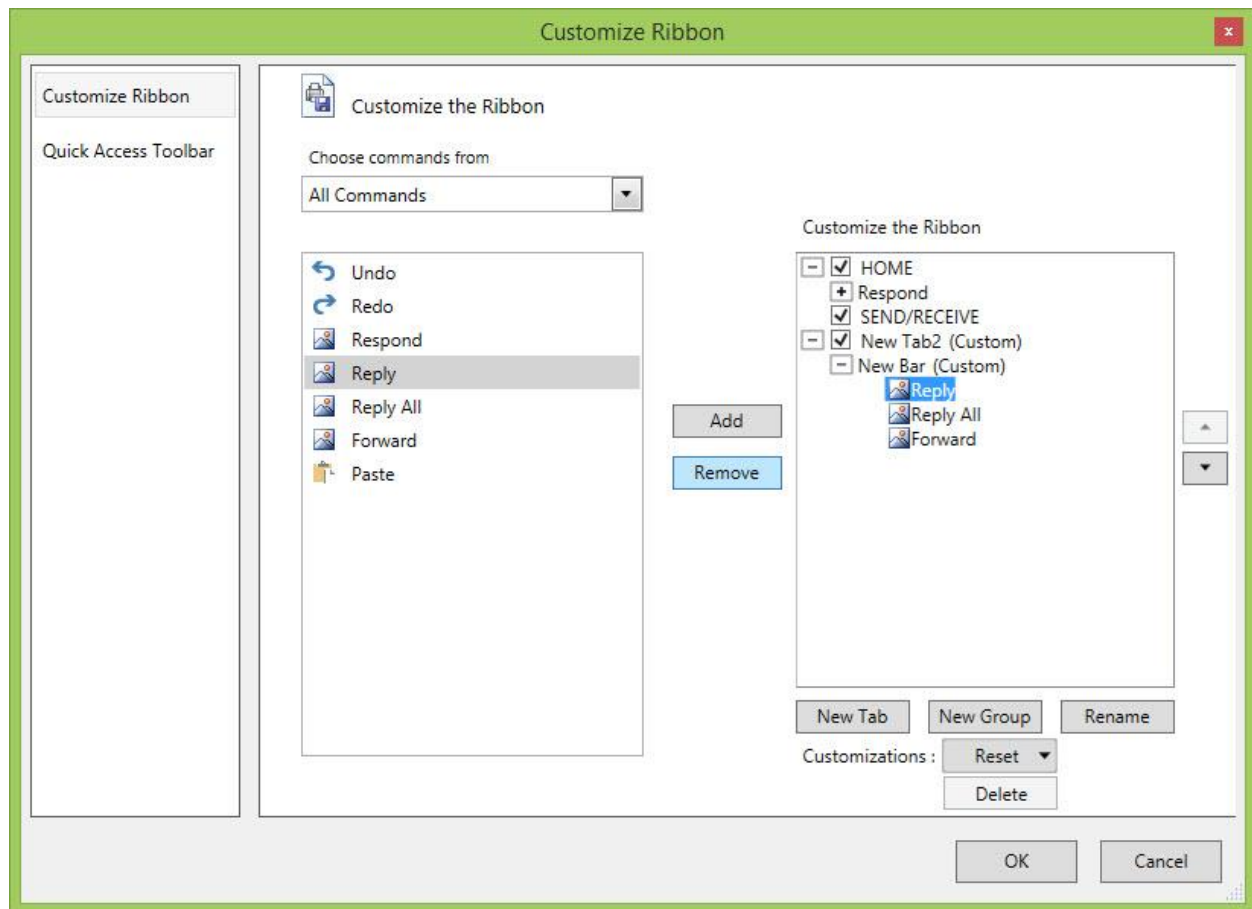
[Add items to the customized RibbonTab](#)

The steps to customize the Ribbon Item are as follows

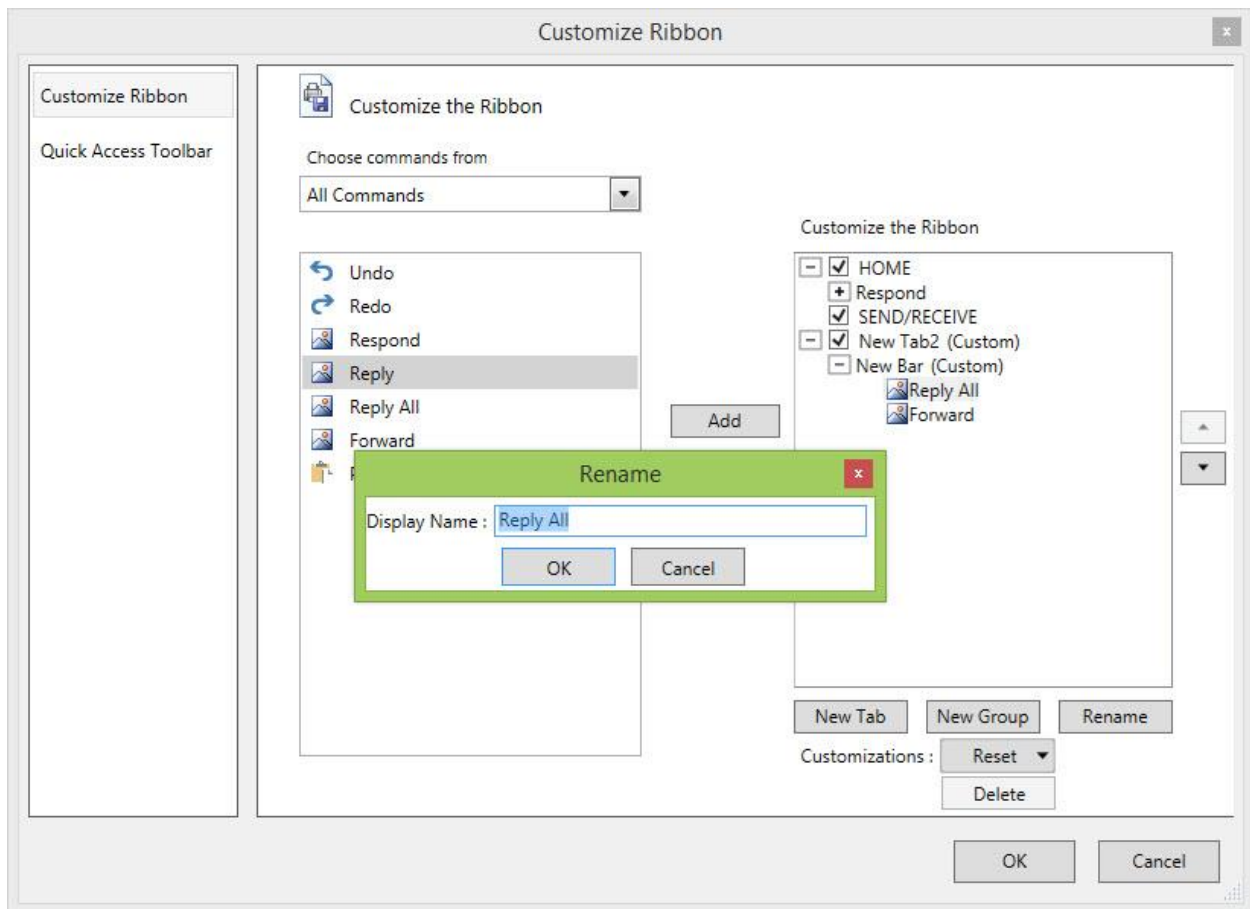
1. The **Choose commands from** drop-down list is used to filter the commands only from the Particular Tab.
2. To add the item under the right-side column of a newly added **RibbonBar** and **RibbonTab**, select the item from left-side column and click the **Add** button.



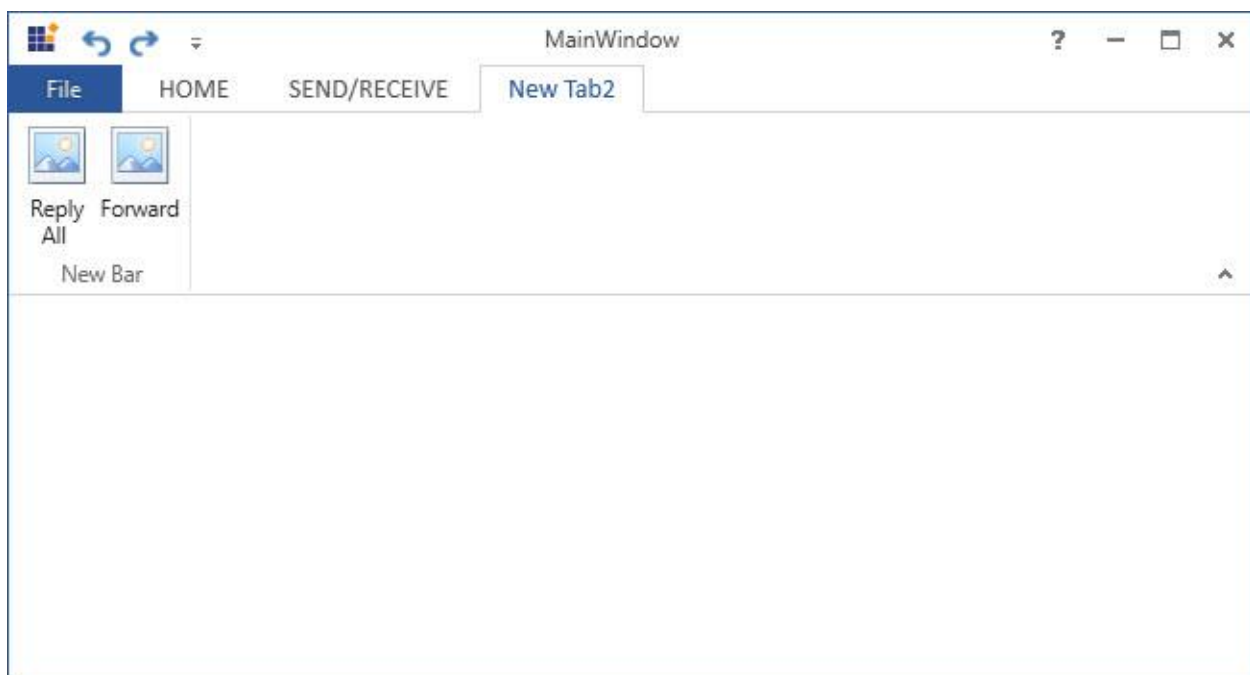
3. Also, the Ribbon Item from the RibbonBar can be removed that is added recently by using **Remove** option



4. The order of the RibbonItem within the RibbonBar can be changed by using **Up** and **Down** arrow
5. By using the Rename option, the RibbonItem can be renamed



6. Click **OK** button. The Changes are reflected in the output window.



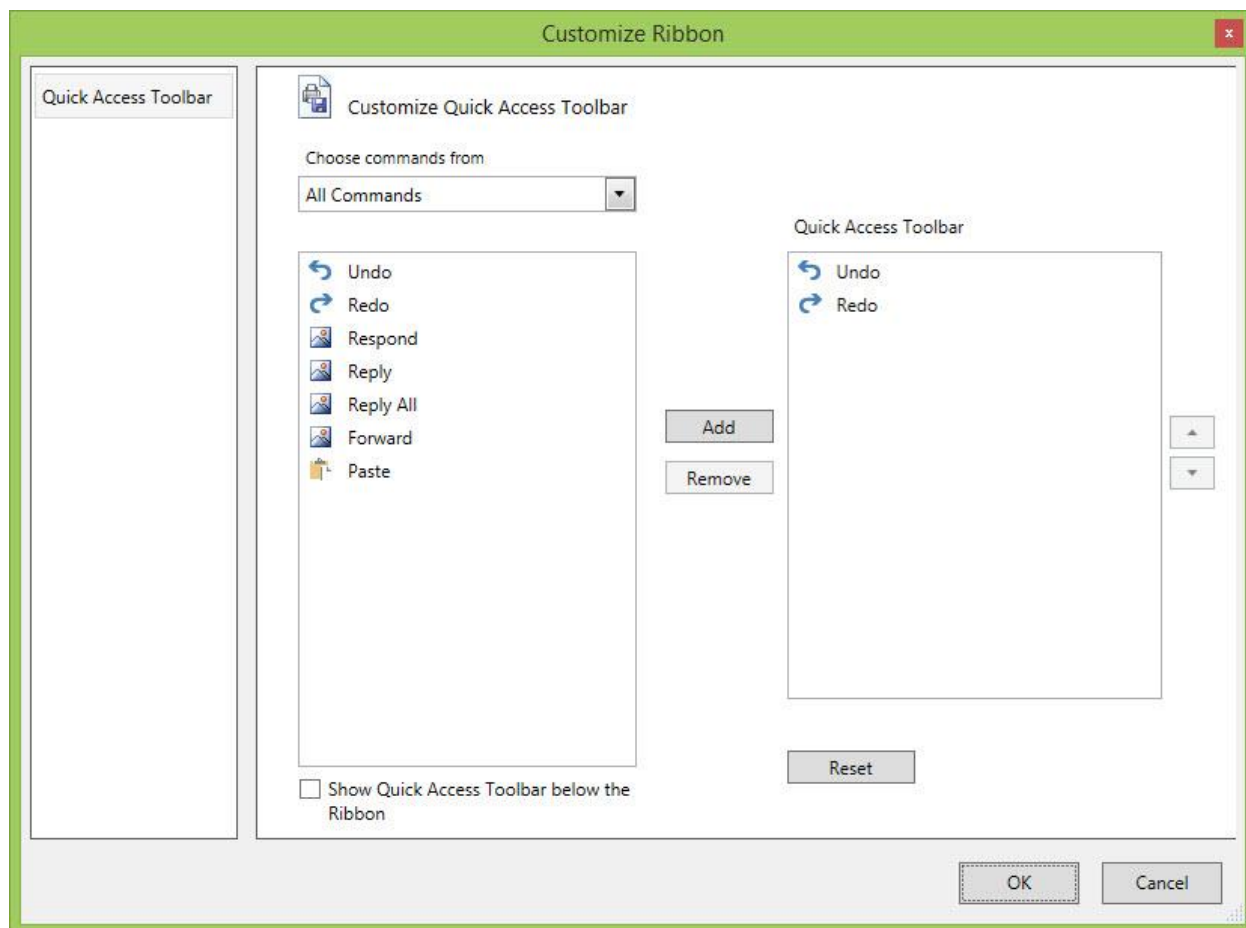
How to disable the customization in Ribbon

To disable the customization in the Ribbon, set `ShowCustomizeRibbon` property of the Ribbon as `False`

XML

```
<syncfusion:Ribbon VerticalAlignment="Top" x:Name="Ribbon"
ShowCustomizeRibbon="False">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar >
<syncfusion:RibbonButton Label="Undo" SmallIcon="/Resources/Undo16.png"
SizeForm="ExtraSmall"
ToolTip="Undo" syncfusion:RibbonCommandManager.SynchronizedItem="Undo" />
<syncfusion:RibbonButton Label="Redo" SmallIcon="/Resources/Redo16.png"
SizeForm="ExtraSmall"
ToolTip="Redo" syncfusion:RibbonCommandManager.SynchronizedItem="Redo" />
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="HOME" >
<syncfusion:RibbonBar Header="Respond">
<syncfusion:RibbonButton Label="Reply" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Reply All" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Forward" SizeForm="Large"/>
<syncfusion:RibbonButton Label="Paste" SizeForm="Large"
SmallIcon="/Resources/Paste32.png"
syncfusion:RibbonCommandManager.SynchronizedItem="Paste" />
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="SEND/RECIEVE"/>
</syncfusion:Ribbon>
```

In the following screenshot, customizing Ribbon is disabled and it shows only the QAT Customize window



Simplified Layout in WPF Ribbon

The Ribbon is available in simplified layout which is designed to display the most commonly used Ribbon commands in a single line interface, allowing more screen space for compact viewing of the content. For the best user experience, the other Ribbon commands are located under the overflow menu. It also provides option to switch back and forth between the simplified and the normal layout using the minimize button.

The [LayoutMode](#) enumeration property provides an option to load the Ribbon control in simplified layout. It contains the following options like:

- **Normal** - The Ribbon items are arranged in the standard layout. This is the default value.
- **Simplified** - The Ribbon items are arranged in the simplified layout.

XML

```
<Grid>
<syncfusion:Ribbon x:Name="ribbon" LayoutMode="Simplified"
VerticalAlignment="Top">
</syncfusion:Ribbon>
</Grid>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.LayoutMode = LayoutMode.Simplified;
grid.Children.Add(ribbon);
```

Switching between simplified and normal layouts

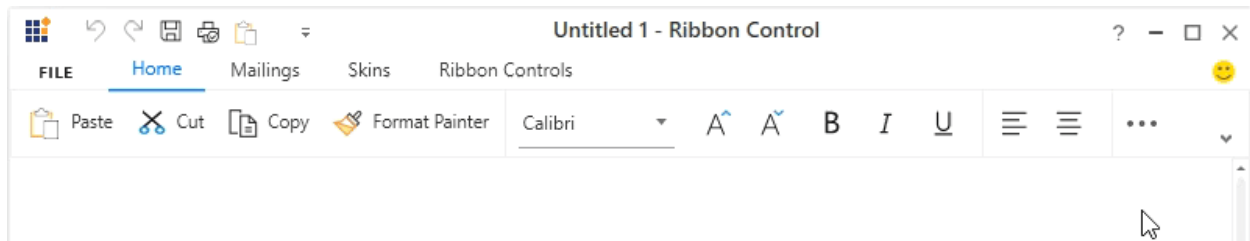
The Ribbon control allows to switch between simplified and normal layouts at runtime using the Ribbon minimize button located in the lower right corner of the Ribbon. To enable this option, set the [EnableSimplifiedLayoutMode](#) property to **True**. By default, its value is **False**.

XML

```
<Grid>
<syncfusion:Ribbon x:Name="ribbon" EnableSimplifiedLayoutMode="True"
VerticalAlignment="Top">
</syncfusion:Ribbon>
</Grid>
```

C#

```
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
grid.Children.Add(ribbon);
```



Visibility of the Ribbon items between normal and simplified layout

The Ribbon items can be set common between different layouts or can be made visible only in a particular layout using the [SimplifiedLayoutSettings.DisplayMode](#) attached property. By default, items will be displayed in both normal and simplified layout. The [DisplayMode](#) is of flag enumeration type that contains the following values.

- **Normal** - The item will be displayed only in the normal layout.
- **Simplified** - The item will be displayed only in the simplified layout.
- **OverflowMenu** - The item will be displayed only inside the overflow menu when simplified layout is enabled.

Also, the [DisplayMode](#) property allows the following value combinations as well.

- **Normal, Simplified** – The item will be displayed in both normal and simplified layout.
- **Normal, OverflowMenu** – The item will be displayed in both normal layout and inside overflow menu during simplified layout.
- **Simplified, OverflowMenu** – The item will be displayed in simplified layout.

- **Normal, Simplified, OverflowMenu** – The item will be displayed in both normal and simplified layout.

In the below code snippet, the [SimplifiedLayoutSettings.DisplayMode](#) property for *Paste* option is set to **“Simplified”**, so it will only be displayed only in the simplified layout. The [SimplifiedLayoutSettings.DisplayMode](#) property for *Underline* option is set to **“Normal, Overflow”**, so it will be displayed in the normal layout and will also be displayed inside the overflow menu in the simplified layout.

XML

```
<!-- This item will only be displayed in simplified layout -->
<syncfusion:RibbonButton Label="Paste"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Simplified" />
<!-- This item will only be displayed in normal layout -->
<syncfusion:RibbonButton
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
SizeForm="ExtraSmall" />
<!-- This item will be displayed in normal layout and inside overflow menu
during simplified layout -->
<syncfusion:RibbonButton Label="Underline"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,OverflowMenu"
SizeForm="ExtraSmall"/>
<!-- This item will be displayed both in normal and simplified layout -->
<syncfusion:RibbonButton Label="Italic" SizeForm="ExtraSmall"/>
```

C#

```
// This item will only be displayed in simplified layout
RibbonButton pasteButton = new RibbonButton();
pasteButton.Label = "Paste";
SimplifiedLayoutSettings.SetDisplayMode(pasteButton,
DisplayMode.Simplified);
// This item will only be displayed in normal layout
RibbonButton boldButton = new RibbonButton();
boldButton.Label = "Bold";
boldButton.SizeForm = SizeForm.ExtraSmall;
SimplifiedLayoutSettings.SetDisplayMode(boldButton, DisplayMode.Normal);
// This item will be displayed in normal layout and inside overflow menu
during simplified layout
RibbonButton underlineButton = new RibbonButton();
underlineButton.Label = "Underline";
SimplifiedLayoutSettings.SetDisplayMode(underlineButton, DisplayMode.Normal
| DisplayMode.OverflowMenu);
// This item will be displayed both in normal and simplified layout
RibbonButton italicButton = new RibbonButton();
italicButton.Label = "Paste";
```

When using simplified layout, the **Margin**, **Width** and **Height** values of the Ribbon items can be ignored as they are changed based on the size form and layout mode. If the item to be shown in both normal and simplified layout, the **Margin**, **Width** and **Height** properties can be set for normal layout alone using triggers.

XML

```

<syncfusion:RibbonButton Label="Copy"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal,Simplified"
SizeForm="Small" MediumIcon="/Resources/Copy_20.png"
SmallIcon="/Resources/Copy16.png" >
<syncfusion:RibbonButton.Style >
<Style TargetType="syncfusion:RibbonButton" BasedOn="{StaticResource
SyncfusionRibbonButtonStyle}">
<Style.Triggers>
<Trigger Property="syncfusion:SimplifiedLayoutSettings.LayoutMode"
Value="Normal">
<Setter Property="Height" Value="48"/>
<Setter Property="Width" Value="48"/>
<Setter Property="Margin" Value="2"/>
</Trigger>
</Style.Triggers>
</Style>
</syncfusion:RibbonButton.Style>
</syncfusion:RibbonButton>

```

Setting image for Ribbon items

For “**Normal**” layout mode, the images from the [SmallIcon](#) and [LargeIcon](#) properties are used inside the Ribbon items based on the size mode (extra small, small and large). However, the simplified layout mode uses 20 * 20 image size for the Ribbon items as standard and it can be obtained from the [MediumIcon](#) property. In-case if the [IconTemplate](#) property is used to display the image, the simplified layout will automatically resize it to 20 * 20 size.

XML

```

<syncfusion:RibbonButton Label="Paste"
syncfusion:SimplifiedLayoutSettings.DisplayMode="Simplified"
MediumIcon="/Resources/Paste20.png" />
<syncfusion:RibbonButton Label="Italic" SizeForm="ExtraSmall">
<syncfusion:RibbonButton.IconTemplate >
<DataTemplate>
<Path Margin="4,2,2,1"
Data="M2.000005,0 L6.000005,0 6.000005,1 4.4186966,1 2.4888427,8.9999952
4,8.9999952 4,9.9999952 0,9.9999952 0,8.9999952 1.4594386,8.9999952
3.3901918,1 2.000005,1 z"
Fill= "Black" />
</DataTemplate>
</syncfusion:RibbonButton.IconTemplate>
</syncfusion:RibbonButton>

```

C#

```

RibbonButton pasteButton = new RibbonButton();
pasteButton.Label = "Paste";
pasteButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Paste20.png",
UriKind.RelativeOrAbsolute));
SimplifiedLayoutSettings.SetDisplayMode(pasteButton,
DisplayMode.Simplified);
RibbonButton italicButton = new RibbonButton();
italicButton.Label = "Italic";
italicButton.SizeForm = SizeForm.ExtraSmall;
DataTemplate iconDataTemplate = new DataTemplate();

```

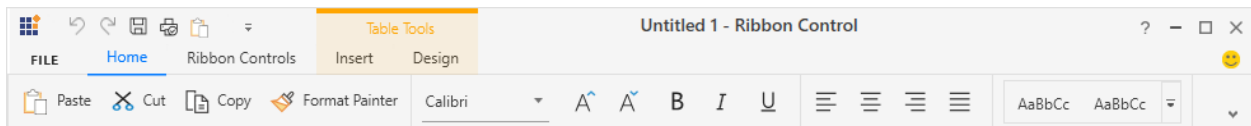


```

FrameworkElementFactory icon_element = new
FrameworkElementFactory(typeof(Path));
icon_element.SetValue(Path.MarginProperty, new Thickness(4, 2, 2, 2));
icon_element.SetValue(Path.FillProperty, new SolidColorBrush(Colors.Black));
icon_element.SetValue(Path.DataProperty, Geometry.Parse("M2.000005,0
L6.000005,0 6.000005,1 4.4186966,1 2.4888427,8.9999952 4,8.9999952
4,9.9999952 0,9.9999952 0,8.9999952 1.4594386,8.9999952 3.3901918,1
2.000005,1 z"));
iconDataTemplate.VisualTree = icon_element;
italicButton.IconTemplate = iconDataTemplate;

```

The following screenshot shows the simplified layout within the Ribbon control.



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

Customizing the Ribbon during runtime through the QAT window

The Ribbon control allows to customize the Ribbon and Ribbon items through the QAT window, where user can add the Ribbon items to a new [RibbonTab](#) or [RibbonBar](#). The newly added [RibbonTab](#) or [RibbonBar](#) will only be visible in the respective layout in which items were added originally. In the below example, the [LayoutMode](#) is set as “**Simplified**” and a new [RibbonTab](#) named *Folder* is created and added using the QAT window. This tab will now be visible only in the simplified layout and not in the normal layout which is the default behavior.



Normal layout



Simplified layout

In the meantime, the Ribbon control also allows to add items to the Quick Access Toolbar (QAT) with the help of the QAT window or through the context menu shortcut. Items added during normal or simplified layout will always be visible even when switching between layouts. In the below example, the [LayoutMode](#) is set as “**Simplified**” and the **Bold** item is added to the QAT through the context menu. This item will now be constantly visible in both normal and simplified layouts.



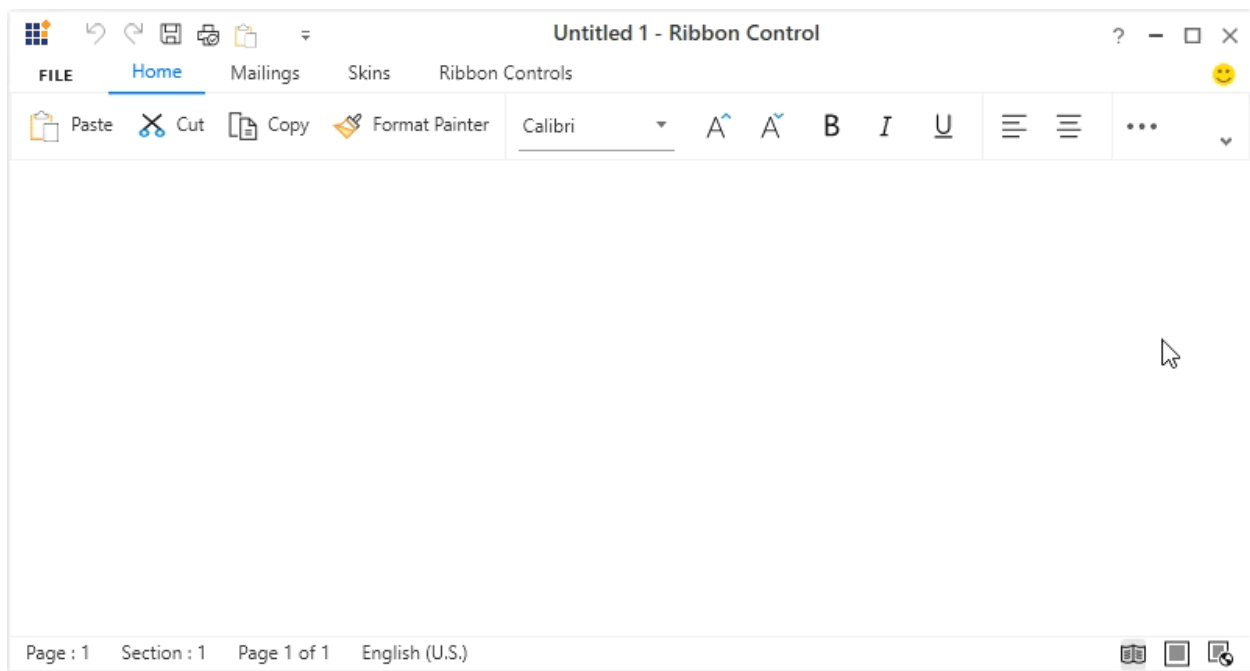
Normal layout



Simplified layout

Resizing Ribbon in simplified layout

While re-sizing the Ribbon, when the width of the window decreases and touches the last positioned item in the Ribbon, the appropriate item will be moved inside the overflow menu automatically. The same behavior will continue for each item when the window is resized continuously.



Dealing with Ribbon in WPF Ribbon

Ribbon can be changed into three different states such as **Normal**, **Hide** and **Adorner**

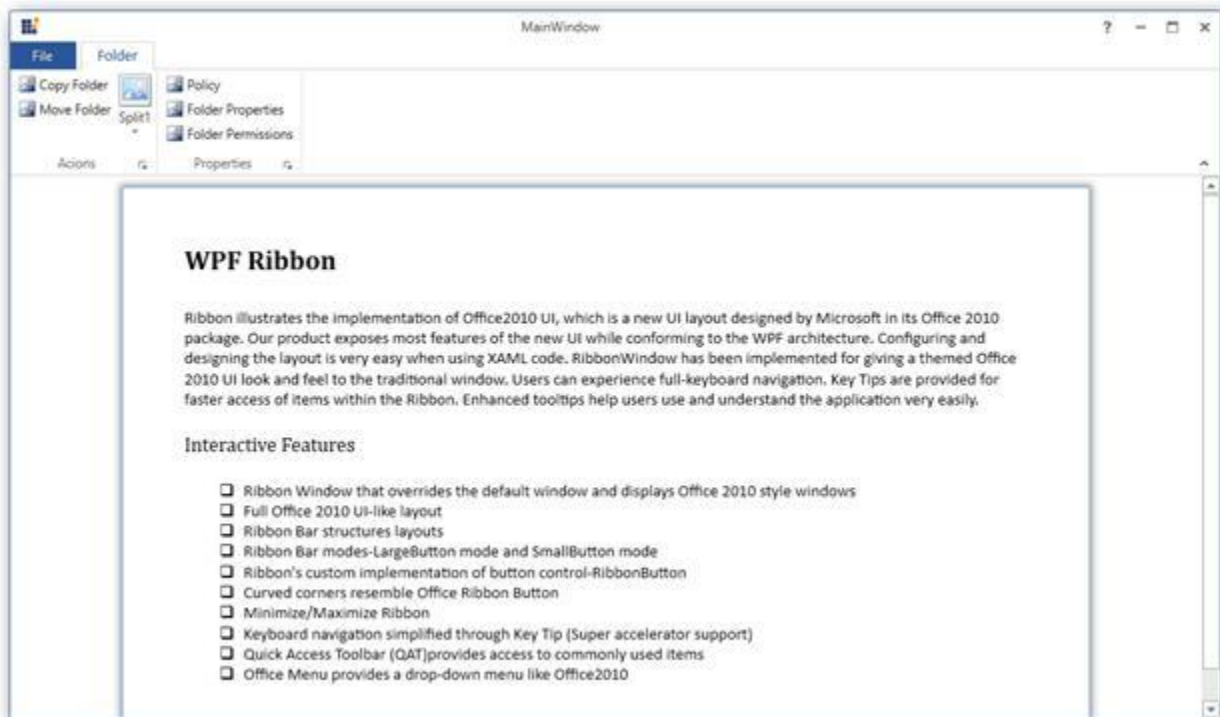
Three types of RibbonState

Normal – Ribbon control displays the RibbonTab content and the window content is arranged below the Ribbon. This is the default state

XML

```
<syncfusion:Ribbon RibbonState="Normal" VerticalAlignment="Top"
x:Name="_ribbon" >
```

```
<syncfusion:RibbonTab Caption="Folder" IsChecked="False" >
<syncfusion:RibbonBar Header="Acions">
<syncfusion:RibbonButton SizeForm="Small" Label="Copy Folder"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Move Folder"/>
<syncfusion:SplitButton Label=" Split1 " SizeForm="Large" >
<syncfusion:RibbonButton SizeForm="Small" Label="Mark to Download"/>
<syncfusion:RibbonButton SizeForm="Small" Label="UnMark to Download"/>
</syncfusion:SplitButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Properties">
<syncfusion:RibbonButton SizeForm="Small" Label="Policy"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Folder Properties"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Folder Permissions"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
```



Hide - RibbonTab content gets hidden in this state

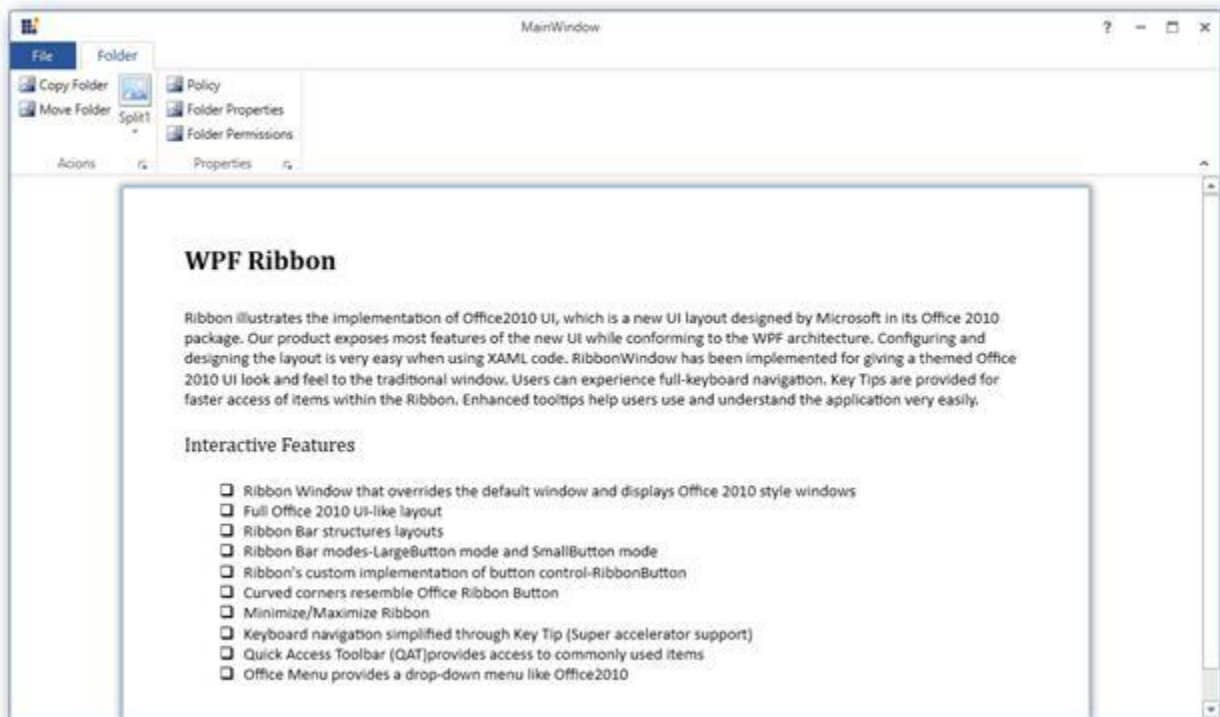
XML

```
<syncfusion:Ribbon RibbonState="Hide" VerticalAlignment="Top"
x:Name="_ribbon" >
<syncfusion:RibbonTab Caption="Folder" IsChecked="False" >
<syncfusion:RibbonBar Header="Acions">
<syncfusion:RibbonButton SizeForm="Small" Label="Copy Folder"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Move Folder"/>
<syncfusion:SplitButton Label=" Split1 " SizeForm="Large" >
<syncfusion:RibbonButton SizeForm="Small" Label="Mark to Download"/>
<syncfusion:RibbonButton SizeForm="Small" Label="UnMark to Download"/>
</syncfusion:SplitButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
```

```

</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Properties">
<syncfusion:RibbonButton SizeForm="Small" Label="Policy"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Folder Properties"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Folder Permissions"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>

```



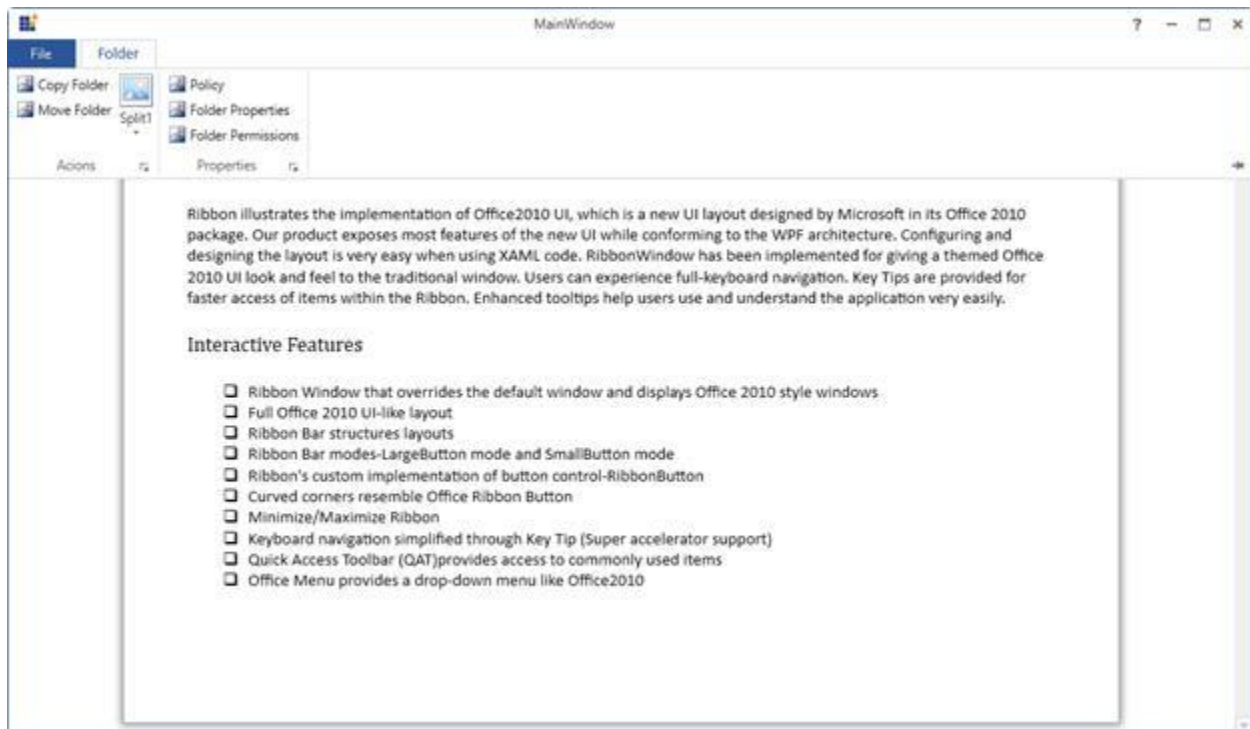
Adorner – In this state, RibbonTab content adorned above the window content

XML

```

<syncfusion:Ribbon RibbonState="Adorner" VerticalAlignment="Top"
x:Name="_ribbon" >
<syncfusion:RibbonTab Caption="Folder" IsChecked="False" >
<syncfusion:RibbonBar Header="Acions">
<syncfusion:RibbonButton SizeForm="Small" Label="Copy Folder"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Move Folder"/>
<syncfusion:SplitButton Label=" Split1 " SizeForm="Large" >
<syncfusion:RibbonButton SizeForm="Small" Label="Mark to Download"/>
<syncfusion:RibbonButton SizeForm="Small" Label="UnMark to Download"/>
</syncfusion:SplitButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Properties">
<syncfusion:RibbonButton SizeForm="Small" Label="Policy"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Folder Properties"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Folder Permissions"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>

```



How to change the RibbonState in run time

Ribbon State can also be changed at the Runtime. In the below code, Ribbon State has been changed dynamically in the button click event

C#

```
_ribbon.RibbonState = Syncfusion.Windows.Tools.RibbonState.Normal;
```

VB.NET

```
_ribbon.RibbonState = Syncfusion.Windows.Tools.RibbonState.Normal
```

C#

```
_ribbon.RibbonState = Syncfusion.Windows.Tools.RibbonState.Hide;
```

VB.NET

```
_ribbon.RibbonState = Syncfusion.Windows.Tools.RibbonState.Hide
```

C#

```
_ribbon.RibbonState = Syncfusion.Windows.Tools.RibbonState.Adorner;
```

VB.NET

```
_ribbon.RibbonState = Syncfusion.Windows.Tools.RibbonState.Adorner
```

Resize Ribbon Window

The ribbon control dynamically resizes as width of the window decreases, when the window's border touches the last placed Ribbon bar, the Ribbon will begin to resize its elements with the following priority.

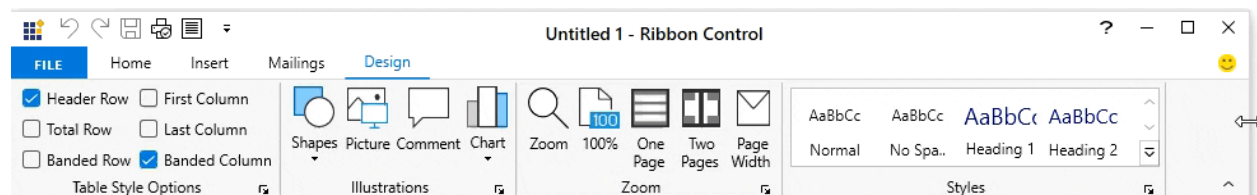
1. **Compress Gallery** - If there is a [RibbonGallery](#) in a [RibbonBar](#), it will be resized until it is converted to a dropdown button. Once it is converted into dropdown button, then the Ribbon will begin to resize the [Large](#) size form items.
2. **Compress Large items** - When there are three or more continuous [Large](#) items, then Ribbon will reduce each pair of three continuous [Large](#) size form items, starting from the right side, into [Small](#) size form items. Once each pair of three continuous [Large](#) size form items are converted into [Small](#) size form, then Ribbon will begin to resize the [Small](#) size form items.
3. **Compress Small items** - When there are three or more continuous [Small](#) items, then Ribbon will reduce each pair of three continuous [Small](#) size form items, starting from the right side, into [ExtraSmall](#) size form items. Once each pair of three continuous [Small](#) size form items are converted into [ExtraSmall](#) size form, then Ribbon will begin convert the [RibbonBar](#) into dropdown button.
4. **Collapsing the RibbonBar** - When there are no three or more continuous [Large](#) or [Small](#) size form items and when there are other ribbon items such as CheckBox, RadioButton, ComboBox, TextBox, ListBox or any other custom items, then, starting with the last bar, each [RibbonBar](#) will be converted into a dropdown button, with its items accessible by clicking on the dropdown arrow.

When the size of the window is increased, the [RibbonBar](#) will become visible if the spacing between the window border and the [RibbonBar](#) is large enough to accommodate its items. These items will then be expanded in the order, starting with the most recently collapsed item and ending with the initially collapsed item.

The [IsAutoSizeFormEnabled](#) property must be set to True to get responsive ribbon window. The following snippet is used for resize the ribbon window.

XML

```
<Grid>
<syncfusion:Ribbon syncfusion:Ribbon.IsAutoSizeFormEnabled="True"
x:Name="_ribbon" VerticalAlignment="Top">
</syncfusion:Ribbon>
</Grid>
```



RibbonBar Positioning

The [RibbonBar](#) can be positioned either at the left or right side of the [Ribbon](#) by using the [Position](#) property. The RibbonBar's [Position](#) property has following two values,

- Left - RibbonBar is placed on the left side of the Ribbon
- Right - RibbonBar is placed on the right side of the Ribbon

The following code snippet illustrates this RibbonBar positioning support.

XML

```
<syncfusion:RibbonWindow
x:Class="RightRibbonBar.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:local="clr-namespace:RightRibbonBar"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:skinManager="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:system="clr-namespace:System;assembly=mscorlib"
Title="Right RibbonBar"
Width="950"
Height="475"
syncfusion:SfSkinManager.Theme="{syncfusion:SkinManagerExtension
ThemeName=FluentLight}"
WindowStartupLocation="CenterScreen"
mc:Ignorable="d">
<Grid>
<syncfusion:Ribbon>
<syncfusion:RibbonTab Caption="Home" IsChecked="True">
<syncfusion:RibbonBar Header="Clipboard">
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Paste"
IconType="VectorImage"
Label="Paste"
SizeForm="Large">
<syncfusion:RibbonButton.VectorImage>
<Path
Margin="0,4,6,2"
Data="M17,0 L21,0 21,7 20,7 20,1.0000001 17,1.0000001 z M0,0 L4,0
4,1.0000001 1.0000001,1.0000001 1.0000001,23 12,23 12,24 0,24 z"
Fill="#FFED8733"
Stretch="Fill" />
<Path
Margin="13,12,0,0"
Data="M1.0000002,0.99999994 L1.0000002,17 13,17 13,0.99999994 z M0,0 L14,0
14,18 0,18 z"
Fill="#FF3C3B39"
Stretch="Fill" />
<Path
Margin="1,5,7,3"
Data="M16,0 L17,0 19,0 19,6 17,6 17,2 16,2 z M0,0 L2,0 3,0 3,2 2,2 2,20
11,20 11,22 0,22 z"
Fill="#FFF8DB8F"
Stretch="Fill" />
<Path
Margin="3,0.5,1,1"
Data="M10.999956,12.5 L22.999956,12.5 22.999956,28.5 10.999956,28.5 z
M7.4999558,0 C9.1569382,0 10.499956,1.3439941 10.499956,3 L13.499956,3
```

```

13.499956,6.5 15,6.5 15,10.5 9.0000001,10.5 9.0000001,24.5 0,24.5 0,6.5
1.4999557,6.5 1.4999557,3 4.4999558,3 C4.4999558,1.3439941 5.8439499,0
7.4999558,0 z"
Fill="White"
Stretch="Fill" />
<Path
Height="8"
Margin="4,0,10,0"
VerticalAlignment="Top"
Data="M6.5,0.99999996 C5.1209717,0.99999996 4,2.1209716 4,3.5 L4,4
0.99999994,4 0.99999994,7 12,7 12,4 9,4 9,3.5 C9,2.1209716
7.8790283,0.99999996 6.5,0.99999996 z M6.5,0 C8.2600098,-4.4703484E-08
9.7209473,1.3060302 9.9649658,3 L13,3 13,8 0,8 0,3 3.0350342,3
C3.2790527,1.3060302 4.7399902,-4.4703484E-08 6.5,0 z"
Fill="#FF797774"
Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Content="Paste the contents of clipboard."
Description="Paste (Ctrl+V)" />
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
HorizontalAlignment="Left"
IconType="VectorImage"
Label="Cut"
SizeForm="Small">
<syncfusion:RibbonButton.VectorImage>
<Path
Width="10"
Height="8"
Margin="3.747,0,1.805,4.614"
Data="M0.4800034,0 L3.2370005,5.6329948 5.9950049,0 6.4480002,1.3919982
3.8000043,6.7859942 6.4040015,12.108999 5.4240053,12.385 3.2370005,7.9400011
1.0859987,12.314001 0,12.249991 2.675004,6.7859942 0.027000348,1.3919982 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
Stretch="Fill" />
<Path
Width="13"
Height="4"
Margin="2,0,0,1"
VerticalAlignment="Bottom"
Data="M2.0000019,1.0000033 C1.4480005,1.0000033 1.0000028,1.4489932
1.0000028,2.0000033 1.0000028,2.5509982 1.4480005,3.0000033
2.0000019,3.0000033 2.5519957,3.0000033 3.0000009,2.5509982
3.0000009,2.0000033 3.0000009,1.4489932 2.5519957,1.0000033
2.0000019,1.0000033 z M7.9999966,0.99999999 C7.4479966,0.99999993
6.9999966,1.449 6.9999966,2 6.9999966,2.5509999 7.4479966,3 7.9999966,3
8.5519962,3 8.9999962,2.5509999 8.9999962,2 8.9999962,1.449
8.5519962,0.99999993 7.9999966,0.99999999 z M2.0000019,3.2782542E-06
C3.1029978,3.3312692E-06 4,0.89700651 4,2.0000033 4,3.1030002
3.1029978,4.0000033 2.0000019,4.0000033 0.8969985,4.0000033 0,3.1030002
0,2.0000033 0,0.89700651 0.8969985,3.3312692E-06 2.0000019,3.2782542E-06 z
M7.9999966,0 C9.1029968,-3.7871359E-08 9.9999962,0.89699995 9.9999962,2
9.9999962,3.1029999 9.1029968,4 7.9999966,4 6.8969965,4 5.9999966,3.1029999
5.9999966,2 5.9999966,0.89699995 6.8969965,-3.7871359E-08 7.9999966,0 z"

```



```

Fill="#FF1D8BCC"
Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Cut (Ctrl+X)">
<TextBlock
Width="130"
HorizontalAlignment="Left"
Text="Cut the selection and put it on the clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
HorizontalAlignment="Left"
IconType="VectorImage"
Label="Copy"
SizeForm="Small">
<syncfusion:RibbonButton.VectorImage>
<Path
Margin="3,1,0.5,0.5"
Data="M5.5000009,2.500005 L10.500001,2.500005 14.500001,6.500005
14.500001,14.500005 5.5000009,14.500005 z M0,0 L4.0000037,0 4.0000037,12
0,12 z"
Fill="White"
Stretch="Fill" />
<Path
Margin="2,0,0,0"
Data="M9.0000026,11.999999 L13.000003,11.999999 13.000003,12.999999
9.0000026,12.999999 z M9.0000026,9.9999986 L13.000003,9.9999986
13.000003,10.999999 9.0000026,10.999999 z M12,4.7070035 L12,7.0000033
14.293,7.0000033 z M6.9999967,4.0000001 L6.9999967,15 14.999997,15
14.999997,8.0000033 11,8.0000033 11,4.0000001 z M5.9999967,2.9999999
L11.706997,2.9999999 15.999997,7.293 15.999997,16 5.9999967,16 z M0,0
L6.9999967,0 6.9999967,2 5.9999971,2 5.9999971,1 1,1 1,13 4.9999976,13
4.9999976,14 0,14 z"
Fill="#FF3A3939"
Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Copy (Ctrl+C)">
<TextBlock
Width="130"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
Text="Copy the selection and put it on the clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
IconType="VectorImage"
Label="Format Painter"
SizeForm="Small">
<syncfusion:RibbonButton.VectorImage>
<Path
Margin="2.048,0.498,0.501,0.705"

```

```

Data="M13.434632,4.827646E-05 C13.680801,0.0024180976 13.926775,0.092046673
14.121765,0.27108389 14.544701,0.66010981 14.565695,1.3190858
14.167659,1.7340507 L11.531004,4.480899 13.369993,6.3119885
11.726018,7.9489524 6.8310282,12.794981 0,5.9919826 3.8440161,5.3719827
6.5970273,2.8419837 11.564087,7.7877166 6.5979662,2.8409979
8.2409729,1.2050026 10.105034,3.0610356 12.670845,0.32509833
C12.8764,0.10675568 13.155641,-0.0026375318 13.434632,4.827646E-05 z"
Fill="White"
Stretch="Fill" />
<Path
Margin="7.936,0,0,4.845"
Data="M2.3529817,2.4090486 L1.417989,3.3400479 5.8379548,7.7410448
6.7719474,6.8110455 z M7.5172076,0.99873667 C7.3818266,1.0037418
7.2480633,1.0597443 7.1480229,1.165737 L4.9139335,3.5478707
5.6360054,4.2659228 7.9211223,1.8858034 C8.0180495,1.782771
8.0700533,1.6487924 8.0661471,1.5067568 8.0611417,1.3647821
8.0010812,1.2347711 7.8960969,1.1377204 7.7895868,1.0397238
7.6525886,0.99373155 7.5172076,0.99873667 z M7.5527165,7.1653047E-05
C7.9184291,0.0035863224 8.2838595,0.13640766 8.5721467,0.40172305
8.8770893,0.68072825 9.0521443,1.0617281 9.0650842,1.4747729
9.0791228,1.8877565 8.9290931,2.2788277 8.6430719,2.5777922
L6.3439374,4.9723075 8.1899364,6.8110455 5.8379548,9.153044 0,3.3400479
2.3529817,0.99704962 4.205102,2.8418849 6.4200914,0.48174404
C6.7234051,0.15819254 7.1382422,-0.0039115331 7.5527165,7.1653047E-05 z"
Fill="#FF484644"
Stretch="Fill" />
<Path
Margin="2.049,6.365,4.368,0.705"
Data="M0.77698034,0 L10.585,3.2109802 6.8309581,6.9279997 0,0.12499928 z"
Fill="#FFF8DB8F"
Stretch="Fill" />
<Path
Margin="1,2.648,2.518,0"
Data="M2.3239305,4.4059882 L7.8809197,9.9390366 10.69988,7.1480067 z
M7.630995,1.3829954 L5.1189968,3.6919881 3.9397421,3.8823536
11.49895,6.3573499 12.063992,5.7979813 z M7.657995,0 L13.483991,5.7999814
7.9662354,11.262604 7.8789665,11.349017 7.8779948,11.349963 0,3.5039886
4.6639969,2.7519911 z"
Fill="#FFEE9243"
Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Format painter (Ctrl+Shift+C)"
HelpText="Press F1 for more help.">
<TextBlock
Width="175"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
TextWrapping="Wrap">
<Run Text="Copy formatting from one place and apply it to another." />
<LineBreak />
<LineBreak />
<Run Text="Double-click this button to apply the same formatting to multiple
places in the document." />
</TextBlock>
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>

```

```

</syncfusion:RibbonButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Font" IsLargeButtonPanel="False">
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox Width="100" IsEditable="True">
<syncfusion:RibbonComboBox.ToolTip>
<syncfusion:ScreenTip Description="Font (Ctrl+Shift+F)">
<TextBlock
Width="165"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
Text="Change the font face."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonComboBox.ToolTip>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox Width="40" IsEditable="True" />
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
IconType="VectorImage"
Label="Increase Font Size"
SizeForm="ExtraSmall">
<syncfusion:RibbonButton.VectorImage>
<Path
Width="5.005"
Height="3.002"
Margin="3,-1,1,2"
HorizontalAlignment="Right"
VerticalAlignment="Top"
Data="M2.5189795,0 L5.0050001,2.2590036 4.3330035,3.0000011
2.5149817,1.3470006 0.66799865,3.002 0,2.2570047 z"
Fill="#FF3094D0"
Stretch="Fill" />
<Path
Margin="2,1,4,0"
Data="M4.4360077,0.8710022 C4.3949921,0.98799133 4.3580048,1.1040039
4.321994,1.2200012 4.2859833,1.3549957 4.2420075,1.4919891
4.1850006,1.6309967 L2.3469865,6.2549896 6.5470016,6.2549896
4.7169835,1.6179962 C4.6359894,1.427002 4.5530121,1.1779938
4.4660065,0.8710022 z M4.0639984,0 L4.7779882,0 8.8869998,10 8.0589959,10
6.8140003,6.8899994 2.0889907,6.8899994 0.84298758,10 0,10 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
IconType="VectorImage"
Label="Decrease Font Size"
SizeForm="ExtraSmall">
<syncfusion:RibbonButton.VectorImage>
<Path
Width="5.006"
Height="3.002"
Margin="2,-1,2,2"
HorizontalAlignment="Right"

```

```

VerticalAlignment="Top"
Data="M4.3379981,0 L5.0059998,0.74500296 2.4870088,3.002 0,0.74300412
0.67199955,0.0019988532 2.4910066,1.6549994 z"
Fill="#FF3094D0"
Stretch="Fill" />
<Path
Margin="2,1,4,0"
Data="M4.4360077,0.8710022 C4.3949921,0.98799133 4.3580048,1.1040039
4.321994,1.2200012 4.2859833,1.3549957 4.2420075,1.4919891
4.1850006,1.6309967 L2.3469865,6.2549896 6.5470016,6.2549896
4.7169835,1.6179962 C4.6359894,1.427002 4.5530121,1.1779938
4.4660065,0.8710022 z M4.0639984,0 L4.7779882,0 8.8869998,10 8.0589959,10
6.8140003,6.8899994 2.0889907,6.8899994 0.84298758,10 0,10 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton
IconType="VectorImage"
IsToggle="True"
Label="Bold"
SizeForm="ExtraSmall">
<syncfusion:RibbonButton.VectorImage>
<Path
Width="10"
Height="12"
Margin="4,2,2,2"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M1.3320006,5.8759918 L1.3320006,9.8339996 3.1399993,9.8339996
C3.9220123,9.8339996 4.5279998,9.6549988 4.9589996,9.2969971
5.3899993,8.9389954 5.6040038,8.447998 5.6040038,7.8239899
5.6040038,6.5249939 4.6900024,5.8759918 2.8619995,5.8759918 z
M1.3320006,1.1660004 L1.3320006,4.7180023 2.6950073,4.7180023
C3.425003,4.7180023 3.998001,4.5469971 4.4160003,4.2079926
4.8330078,3.8669891 5.0420074,3.3880005 5.0420074,2.769989
5.0420074,1.701004 4.3150024,1.1660004 2.8619995,1.1660004 z M0,0
L3.2340087,0 C4.2180023,0 4.9970092,0.23300171 5.572998,0.69799811
6.149002,1.1629944 6.4370117,1.769989 6.4370117,2.5159912
6.4370117,3.1399994 6.2630004,3.6819916 5.9140014,4.1419983
5.5650024,4.6029968 5.0839996,4.9299927 4.4710082,5.125 L4.4710082,5.1549988
C5.2369995,5.2419891 5.8509979,5.5220032 6.3110046,5.9949951
6.7700042,6.4680023 6.9999999,7.0829926 6.9999999,7.8399963
6.9999999,8.7799988 6.6510009,9.5429993 5.95401,10.125992 5.2560119,10.709
4.376007,11 3.3130035,11 L0,11 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
IconType="VectorImage"
IsToggle="True"
Label="Italic"
SizeForm="ExtraSmall">

```

```

<syncfusion:RibbonButton.VectorImage>
<Path
Width="10"
Height="12"
Margin="4,3,2,1"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M2.000005,0 L6.000005,0 6.000005,1 4.4186966,1 2.4888427,8.9999952
4,8.9999952 4,9.9999952 0,9.9999952 0,8.9999952 1.4594386,8.9999952
3.3901918,1 2.000005,1 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
IconType="VectorImage"
Label="Underline"
SizeForm="ExtraSmall">
<syncfusion:RibbonButton.VectorImage>
<Path
Width="10"
Height="12"
Margin="4,2,2,2"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M0,10.999995 L9.0000002,10.999995 9.0000002,11.999995 0,11.999995 z
M1.000005,0 L2.0000049,0 2.0000049,6.5 C2.000005,7.8779907 3.1210072,9
4.500005,9 5.8790028,9 7.000005,7.8779907 7.000005,6.5 L7.000005,0
8.000005,0 8.000005,6.5 C8.000005,8.4299927 6.4299977,10 4.500005,10
2.5699971,10 1.000005,8.4299927 1.000005,6.5 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel
syncfusion:SimplifiedLayoutSettings.DisplayMode="Normal"
SeparatorVisibility="Collapsed">
<syncfusion:SplitButton
IconType="VectorImage"
Label="Text Highlight Color"
SizeForm="ExtraSmall">
<syncfusion:SplitButton.VectorImage>
<Path
Height="4"
Margin="1"
VerticalAlignment="Bottom"
Data="M0,0 L16,0 16,4 0,4 z"
Fill="Yellow"
Stretch="Fill" />
<Path
Margin="6.5,0.502,1.934,8.048"
Data="M6.4365052,0.00010436084 C6.6881317,-0.0035195307
6.9411348,0.087229683 7.1381453,0.27322931 L8.254165,1.3292008
C8.6621631,1.7171909 8.6711658,2.3651685 8.2731775,2.763153
L2.5470487,8.4889981 0,6.0360758 5.7431277,0.29321776 C5.9346298,0.10172514
6.1848787,0.0037282004 6.4365052,0.00010436084 z"

```

```

Fill="#FFF3F2F1"
Stretch="Fill" />
<Path
Margin="2.061,0,1.434,5"
Data="M10.881433,1.0010477 C10.756186,1.0027986 10.631936,1.052303
10.535929,1.1488092 L5.1529995,6.5319232 6.9789655,8.2909641
12.358904,2.9118485 C12.454911,2.815838 12.506913,2.6878447
12.504899,2.55184 12.502915,2.4158355 12.446916,2.2888338
12.347918,2.1958293 L11.232911,1.1398066 C11.132922,1.0452991
11.00668,0.99929665 10.881433,1.0010477 z M10.868932,0.0001521778
C11.245179,-0.005222887 11.623424,0.13178115 11.920921,0.41278561
L13.035898,1.4688085 C13.332891,1.7498205 13.498905,2.1298184
13.504886,2.5378322 13.509891,2.9468378 13.354894,3.329857
13.065897,3.6188647 L6.9929731,9.6919965 6.6261415,9.3388461
5.5570025,10.219989 C4.9130021,10.748988 4.1050018,11.038988
3.2720014,11.038988 L0,11.038988 0.22000027,10.953988 C2.0155009,10.255738
3.3807202,8.7842073 3.9559099,6.9755153 L4.0042109,6.8147053
3.725008,6.5459155 9.8289366,0.44179319 C10.118438,0.15328126
10.492686,0.0055271609 10.868932,0.0001521778 z"
Fill="#FF3A3937"
Stretch="Fill" />
</syncfusion:SplitButton.VectorImage>
</syncfusion:SplitButton>
<syncfusion:SplitButton
IconType="VectorImage"
Label="Font Color"
SizeForm="ExtraSmall">
<syncfusion:SplitButton.VectorImage>
<Path
Height="4"
VerticalAlignment="Bottom"
Data="M0,0 L16,0 16,4 0,4 z"
Fill="#FFFE0000"
Stretch="Fill" />
<Path
Margin="3.344,0,3.352,5"
Data="M4.6480023,0.95898432 C4.6079937,1.0870056 4.5689923,1.2149963
4.533012,1.34198 4.4980089,1.4909973 4.4510118,1.6419983 4.394005,1.7949829
L2.5330156,6.8809814 6.787006,6.8809814 4.9330055,1.7799988
C4.8510047,1.5699768 4.7659832,1.2969971 4.6790081,0.95898432 z M4.0779959,0
L5.209006,0 9.304,11 8.3170024,11 7.2039977,8.0019836 2.1150171,8.0029907
1.0100081,11 0,11 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
Stretch="Fill" />
</syncfusion:SplitButton.VectorImage>
</syncfusion:SplitButton>
<syncfusion:SplitButton
IconType="VectorImage"
Label="Shading"
SizeForm="ExtraSmall">
<syncfusion:SplitButton.VectorImage>
<Path
Margin="1"
Data="M0,12 L15,12 15,15 0,15 z M7.9999695,0 C8.5529652,0
8.9999628,0.44699955 8.9999628,1 L8.9999628,6 10.999949,3.5 11.002948,6.414

```

```

6.7919779,10.625 2.5420079,6.375 6.9999766,1.9169998 6.9999766,1
C6.9999766,0.44699955 7.4469733,0 7.9999695,0 z"
Fill="White"
Stretch="Fill" />
<Path
Height="4"
Margin="1"
VerticalAlignment="Bottom"
Data="M1,1 L1,3 15,3 15,1 z M0,0 L16,0 16,4 0,4 z"
Fill="#FFA1A0A0"
Stretch="Fill" />
<Path
Margin="2.335,0,3.997,4.168"
Data="M6.1649812,0 C6.9919994,0 7.6649669,0.6729753 7.6649669,1.5000038
L7.6649669,6.5000162 6.6649764,6.5000162 6.6649764,1.5000038
C6.6649764,1.2249787 6.4400027,1.0000025 6.1649812,1.0000025
5.8899591,1.0000025 5.6649859,1.2249787 5.6649859,1.5000038
L5.6649859,2.6240301 1.4140491,6.8750172 4.9569843,10.418056
8.6680091,6.7069869 8.6649573,4.0009866 9.6649478,3.9990335
9.6679995,7.1209893 4.9569843,11.832 0,6.8750172 4.664995,2.2100275
4.664995,1.5000038 C4.664995,0.6729753 5.3379624,0 6.1649812,0 z"
Fill="#FF505050"
Stretch="Fill" />
<Path
Width="2"
Margin="0,4,2,6"
HorizontalAlignment="Right"
Data="M0,0 C1.1040039,0 2,0.89599609 2,2 L2,6 1,6 C1,6 1.1300049,2.6829834
0.68994144,2.1170044 0.53698733,1.9199829 2.9802322E-08,2 0,2 z"
Fill="#FF135C9A"
Stretch="Fill" />
</syncfusion:SplitButton.VectorImage>
</syncfusion:SplitButton>
<syncfusion:SplitButton
IconType="VectorImage"
Label="Borders"
SizeForm="ExtraSmall">
<syncfusion:SplitButton.VectorImage>
<Path
Margin="1"
Data="M0,0 L15,0 15,14 0,14 z"
Fill="White"
Stretch="Fill" />
<Path
Margin="1"
Data="M0,14 L15,14 15,15 0,15 z M14,12 L15,12 15,13 14,13 z M7,12 L8,12 8,13
7,13 z M0,12 L1,12 1,13 0,13 z M14,10 L15,10 15,11 14,11 z M7,10 L8,10 8,11
7,11 z M0,10 L1,10 1,11 0,11 z M14,8 L15,8 15,8.9999999 14,8.9999999 z M7,8
L8,8 8,8.9999999 7,8.9999999 z M0,8 L1,8 1,8.9999999 0,8.9999999 z M12,7
L13,7 13,8 12,8 z M10,7 L11,7 11,8 10,8 z M8,7 L8.9999999,7 8.9999999,8 8,8
z M6,7 L7,7 7,8 6,8 z M4,7 L5,7 5,8 4,8 z M2,7 L3,7 3,8 2,8 z M14,6.0000001
L15,6.0000001 15,7 14,7 z M7,6.0000001 L8,6.0000001 8,7 7,7 z M0,6.0000001
L1,6.0000001 1,7 0,7 z M14,4 L15,4 15,5 14,5 z M7,4 L8,4 8,5 7,5 z M0,4 L1,4
1,5 0,5 z M14,2 L15,2 15,3 14,3 z M7,2 L8,2 8,3 7,3 z M0,2 L1,2 1,3 0,3 z
M14,0 L15,0 15,1 14,1 z M12,0 L13,0 13,1 12,1 z M10,0 L11,0 11,1 10,1 z M8,0
L8.9999999,0 8.9999999,1 8,1 z M6,0 L7,0 7,1 6,1 z M4,0 L5,0 5,1 4,1 z M2,0
L3,0 3,1 2,1 z M0,0 L1,0 1,1 0,1 z"

```

```

Fill="#FF3A3A38"
Stretch="Fill" />
</syncfusion:SplitButton.VectorImage>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Name="Delete" Header="Delete">
<syncfusion:RibbonButton Label="Ignore" />
<syncfusion:SplitButton Label="Clean Up">
<syncfusion:DropDownMenuItem Header="Clean Up Folder" />
<syncfusion:DropDownMenuItem Header="Clean Up Conversation" />
<syncfusion:DropDownMenuItem Header="Clean Up Folder/SubFolder" />
</syncfusion:SplitButton>
<syncfusion:SplitButton HorizontalAlignment="Left" Label="Junk" />
<syncfusion:RibbonButton
IconType="VectorImage"
Label="Delete"
SizeForm="Large">
<syncfusion:RibbonButton.VectorImage>
<Path
Margin="4"
Data="M1.4139423,0L7.0029922,5.5845888 12.592018,0 14.006015,1.4149939
8.4180527,6.9985202 14.006,12.582007 12.591996,13.997001 7.0030056,8.4124444
1.4140122,13.997001 1.5026823E-05,12.582007 5.5879484,6.9985092 0,1.4149939z
"
Fill="Red"
SnapsToDevicePixels="True"
Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar
x:Name="help"
Header="Help"
Position="Right">
<syncfusion:RibbonButton
IconType="VectorImage"
Label="About"
SizeForm="Large">
<syncfusion:RibbonButton.VectorImage>
<Path
Width="18"
Height="18"
Margin="0.5,0.5,8.354,8.353"
Data="M10,0 C15.52301,0 20,4.4769897 20,10 20,15.522003 15.52301,20 10,20
4.4780121,20 0,15.522003 0,10 0,4.4769897 4.4780121,0 10,0 z"
Fill="White"
Stretch="Fill" />
<Path
Margin="4"
Data="M10.5,1 C5.262001,1 1,5.2619991 1,10.499999 1,15.737999
5.262001,19.999999 10.5,19.999999 15.738001,19.999999 20.000001,15.737999
20.000001,10.499999 20.000001,5.2619991 15.738001,1 10.5,1 z M10.5,0
C16.290001,0 21.000001,4.7099991 21.000001,10.499999 21.000001,13.214061
19.965089,15.690819 18.269143,17.556394 L18.266851,17.558856
28.853999,28.146005 28.146998,28.853005 17.559885,18.265892

```



```

17.556396,18.269141 C15.690821,19.965087 13.214063,20.999999 10.5,20.999999
4.710001,20.999999 0,16.289999 0,10.499999 0,4.709999 4.710001,0 10.5,0 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
IconType="VectorImage"
Label="Help"
SizeForm="Large">
<syncfusion:RibbonButton.VectorImage>
<Path
Margin="6"
Data="M3.9400001,13.238 L5.5349999,13.238 5.5349999,14.833 3.9400001,14.833
z M4.7539988,0 C6.2060028,8.8817842E-16 7.3750015,0.39599991
8.2300044,1.1770003 9.0749989,1.9500008 9.5039998,2.8959999
9.5039998,3.9900017 9.5039998,4.6220016 9.3539982,5.2110023 9.0599995,5.743
8.7549992,6.2900009 8.1419993,6.9750023 7.2350021,7.7770004
6.5800033,8.3570023 6.1620041,8.7770004 5.9580017,9.0590019
5.7429972,9.3530006 5.5830012,9.6930008 5.4789973,10.070999
5.3929988,10.394001 5.3399974,10.871002 5.3170024,11.521999
L4.0500041,11.521999 C4.0479975,11.409 4.0459986,11.316002
4.0459986,11.244999 4.0459986,10.528 4.1480036,9.9029999 4.3499995,9.387001
4.4899989,9.0110016 4.7289973,8.618 5.0599986,8.2180023 5.310998,7.9189987
5.7679992,7.4770012 6.4180008,6.9049988 7.1190048,6.2859993
7.5660034,5.7989998 7.7829991,5.4169998 8.0100032,5.0200005
8.1240016,4.5839996 8.1240016,4.1189995 8.1240016,3.288002
7.796999,2.5480003 7.1510025,1.9220008 6.5110031,1.2989998
5.7139979,0.98400116&#xd;&#xa;4.784997,0.9840011 3.8870018,0.98400116
3.1250005,1.2709999 2.5199972,1.8380011 1.9710011,2.3500004
1.5930027,3.1230011 1.3939974,4.1389999 L0,3.9729996 C0.19999708,2.7350006
0.6869967,1.7670002 1.4499972,1.0950012 2.2720037,0.36900139
3.3850029,8.8817842E-16 4.7539988,0 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</syncfusion:RibbonButton.VectorImage>
</syncfusion:RibbonButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert" />
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

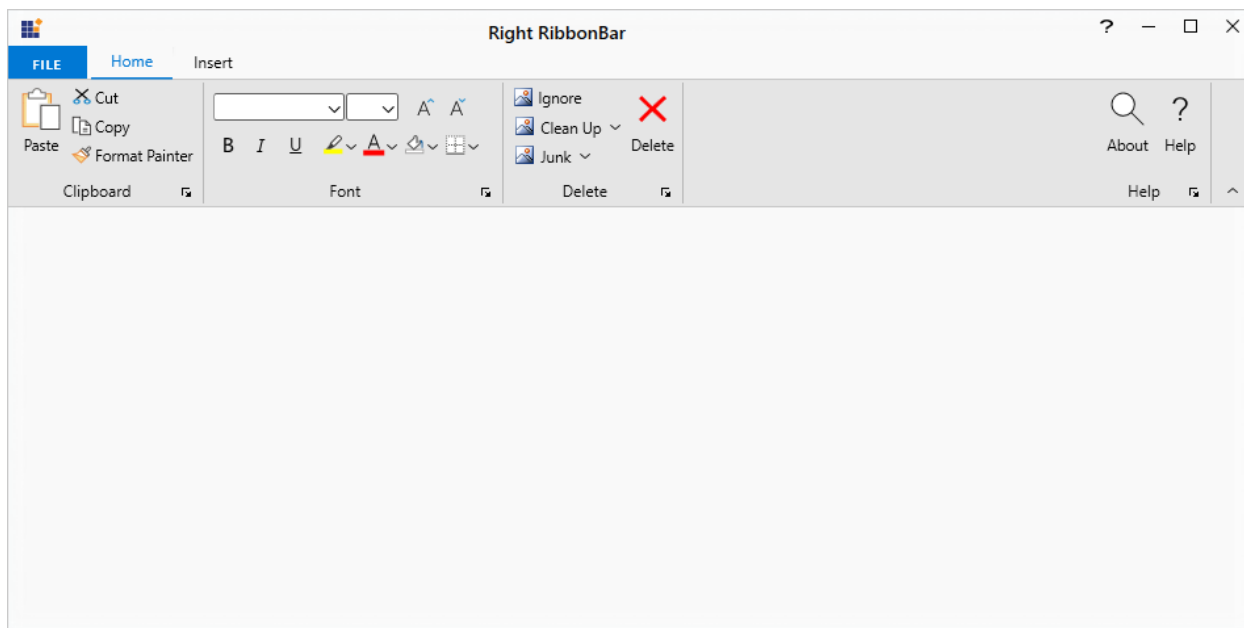
```

C#

```

this.help.Position =
Syncfusion.Windows.Tools.Controls.HorizontalPosition.Right;

```



Note: [View sample in GitHub](#)

Resize based on collapse order

The Ribbon control provides support to resize the RibbonBar's based on the order specified in the `SizeReductionOrder` property in the `RibbonTab`. The `SizeReductionOrder` property accepts the names of the `RibbonBar` that determines the order in which `RibbonBar` size is reduced. The Ribbon will begin to resize its elements with the following priority.

1. Compressing the RibbonBar specified in the SizeReductionOrder:

- a) The Ribbon will initially retrieve the `RibbonBar` corresponding to the first name specified in the `SizeReductionOrder` and resize its elements based on its `SizeForm`. This `RibbonBar` will be converted to a dropdown if there are no three or more consecutive Large or Small size form items.
- b) Once the `RibbonBar` corresponding to the first name specified in the `SizeReductionOrder` is collapsed, the Ribbon will retrieve the next name and its corresponding `RibbonBar` and begin resizing in the same manner as in the previous step. This process continues until all of the `SizeReductionOrder` property's corresponding `RibbonBar`'s are collapsed into a dropdown.

2. Compressing the other RibbonBar's:

The remaining `RibbonBar`'s that are not specified in the `SizeReductionOrder` property will be resized according to the Ribbon's default resizing behaviour once the corresponding `RibbonBar` of the names specified in the `SizeReductionOrder` property are resized and collapsed. To know more about the default resizing behavior, refer [here](#).

When the size of the window is increased, the `RibbonBar` will become visible if the spacing between the window border and the `RibbonBar` is large enough to accommodate its items. These items will then be expanded in the order, starting with the most recently collapsed item and ending with the initially collapsed item.

In the below code snippet, the `SizeReductionOrder` property in the **Home RibbonTab** contains the name of **Clipboard** and **Paragraph RibbonBar**. So, when resizing the Ribbon, the Clipboard **RibbonBar** will be compressed and converted to a dropdown first, followed by the Paragraph **RibbonBar**. The remaining RibbonBar's will be collapsed as per the default resizing behaviour once the **Clipboard** and **Paragraph** RibbonBar's are converted to dropdown.

XML

```
<syncfusion:RibbonWindow
x:Class="RibbonSample.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:interactivity="clr-
namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity
"
xmlns:local="clr-namespace:RibbonSample"
xmlns:skinManager="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:system="clr-namespace:System;assembly=mscorlib"
x:Name="ribbonWindow"
Title="Untitled 1 - Ribbon Control"
Width="980" Height="700"
skinManager:SfSkinManager.VisualStudio="FluentLight"
FlowDirection="{Binding WindowFlowDirectionProperty}"
Icon="/Resources/App.ico" IsGlassActive="False"
Office2010Icon="/Resources/syncfusion.png"
SnapsToDevicePixels="True" WindowStartupLocation="CenterScreen"
WindowState="Normal">
<syncfusion:RibbonWindow.DataContext>
<local:ViewModel />
</syncfusion:RibbonWindow.DataContext>
<syncfusion:RibbonWindow.Resources>
<syncfusion:ColorToBrushConverter x:Key="ColorToBrushConverter" />
<syncfusion:RibbonContextMenu
x:Key="GalleryContextMenu"
x:Name="contextMenu"
ItemsSource="{Binding}">
<syncfusion:RibbonMenuItem
Command="{Binding ButtonCommand}"
Header="Apply Style"
IconBarEnabled="True" />
<syncfusion:RibbonMenuItem
Command="{Binding RemoveStyleCommand}"
Header="Remove from Style Gallery"
IconBarEnabled="True" />
<Separator />
<syncfusion:RibbonMenuItem
Command="{Binding MinimizeRibbonCommand}"
Header="Minimize Ribbon"
IconBarEnabled="True" />
</syncfusion:RibbonContextMenu>
</syncfusion:RibbonWindow.Resources>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
```

```

<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary Source="/Resources/PathIcon.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Grid.Resources>
<Grid x:Name="ribbonGrid">
<syncfusion:Ribbon
Name="mainRibbon"
local:ViewModel.Ribbon="{Binding ElementName=mainRibbon}"
syncfusion:Ribbon.IsAutoSizeFormEnabled="True"
BackStageCornerImageVisibility="Collapsed"
BackStageHeader="File"
EnableMoreCommands="True"
ShowCustomizeRibbon="True">
<syncfusion:Ribbon.TabPanelItem>
<syncfusion:RibbonButton SizeForm="ExtraSmall"
SmallIcon="Resources/Smile16.png" />
</syncfusion:Ribbon.TabPanelItem>
<syncfusion:RibbonTab
syncfusion:Ribbon.KeyTip="H" SizeReductionOrder="clipboardBar,paragraphBar"
Caption="Home"
IsChecked="True">
<syncfusion:RibbonBar
Name="clipboardBar"
syncfusion:Ribbon.KeyTip="FN"
syncfusion:Ribbon.ShowInMoreCommands="False"
Header="Clipboard"
IconTemplate="{StaticResource Paste}">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:RibbonButton
syncfusion:Ribbon.KeyTip="CP"
syncfusion:RibbonCommandManager.SynchronizedItem="Paste"
Command="ApplicationCommands.Paste"
IconTemplate="{StaticResource Paste}"
Label="Paste"
SizeForm="Large">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Paste (Ctrl+V)">
<TextBlock
Width="130"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
Text="Paste the contents of clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
x:Name="cut"

```

```

HorizontalAlignment="Left"
syncfusion:Ribbon.KeyTip="CT"
Command="ApplicationCommands.Cut"
IconTemplate="{StaticResource Cut}"
Label="Cut"
SizeForm="Small">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Cut (Ctrl+X)">
<TextBlock
Width="130"
HorizontalAlignment="Left"
Text="Cut the selection and put it on the clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
x:Name="copy"
HorizontalAlignment="Left"
syncfusion:Ribbon.KeyTip="CY"
Command="ApplicationCommands.Copy"
IconTemplate="{StaticResource Copy}"
Label="Copy"
SizeForm="Small">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Copy (Ctrl+C)">
<TextBlock
Width="130"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
Text="Copy the selection and put it on the clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
x:Name="formatPainter"
syncfusion:Ribbon.KeyTip="CR"
IconTemplate="{StaticResource FormatPainter}"
Label="Format Painter"
SizeForm="Small">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Format painter (Ctrl+Shift+C)"
HelpText="Press F1 for more help.">
<TextBlock
Width="175"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
TextWrapping="Wrap">
<Run Text="Copy formatting from one place and apply it to another." />
<LineBreak />
<LineBreak />
<Run Text="Double-click this button to apply the same formatting to multiple
places in the document." />
</TextBlock>
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>

```

```

</syncfusion:RibbonButton>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar
Name="barFont"
syncfusion:Ribbon.KeyTip="HF"
Header="Font" IconTemplate="{StaticResource FormatBorder}"
IsLargeButtonPanel="False"
KeyTipOnCollapsed="ZF">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonComboBox
Name="fontFamilyComboBox"
Width="110"
syncfusion:Ribbon.KeyTip="FF"
syncfusion:Ribbon.ShowInMoreCommands="False"
DisplayMemberPath="FontFamily"
IsEditable="True"
ItemsSource="{Binding FontFamilyList}"
SelectedIndex="0">
<syncfusion:RibbonComboBox.ToolTip>
<syncfusion:ScreenTip Description="Font (Ctrl+Shift+F)">
<TextBlock
Width="165"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
Text="Change the font face."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonComboBox.ToolTip>
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction CommandParameter="{Binding
ElementName=fontFamilyComboBox, Path=SelectedIndex}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonComboBox
Name="fontSizeComboBox"
Width="40"
syncfusion:Ribbon.KeyTip="FZ"
syncfusion:Ribbon.ShowInMoreCommands="False"
DisplayMemberPath="FontSize"
IsEditable="True"
ItemsSource="{Binding FontSizeList}"
SelectedIndex="4">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction CommandParameter="{Binding
ElementName=fontSizeComboBox, Path=SelectedIndex}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</syncfusion:RibbonComboBox>
</syncfusion:RibbonButton

```

```

Name="increaseFontSize"
syncfusion:Ribbon.KeyTip="IF"
Command="EditingCommands.IncreaseFontSize"
IconTemplate="{StaticResource IncreaseFontSize}"
SizeForm="ExtraSmall" />
<syncfusion:RibbonButton
Name="decreaseFontSize"
syncfusion:Ribbon.KeyTip="DF"
Command="EditingCommands.DecreaseFontSize"
IconTemplate="{StaticResource DecreaseFontSize}"
SizeForm="ExtraSmall" />
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24">
<syncfusion:RibbonButton
Name="formatBold"
syncfusion:Ribbon.KeyTip="B"
Command="EditingCommands.ToggleBold"
IconTemplate="{StaticResource Bold}"
IsSelected="{Binding IsFormatBoldSelected}"
IsToggle="True"
SizeForm="ExtraSmall" />
<syncfusion:RibbonButton
Name="formatItalic"
syncfusion:Ribbon.KeyTip="I"
Command="EditingCommands.ToggleItalic"
IconTemplate="{StaticResource Italics}"
IsSelected="{Binding IsFormatItalicSelected}"
IsToggle="True"
SizeForm="ExtraSmall" />
<syncfusion:RibbonButton
Name="formatUnderLine"
syncfusion:Ribbon.KeyTip="U"
Command="EditingCommands.ToggleUnderline"
IconTemplate="{StaticResource Underline}"
IsSelected="{Binding IsFormatUnderLineSelected}"
SizeForm="ExtraSmall" />
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24" SeparatorVisibility="Collapsed">
<syncfusion:SplitButton
x:Name="textHighlight"
syncfusion:Ribbon.KeyTip="TH"
IconTemplate="{StaticResource TextHighlight}"
SizeForm="ExtraSmall">
<syncfusion:ColorPickerPalette
x:Name="highlightColorPicker"
BorderThickness="0"
IsExpanded="True"
Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
x:Name="fontColor"
syncfusion:Ribbon.KeyTip="FC"
IconTemplate="{StaticResource FontColor}"
SizeForm="ExtraSmall">
<syncfusion:ColorPickerPalette
x:Name="fontColorPicker"
BorderThickness="0"

```

```

IsExpanded="True"
Color="Black" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatShading"
syncfusion:Ribbon.KeyTip="FS"
IconTemplate="{StaticResource Shading}"
SizeForm="ExtraSmall">
<syncfusion:ColorPickerPalette
x:Name="shadingColorPicker"
Margin="3"
BorderThickness="0"
IsExpanded="True"
Color="White" />
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatBorder"
syncfusion:Ribbon.KeyTip="BF"
IconTemplate="{StaticResource FormatBorder}"
SizeForm="ExtraSmall">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem Margin="5">Full Border</ListBoxItem>
<ListBoxItem Margin="5">Half Border</ListBoxItem>
<ListBoxItem Margin="5">Inside Border</ListBoxItem>
<ListBoxItem Margin="5">Outside Border</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar
Name="paragraphBar"
syncfusion:Ribbon.KeyTip="HP"
Header="Paragraph" IconTemplate="{StaticResource AlignJustifyLarge}"
IsLargeButtonPanel="False"
KeyTipOnCollapsed="ZP">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:ButtonPanel Height="24">
<syncfusion:SplitButton
Name="formatBullet"
syncfusion:Ribbon.KeyTip="FN"
Command="EditingCommands.ToggleBullets"
IconTemplate="{StaticResource Bullets}"
SizeForm="ExtraSmall">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction />
</interactivity:EventTrigger>

```



```

</interactivity:Interaction.Triggers>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Grid
x:Name="Bullets_Arrow"
Width="40"
Height="40"
HorizontalAlignment="Left"
VerticalAlignment="Top"
Background="Transparent">
<Path
Margin="0.5"
Data="M0,0 L39,0 39,39 0,39 z"
Fill="Transparent"
Stretch="Fill" />
<Path
Margin="9.181,7.2,9.502,8.146"
Data="M2.6369915,2.600007 L7.8879943,11.789005 7.8822666,11.799983
19.284758,11.799983 z M0,0 L21.316999,11.78101 0.049011266,24.654001
6.7489877,11.811009 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</Grid>
<TextBlock
Margin="5"
VerticalAlignment="Center"
Text="Arrow Bullet" />
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Grid
x:Name="Bullets_Circle"
Width="40"
Height="40"
HorizontalAlignment="Left"
VerticalAlignment="Top"
Background="Transparent">
<Path
Margin="0.5"
Data="M0,0 L39,0 39,39 0,39 z"
Fill="Transparent"
Stretch="Fill" />
<Path
Margin="14,16,15,11"
Data="M7.5,0.99999991 C3.9160004,1 1,3.6910096 0.99999994,7 1,10.308991
3.9160004,13 7.5,13 11.084,13 14,10.308991 14,7 14,3.6910096 11.084,1
7.5,0.99999991 z M7.5,0 C11.636002,2.9802322E-08 15,3.1409912 15,7
15,10.859009 11.636002,14 7.5,14 3.3639984,14 0,10.859009 0,7 0,3.1409912
3.3639984,2.9802322E-08 7.5,0 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</Grid>
<TextBlock
Margin="5"

```

```

VerticalAlignment="Center"
Text="Circle Bullet" />
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Grid
x:Name="Bullets_Dot"
Width="40"
Height="40"
HorizontalAlignment="Left"
VerticalAlignment="Top"
Background="Transparent">
<Path
Margin="0.5"
Data="M0,0 L39,0 39,39 0,39 z"
Fill="Transparent"
Stretch="Fill" />
<Path
Margin="14,16,15,11"
Data="M5.5,0 C8.5370026,0 11,2.4630127 11,5.5 11,8.5369873 8.5370026,11
5.5,11 2.4629974,11 0,8.5369873 0,5.5 0,2.4630127 2.4629974,0 5.5,0 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground)"
Stretch="Fill" />
</Grid>
<TextBlock
Margin="5"
VerticalAlignment="Center"
Text="Dot Bullet" />
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Grid
x:Name="Bullets_None"
Width="40"
Height="40"
HorizontalAlignment="Left"
VerticalAlignment="Top"
Background="Transparent">
<Path
Margin="0.5"
Data="M0,0 L39,0 39,39 0,39 z"
Fill="Transparent"
Stretch="Fill" />
<Path
Margin="7.449,16.465,7.849,15.607"
Data="M22.333007,2.7070129 C21.957,2.707013 21.640006,2.8580138
21.380004,3.160992 21.120002,3.4640006 20.954001,3.8559983
20.883002,4.3400065 L23.708008,4.3400065 23.708008,4.1999923
C23.708008,3.7670091 23.594002,3.410015 23.364006,3.1290095
23.135002,2.848004 22.791,2.707013 22.333007,2.7070129 z
M9.9689951,2.7069989 C9.4389954,2.7069988 9.0409956,2.9089988
8.7759953,3.3139987 8.5119953,3.7179987 8.3789959,4.2319986
8.3789959,4.8549985 L8.3789959,4.9839984 C8.3789959,5.6179985
8.5119953,6.1329983 8.7759953,6.5309981 9.0409956,6.9279979

```

```

9.4419956,7.1269983 9.9799957,7.1269983 10.509995,7.1269983
10.908996,6.926998 11.176995,6.5279983 11.445995,6.1289982
11.579995,5.6139983 11.579995,4.9839984 L11.579995,4.8549985
C11.579995,4.2319986 11.444995,3.7179987 11.174995,3.3139987
10.903996,2.9089988 10.501995,2.7069988 9.9689951,2.7069989 z
M16.590996,1.9020022 C17.225996,1.9020023 17.709996,2.0800021
18.046996,2.4390017 18.383996,2.7970013 18.551996,3.3670007
18.551996,4.1529998 L18.551996,7.8209957 17.552996,7.8209957
17.552996,4.1739999 C17.552996,3.6400003 17.451996,3.2660007
17.249996,3.0510012 17.046996,2.8360013 16.729996,2.7290013
16.295996,2.7290014 15.969996,2.7290013 15.688997,2.8060013
15.450997,2.9590012 15.211997,3.113001 15.019997,3.3280007
14.872997,3.6040006 L14.872997,7.8209957 13.874998,7.8209957
13.874998,2.0090022 14.770997,2.0090022 14.840997,2.8790012
14.856997,2.885001 C15.045997,2.5720015 15.287997,2.3310018
15.579997,2.1590019 15.871997,1.9880022 16.207996,1.9020023
16.590996,1.9020022 z M22.333007,1.9010142 C23.121002,1.9010143
23.713005,2.1330086 24.108002,2.5940064 24.504005,3.0570186
24.702004,3.6879992 24.702004,4.4899999 L24.702004,5.1410002
20.851005,5.1410002 C20.865004,5.734017 21.012999,6.2150038
21.294005,6.5799936 21.575004,6.9450139 21.975005,7.1269902
22.494003,7.1269902 22.855003,7.1269902 23.168007,7.0770024
23.432006,6.9769963 23.694008,6.8769907 23.932007,6.7349924
24.144005,6.5530161 L24.535004,7.2129887 C24.324005,7.4209961
24.053001,7.5919861 23.722008,7.7269953 23.391006,7.859991
22.981003,7.9280146 22.494003,7.9280146 21.678001,7.9280146
21.033004,7.6640071 20.561003,7.1359929 20.088004,6.6070017
19.852004,5.9060138 19.852003,5.0329987 L19.852003,4.7909948
C19.852004,3.9459945 20.094,3.2530023 20.579001,2.7129943
21.064002,2.1720101 21.649001,1.9010143 22.333007,1.9010142 z
M9.9689951,1.9009991 C10.784995,1.900999 11.423995,2.1739989
11.883995,2.7199987 12.343995,3.2659987 12.572995,3.9779986
12.572995,4.8549985 L12.572995,4.9839984 C12.572995,5.8649982
12.344995,6.5759982 11.885995,7.1169981 11.427995,7.656998
10.791995,7.926998 9.9799957,7.926998 9.1629953,7.926998 8.5249958,7.656998
8.0669956,7.1169981 7.6089954,6.5759982 7.3799953,5.8649982
7.3799953,4.9839984 L7.3799953,4.8549985 C7.3799953,3.9779986
7.6089954,3.2659987 8.0669956,2.7199987 8.5249958,2.1739989
9.1589956,1.900999 9.9689951,1.9009991 z M0,0 L0.99299812,0
4.8769979,6.1550182 4.8939981,6.1550182 4.8939981,0 5.8869982,0
5.8869982,7.8200229 4.8939981,7.8200229 1.0099983,1.6600049
0.99299812,1.6600049 0.99299812,7.8200229 0,7.8200229 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</Grid>
<TextBlock
Margin="5"
VerticalAlignment="Center"
Text="Remove Bullet" />
</StackPanel>
</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatNumbering"
syncfusion:Ribbon.KeyTip="FB"

```

```

Command="EditingCommands.ToggleNumbering"
IconTemplate="{StaticResource Numbering}"
SizeForm="ExtraSmall">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Grid
x:Name="Numbering_LowLetterBrace"
Width="60"
Height="60"
HorizontalAlignment="Right"
VerticalAlignment="Top">
<Path
Margin="0.5"
Data="M0,0 L75,0 75,75 0,75 z"
Fill="Transparent"
Stretch="Fill" />
<Path
Width="9.369"
Margin="6.948,11.249,0,14.376"
HorizontalAlignment="Left"
Data="M2.8240006,41.163951 C3.5700009,41.163951 4.177001,41.378952
4.6430013,41.807953 5.1100011,42.237955 5.3450012,42.788957
5.3490009,43.46096 L4.318001,43.46096 C4.314001,43.053958 4.174001,42.716957
3.8960007,42.446956 3.6190009,42.177955 3.2610009,42.042954
2.8240006,42.042954 2.1950006,42.042954 1.7490005,42.260955
1.4850001,42.695956 1.2210002,43.131958 1.0900006,43.675961
1.0900006,44.327963 L1.0900006,44.573964 C1.0900006,45.241967
1.2200003,45.790969 1.4820008,46.219971 1.7440004,46.649972
2.1910007,46.864973 2.8240006,46.864973 3.2260008,46.864973
3.5750008,46.747972 3.8700006,46.512972 4.1650009,46.278971
4.314001,45.981969 4.318001,45.622968 L5.3490009,45.622968
C5.3450012,46.200971 5.0950012,46.697972 4.5960011,47.113974
4.098001,47.529976 3.5080009,47.737976 2.8240006,47.737976
1.8940005,47.737976 1.1910005,47.441975 0.71500015,46.849973
0.23800039,46.25897 0,45.499968 0,44.573964 L0,44.327963 C0,43.40596
0.23800039,42.647956 0.71500015,42.053954 1.1910005,41.460952
1.8940005,41.163951 2.8240006,41.163951 z M6.052991,37.999981
C6.7559884,38.399029 7.4089983,39.136028 8.0129974,40.212017
8.6159894,41.288006 8.9179888,42.603985 8.9179888,44.158001
L8.9179888,44.217021 C8.9179888,45.783001 8.6159894,47.102032
8.0129974,48.171978 7.4089983,49.241985 6.7559884,49.977032
6.052991,50.374981 L5.8239892,49.725018 C6.3669839,49.306988
6.8379884,48.632 7.2359951,47.699993 7.6349791,46.769024 7.8339827,45.61198
7.8339827,44.228985 L7.8339827,44.147014 C7.8339827,42.787029
7.6289977,41.639996 7.2189975,40.704021 6.808997,39.769024
6.3440043,39.084026 5.8239892,38.650982 z M2.9069784,23.066991
C2.5279789,23.066991 2.2139792,23.15499 1.9659796,23.330991
1.7179794,23.505991 1.5199795,23.748991 1.3719802,24.056991
L1.3719802,26.839993 C1.5239797,27.152992 1.7239795,27.396993
1.9719794,27.572992 2.219979,27.748993 2.5359788,27.835993
2.9189782,27.835993 3.4849782,27.835993 3.9009777,27.642993

```

4.1669779,27.255993 4.4319773,26.869992 4.5649772,26.345992
4.5649769,25.685991 L4.5649769,25.562992 C4.5649772,24.812991
4.4299773,24.208991 4.160978,23.75199 3.8909777,23.294991
3.4729781,23.066991 2.9069784,23.066991 z M0.28198004,19.480989
L1.3719802,19.480989 1.3719802,23.042991 1.3889799,23.04899
C1.5849795,22.759991 1.8289795,22.54099 2.1219795,22.38999 2.414979,22.23999
2.7659786,22.16499 3.1759784,22.16499 3.9689777,22.16499 4.5799772,22.47099
5.0069767,23.08399 5.4349765,23.69799 5.6489763,24.523992
5.6489762,25.562992 L5.6489762,25.685991 C5.6489763,26.623992
5.4349765,27.366993 5.0069767,27.915993 4.5799772,28.463993
3.9729776,28.738993 3.1879783,28.738993 2.7619786,28.738993
2.3969791,28.658993 2.0919793,28.498993 1.7879796,28.337993
1.5319796,28.097993 1.3249798,27.777992 L1.2189798,28.621993
0.28198004,28.621993 z M6.5039962,19.000002 C7.2069905,19.399002
7.8599851,20.136002 8.4639802,21.212002 9.0669756,22.288002
9.3689733,23.604002 9.3689728,25.158002 L9.3689728,25.217002
C9.3689733,26.783002 9.0669756,28.102001 8.4639802,29.172002
7.8599851,30.242002 7.2069905,30.977002 6.5039962,31.375001
L6.2749974,30.725002 C6.8179935,30.307002 7.2889898,29.632002
7.6869868,28.700002 8.0859835,27.769002 8.2849817,26.612001
8.284982,25.229002 L8.284982,25.147001 C8.2849817,23.787002
8.0799835,22.640001 7.669987,21.704002 7.25999,20.769001 6.7949937,20.084002
6.2749974,19.651001 z M2.8069894,6.6679878 C2.302989,6.6679878
1.9069889,6.7859879 1.6179886,7.0229893 1.3289886,7.2589912
1.1839881,7.5429935 1.1839886,7.8749962 1.1839881,8.1679993
1.2749882,8.3969994 1.4569883,8.5610008 1.6379886,8.7250023
1.9109888,8.807003 2.273989,8.807003 2.6799893,8.807003 3.0449896,8.7130032
3.3669899,8.526001 3.6889901,8.3379993 3.9219903,8.1079979
4.0669904,7.8339958 L4.0669904,6.6679878 z M2.7489893,3.1639595
C3.4789898,3.1639595 4.0619904,3.3439636 4.4979901,3.7029648
4.932991,4.0629692 5.1509911,4.5899734 5.150991,5.2859764
L5.150991,8.3439999 C5.1509911,8.5710011 5.1629911,8.7870026
5.1859914,8.9940052 5.2099912,9.2020054 5.2529912,9.4110069
5.3149914,9.6210098 L4.1959903,9.6210098 C4.1569904,9.4220085
4.1279904,9.258007 4.1079899,9.1290054 4.0879904,9.0000038
4.0749904,8.866003 4.0669904,8.7250023 3.8399901,9.0220051 3.55899,9.265007
3.2229898,9.4540081 2.8869896,9.6440086 2.5179892,9.7390099
2.1159892,9.7390099 1.4479885,9.7390099 0.94498825,9.5740089
0.60698795,9.2440071 0.26898766,8.9130039 0.099987507,8.4500008
0.099987507,7.8519955 0.099987507,7.2349911 0.34198761,6.757988
0.82698822,6.4219856 1.3109884,6.0859833
1.9829888,5.9179802
2.8419898,5.9179802 L4.0669904,5.9179802
4.0669904,5.2739754 C4.0669904,4.8869743 3.9479903,4.5849724
3.70999,4.368969 3.4709899,4.1519699 3.1309898,4.0429688 2.6899893,4.0429688
2.279989,4.0429688 1.9489889,4.1399689 1.6969886,4.3329697
1.4449887,4.5269699 1.3189883,4.7619743 1.3189888,5.0389748
L0.23498821,5.0389748 C0.23498774,4.550972 0.46698761,4.1169701
0.93198824,3.7359657 1.3969884,3.3549614 2.0029888,3.1639595
2.7489893,3.1639595 z M6.2819852,0 C6.9849864,0.39900208 7.6379876,1.1360016
8.2419887,2.2120018 8.84499,3.288002 9.1469905,4.6040001 9.1469905,6.1580009
L9.1469905,6.217001 C9.1469905,7.7830009 8.84499,9.1020012
8.2419887,10.172001 7.6379876,11.242002 6.9849864,11.977001
6.2819852,12.375002 L6.052985,11.725002 C6.5959857,11.307001
7.0669866,10.632002 7.464987,9.7000008 7.8639882,8.769001
8.0629885,7.6120014 8.0629883,6.229002 L8.0629883,6.1470013
C8.0629885,4.7870026 7.8579881,3.6400032 7.4479872,2.7040024
7.0379865,1.769001 6.5729856,1.0840034 6.052985,0.65100098 z"

```

Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
Stretch="Fill" />
<Path
Margin="18,16,7,20"
Data="M0,37.999999 L51,37.999999 51,39.999999 0,39.999999 z
M0.99999994,18.999999 L51,18.999999 51,20.999999 0.99999994,20.999999 z
M0.99999994,0 L51,0 51,1.9999992 0.99999994,1.9999992 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}"
Stretch="Fill" />
</Grid>
<TextBlock
Margin="5"
VerticalAlignment="Center"
Text="Lowercase Braces" />
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Grid
x:Name="Numbering_Lowletter_Dot"
Width="60"
Height="60"
HorizontalAlignment="Right"
VerticalAlignment="Top">
<Path
Margin="0.5"
Data="M0,0 L75,0 75,75 0,75 z"
Fill="Transparent"
Stretch="Fill" />
<Path
Width="8.28"
Margin="6.948,14.413,0,17.013"
HorizontalAlignment="Left"
Data="M6.7260109,43.302991 L7.8280107,43.302991 7.8280107,44.456992
6.7260109,44.456992 z M2.8240008,37.999999 C3.5700009,37.99999
4.177001,38.214991 4.6430008,38.643993 5.1100011,39.073995
5.3450011,39.624997 5.3490012,40.296999 L4.318001,40.296999
C4.3140008,39.889998 4.174001,39.552996 3.8960011,39.282995
3.6190009,39.013994 3.2610006,38.878994 2.8240008,38.878994
2.1950006,38.878994 1.7490005,39.096994 1.4850001,39.531996
1.2210002,39.967998 1.0900002,40.512 1.0900006,41.164003
L1.0900006,41.410003 C1.0900002,42.078006 1.2200003,42.627008
1.4819999,43.05601 1.7440004,43.486012 2.1910005,43.701012
2.8240008,43.701012 3.2260008,43.701012 3.5750008,43.584012
3.8700008,43.349011 4.1650009,43.11501 4.3140008,42.818009
4.318001,42.459008 L5.3490012,42.459008 C5.3450011,43.03701
5.0950011,43.534012 4.5960011,43.950013 4.098001,44.366015
3.5080009,44.574016 2.8240008,44.574016 1.8940005,44.574016
1.1910005,44.278015 0.71499968,43.686012 0.23800039,43.09501 0,42.336007
0,41.410003 L0,41.164003 C0,40.241999 0.23800039,39.483996
0.71499968,38.889994 1.1910005,38.296991 1.8940005,37.99999
2.8240008,37.99999 z M7.1780095,24.303017 L8.2799587,24.303017
8.2799587,25.457026 7.1780095,25.457026 z M2.9070091,19.90303
C2.5280094,19.90303 2.2140098,19.99103 1.9660091,20.16703 1.7180099,20.34203
1.52001,20.58503 1.3720102,20.89303 L1.3720102,23.676031

```

```

C1.5240102,23.989033 1.72401,24.233032 1.9720106,24.409033
2.2200098,24.585032 2.5360093,24.672031 2.9190087,24.672031
3.4850085,24.672031 3.9010081,24.479033 4.1670079,24.092031
4.4320078,23.706032 4.5650077,23.182032 4.5650079,22.522032
L4.5650079,22.399031 C4.5650077,21.649031 4.4300077,21.045031
4.1610081,20.58803 3.8910081,20.131031 3.4730086,19.90303 2.9070091,19.90303
z M0.28201103,16.317028 L1.3720102,16.317028 1.3720102,19.87903
1.38901,19.885031 C1.5850101,19.596029 1.82901,19.377029 2.1220098,19.22603
2.4150095,19.076029 2.7660091,19.00103 3.1760087,19.00103 3.9690082,19.00103
4.5800076,19.30703 5.0070069,19.920031 5.435007,20.534031
5.6490068,21.360031 5.6490072,22.399031 L5.6490072,22.522032
C5.6490068,23.460032 5.435007,24.203032 5.0070069,24.752033
4.5800076,25.300032 3.9730082,25.575033 3.188009,25.575033
2.7620091,25.575033 2.3970094,25.495033 2.0920095,25.335033
1.7880101,25.174032 1.5320101,24.934032 1.3250098,24.614033
L1.2190108,25.458033 0.28201103,25.458033 z M6.9549709,5.3030167
L8.0569813,5.3030167 8.0569813,6.4570236 6.9549709,6.4570236 z
M2.8069746,3.5040283 C2.3029714,3.5040283 1.9069686,3.6220284
1.6179676,3.8590279 1.3289652,4.0950317 1.1839643,4.3790321
1.1839643,4.7110367 1.1839643,5.0040398 1.2749648,5.2330399
1.4569664,5.3970413 1.6379671,5.5610428 1.9109688,5.6430435
2.2739706,5.6430435 2.6799741,5.6430435 3.0449762,5.5490417
3.3669782,5.3620415 3.6889806,5.1740379 3.921982,4.9440384
4.0669835,4.6700363 L4.0669835,3.5040283 z M2.7489743,0 C3.4789791,0
4.0619829,0.18000031 4.4979858,0.53900528 4.9329886,0.89900589
5.1509901,1.4260101 5.1509905,2.1220169 L5.1509905,5.1800385
C5.1509901,5.4070396 5.1629902,5.6230431 5.1859901,5.8300438
5.2099905,6.038044 5.2529907,6.2470474 5.3149912,6.4570503
L4.1959839,6.4570503 C4.1569836,6.258049 4.1279833,6.0940475
4.1079834,5.9650459 4.0879831,5.8360443 4.0749831,5.7020416
4.0669835,5.5610428 3.8399816,5.8580437 3.5589797,6.1010475
3.2229779,6.2900467 2.8869753,6.4800491 2.5179729,6.5750504
2.1159701,6.5750504 1.4479656,6.5750504 0.9449625,6.4100494
0.6069603,6.0800476 0.26895809,5.7490425 0.099956989,5.2860413
0.099956512,4.6880341 0.099956989,4.0710297 0.34195852,3.5940285
0.82696152,3.2580261 1.3109651,2.9220238
1.9829693,2.7540207&#xd;&#xa;2.841975,2.7540207 L4.0669835,2.7540207
4.0669835,2.1100159 C4.066983,1.7230148 3.9479823,1.4210129
3.7099807,1.2050095 3.470979,0.98800659 3.1309769,0.87900543
2.6899738,0.87900543 2.2799711,0.87900543 1.9489689,0.97600937
1.6969681,1.1690102 1.4449658,1.3630104 1.318965,1.598011
1.3189645,1.8750153 L0.23495817,1.8750153 C0.2349577,1.3870125
0.46695948,0.95300674 0.93196249,0.57200623 1.3969655,0.19100189 2.0029693,0
2.7489743,0 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
<Path
Margin="17,16,6,20"
Data="M0,38 L53,38 53,40 0,40 z M1,19 L53,19 53,21 1,21 z M1,0 L53,0 53,2
1,2 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</Grid>
<TextBlock
Margin="5"

```

```

VerticalAlignment="Center"
Text="Lowercase Dot" />
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Grid
x:Name="Numbering_Dot"
Width="60"
Height="60"
HorizontalAlignment="Left"
VerticalAlignment="Top">
<Path
Margin="0.5"
Data="M0,0 L75,0 75,75 0,75 z"
Fill="Transparent"
Stretch="Fill" />
<Path
Width="8.244"
Margin="6.925,12.339,0,17.007"
HorizontalAlignment="Left"
Data="M7.1420118,45.376981 L8.2440114,45.376981 8.2440114,46.530981
7.1420118,46.530981 z M2.7070026,37.877029 C3.5110023,37.877029
4.1420021,38.083028 4.5990024,38.495026 5.0560019,38.907024
5.2850019,39.508022 5.285002,40.297019 5.2850019,40.660017
5.1780019,41.014015 4.963002,41.357013 4.7480021,41.701012
4.4220021,41.967011 3.9840024,42.15401 4.5110021,42.32201 4.883002,42.584008
5.0970023,42.939007 5.3120018,43.295005 5.4200019,43.701003
5.420002,44.158001 5.4200019,44.950998 5.1700019,45.564995
4.6700022,46.000993 4.1700022,46.436991
3.5170023,46.65399&#xd;&#xa;2.7130027,46.65399 1.9240026,46.65399
1.275003,46.445991 0.76700258,46.029993 0.26000309,45.613995
0.0060033798,45.028998 0.0060033798,44.275001 L1.0950031,44.275001
C1.0950031,44.739999 1.2380028,45.106997 1.5230026,45.376996
1.8080029,45.645995 2.2050028,45.780994 2.7130027,45.780994
3.2320025,45.780994 3.6330023,45.648995 3.9140022,45.385996
4.1950021,45.121997 4.3360021,44.720999 4.3360023,44.182001
4.3360021,43.639004 4.2070022,43.242005 3.949002,42.992007
3.6910024,42.742008 3.2810025,42.617008 2.7190032,42.617008
L1.7400026,42.617008 1.7400026,41.750012 2.7190032,41.750012
C3.2580025,41.750012 3.6390023,41.621012 3.8640022,41.363014
4.0890021,41.105015 4.2010021,40.742016 4.2010024,40.273019
4.2010021,39.77002 4.0780022,39.391022 3.8320022,39.137024
3.5860023,38.883025 3.2110023,38.756025 2.7070026,38.756025
2.2340026,38.756025 1.8590026,38.889025 1.5820031,39.154024
1.3040028,39.420022 1.1660032,39.779021 1.1660028,40.232018
L0.082003593,40.232018 C0.082003117,39.557022 0.32200336,38.995024
0.80300379,38.548026 1.2830029,38.101028 1.9180026,37.877029
2.7070026,37.877029 z M7.1420013,26.377007 L8.2439969,26.377007
8.2439969,27.531015 7.1420013,27.531015 z M2.8469973,18.876986
C3.6209977,18.876986 4.2309973,19.094986 4.6789968,19.532986
5.1259966,19.969986 5.3489965,20.542986 5.3489965,21.249987
5.3489965,21.721987 5.2129966,22.185987 4.9389961,22.640989
4.665997,23.095989 4.2729971,23.60499 3.761997,24.16799 L1.4709992,26.657991
5.6599964,26.657991 5.6599964,27.530993 0.16400003,27.530993
0.16400003,26.762993 2.9589972,23.651989 C3.4739978,23.081989
3.8199975,22.633989 3.9959974,22.306988 4.1719973,21.980988

```



```

4.2599974,21.641987 4.2599976,21.290987 4.2599974,20.844986
4.1339974,20.477987 3.8819973,20.188986 3.6299977,19.899986
3.2849979,19.755985 2.8469973,19.755985 2.2379985,19.755985
1.7919989,19.905987 1.5089989,20.206985 1.2249994,20.507986
1.0839992,20.940987 1.0839996,21.507988 L0,21.507988 C0,20.745987
0.24799967,20.116985 0.74399948,19.620985 1.2399993,19.124985
1.9409986,18.876986 2.8469973,18.876986 z M7.142002,7.3770065
L8.2439969,7.3770065 8.2439969,8.5310135 7.142002,8.5310135 z M3.539005,0
L3.539005,8.5309944 2.4550076,8.5309944 2.4550076,1.2419968
0.71501112,1.2889977 0.71501112,0.59199905 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
<Path
Margin="18,16,6,20"
Data="M0,38 L52,38 52,40 0,40 z M0,19 L52,19 52,21 0,21 z M0,0 L52,0 52,2
0,2 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</Grid>
<TextBlock
Margin="5"
VerticalAlignment="Center"
Text="Number Dot" />
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Grid
x:Name="Numbering_Upletter"
Width="60"
Height="60"
HorizontalAlignment="Right"
VerticalAlignment="Top">
<Path
Margin="0.5"
Data="M0,0 L75,0 75,75 0,75 z"
Fill="Transparent"
Stretch="Fill" />
<Path
Width="9.756"
Margin="5.608,12.339,0,17.007"
HorizontalAlignment="Left"
Data="M8.396025,45.377015 L9.4980357,45.377015 9.4980357,46.531025
8.396025,46.531025 z M3.798023,37.877038 C4.7550271,37.877038
5.4960306,38.124035 6.0210326,38.618035 6.547035,39.112034
6.856036,39.812031 6.9500363,40.719025 L5.8660315,40.719025
C5.7680314,40.070027 5.5600308,39.581032 5.2420295,39.25103
4.9240282,38.921032 4.442026,38.756035 3.798023,38.756035
3.1260205,38.756035 2.5960184,39.015034 2.2070163,39.532033
1.818015,40.05003 1.6240142,40.707024 1.6240142,41.504021
L1.6240142,43.021016 C1.6240142,43.826012 1.818015,44.487008
2.2070163,45.005005 2.5960184,45.522004 3.1260205,45.781003
3.798023,45.781003 4.4460261,45.781003 4.9290279,45.621003
5.2450293,45.301007 5.5610307,44.980008 5.7680314,44.49001
5.8660315,43.830014 L6.9500363,43.830014 C6.856036,44.689007

```

```

6.5430348,45.375004 6.0100325,45.887002 5.4760302,46.397999 4.7390273,46.654
3.798023,46.654 2.8130191,46.654 2.024016,46.318001 1.4300135,45.646004
0.83701092,44.975007 0.54000967,44.10001 0.54000955,43.021016
L0.54000955,41.516022 C0.54000967,40.441025 0.83701092,39.566029
1.4300135,38.891034 2.024016,38.215035 2.8130191,37.877038
3.798023,37.877038 z M8.3550103,26.377017 L9.4570211,26.377017
9.4570211,27.531024 8.3550103,27.531024 z M1.9220321,23.517018
L1.9220321,26.658001 3.9550276,26.658001 C4.506027,26.658001
4.9310263,26.528002 5.2300256,26.268004 5.5280254,26.008006
5.6780251,25.632006 5.6780251,25.140009 5.6780251,24.628012
5.5520251,24.230013 5.3000258,23.945015 5.0480259,23.660018
4.6540268,23.517018 4.1190279,23.517018 z M1.9220321,19.878037
L1.9220321,22.644022 3.8670285,22.644022 C4.3170274,22.632021
4.6730269,22.510024 4.9370263,22.278023 5.2000259,22.045025
5.3320258,21.712027 5.3320258,21.279028 5.3320258,20.802031
5.185026,20.450034 4.8900259,20.221034 4.5950268,19.993035
4.1560276,19.878037 3.5740295,19.878037 z M0.83803361,18.999041
L3.5740295,18.999041 C4.4810274,18.999041 5.1810258,19.18804
5.6750253,19.565037 6.1690238,19.942036 6.4160235,20.521033
6.4160235,21.302029 6.4160235,21.693026 6.298024,22.037026
6.0620239,22.333024 5.8250249,22.630023 5.514025,22.849021
5.1270254,22.990021 L5.1270254,23.00702 C5.639025,23.11302
6.0390242,23.360018 6.3280237,23.748017 6.6170233,24.137015
6.7620228,24.597012 6.7620228,25.12801 6.7620228,25.921006
6.5140231,26.520003 6.0180243,26.924 5.5220254,27.328999 4.8340266,27.530997
3.9550276,27.530997 L0.83803361,27.530997 z M8.6540301,7.3770161
L9.7560094,7.3770161 9.7560094,8.5310245 8.6540301,8.5310245 z
M3.7560075,1.3419791 L2.2680033,5.3849449 5.2210024,5.3849449 z M3.292996,0
L4.2309831,0 7.471,8.5309997 6.3639763,8.5309997 5.537988,6.2579918
1.9459817,6.2579918 1.1080008,8.5309997 0,8.5309997 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
<Path
Margin="18,16,7,20"
Data="M1.9999999,38 L51,38 51,40 1.9999999,40 z M1,19 L51,19 51,21 1,21 z
M0,0 L51,0 51,2 0,2 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self}},
Path=(TextBlock.Foreground) }"
Stretch="Fill" />
</Grid>
<TextBlock
Margin="5"
VerticalAlignment="Center"
Text="Uppercase Dot" />
</StackPanel>
</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
<syncfusion:SplitButton
Name="formatLineSpacing"
syncfusion:Ribbon.KeyTip="FA"
IconTemplate="{StaticResource LineSpacing}"
SizeForm="ExtraSmall">
<ListBox BorderThickness="0">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">

```

```

<interactivity:InvokeCommandAction      />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<ListBoxItem Margin="5">1</ListBoxItem>
<ListBoxItem Margin="5">1.15</ListBoxItem>
<ListBoxItem Margin="5">1.5</ListBoxItem>
<ListBoxItem Margin="5">2</ListBoxItem>
<ListBoxItem Margin="5">2.5</ListBoxItem>
<ListBoxItem Margin="5">3</ListBoxItem>
</ListBox>
</syncfusion:SplitButton>
</syncfusion:ButtonPanel>
<syncfusion:ButtonPanel Height="24">
<syncfusion:RibbonButton
Name="formatLeftAlign"
syncfusion:Ribbon.KeyTip="LA"
Command="EditingCommands.AlignLeft"
IconTemplate="{StaticResource AlignLeft}"
SizeForm="ExtraSmall" />
<syncfusion:RibbonButton
Name="formatCenterAlign"
syncfusion:Ribbon.KeyTip="CA"
Command="EditingCommands.AlignCenter"
IconTemplate="{StaticResource AlignCenter}"
SizeForm="ExtraSmall" />
<syncfusion:RibbonButton
Name="formatRightAlign"
syncfusion:Ribbon.KeyTip="RA"
Command="EditingCommands.AlignRight"
IconTemplate="{StaticResource AlignRight}"
SizeForm="ExtraSmall" />
<syncfusion:RibbonButton
Name="formatJustify"
syncfusion:Ribbon.KeyTip="JA"
Command="EditingCommands.AlignJustify"
IconTemplate="{StaticResource AlignJustify}"
SizeForm="ExtraSmall" />
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="Styles" IconTemplate="{StaticResource
IncreaseFontSize}">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="LauncherClick">
<interactivity:InvokeCommandAction      />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:RibbonGallery
x:Name="gallery"
Width="350"
Height="67"
ContextMenu="{DynamicResource GalleryContextMenu}"
IconTemplate="{StaticResource Styling}"
ItemHeight="60"
ItemWidth="65"
Label="Styles"
MenuIconBarEnabled="True"
SelectedItem="{Binding RibbonGallerySelectedItem}"

```

```
SizeForm="Large">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectedItemChanged">
<interactivity:InvokeCommandAction />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
<syncfusion:RibbonGalleryItem
x:Name="firstItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemOneCommand}"
Tag="firstItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,11,0,0"
Text="Normal"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="secondItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemTwoCommand}"
Tag="secondItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,11,0,0"
Text="No Spa.."
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="thirdItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemThreeCommand}"
Tag="thirdItem">
<StackPanel>
<TextBlock
Margin="0,5,0,0"
FontSize="18"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,6,0,0"
Text="Heading 1"
TextAlignment="Center" />
```

```
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="fourthItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemFourCommand}"
Tag="fourthItem">
<StackPanel>
<TextBlock
Margin="0,6,0,0"
FontSize="16"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,8,0,0"
Text="Heading 2"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="fifthItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemFiveCommand}"
Tag="fifthItem">
<StackPanel>
<TextBlock
Margin="0,7,0,0"
FontSize="15"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,9,0,0"
Text="Heading 3"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="sixthItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemSixCommand}"
Tag="sixthItem">
<StackPanel>
<TextBlock
Margin="0,7,0,0"
FontSize="14"
Foreground="DarkBlue"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,9,0,0"
Text="Heading 4"
TextAlignment="Center" />
```

```
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="seventhItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemSevenCommand}"
Tag="seventhItem">
<StackPanel>
<TextBlock
FontSize="24"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,5,0,0"
Text="Title"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="eighthItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemEightCommand}"
Tag="eighthItem">
<StackPanel>
<TextBlock
Margin="0,7,0,0"
FontStyle="Italic"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,13,0,0"
Text="Emphasis"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="ninthItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemNineCommand}"
Tag="ninthItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
FontWeight="Bold"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Strong"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="tenthItem"
```

```
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemTenCommand}"
Tag="tenthItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Text="AaBbCc"
TextAlignment="Center"
TextDecorations="Underline" />
<TextBlock
Margin="0,10,0,0"
Text="Underline"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="eleventhItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemElevenCommand}"
Tag="eleventhItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
Foreground="Red"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Important"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="italicItem"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemTwelveCommand}"
Tag="italicItem">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
FontStyle="Italic"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Italic"
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGalleryItem
x:Name="superString"
Margin="2"
CheckOnClick="True"
Command="{Binding GalleryItemThirteenCommand}"
```

```

Tag="superString">
<StackPanel>
<TextBlock
Margin="0,9,0,0"
FontWeight="ExtraBold"
Text="AaBbCc"
TextAlignment="Center" />
<TextBlock
Margin="0,10,0,0"
Text="Super Str.."
TextAlignment="Center" />
</StackPanel>
</syncfusion:RibbonGalleryItem>
<syncfusion:RibbonGallery.MenuItems>
<syncfusion:RibbonButton IconTemplate="{StaticResource IncreaseFontSize}"
Label="Create a style" />
<syncfusion:RibbonButton
IconTemplate="{StaticResource ClearFormatting}"
Label="Clear Formatting" />
</syncfusion:RibbonGallery.MenuItems>
</syncfusion:RibbonGallery>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab
syncfusion:Ribbon.KeyTip="I"
Caption="Insert"
IsChecked="False">
</syncfusion:RibbonTab>
<syncfusion:RibbonTab
syncfusion:Ribbon.KeyTip="D"
Caption="Design"
IsChecked="False">
</syncfusion:RibbonTab>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage Name="ribbonBackStage">
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.ShowInMoreCommands="False"
Command="{Binding ButtonCommand}"
Header="Save" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.ShowInMoreCommands="False"
Command="{Binding SaveAsCommand}"
Header="Save As" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.ShowInMoreCommands="False"
Command="{Binding OpenCommand}"
Header="Open" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.ShowInMoreCommands="False"
Command="{Binding CloseCommand}"
Header="Close" />
<syncfusion:BackStageCommandButton
syncfusion:Ribbon.ShowInMoreCommands="False"
Command="{Binding BackStageExitCommand}"
Header="Exit" />
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>

```



```

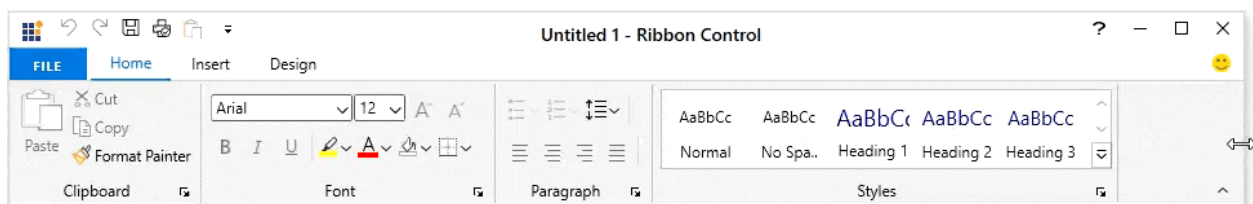
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Save" Label="Save" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Quick Print" Label="Quick
Print" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Print Preview"
Label="Print Preview" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Undo" Label="Undo" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Redo" Label="Redo" />
<syncfusion:RibbonButton
syncfusion:RibbonCommandManager.SynchronizedItem="Paste" Label="Paste" />
</syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Undo"
Command="ApplicationCommands.Undo"
IconTemplate="{StaticResource Undo}"
Label="Undo"
SizeForm="ExtraSmall"
ToolTip="Undo"/>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Redo"
Command="ApplicationCommands.Redo"
IconTemplate="{StaticResource Redo}"
Label="Redo"
SizeForm="ExtraSmall"
ToolTip="Redo"/>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Save"
Command="{Binding ButtonCommand}"
IconTemplate="{StaticResource Save}"
Label="Save"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Quick Print"
IconTemplate="{StaticResource QuickPrint}"
Label="QuickPrint"
SizeForm="ExtraSmall"/>
<syncfusion:RibbonButton
Width="24"
Height="24"
syncfusion:RibbonCommandManager.SynchronizedItem="Paste"
IconTemplate="{StaticResource Paste}"
Label="Paste"

```

```

SizeForm="ExtraSmall"
ToolTip="Paste"/>
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
</syncfusion:Ribbon>
</Grid>
<ScrollViewer
x:Name="scrollView"
Grid.Row="1"
Grid.ColumnSpan="2"
VerticalScrollBarVisibility="Auto">
<Grid>
<RichTextBox
Name="editor"
Margin="100,10"
Padding="50"
local:ViewModel.RichTextBox="richTextBoxText"
AcceptsTab="True"
Background="{Binding ElementName=shadingColorPicker, Path=Color,
Mode=OneWay, Converter={StaticResource ColorToBrushConverter}}"
ContextMenu="{x:Null}"
FontSize="14"
Foreground="{Binding ElementName=fontColorPicker, Path=Color, Mode=OneWay,
Converter={StaticResource ColorToBrushConverter}}"
SelectionBrush="{Binding ElementName=highlightColorPicker, Path=Color,
Mode=OneWay, Converter={StaticResource ColorToBrushConverter}}">
<interactivity:Interaction.Triggers>
<interactivity:EventTrigger EventName="SelectionChanged">
<interactivity:InvokeCommandAction Command="{Binding
Path=RichTextBoxSelectionChangedCommand}" />
</interactivity:EventTrigger>
<interactivity:EventTrigger EventName="PreviewMouseLeftButtonUp">
<interactivity:InvokeCommandAction Command="{Binding
Path=RichTextBoxPreviewMouseLeftButtonUpCommand}" />
</interactivity:EventTrigger>
</interactivity:Interaction.Triggers>
</RichTextBox>
</Grid>
</ScrollViewer>
</Grid>
</syncfusion:RibbonWindow>

```



Note: [View sample in GitHub](#)

Setting collapse image for RibbonBar

When the RibbonBar is converted into a dropdown button during resizing, the [RibbonBar](#) allows us to display a image using its [IconTemplate](#) or [CollapseImage](#) property.

Note: The [RibbonBar](#) loads icon in the following priority order,

- [IconTemplate](#)
- [CollapselImage](#)

Setting icon template

The [IconTemplate](#) property provides support to set any type of image such as glyph, font or any custom content to the [RibbonBar](#). The [RibbonBar](#) displays the [IconTemplate](#) in 16 * 16 size.

XML

```
<syncfusion:RibbonWindow x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="350" Width="600">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Clipboard" >
<syncfusion:RibbonBar.IconTemplate>
<DataTemplate>
<Grid Width="14" Height="16">
<Path Width="7" Margin="0,7,0,0" HorizontalAlignment="Right"
Fill="#FF3A3A38" Stretch="Fill"
Data="F1M122,367L127,367L127,360L122,360z M128,368L121,368L121,359L128,359z"
/>
<Path Margin="0,2,2,0" Fill="#FFDE6C00" Stretch="Fill"
Data="M0,0 L12,0 12,4 11,4 11,0.99999994 1.0000002,0.99999994 1.0000002,13
6.0000001,13 6.0000001,14 0,14 z" />
<Path Margin="1,3,3,1" Fill="#FFF8DB8F" Stretch="Fill"
Data="M0,0 L10,0 10,3 9.0000001,3 9.0000001,0.99999994 1.0000001,0.99999994
1.0000001,1.5829999 1.0000001,2.5 1.0000001,11 5.0000001,11 5.0000001,12
0,12 z" />
<Path Margin="2.011,0.5,0.983,0.983" Fill="#FFFAFAFA" Stretch="Fill"
Data="M5.9873815,7.496151 L11.006,7.496151 11.006,14.516999
5.9873815,14.516999 z M0,5.4959998 L3.9880071,5.4964137 3.9880071,13.51695
0,13.51695 z M3.9889999,2.2337155E-15 C4.8170028,-4.4703477E-08
5.4889999,0.67098993 5.4889999,1.5 L5.4889999,2 7.4889999,2 7.4889999,5
0.4889999,5 0.4889999,2 2.4889999,2 2.4889999,1.5 C2.4889999,0.67098993
3.1609969,-4.4703477E-08 3.9889999,2.2337155E-15 z" />
<Path Height="6" Margin="2,0,4,0" VerticalAlignment="Top" Fill="#FF797774"
Stretch="Fill"
Data="M4,1 C3.447998,1 3,1.4490051 3,2 L3,3 1,3 1,5 7,5 7,3 5,3 5,2
C5,1.4490051 4.552002,1 4,1 z M4,0 C5.1029968,0 6,0.89700317 6,2 L8,2 8,6
0,6 0,2 2,2 C2,0.89700317 2.8970032,0 4,0 z" />
</Grid>
</DataTemplate>
</syncfusion:RibbonBar.IconTemplate>
</syncfusion:RibbonBar>
```

```

<syncfusion:RibbonBar Header="Font" >
<syncfusion:RibbonBar.IconTemplate>
<DataTemplate>
<Grid>
<Path
Height="4" VerticalAlignment="Bottom" Data="M0,0 L16,0 16,4 0,4 z"
Fill="#FFFE0000" Stretch="Fill" />
<Path
Margin="3.344,0,3.352,5"
Data="M4.6480023,0.95898432 C4.6079937,1.0870056 4.5689923,1.2149963
4.533012,1.34198 4.4980089,1.4909973 4.4510118,1.6419983 4.394005,1.7949829
L2.5330156,6.8809814 6.787006,6.8809814 4.9330055,1.7799988
C4.8510047,1.5699768 4.7659832,1.2969971 4.6790081,0.95898432 z M4.0779959,0
L5.209006,0 9.304,11 8.3170024,11 7.2039977,8.0019836 2.1150171,8.0029907
1.0100081,11 0,11 z"
Fill="{Binding RelativeSource={RelativeSource Mode=Self},
Path=(TextBlock.Foreground)}" Stretch="Fill" />
</Grid>
</DataTemplate>
</syncfusion:RibbonBar.IconTemplate>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Clipboard";
DataTemplate iconDataTemplate = new DataTemplate();
FrameworkElementFactory gridElement = new
FrameworkElementFactory(typeof(Grid));
FrameworkElementFactory pathElement1 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement2 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement3 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement4 = new
FrameworkElementFactory(typeof(Path));
FrameworkElementFactory pathElement5 = new
FrameworkElementFactory(typeof(Path));
gridElement.SetValue(Grid.WidthProperty, (double)14);
gridElement.SetValue(Grid.HeightProperty, (double)16);
pathElement1.SetValue(Path.DataProperty,
Geometry.Parse("F1M122,367L127,367L127,360L122,360z
M128,368L121,368L121,359L128,359z"));
pathElement1.SetValue(Path.MarginProperty, new Thickness(0, 7, 0, 0));

```

```

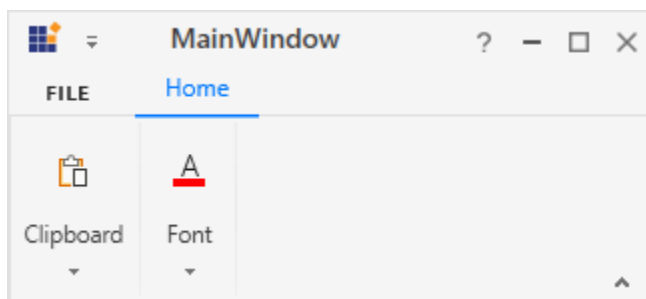
pathElement1.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(58, 58, 56)));
pathElement1.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement1.SetValue(Path.WidthProperty, (double)7);
pathElement1.SetValue(Path.HorizontalAlignmentProperty,
HorizontalAlignment.Right);
pathElement2.SetValue(Path.DataProperty, Geometry.Parse("M0,0 L12,0 12,4
11,4 11,0.99999994 1.0000002,0.99999994 1.0000002,13 6.0000001,13
6.0000001,14 0,14 z"));
pathElement2.SetValue(Path.MarginProperty, new Thickness(0, 2, 2, 0));
pathElement2.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(222, 108, 0)));
pathElement2.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement3.SetValue(Path.DataProperty, Geometry.Parse("M0,0 L10,0 10,3
9.0000001,3 9.0000001,0.99999994 1.0000001,0.99999994 1.0000001,1.5829999
1.0000001,2.5 1.0000001,11 5.0000001,11 5.0000001,12 0,12 z"));
pathElement3.SetValue(Path.MarginProperty, new Thickness(1, 3, 3, 1));
pathElement3.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(248, 219, 143)));
pathElement3.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement4.SetValue(Path.DataProperty, Geometry.Parse("M5.9873815,7.496151
L11.006,7.496151 11.006,14.516999 5.9873815,14.516999 z M0,5.4959998
L3.9880071,5.4964137 3.9880071,13.51695 0,13.51695 z M3.9889999,2.2337155E-
15 C4.8170028,-4.4703477E-08 5.4889999,0.67098993 5.4889999,1.5 L5.4889999,2
7.4889999,2 7.4889999,5 0.4889999,5 0.4889999,2 2.4889999,2 2.4889999,1.5
C2.4889999,0.67098993 3.1609969,-4.4703477E-08 3.9889999,2.23
pathElement4.SetValue(Path.MarginProperty, new Thickness(2.011, 0.5, 0.983,
0.983));
pathElement4.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(250, 250, 250)));
pathElement4.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement5.SetValue(Path.DataProperty, Geometry.Parse("M4,1 C3.447998,1
3,1.4490051 3,2 L3,3 1,3 1,5 7,5 7,3 5,3 5,2 C5,1.4490051 4.552002,1 4,1 z
M4,0 C5.1029968,0 6,0.89700317 6,2 L8,2 8,6 0,6 0,2 2,2 C2,0.89700317
2.8970032,0 4,0 z"));
pathElement5.SetValue(Path.MarginProperty, new Thickness(2, 0, 4, 0));
pathElement5.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(121, 119, 116)));
pathElement5.SetValue(Path.StretchProperty, Stretch.Fill);
pathElement5.SetValue(Path.HeightProperty, (double)6);
pathElement5.SetValue(Path.VerticalAlignmentProperty,
VerticalAlignment.Top);
gridElement.AppendChild(pathElement1);
gridElement.AppendChild(pathElement2);
gridElement.AppendChild(pathElement3);
gridElement.AppendChild(pathElement4);
gridElement.AppendChild(pathElement5);
iconDataTemplate.VisualTree = gridElement;
clipboardBar.IconTemplate = iconDataTemplate;
RibbonBar fontBar = new RibbonBar();
fontBar.Header = "Font";
DataTemplate iconDataTemplate2 = new DataTemplate();
FrameworkElementFactory gridElement2 = new
FrameworkElementFactory(typeof(Grid));
FrameworkElementFactory fontElement1 = new
FrameworkElementFactory(typeof(Path));

```

```

FrameworkElementFactory fontElement2 = new
FrameworkElementFactory(typeof(Path));
fontElement1.SetValue(Path.DataProperty, Geometry.Parse("M0,0 L16,0 16,4 0,4
z"));
fontElement1.SetValue(Path.FillProperty, new
SolidColorBrush(Color.FromRgb(254, 0, 0)));
fontElement1.SetValue(Path.StretchProperty, Stretch.Fill);
fontElement1.SetValue(Path.HeightProperty, (double)4);
fontElement1.SetValue(Path.VerticalAlignmentProperty,
VerticalAlignment.Bottom);
fontElement2.SetValue(Path.DataProperty,
Geometry.Parse("M4.6480023,0.95898432 C4.6079937,1.0870056
4.5689923,1.2149963 4.533012,1.34198 4.4980089,1.4909973 4.4510118,1.6419983
4.394005,1.7949829 L2.5330156,6.8809814 6.787006,6.8809814
4.9330055,1.7799988 C4.8510047,1.5699768 4.7659832,1.2969971
4.6790081,0.95898432 z M4.0779959,0 L5.209006,0 9.304,11 8.3170024,11
7.2039977,8.0019836 2.1150171,8.0029907 1.0100081,11 0,11 z"));
fontElement2.SetValue(Path.MarginProperty, new Thickness(3.344, 0, 3.352,
5));
fontElement2.SetValue(Path.FillProperty, new SolidColorBrush(Colors.Black));
fontElement2.SetValue(Path.StretchProperty, Stretch.Fill);
gridElement2.AppendChild(fontElement1);
gridElement2.AppendChild(fontElement2);
iconDataTemplate2.VisualTree = gridElement2;
clipboardBar.IconTemplate = iconDataTemplate2;
// Adding bars to the tabs
homeTab.Items.Add(clipboardBar);
homeTab.Items.Add(fontBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Note: [View sample in GitHub](#)

Setting image path

The [RibbonBar](#) allows to set the image for the dropdown button using its [CollapseImage](#) property. The [RibbonBar](#) displays the image in 16 * 16 size.

XML

```

<syncfusion:RibbonWindow x:Class="WpfAppl.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

```

```

xmlns:local="clr-namespace:WpfApp1"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="MainWindow" Height="350" Width="600">
<Grid>
<syncfusion:Ribbon VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Clipboard"
CollapseImage="Resources/Pastel6.png"/>
<syncfusion:RibbonBar Header="Font"
CollapseImage="Resources/FontColor.png"/>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

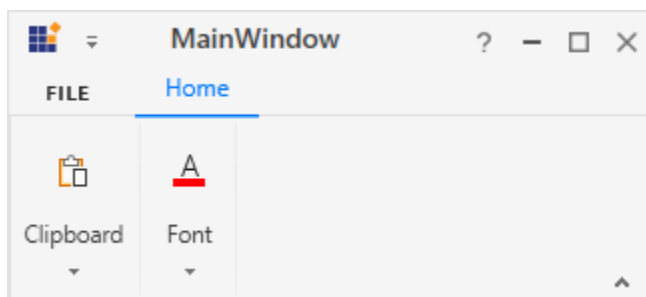
```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
// Creating new tabs
RibbonTab homeTab = new RibbonTab();
homeTab.Caption = "Home";
homeTab.IsChecked = true;
// Creating new bar
RibbonBar clipboardBar = new RibbonBar();
clipboardBar.Header = "Clipboard";
clipboardBar.CollapseImage = new BitmapImage(new
Uri(@"Resources/Pastel6.png", UriKind.RelativeOrAbsolute));
RibbonBar fontBar = new RibbonBar();
fontBar.Header = "Font";
fontBar.CollapseImage = new BitmapImage(new Uri(@"Resources/FontColor.png",
UriKind.RelativeOrAbsolute));
// Adding bars to the tabs
homeTab.Items.Add(clipboardBar);
homeTab.Items.Add(fontBar);
// Adding tabs to ribbon
ribbon.Items.Add(homeTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Grouping RibbonTabs using ContextTabGroups

ContextualTabGroups are used to group the RibbonTabs for easy Navigation. This ContextTabGroups appear when a user enable their context.

Creating ContextTabGroup

This ContextTabGroup can also be kept hidden and shown while required cases like in Word Document's TABLETOOLS ContextTabGroups which gets displayed automatically, while selecting the table. The following code snippet used to create a ContextTabGroup

XML

```
<syncfusion:Ribbon.ContextTabGroups>
<syncfusion:ContextTabGroup Label="Table tools" IsGroupVisible="True"
BackColor="Green">
<syncfusion:RibbonTab Caption="Tables" IsChecked="True" />
<syncfusion:RibbonTab Caption="Design" IsChecked="False" />
</syncfusion:ContextTabGroup>
</syncfusion:Ribbon.ContextTabGroups>
```



Add ContextTabGroup to the simplified layout

When the simplified layout is enabled, the ContextTabGroup can be added and its items will be displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
```



```

mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:Ribbon.ContextTabGroups>
<syncfusion:ContextTabGroup
BackColor="Orange" IsGroupVisible="True" Label="Tools">
<syncfusion:RibbonTab
Caption="Insert"
IsChecked="True">
<syncfusion:RibbonBar
Header="Illustrations"
IsLauncherButtonVisible="True">
<syncfusion:RibbonButton
Label="Picture"
MediumIcon="Resources/Picture20.png"
SizeForm="Large">
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
Label="Comment"
MediumIcon="Resources/New Comment20.png"
SizeForm="Large">
</syncfusion:RibbonButton>
<syncfusion:DropDownButton
Label="Shapes"
MediumIcon="Resources/Insert Shapes20.png"
SizeForm="Small">
</syncfusion:DropDownButton>
<syncfusion:DropDownButton
Label="Chart"
MediumIcon="Resources/Base chart20.png"
SizeForm="Small">
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:ContextTabGroup>
</syncfusion:Ribbon.ContextTabGroups>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

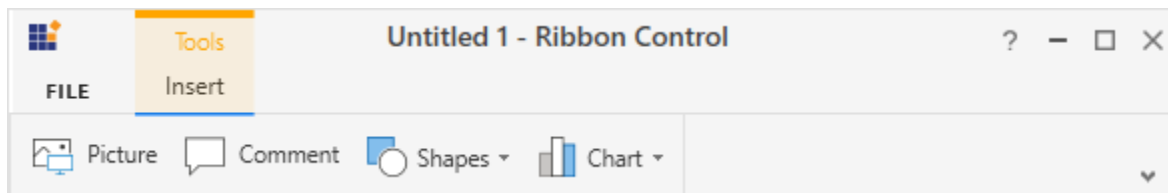
Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
//Creating context tab group
ContextTabGroup contextTab = new ContextTabGroup();
contextTab.Label = "Tools";
contextTab.BackColor = Colors.Orange;
contextTab.IsGroupVisible = true;
// Creating new tabs
RibbonTab insertTab = new RibbonTab();

```

```

insertTab.Caption = "Insert";
insertTab.IsChecked = true;
// Creating new bar
RibbonBar illustrationsBar = new RibbonBar();
illustrationsBar.Header = "Illustrations";
// Creating items
// Creating items
RibbonButton pictureButton = new RibbonButton();
pictureButton.Label = "Picture";
pictureButton.SizeForm = SizeForm.Large;
pictureButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/Picture20.png", UriKind.RelativeOrAbsolute));
RibbonButton commentButton = new RibbonButton();
commentButton.Label = "Comment";
commentButton.SizeForm = SizeForm.Large;
commentButton.MediumIcon = new BitmapImage(new Uri(@"Resources/New
Comment20.png", UriKind.RelativeOrAbsolute));
DropDownButton shapesButton = new DropDownButton();
shapesButton.Label = "Shapes";
shapesButton.SizeForm = SizeForm.Small;
shapesButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Insert
Shapes20.png", UriKind.RelativeOrAbsolute));
DropDownButton chartButton = new DropDownButton();
chartButton.Label = "Chart";
chartButton.SizeForm = SizeForm.Small;
chartButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Base
chart20.png", UriKind.RelativeOrAbsolute));
// Adding items to bar
illustrationsBar.Items.Add(pictureButton);
illustrationsBar.Items.Add(commentButton);
illustrationsBar.Items.Add(shapesButton);
illustrationsBar.Items.Add(chartButton);
// Adding bars to the tabs
insertTab.Items.Add(illustrationsBar);
//Adding ribbon tab to the context tab
contextTab.RibbonTabs.Add(insertTab);
// Adding context tab
ribbon.ContextTabGroups.Add(contextTab);
grid.Children.Add(ribbon);
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



Multiple ContextTabs

To differentiate one ContextTabGroup with one another, change its **BackColor** property of the ContextTabGroup, Since a ContextTabGroup Support to have multiple context Tab.

XML

```
<syncfusion:Ribbon.ContextTabGroups>
```

```
<syncfusion:ContextTabGroup Label="Table Tools" BackColor="Green"
IsGroupVisible="True">
<syncfusion:RibbonTab Caption="Tables" IsChecked="True" />
<syncfusion:RibbonTab Caption="Design" IsChecked="False" />
</syncfusion:ContextTabGroup>
<syncfusion:ContextTabGroup Label="Table Grid" BackColor="Red"
IsGroupVisible="True">
<syncfusion:RibbonTab Caption="Tables" IsChecked="False" />
<syncfusion:RibbonTab Caption="Design" IsChecked="False" />
</syncfusion:ContextTabGroup>
</syncfusion:Ribbon.ContextTabGroups>
```



ContextTabGroup heading

The **Label** property of the ContextTabGroup is used to define the Heading for the ContextTabGroup.

XML

```
<syncfusion:Ribbon.ContextTabGroups>
<syncfusion:ContextTabGroup Label="Table tools" BackColor="Green"
IsGroupVisible="True">
<syncfusion:RibbonTab Caption="Tables" IsChecked="True" />
<syncfusion:RibbonTab Caption="Design" IsChecked="False" />
</syncfusion:ContextTabGroup>
</syncfusion:Ribbon.ContextTabGroups>
```



Changing the visibility at run time

ContextTabGroup visibility can also be changed at the runtime. To change the visibility, enable `IsGroupVisible` property of the ContextTabGroup

XML

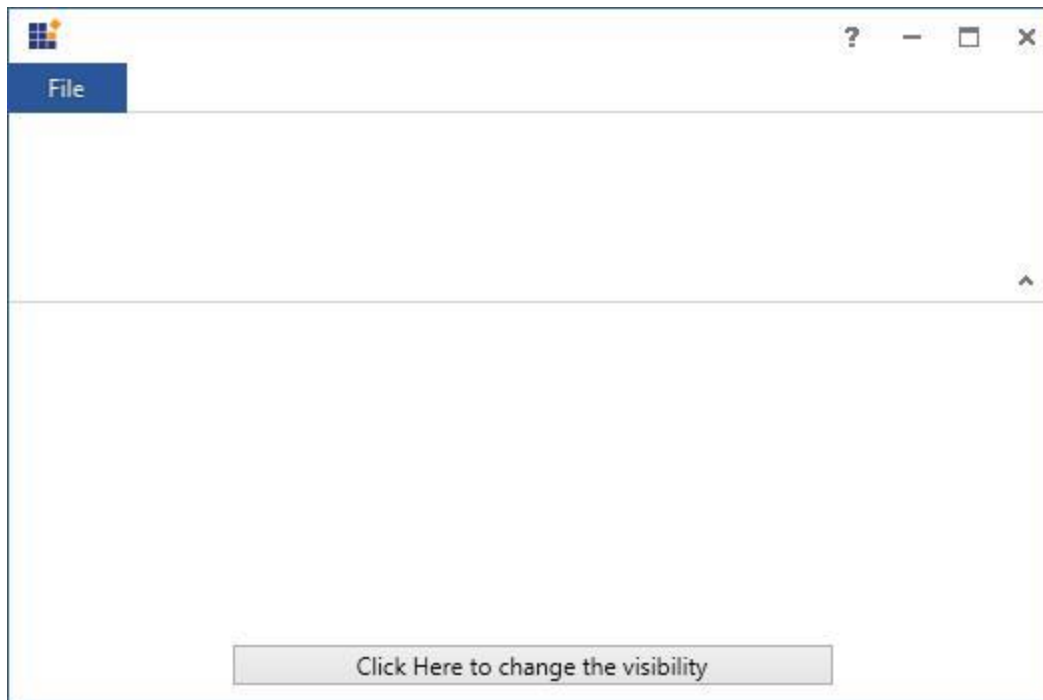
```
<syncfusion:Ribbon VerticalAlignment="Top" >
  <syncfusion:Ribbon.ContextTabGroups>
    <syncfusion:ContextTabGroup x:Name="_contextTabGroup" Label="Table tools"
      BackColor="Red" >
      <syncfusion:RibbonTab Caption="Tables" IsChecked="True" >
      </syncfusion:RibbonTab>
      <syncfusion:RibbonTab Caption="Design" IsChecked="False" >
      </syncfusion:RibbonTab>
    </syncfusion:ContextTabGroup>
  </syncfusion:Ribbon.ContextTabGroups>
</syncfusion:Ribbon>
```

C#

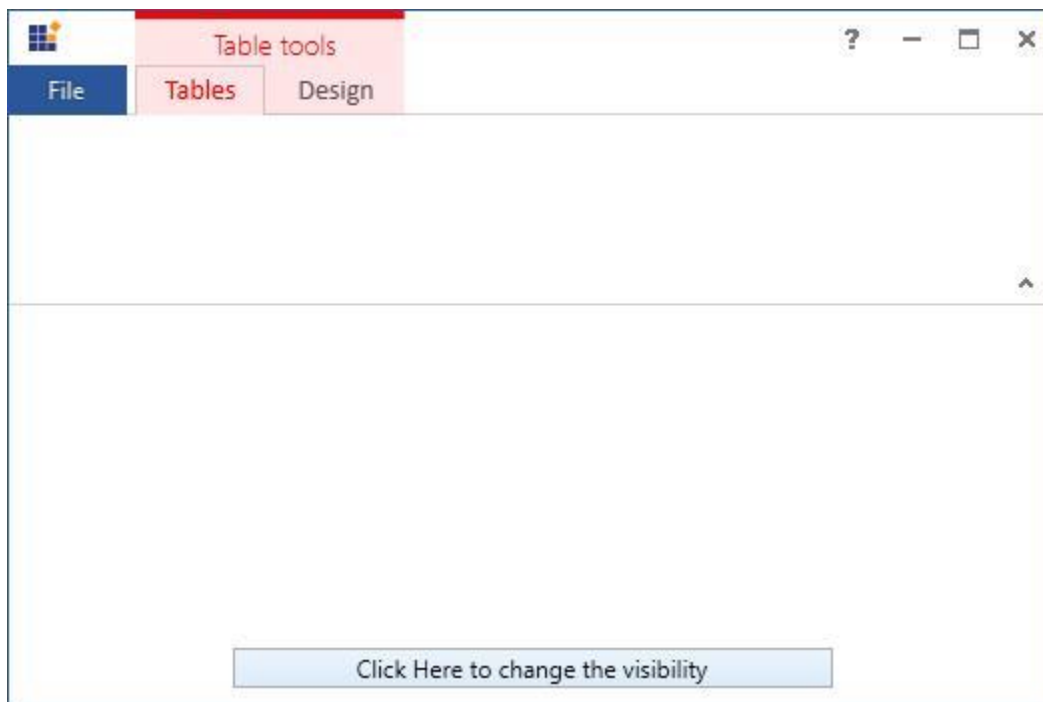
```
private void Button_Click(object sender, RoutedEventArgs e)
{
    _contextTabGroup.IsGroupVisible = true;
}
```

VB.NET

```
Private Sub Button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    _contextTabGroup.IsGroupVisible = True
End Sub
```



After the Button is clicked, the ContextTabGroup visibility changes as follows



Creating ContextTabGroup in MVVM

To create context tab group in Ribbon, use `ContextGroupSource` and `ContextGroupContainerStyle` properties. Similarly we can populate ContextTabGroup to corresponding RibbonTab using `ItemsSource` and `ItemContainerStyle` properties of ContextTabGroup.

To create a ContextTabGroup in MVVM, follow below steps.

1. Create a class for ContextTabGroup, RibbonBar and RibbonItems in Model CS file.

C#

```
public class CustomContextTab : INotifyPropertyChanged
{
    private string tabheader;
    public string TabHeader
    {
        get { return tabheader; }
        set
        {
            tabheader = value;
            RaisePropertyChanged("TabHeader");
        }
    }
    private Color _backcolor;
    public Color BackColor
    {
        get { return _backcolor; }
        set
        {
            _backcolor = value;
            RaisePropertyChanged("BackColor");
        }
    }
    private bool _isgroupvisible;
    public bool IsGroupVisible
    {
        get { return _isgroupvisible; }
        set
        {
            _isgroupvisible = value;
            RaisePropertyChanged("IsGroupVisible");
        }
    }
    public ObservableCollection<CustomRibbonTab> CustomContextRibbonTabs { get; set; }
    public CustomContextTab()
    {
        CustomContextRibbonTabs = new ObservableCollection<CustomRibbonTab>();
    }
    public void RaisePropertyChanged(string name)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(name));
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
}
public class CustomRibbonBar
{
    public string BarHeader { get; set; }
    public ObservableCollection<CustomRibbonItem> CustomRibbonItems { get; set; }
}
```

```
public CustomRibbonBar()
{
    CustomRibbonItems = new ObservableCollection<CustomRibbonItem>();
}
}
public class CustomRibbonItem
{
    public CustomRibbonItem()
    {
        IsSplitButton = false;
        IsBoolean = true;
    }
    public string ItemHeader
    {
        get;
        set;
    }
    public string Image { get; set; }
    public bool IsBoolean { get; set; }
    public bool IsLarge { get; set; }
    public bool IsSplitButton { get; set; }
}
```

VB.NET

```
Public Class CustomContextTab
Inherits INotifyPropertyChanged
Private tabheader As String
Public Property TabHeader As String
Get
Return tabheader
End Get
Set(ByVal value As String)
tabheader = value
RaisePropertyChanged("TabHeader")
End Set
End Property
Private _backcolor As Color
Public Property BackColor As Color
Get
Return _backcolor
End Get
Set(ByVal value As Color)
_backcolor = value
RaisePropertyChanged("BackColor")
End Set
End Property
Private _isgroupvisible As Boolean
Public Property IsGroupVisible As Boolean
Get
Return _isgroupvisible
End Get
Set(ByVal value As Boolean)
_isgroupvisible = value
RaisePropertyChanged("IsGroupVisible")
End Set
```

```

End Property
Public Property CustomContextRibbonTabs As ObservableCollection(Of
CustomRibbonTab)
Public Sub New()
CustomContextRibbonTabs = New ObservableCollection(Of CustomRibbonTab) ()
End Sub
Public Sub RaisePropertyChanged(ByVal name As String)
RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(name))
End Sub
Public Event PropertyChanged As PropertyChangedEventHandler
End Class
Public Class CustomRibbonBar
Public Property BarHeader As String
Public Property CustomRibbonItems As ObservableCollection(Of
CustomRibbonItem)
Public Sub New()
CustomRibbonItems = New ObservableCollection(Of CustomRibbonItem) ()
End Sub
End Class
Public Class CustomRibbonItem
Public Sub New()
IsSplitButton = False
IsBoolean = True
End Sub
Public Property ItemHeader As String
Public Property Image As String
Public Property IsBoolean As Boolean
Public Property IsLarge As Boolean
Public Property IsSplitButton As Boolean
End Class

```

2. Create a View model class and then add ContextTabGroup , RibbonBars and RibbonItems with some properties.

C#

```

public class ViewModel
{
public ObservableCollection<CustomContextTab> CustomContextTabs { get; set; }
}
public ViewModel()
{
CustomContextTabs = new ObservableCollection<CustomContextTab>();
PopulateContextTabs();
}
private void PopulateContextTabs()
{
CustomContextTab contexttab1 = new CustomContextTab() { TabHeader =
"ContextTabGroup1", BackColor=Colors.Red, IsGroupVisible=true };
CustomRibbonTab Tab1 = new CustomRibbonTab() { TabHeader = "TabGroup1" };
PopulateRibbonHomeBars(Tab1);
CustomRibbonTab Tab2 = new CustomRibbonTab() { TabHeader = "TabGroup2" };
PopulateRibbonInsertBars(Tab2);
contexttab1.CustomContextRibbonTabs.Add(Tab1);
contexttab1.CustomContextRibbonTabs.Add(Tab2);
}
}

```



```

CustomContextTabs.Add(contexttab1);
CustomContextTab contexttab2 = new CustomContextTab() { TabHeader =
"ContextTabGroup2", BackColor=Colors.Green, IsGroupVisible=true };
CustomRibbonTab Tab11 = new CustomRibbonTab() { TabHeader = "TabGroup11" };
PopulateRibbonHomeBars(Tab11);
CustomRibbonTab Tab21 = new CustomRibbonTab() { TabHeader = "TabGroup21" };
PopulateRibbonInsertBars(Tab21);
contexttab2.CustomContextRibbonTabs.Add(Tab11);
contexttab2.CustomContextRibbonTabs.Add(Tab21);
CustomContextTabs.Add(contexttab2);
}
void PopulateRibbonHomeBars(CustomRibbonTab Tab)
{
CustomRibbonBar Bar1 = new CustomRibbonBar() { BarHeader = "Clipboard" };
PopulateRibbonNewItemS(Bar1);
CustomRibbonBar Bar2 = new CustomRibbonBar() { BarHeader = "Editing" };
PopulateRibbonEditingItems(Bar2);
Tab.CustomRibbonBars.Add(Bar1);
Tab.CustomRibbonBars.Add(Bar2);
}
void PopulateRibbonNewItemS(CustomRibbonBar Bar)
{
CustomRibbonItem Item1 = new CustomRibbonItem() { ItemHeader = "Paste",
IsLarge = true, Image = "Paste32.png" };
CustomRibbonItem Item2 = new CustomRibbonItem() { ItemHeader = "Cut", Image
= "Cut16.png" };
CustomRibbonItem Item3 = new CustomRibbonItem() { ItemHeader = "Copy", Image
= "Copy16.png" };
CustomRibbonItem Item4 = new CustomRibbonItem() { ItemHeader = "Format
Painter", Image = "FormatPainter16.png" };
Bar.CustomRibbonItems.Add(Item1);
Bar.CustomRibbonItems.Add(Item2);
Bar.CustomRibbonItems.Add(Item3);
Bar.CustomRibbonItems.Add(Item4);
}
private void PopulateRibbonEditingItems(CustomRibbonBar Bar)
{
CustomRibbonItem Item1 = new CustomRibbonItem() { ItemHeader = "Hyperlink",
IsLarge = true, Image = "hyperlink32.png" };
CustomRibbonItem Item2 = new CustomRibbonItem() { ItemHeader = "Replace",
IsLarge = true, Image = "replace_32.png" };
CustomRibbonItem Item3 = new CustomRibbonItem() { ItemHeader = "Zoom",
IsLarge = true, Image = "Zoom_32x32.png" };
Bar.CustomRibbonItems.Add(Item1);
Bar.CustomRibbonItems.Add(Item2);
Bar.CustomRibbonItems.Add(Item3);
}
private void PopulateRibbonInsertBars(CustomRibbonTab Tab)
{
CustomRibbonBar Bar2 = new CustomRibbonBar() { BarHeader = "Mail" };
PopulateRibbonMailItems(Bar2);
CustomRibbonBar Bar1 = new CustomRibbonBar() { BarHeader = "Tables" };
PopulateRibbonTablesItems(Bar1);
Tab.CustomRibbonBars.Add(Bar2);
Tab.CustomRibbonBars.Add(Bar1);
}
private void PopulateRibbonMailItems(CustomRibbonBar Bar)

```

```

{
CustomRibbonItem Item1 = new CustomRibbonItem() { ItemHeader = "Attach
File", IsLarge = true, Image = "base_paperclip_32.png", IsSplitButton=true };
CustomRibbonItem Item2 = new CustomRibbonItem() { ItemHeader = "Business
card", IsLarge = true, Image = "base_business_contacts.png" };
CustomRibbonItem Item3 = new CustomRibbonItem() { ItemHeader = "Audio",
IsLarge = true, Image = "base_speaker_32.png" };
Bar.CustomRibbonItems.Add(Item1);
Bar.CustomRibbonItems.Add(Item2);
Bar.CustomRibbonItems.Add(Item3);
}
private void PopuplateRibbonTablesItems(CustomRibbonBar Bar)
{
CustomRibbonItem Item1 = new CustomRibbonItem() { ItemHeader = "Tables",
IsLarge = true, IsSplitButton = true, Image = "Table_32.png" };
Bar.CustomRibbonItems.Add(Item1);
}
}

```

VB.NET

```

Public Class ViewModel
Public Property CustomContextTabs As ObservableCollection(Of
CustomContextTab)
Public Sub New()
CustomContextTabs = New ObservableCollection(Of CustomContextTab) ()
PopulateContextTabs ()
End Sub
Private Sub PopulateContextTabs ()
Dim contexttab1 As CustomContextTab = New CustomContextTab() With {
.TabHeader = "ContextTabGroup1",
.BackColor = Colors.Red,
.IsGroupVisible = True
}
Dim Tab1 As CustomRibbonTab = New CustomRibbonTab() With {
.TabHeader = "TabGroup1"
}
PopulateRibbonHomeBars (Tab1)
Dim Tab2 As CustomRibbonTab = New CustomRibbonTab() With {
.TabHeader = "TabGroup2"
}
PopulateRibbonInsertBars (Tab2)
contexttab1.CustomContextRibbonTabs.Add (Tab1)
contexttab1.CustomContextRibbonTabs.Add (Tab2)
CustomContextTabs.Add (contexttab1)
Dim contexttab2 As CustomContextTab = New CustomContextTab() With {
.TabHeader = "ContextTabGroup2",
.BackColor = Colors.Green,
.IsGroupVisible = True
}
Dim Tab11 As CustomRibbonTab = New CustomRibbonTab() With {
.TabHeader = "TabGroup11"
}
PopulateRibbonHomeBars (Tab11)
Dim Tab21 As CustomRibbonTab = New CustomRibbonTab() With {
.TabHeader = "TabGroup21"
}

```

```
}
PopulateRibbonInsertBars(Tab21)
contexttab2.CustomContextRibbonTabs.Add(Tab11)
contexttab2.CustomContextRibbonTabs.Add(Tab21)
CustomContextTabs.Add(contexttab2)
End Sub
Private Sub PopulateRibbonHomeBars(ByVal Tab As CustomRibbonTab)
Dim Bar1 As CustomRibbonBar = New CustomRibbonBar() With {
    .BarHeader = "Clipboard"
}
PopulateRibbonNewItem(Bar1)
Dim Bar2 As CustomRibbonBar = New CustomRibbonBar() With {
    .BarHeader = "Editing"
}
PopulateRibbonEditingItems(Bar2)
Tab.CustomRibbonBars.Add(Bar1)
Tab.CustomRibbonBars.Add(Bar2)
End Sub
Private Sub PopulateRibbonNewItem(ByVal Bar As CustomRibbonBar)
Dim Item1 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Paste",
    .IsLarge = True,
    .Image = "Paste32.png"
}
Dim Item2 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Cut",
    .Image = "Cut16.png"
}
Dim Item3 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Copy",
    .Image = "Copy16.png"
}
Dim Item4 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Format Painter",
    .Image = "FormatPainter16.png"
}
Bar.CustomRibbonItems.Add(Item1)
Bar.CustomRibbonItems.Add(Item2)
Bar.CustomRibbonItems.Add(Item3)
Bar.CustomRibbonItems.Add(Item4)
End Sub
Private Sub PopulateRibbonEditingItems(ByVal Bar As CustomRibbonBar)
Dim Item1 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Hyperlink",
    .IsLarge = True,
    .Image = "hyperlink32.png"
}
Dim Item2 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Replace",
    .IsLarge = True,
    .Image = "replace_32.png"
}
Dim Item3 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Zoom",
    .IsLarge = True,
    .Image = "Zoom_32x32.png"
}
```

```

Bar.CustomRibbonItems.Add(Item1)
Bar.CustomRibbonItems.Add(Item2)
Bar.CustomRibbonItems.Add(Item3)
End Sub
Private Sub PopulateRibbonInsertBars(ByVal Tab As CustomRibbonTab)
Dim Bar2 As CustomRibbonBar = New CustomRibbonBar() With {
    .BarHeader = "Mail"
}
PopulateRibbonMailItems(Bar2)
Dim Bar1 As CustomRibbonBar = New CustomRibbonBar() With {
    .BarHeader = "Tables"
}
PopulateRibbonTablesItems(Bar1)
Tab.CustomRibbonBars.Add(Bar2)
Tab.CustomRibbonBars.Add(Bar1)
End Sub
Private Sub PopulateRibbonMailItems(ByVal Bar As CustomRibbonBar)
Dim Item1 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Attach File",
    .IsLarge = True,
    .Image = "base_paperclip_32.png",
    .IsSplitButton = True
}
Dim Item2 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Business card",
    .IsLarge = True,
    .Image = "base_business_contacts.png"
}
Dim Item3 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Audio",
    .IsLarge = True,
    .Image = "base_speaker_32.png"
}
Bar.CustomRibbonItems.Add(Item1)
Bar.CustomRibbonItems.Add(Item2)
Bar.CustomRibbonItems.Add(Item3)
End Sub
Private Sub PopulateRibbonTablesItems(ByVal Bar As CustomRibbonBar)
Dim Item1 As CustomRibbonItem = New CustomRibbonItem() With {
    .ItemHeader = "Tables",
    .IsLarge = True,
    .IsSplitButton = True,
    .Image = "Table_32.png"
}
Bar.CustomRibbonItems.Add(Item1)
End Sub
End Class

```

3. Now bind the properties of View model in XAML.

XML

```

<DataTemplate x:Key="Ribbonbutton">
<syncfusion:RibbonButton Label="{Binding ItemHeader}" SizeForm="{Binding
IsLarge, Converter={StaticResource sizeform}}" Visibility="{Binding

```

```

IsBoolean, Converter={StaticResource visibility}}" LargeIcon="{Binding
Image,Converter={StaticResource image}}" SmallIcon="{Binding
Image,Converter={StaticResource image}}"/>
</DataTemplate>
<DataTemplate x:Key="Splitbutton">
<syncfusion:SplitButton Label="{Binding ItemHeader}" SizeForm="{Binding
IsLarge, Converter={StaticResource sizeform}}" Visibility="{Binding
IsBoolean, Converter={StaticResource visibility}}" LargeIcon="{Binding
Image,Converter={StaticResource image}}" SmallIcon="{Binding
Image,Converter={StaticResource image}}"/>
</DataTemplate>
<syncfusion:Ribbon VerticalAlignment="Top"
Name="Ribbon1" RibbonBarCollapseImage="App.ico"
ItemsSource="{Binding CustomRibbonTabs}" ContextGroupSource="{Binding
CustomContextTabs}" >
<syncfusion:Ribbon.ItemContainerStyle>
<Style TargetType="{x:Type syncfusion:RibbonTab}">
<Setter Property="Caption" Value="{Binding TabHeader}"></Setter>
<Setter Property="ItemsSource" Value="{Binding CustomRibbonBars}" />
<Setter Property="ItemContainerStyle">
<Setter.Value>
<Style BasedOn="{StaticResource Office2013RibbonBarStyle}"
TargetType="{x:Type syncfusion:RibbonBar}">
<Setter Property="Header" Value="{Binding BarHeader}" />
<Setter Property="ItemsSource" Value="{Binding CustomRibbonItems}" />
<Setter Property="ItemTemplateSelector" Value="{StaticResource selector}" />
</Style>
</Setter.Value>
</Setter>
</Style>
</syncfusion:Ribbon.ItemContainerStyle>
<syncfusion:Ribbon.ContextGroupContainerStyle>
<Style TargetType="syncfusion:ContextTabGroup">
<Setter Property="Label" Value="{Binding TabHeader}" />
<Setter Property="BackColor" Value="{Binding BackColor, Mode=TwoWay}" />
<Setter Property="IsGroupVisible" Value="{Binding IsGroupVisible,
Mode=TwoWay}" />
<Setter Property="ItemsSource" Value="{Binding CustomContextRibbonTabs}" />
<Setter Property="ItemContainerStyle">
<Setter.Value>
<Style TargetType="syncfusion:RibbonTab">
<Setter Property="Caption" Value="{Binding TabHeader}" />
<Setter Property="ItemsSource" Value="{Binding CustomRibbonBars}" />
<Setter Property="ItemContainerStyle">
<Setter.Value>
<Style BasedOn="{StaticResource Office2013RibbonBarStyle}"
TargetType="{x:Type syncfusion:RibbonBar}">
<Setter Property="Header" Value="{Binding BarHeader}" />
<Setter Property="ItemsSource" Value="{Binding CustomRibbonItems}" />
<Setter Property="ItemTemplateSelector" Value="{StaticResource selector}" />
<Setter Property="CollapseImage" Value="/Images/FormatPainter16.png" />
</Style>
</Setter.Value>
</Setter>
</Style>
</Setter.Value>
</Setter>

```

```
</Style>
</syncfusion:Ribbon.ContextGroupContainerStyle>
</syncfusion:Ribbon>
```

4. Add a converter class for binding conversions

C#

```
public class BooltoSizeformConverter: IValueConverter
{
    #region IValueConverter Members
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        if (value.Equals(true))
        {
            return "Large";
        }
        else
        {
            return "Small";
        }
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
    #endregion
}
public class BooltoVisibilityConverter : IValueConverter
{
    #region IValueConverter Members
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        if (value.Equals(true))
        {
            return Visibility.Visible;
        }
        else
        {
            return Visibility.Collapsed;
        }
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
    #endregion
}
public class ItemDataTemplateSelector : DataTemplateSelector
{

```

```

public override DataTemplate SelectTemplate(object item, DependencyObject
container)
{
    FrameworkElement element = container as FrameworkElement;
    if (element != null && item != null )
    {
        if (item is CustomRibbonItem && (item as CustomRibbonItem).IsSplitButton)
        {
            return
            element.FindResource("Splitbutton") as DataTemplate;
        }
        else
        {
            return element.FindResource("Ribbonbutton") as DataTemplate;
        }
    }
    return null;
}

public class ImageConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
    {
        if (value != null)
        {
            string str = value.ToString();
            return "../Images/" + str;
        }
        else
        {
            return value;
        }
    }
    public object ConvertBack(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

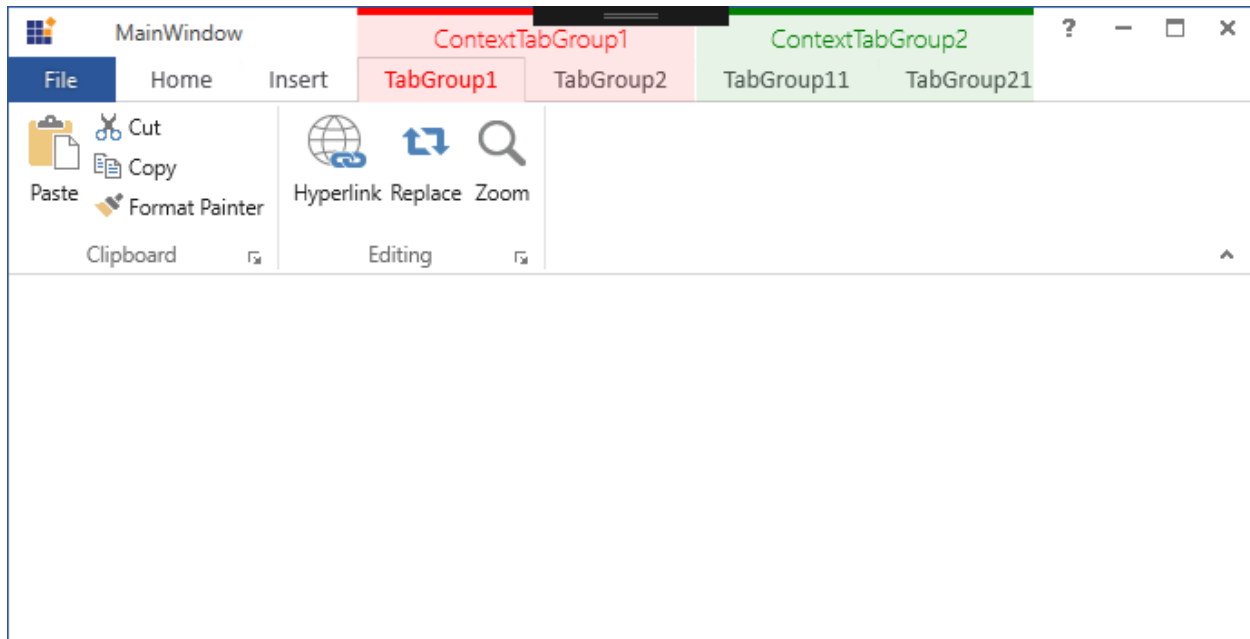
VB.NET

```

Public Class BooltoSizeformConverter
    Inherits IValueConverter
    Public Function Convert(ByVal value As Object, ByVal targetType As Type,
    ByVal parameter As Object, ByVal culture As
    System.Globalization.CultureInfo) As Object
    If value.Equals(True) Then
    Return "Large"
    Else
    Return "Small"
    End If
    End Function
    Public Function ConvertBack(ByVal value As Object, ByVal targetType As Type,
    ByVal parameter As Object, ByVal culture As
    System.Globalization.CultureInfo) As Object
    Throw New NotImplementedException()
    End Function
End Class

```

```
End Function
End Class
Public Class BooltoVisibilityConverter
Inherits IValueConverter
Public Function Convert(ByVal value As Object, ByVal targetType As Type,
ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object
If value.Equals(True) Then
Return Visibility.Visible
Else
Return Visibility.Collapsed
End If
End Function
Public Function ConvertBack(ByVal value As Object, ByVal targetType As Type,
ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object
Throw New NotImplementedException()
End Function
End Class
Public Class ItemDataTemplateSelector
Inherits DataTemplateSelector
Public Overrides Function SelectTemplate(ByVal item As Object, ByVal
container As DependencyObject) As DataTemplate
Dim element As FrameworkElement = TryCast(container, FrameworkElement)
If element IsNot Nothing AndAlso item IsNot Nothing Then
If TypeOf item Is CustomRibbonItem AndAlso (TryCast(item,
CustomRibbonItem)).IsSplitButton Then
Return TryCast(element.FindResource("Splitbutton"), DataTemplate)
Else
Return TryCast(element.FindResource("Ribbonbutton"), DataTemplate)
End If
End If
Return Nothing
End Function
End Class
Public Class ImageConverter
Inherits IValueConverter
Public Function Convert(ByVal value As Object, ByVal targetType As Type,
ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object
If value IsNot Nothing Then
Dim str As String = value.ToString()
Return "../Images/" & str
Else
Return value
End If
End Function
Public Function ConvertBack(ByVal value As Object, ByVal targetType As Type,
ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object
Throw New NotImplementedException()
End Function
End Class
```

Detecting selection changes in RibbonTab

The [SelectionChanging](#) event notifies when the user attempts to switch tab in the Ribbon control. The [SelectionChanging](#) event receives an argument of the type **CancelEventArgs** that allows us to cancel the switching operation.

XML

```
<syncfusion:Ribbon x:Name="ribbon" VerticalAlignment="Top"
SelectionChanging="Ribbon_SelectionChanging">
<syncfusion:RibbonTab Caption="HOME" IsChecked="True"/>
<syncfusion:RibbonTab Caption="SEND/RCEIVE" IsChecked="False"/>
<syncfusion:RibbonTab Caption="FOLDER" IsChecked="False"/>
<syncfusion:RibbonTab Caption="VIEW" IsChecked="False"/>
</syncfusion:Ribbon>
```

C#

```
private void Ribbon_SelectionChanging(object sender,
System.ComponentModel.CancelEventArgs e)
{
e.Cancel = true;
}
```

VB.NET

```
Private Sub Ribbon_SelectionChanging(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs)
e.Cancel = True
End Sub
```

Dealing with Ribbon Items in WPF Ribbon

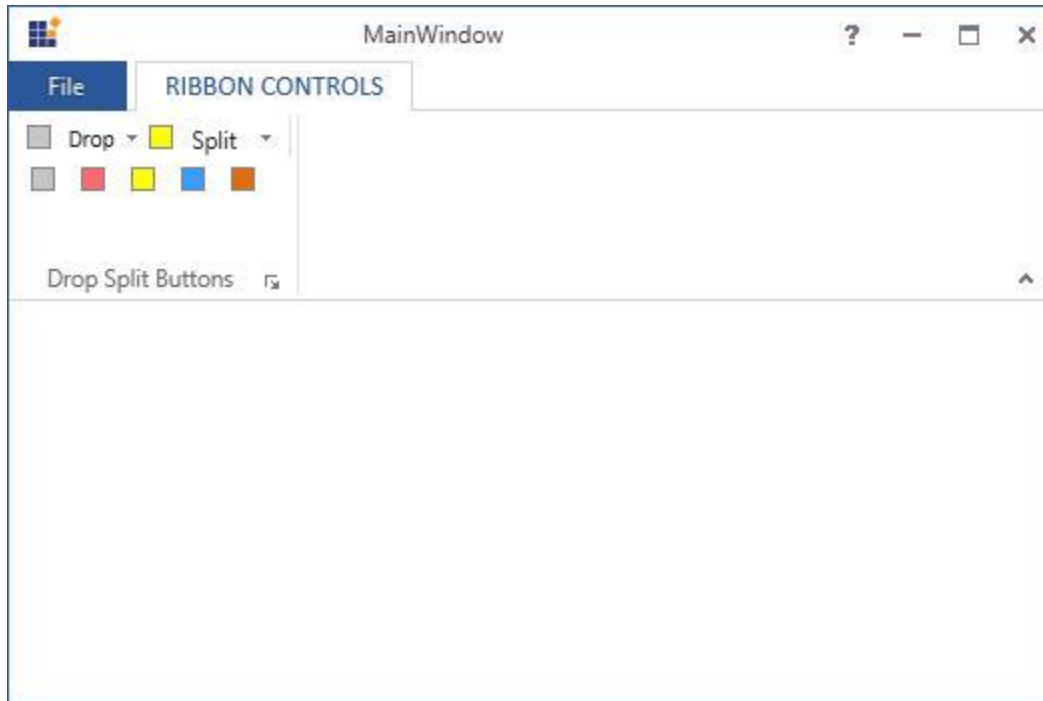
Button Panel represents a panel control that can display horizontal row of Ribbon Items in a single border. It is used to manage the place for the controls.

Adding items to ButtonPanel in XAML

Use the below code to create ButtonPanel in XAML

XML

```
<syncfusion:Ribbon VerticalAlignment="Top" x:Name="_ribbon">
<syncfusion:RibbonTab Caption="RIBBON CONTROLS" >
<syncfusion:RibbonBar Header="Drop Split Buttons">
<syncfusion:ButtonPanel>
<syncfusion:DropDownButton Label=" Drop "
SizeForm="Small"
SmallIcon="Images/Gray.png" />
<syncfusion:SplitButton Label=" Split "
SizeForm="Small"
SmallIcon="Images/Yellow.png" />
</syncfusion:ButtonPanel >
<syncfusion:ButtonPanel SeparatorVisibility="Collapsed">
<syncfusion:RibbonButton Label="Simple"
SizeForm="ExtraSmall"
SmallIcon="Images/Gray.png" />
<syncfusion:RibbonButton IsToggle="True"
Label="Toggle"
SizeForm="ExtraSmall"
SmallIcon="Images/Red.png" />
<syncfusion:RibbonButton Label="Simple"
SizeForm="ExtraSmall"
SmallIcon="Images/Yellow.png" />
<syncfusion:RibbonButton IsToggle="True"
Label="Toggle"
SizeForm="ExtraSmall"
SmallIcon="Images/Blue.png" />
<syncfusion:RibbonButton IsToggle="True"
Label="Toggle"
SizeForm="ExtraSmall"
SmallIcon="Images/Orange.png" />
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
```



Adding items to ButtonPanel in code behind

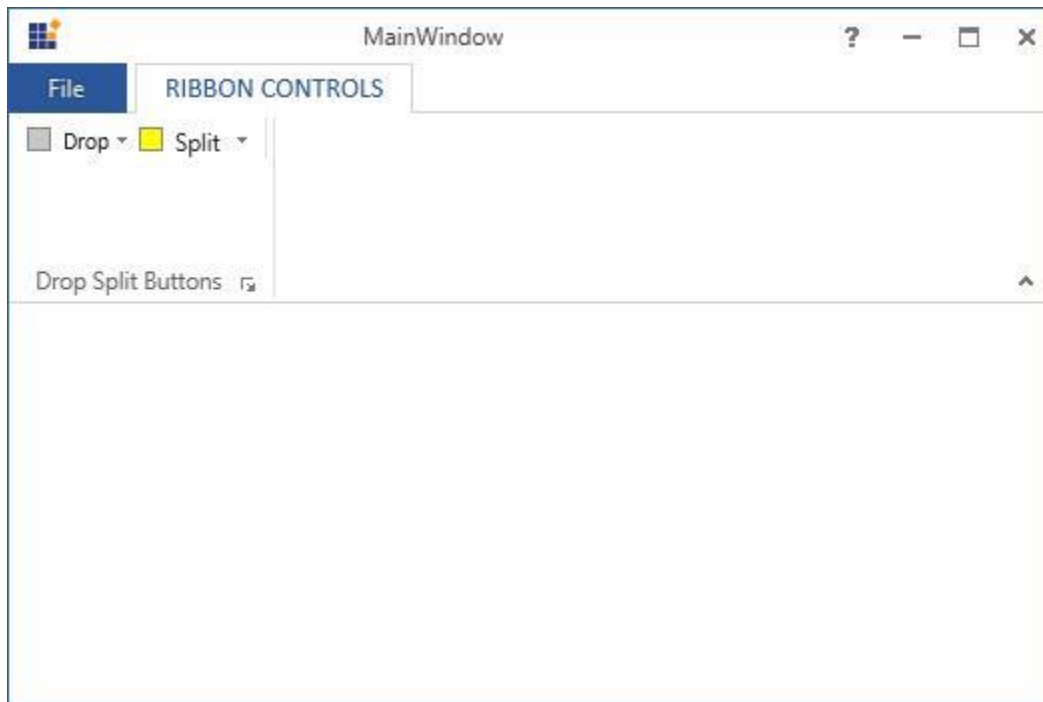
To create the ButtonPanel control in Code Behind, use the below code.

C#

```
DropDownButton _dropDownButton = new DropDownButton() { Label = "Drop",  
    SizeForm = Syncfusion.Windows.Tools.SizeForm.Small, SmallIcon = new  
    BitmapImage(new Uri("/Resources/Gray.png", UriKind.Relative)) };  
SplitButton _splitButton = new SplitButton() { Label = "Split", SizeForm =  
    Syncfusion.Windows.Tools.SizeForm.Small, SmallIcon = new BitmapImage(new  
    Uri("/Resources/Yellow.png", UriKind.Relative)) };  
ButtonPanel _buttonPanel = new ButtonPanel();  
_buttonPanel.Items.Add(_dropDownButton);  
_buttonPanel.Items.Add(_splitButton);
```

VB.NET

```
Dim _dropDownButton As New DropDownButton() With {  
    .Label = "Drop",  
    .SizeForm = Syncfusion.Windows.Tools.SizeForm.Small,  
    .SmallIcon = New BitmapImage(New Uri("/Resources/Gray.png",  
    UriKind.Relative))  
}  
Dim _splitButton As New SplitButton() With {  
    .Label = "Split",  
    .SizeForm = Syncfusion.Windows.Tools.SizeForm.Small,  
    .SmallIcon = New BitmapImage(New Uri("/Resources/Yellow.png",  
    UriKind.Relative))  
}  
Dim _buttonPanel As New ButtonPanel()  
_buttonPanel.Items.Add(_dropDownButton)  
_buttonPanel.Items.Add(_splitButton)
```



Changing size of ribbon items

SizeForm is used to set the size of the Ribbon items that are added inside the Ribbon control.

Possible values of **SizeForm** are Large, Small and ExtraSmall.

List of controls which support SizeForm

- RibbonButton
- DropDownButton
- SplitButton

The code to set SizeForm for the above controls is illustrated below

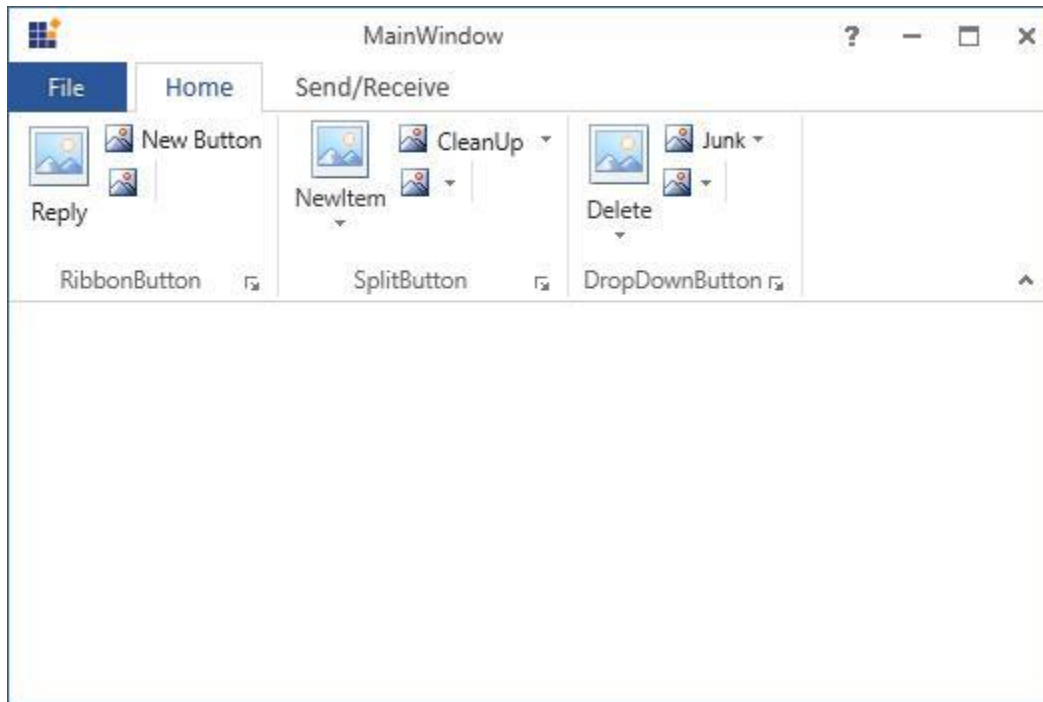
XML

```
<syncfusion:Ribbon VerticalAlignment="Top" >
<syncfusion:RibbonTab Caption="Home" IsChecked="True">
<syncfusion:RibbonBar Header="RibbonButton">
<syncfusion:RibbonButton SizeForm="Large" Label="Reply"/>
<syncfusion:RibbonButton SizeForm="Small" />
<syncfusion:ButtonPanel>
<syncfusion:RibbonButton SizeForm="ExtraSmall" Label="ReplyAll"/>
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
<syncfusion:RibbonBar Header="SplitButton">
<syncfusion:SplitButton SizeForm="Large" Label="NewItem"/>
<syncfusion:SplitButton SizeForm="Small" Label="CleanUp"/>
<syncfusion:ButtonPanel>
<syncfusion:SplitButton SizeForm="ExtraSmall" />
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
```

```

<syncfusion:RibbonBar Header="DropDownButton">
<syncfusion:DropDownButton SizeForm="Large" Label="Delete"/>
<syncfusion:DropDownButton SizeForm="Small" Label="Junk"/>
<syncfusion:ButtonPanel>
<syncfusion:DropDownButton SizeForm="ExtraSmall" />
</syncfusion:ButtonPanel>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Send/Receive" IsChecked="False"/>
</syncfusion:Ribbon>

```



Add command to RibbonLauncherButton

Ribbon instance now provides command support for RibbonBar LauncherButton. It provides the following options.

- Setting LauncherButton Command
- Setting LauncherButton CommandTarget
- Setting LauncherButton CommandParameter

Set RibbonBar LauncherButton Command

Command for RibbonBar LauncherButton can be set by using **LauncherCommand** attached property. The following code example illustrates this.

XML

```

<syncfusion:Ribbon VerticalAlignment="Top" >
<syncfusion:RibbonTab Caption="Home" IsChecked="False">
<syncfusion:RibbonBar Header="Acions" IsLauncherButtonVisible="True"
syncfusion:RibbonBar.LauncherCommand="EditingCommands.IncreaseFontSize"
syncfusion:RibbonBar.LauncherCommandTarget="{Binding ElementName=Editor}">

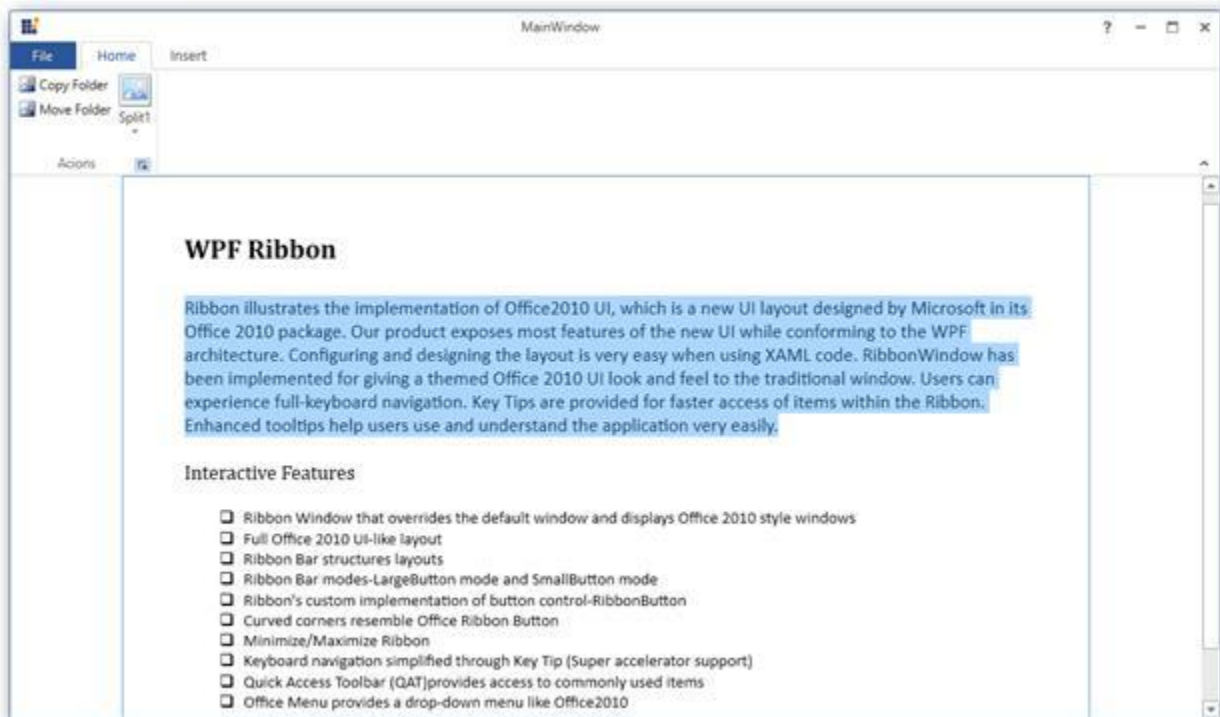
```

```

<syncfusion:RibbonButton SizeForm="Small" Label="Copy Folder"/>
<syncfusion:RibbonButton SizeForm="Small" Label="Move Folder"/>
<syncfusion:SplitButton Label=" Split1 " SizeForm="Large" >
<syncfusion:RibbonButton SizeForm="Small" Label="Mark to Download"/>
<syncfusion:RibbonButton SizeForm="Small" Label="UnMark to Download"/>
</syncfusion:SplitButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert" />
</syncfusion:Ribbon>

```

The font size for the selected text is increased when launcher button is clicked



Show HelpButton in RibbonWindow

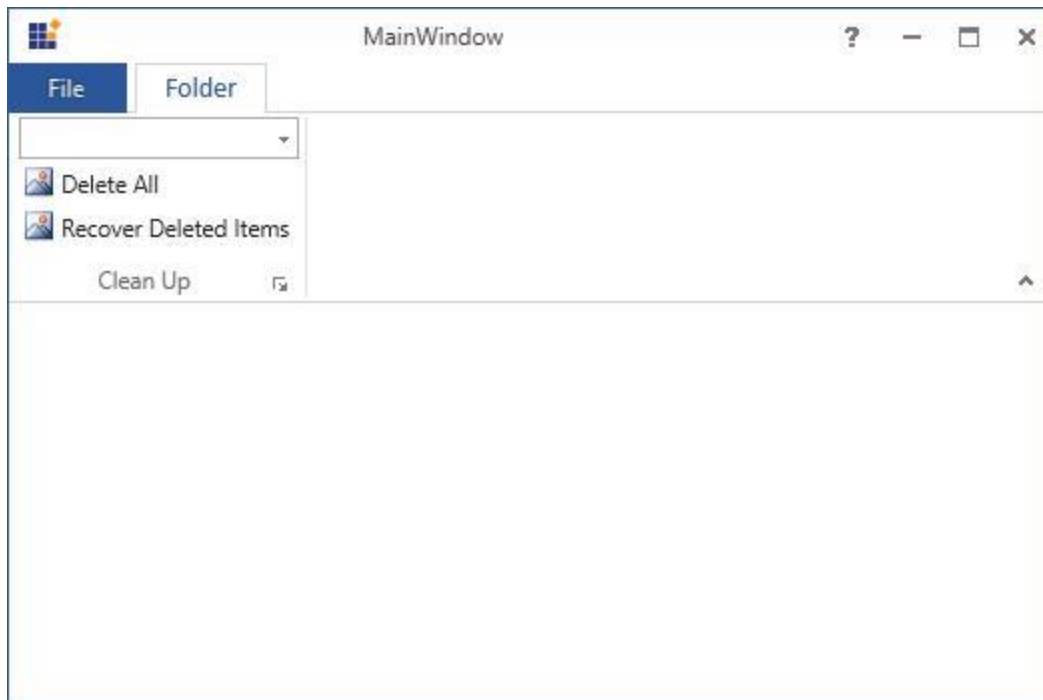
The HelpButton displays the information about the application. To enable this HelpButton in Office2013 theme, set ShowHelpButton as True

XML

```

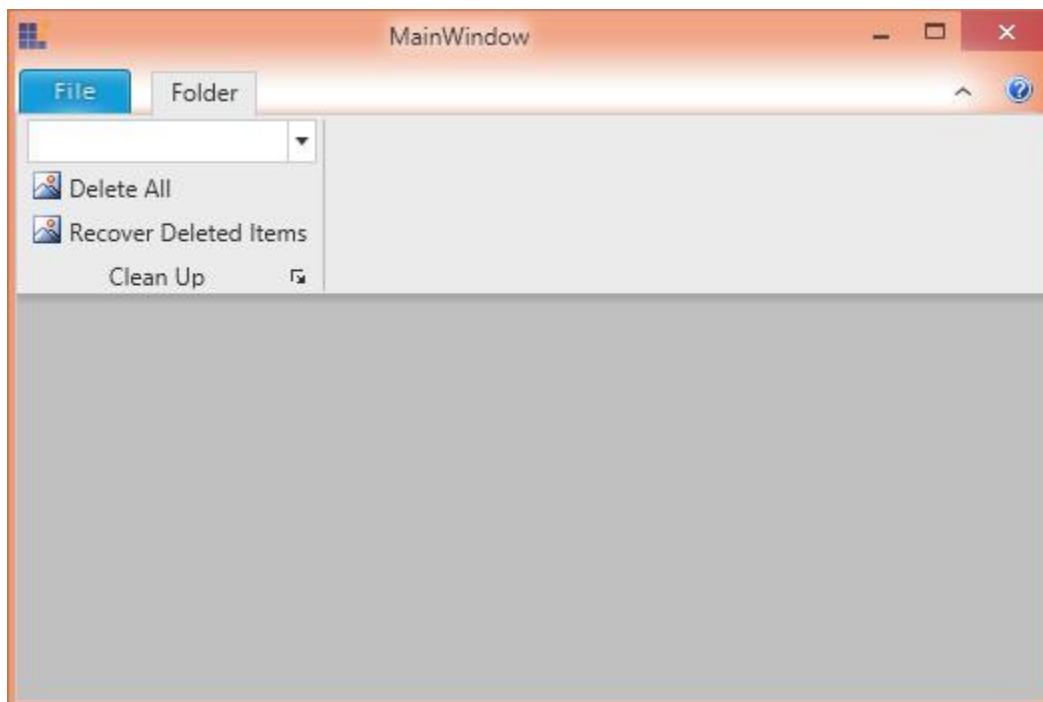
<syncfusion:RibbonWindow
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="RibbonButtonPanel.MainWindow"
Title="MainWindow" Height="350" Width="525"
syncfusion:SkinStorage.VisualStudio="Office2013" ShowHelpButton="True"
x:Name="ribbonWindow"/>

```



To set the HelpButton at the right of the RibbonWindow for the themes except Office2013, use `TabPanelItem` property of the Ribbon

The `TabPanelItem` is located at the right corner below the main window close button in the following screen shot.



This following code snippet explains how to create and configure `TabPanelItem`.

XML

```
<syncfusion:Ribbon.TabPanelItem>  
<syncfusion:RibbonButton SizeForm="ExtraSmall" SmallIcon="help16.png" />  
</syncfusion:Ribbon.TabPanelItem>
```

Serialization and Deserialization in WPF Ribbon

State persistence is a combined process of serialization and deserialization. Serialization is the process of converting the state of the object to the format in which it can be stored as a file in the memory.

Deserialization is a complement process of serialization that converts into the object from the stored state information.

The Ribbon control has built-in serialization support to serialize the entire Ribbon control state and details of the layout mode. It also provides supports to save and load the Ribbon at any time while running the application, either with simplified or normal layout.

The following ribbon control states can persist separately.

1. Quick Access Tool Bar

- a. Quick Access Tool Bar Items
- b. Quick Access Tool Bar State (Above Ribbon, Below Ribbon)

2. Ribbon

- a. Ribbon State (Normal, Hide)
- b. Layout Mode (Normal, Simplified)

3. Ribbon Window

- a. Window Width
- b. Window Height
- c. Window Left
- d. Window Top

Use case scenarios

The Ribbon State Persistence feature helps users to load the state of the Ribbon control that existed when the application was closed. State Persistence feature gives a more consistent workflow for an application that is executed for a long time.

Persist ribbon states at application exit and load

To persist the Ribbon States at application exit and load, use `AutoPersist` property. It is also possible to handle the persisting states in Ribbon control by handling the `AutoPersist` property individually in Ribbon, QAT and Ribbon window.

The following code snippet shows how to handle the property in Ribbon elements.

XML

```
<syncfusion:RibbonWindow  
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="RibbonStatePersistence.MainWindow"
Title="MainWindow" Height="350" Width="525" AutoPersist="True"
x:Name="RibbonWindow" WindowStyle="SingleBorderWindow"
WindowStartupLocation="CenterScreen" WindowState="Normal"
syncfusion:SkinStorage.VisualStudio="Office2013"
Loaded="RibbonWindow_Loaded">
<Grid>
<syncfusion:Ribbon x:Name="MyRibbon" AutoPersist="True"
VerticalAlignment="Top">
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar AutoPersist="True">
</syncfusion:QuickAccessToolBar>
</syncfusion:Ribbon.QuickAccessToolBar>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

Persist ribbon states any time while running the application

WPF Ribbon control supports the persistence of its states any time while running the application. Ribbon states can be saved and loaded at any time by using `SaveRibbonState` and `LoadRibbonState` methods

Before calling the methods, it is important to specify the persisting Ribbon elements in `PersistElements` collection. This collection can be changed at any time. Save and Load states at runtime are fully based on this collection details. The following code snippet shows how to add Ribbon elements that are required to retain its state.

C#

```

private void RibbonWindow_Loaded(object sender, RoutedEventArgs e)
{
    this.MyRibbon.PersistElements.Add(RibbonElements.Ribbon);
    this.MyRibbon.PersistElements.Add(RibbonElements.RibbonWindow);
    this.MyRibbon.PersistElements.Add(RibbonElements.QuickAccessToolbar);
}

```

VB.NET

```

Private Sub RibbonWindow_Loaded(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    Me.MyRibbon.PersistElements.Add(RibbonElements.Ribbon)
    Me.MyRibbon.PersistElements.Add(RibbonElements.RibbonWindow)
    Me.MyRibbon.PersistElements.Add(RibbonElements.QuickAccessToolbar)
End Sub

```

Saving ribbon states

Ribbon State can be saved and loaded dynamically at run time. To save the current Ribbon State, use `SaveRibbonState` method in Ribbon.

This method has two overloaded methods for customizing the Save state process as follows:

1. void `SaveRibbonState()`

2. void SaveRibbonState(IsolatedStorageFile isoStorage, string storeFileName)

In the first method, there is no parameters. Current state of the Ribbon Control is saved to the default Isolated Storage file, which is built in the source.

C#

```
private void SaveRibbonState_Click(object sender, RoutedEventArgs e)
{
    this.MyRibbon.SaveRibbonState();
}
```

VB.NET

```
Private Sub SaveRibbonState_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    Me.MyRibbon.SaveRibbonState()
End Sub
```

To store the current Ribbon State in the custom Isolated Storage file, use the second overloaded method. This method has two arguments namely IsolatedStorageFile and storeFileName.

C#

```
private void SaveRibbonState_Click(object sender, RoutedEventArgs e)
{
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
    IsolatedStorageScope.Assembly, null, null);
    this.MyRibbon.SaveRibbonState(storage, "Customfilename.dat");
}
```

VB.NET

```
Private Sub SaveRibbonState_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
    IsolatedStorageScope.Assembly, null, null)
    Me.MyRibbon.SaveRibbonState(storage, "Customfilename.dat")
End Sub
```

Loading ribbon states

Load state process is also having the similar procedures of save states. We can load the Ribbon State at any time from the last saved Isolated Storage file. **LoadRibbonState** method is used to load the Ribbon state from the last saved state. This method has two overloaded methods as follows:

1. void LoadRibbonState()
2. void LoadRibbonState(IsolatedStorageFile isoStorage, string storeFileName)

The first method with no arguments loads the Ribbon State from the last saved state in the default Isolated Storage file, which is stored by the SaveRibbonState method.

C#

```
private void LoadRibbonState_Click(object sender, RoutedEventArgs e)
{
    this.MyRibbon.LoadRibbonState();
}
```

VB.NET

```
Private Sub LoadRibbonState_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    Me.MyRibbon.LoadRibbonState()
End Sub
```

The second overloaded method loads the Ribbon State from the given file name in the mentioned Isolated Storage file.

C#

```
private void LoadRibbonState_Click(object sender, RoutedEventArgs e)
{
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
    IsolatedStorageScope.Assembly, null, null);
    this.MyRibbon.LoadRibbonState(storage, "Customfilename.dat");
}
```

VB.NET

```
Private Sub LoadRibbonState_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
    IsolatedStorageScope.Assembly, null, null)
    Me.MyRibbon.LoadRibbonState(storage, "Customfilename.dat")
End Sub
```

Save and load many ribbon states

Ribbon control states can easily be maintained in the Isolated Storage files. Further, It supports to Save the consecutive or different states of the Ribbon control in different Isolated Storage files and also load any saved state from the Isolated Storage files which is in old state.

C#

```
private void SaveLevel1State_Click(object sender, RoutedEventArgs e)
{
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
    IsolatedStorageScope.Assembly, null, null);
    this.MyRibbon.SaveRibbonState(storage, "RibbonState1.dat");
}
private void SaveLevel2State_Click(object sender, RoutedEventArgs e)
{
```

```

IsolatedStorageFile storage =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
IsolatedStorageScope.Assembly, null, null);
this.MyRibbon.SaveRibbonState(storage, "RibbonState2.dat");
}
private void SaveLevel3State_Click(object sender, RoutedEventArgs e)
{
IsolatedStorageFile storage =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
IsolatedStorageScope.Assembly, null, null);
this.MyRibbon.SaveRibbonState(storage, "RibbonState3.dat");
}

```

VB.NET

```

Private Sub SaveLevel1State_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
IsolatedStorageFile storage =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
IsolatedStorageScope.Assembly, null, null)
Me.MyRibbon.SaveRibbonState(storage, "RibbonState1.dat")
End Sub
Private Sub SaveLevel2State_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
IsolatedStorageFile storage =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
IsolatedStorageScope.Assembly, null, null)
Me.MyRibbon.SaveRibbonState(storage, "RibbonState2.dat")
End Sub
Private Sub SaveLevel3State_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
IsolatedStorageFile storage =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
IsolatedStorageScope.Assembly, null, null)
Me.MyRibbon.SaveRibbonState(storage, "RibbonState3.dat")
End Sub

```

After saving the different states of the Ribbon Control, load the Ribbon state to any of the old states

C#

```

private void LoadLevel1State_Click(object sender, RoutedEventArgs e)
{
IsolatedStorageFile storage =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
IsolatedStorageScope.Assembly, null, null);
this.MyRibbon.LoadRibbonState(storage, "RibbonState1.dat");
}
private void LoadLevel2State_Click(object sender, RoutedEventArgs e)
{
IsolatedStorageFile storage =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
IsolatedStorageScope.Assembly, null, null);
this.MyRibbon.LoadRibbonState(storage, "RibbonState2.dat");
}
private void LoadLevel3State_Click(object sender, RoutedEventArgs e)

```

```
{
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
    IsolatedStorageScope.Assembly, null, null);
    this.MyRibbon.LoadRibbonState(storage, "RibbonState3.dat");
}
```

VB.NET

```
Private Sub LoadLevel1State_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
    IsolatedStorageScope.Assembly, null, null)
    Me.MyRibbon.LoadRibbonState(storage, "RibbonState1.dat")
End Sub
Private Sub LoadLevel2State_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
    IsolatedStorageScope.Assembly, null, null)
    Me.MyRibbon.LoadRibbonState(storage, "RibbonState2.dat")
End Sub
Private Sub LoadLevel3State_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
    IsolatedStorageScope.Assembly, null, null)
    Me.MyRibbon.LoadRibbonState(storage, "RibbonState3.dat")
End Sub
```

Persisting ribbon states by XML writer

The WPF Ribbon control supports state persistence in the XML file created by the user. The ribbon states can be saved and loaded in the XML file by overloading the following methods:

- SaveRibbonState(XmlWriter xmlWriter)
- LoadRibbonState(XmlReader xmlReader)

The following code illustrates this

XML

```
<syncfusion:Ribbon Name="PART_Ribbon" VerticalAlignment="Top">
<syncfusion:Ribbon.PersistElements>
<syncfusion:RibbonElements>QuickAccessToolBar</syncfusion:RibbonElements>
</syncfusion:Ribbon.PersistElements>
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage />
</syncfusion:Ribbon.BackStage>
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar />
</syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:RibbonTab Caption="RibbonTab">
<syncfusion:RibbonBar Header="RibbonBar">
```

```

<syncfusion:SplitButton Label="SplitButton" SizeForm="Large">
<syncfusion:RibbonMenuItem Header="RibbonMenuItem" />
</syncfusion:SplitButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Disk" IsChecked="False">
<syncfusion:RibbonBar>
<syncfusion:RibbonButton Click="OnSaveDisk" Label="Save to disk" />
<syncfusion:RibbonButton Click="OnLoadDisk" Label="Load from disk" />
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>

```

C#

```

private const string SaveLocation = @"D:\templ.xml";
private void OnLoadDisk(object sender, RoutedEventArgs e)
{
    if (!File.Exists(SaveLocation))
    {
        return;
    }
    using (var reader = new StringReader(File.ReadAllText(SaveLocation)))
    {
        this.PART_Ribbon.LoadRibbonState(reader);
    }
}
private void OnSaveDisk(object sender, RoutedEventArgs e)
{
    using (var stream = new FileStream(SaveLocation, FileMode.Create))
    using (var writer = XmlWriter.Create(stream))
    {
        this.PART_Ribbon.SaveRibbonState(writer);
    }
}

```

VB.NET

```

Private Const SaveLocation As String = "D:\templ.xml"
Private Sub OnLoadDisk(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If Not File.Exists(SaveLocation) Then
        Return
    End If
    Using reader = New StringReader(File.ReadAllText(SaveLocation))
        Me.PART_Ribbon.LoadRibbonState(reader)
    End Using
End Sub
Private Sub OnSaveDisk(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Using stream = New FileStream(SaveLocation, FileMode.Create)
    Using writer = XmlWriter.Create(stream)
        Me.PART_Ribbon.SaveRibbonState(writer)
    End Using
    End Using
End Sub

```

Reset Ribbon States

To load the Normal (Initial) Ribbon state at runtime call the `ResetRibbonState` method. This is a parameter less method. This loads the Normal state of the Ribbon control. Resetting the Ribbon state is applicable while `AutoPersist` is enabled in Ribbon elements.

C#

```
private void ResetState_Click(object sender, RoutedEventArgs e)
{
    this.MyRibbon.ResetRibbonState();
}
```

VB.NET

```
Private Sub ResetState_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
Me.MyRibbon.ResetRibbonState()
End Sub
```

Delete ribbon states

To delete the unused saved Isolated Storage files, use the `DeleteRibbonState` method.

C#

```
private void DeleteRibbonState_Click(object sender, RoutedEventArgs e)
{
    this.MyRibbon.DeleteRibbonState();
}
```

VB.NET

```
Private Sub DeleteRibbonState_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
Me.MyRibbon.DeleteRibbonState()
End Sub
```

Also this method is used to delete any file from specified Isolated Storage location.

C#

```
private void DeleteRibbonState_Click(object sender, RoutedEventArgs e)
{
    IsolatedStorageFile storage =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
    IsolatedStorageScope.Assembly, null, null);
    this.MyRibbon.DeleteRibbonState(storage, "RibbonState1.dat");
}
```

VB.NET

```
Private Sub DeleteRibbonState_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
```

```

IsolatedStorageFile storage =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
IsolatedStorageScope.Assembly, null, null)
Me.MyRibbon.DeleteRibbonState(storage, "RibbonState1.dat")
End Sub

```

Ribbon Merge in WPF Ribbon

WPF [Ribbon](#) allows merging of [RibbonTab](#) and [RibbonBar](#) of two different Ribbon controls in MDI applications. [DocumentContainer.MDIParentRibbon](#), [RibbonTab.MergeType](#) and [RibbonTab.MergeOrder](#) properties helps to perform menu merging.

In WPF, you can create MDI application using [DocumentContainer](#) control. Also, [Ribbon](#) controls allows to merge [RibbonTab](#) and [RibbonBar](#) of active child window to the [Ribbon](#) in parent window.

Creating MDI window and enabling menu merging

[DocumentContainer](#) helps to create MDI window in WPF Application. The [DocumentContainer](#) allows you to create MDI window and Tabbed MDI window layouts.

Follow the below steps to create simple sample to understand ribbon menu merging,

1. Creating main [RibbonWindow](#) with [Ribbon](#) and [DocumentContainer](#)

XML

```

<syncfusion:RibbonWindow
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="Ribbon.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:Ribbon"
mc:Ignorable="d"
WindowStartupLocation="CenterScreen"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="Office2019Colorful"
Title="Ribbon Merging" Height="450" Width="700">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*/"/>
</Grid.RowDefinitions>
<syncfusion:Ribbon x:Name="parentRibbon" >
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Mail">
<syncfusion:RibbonButton Label="New Mail" SizeForm="Large"
Click="RibbonButton_Click"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
<syncfusion:DocumentContainer x:Name="doc" Grid.Row="1" >
</syncfusion:DocumentContainer>
</Grid>
</syncfusion:RibbonWindow>

```


- Next, lets create two `UserControl` views with Ribbon which acts as child MDI windows.

Child View 1

XML

```
<UserControl x:Class="Ribbon.ChildView1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:local="clr-namespace:Ribbon"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>
<ContentPresenter>
<ContentPresenter.Content>
<Grid>
<syncfusion:Ribbon x:Name="childRibbon" >
<syncfusion:RibbonTab Caption="Home" MergeType="MergeItems">
<syncfusion:RibbonBar Header="Message">
<syncfusion:RibbonButton Label="New Message" SizeForm="Large"/>
<syncfusion:RibbonButton x:Name="MergeButton" Label="Merge"
SizeForm="Large" />
<syncfusion:RibbonButton x:Name="UnMergeButton" Label="UnMerge"
SizeForm="Large"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="View1 Tab2" >
<syncfusion:RibbonBar Header="Folders">
<syncfusion:RibbonButton Label="New Folder" SizeForm="Large"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
<TextBlock Text="View 1"/>
</Grid>
</ContentPresenter.Content>
</ContentPresenter>
</Grid>
</UserControl>
```

Child View 2

XML

```
<UserControl x:Class="Ribbon.ChildView2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:local="clr-namespace:Ribbon"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>
```

```

<ContentPresenter>
<ContentPresenter.Content>
<Grid>
<syncfusion:Ribbon x:Name="childRibbon" >
<syncfusion:RibbonTab Caption="View2 Home" MergeType="AddItems">
<syncfusion:RibbonBar Header="Message">
<syncfusion:RibbonButton Label="View Message" SizeForm="Large"/>
<syncfusion:RibbonButton x:Name="MergeButton" Label="Merge"
SizeForm="Large" />
<syncfusion:RibbonButton x:Name="UnMergeButton" Label="UnMerge"
SizeForm="Large"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="View2 Tab2" MergeOrder="1">
<syncfusion:RibbonBar Header="Folders">
<syncfusion:RibbonButton Label="New Folder" SizeForm="Large"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
<TextBlock Text="View 2"/>
</Grid>
</ContentPresenter.Content>
</ContentPresenter>
</Grid>
</UserControl>

```

- Now, add the both child view's into the [DocumentContainer](#) and set the the [DocumentContainer.MDIParentRibbon](#) property of [DocumentContainer](#).

XML

```

<syncfusion:RibbonWindow
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="Ribbon.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:Ribbon"
mc:Ignorable="d"
WindowStartupLocation="CenterScreen"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="Office2019Colorful"
Title="Ribbon Merging" Height="450" Width="700">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*/>
</Grid.RowDefinitions>
<syncfusion:Ribbon x:Name="parentRibbon" >
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Mail">
<syncfusion:RibbonButton Label="New Mail" SizeForm="Large"
Click="RibbonButton_Click"/>

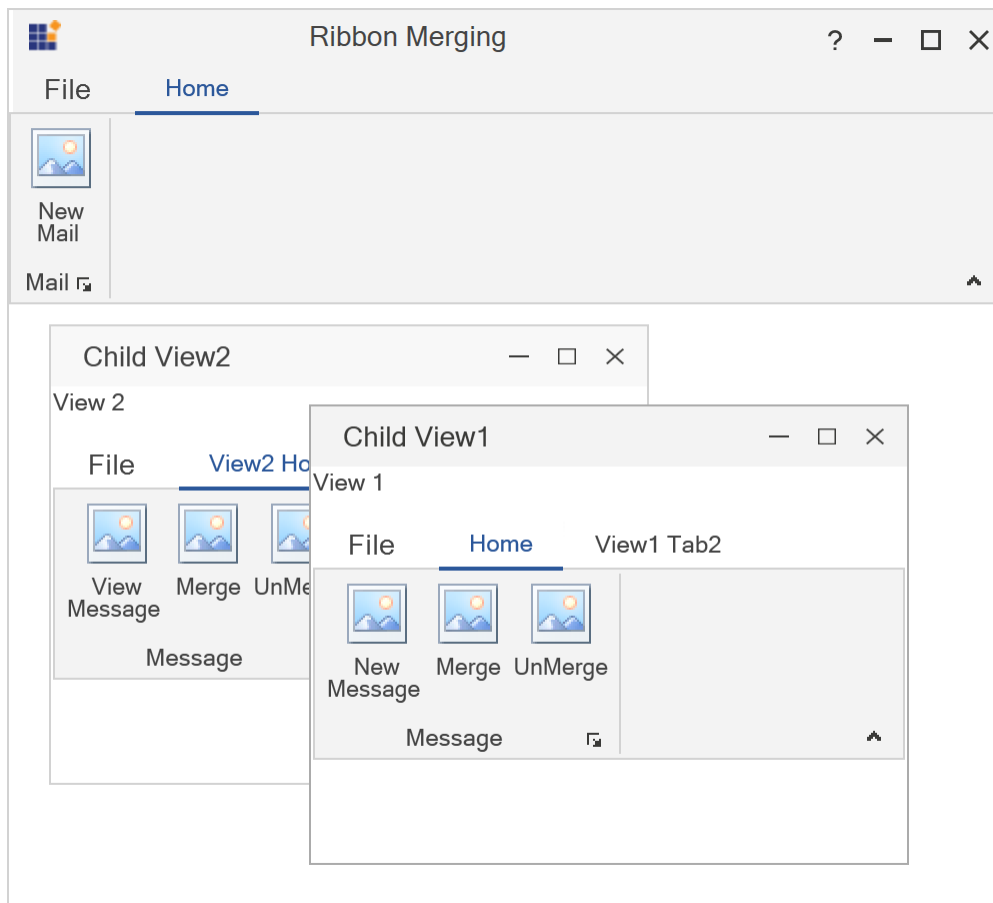
```

```

</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
<syncfusion:DocumentContainer x:Name="doc" Grid.Row="1"
MDIParentRibbon="{Binding ElementName=parentRibbon}">
<local:ChildView1 x:Name="ChildView1"
syncfusion:DocumentContainer.Header="Child View1"
syncfusion:DocumentContainer.MDIBounds="150,50,300,230"/>
<local:ChildView2 x:Name="ChildView2"
syncfusion:DocumentContainer.Header="Child View2"
syncfusion:DocumentContainer.MDIBounds="20,10,300,230"/>
</syncfusion:DocumentContainer>
</Grid>
</syncfusion:RibbonWindow>

```

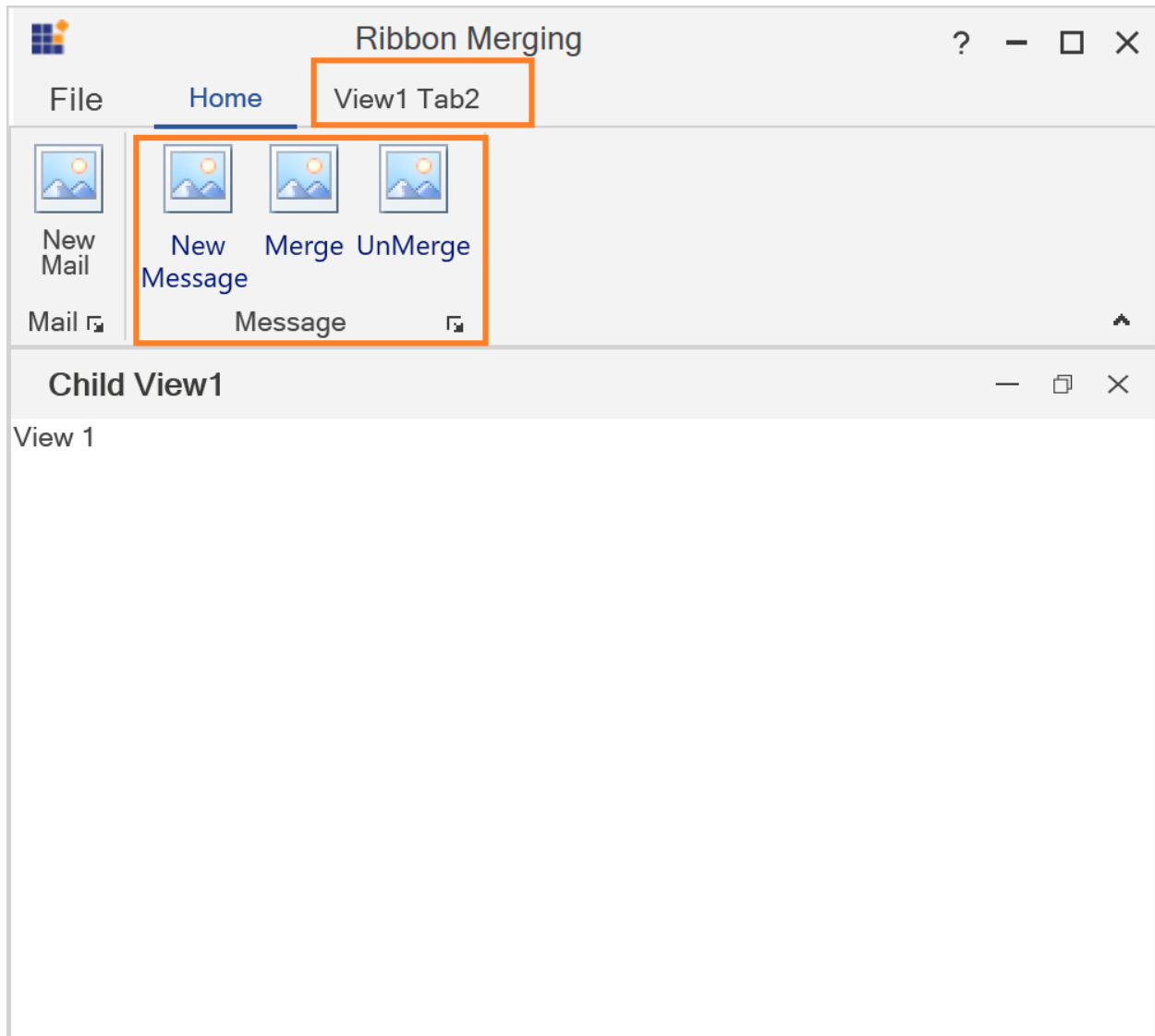
Now run the application and you can see an output like this.



- Now the child view's has been added inside the [DocumentContainer](#), and now lets see how to merge the child Ribbon into the parent Ribbon.

The merging operation performed based on [DocumentContainer.Mode](#) (MDI and TDI) property.

- [MDI](#) - The default value of [DocumentContainer.Mode](#). In the above image child view's are in MDI mode where each view loaded inside a child window. You can merge the [Ribbon](#) in child view's into the MDI parent Ribbon by maximizing the MDI child window.



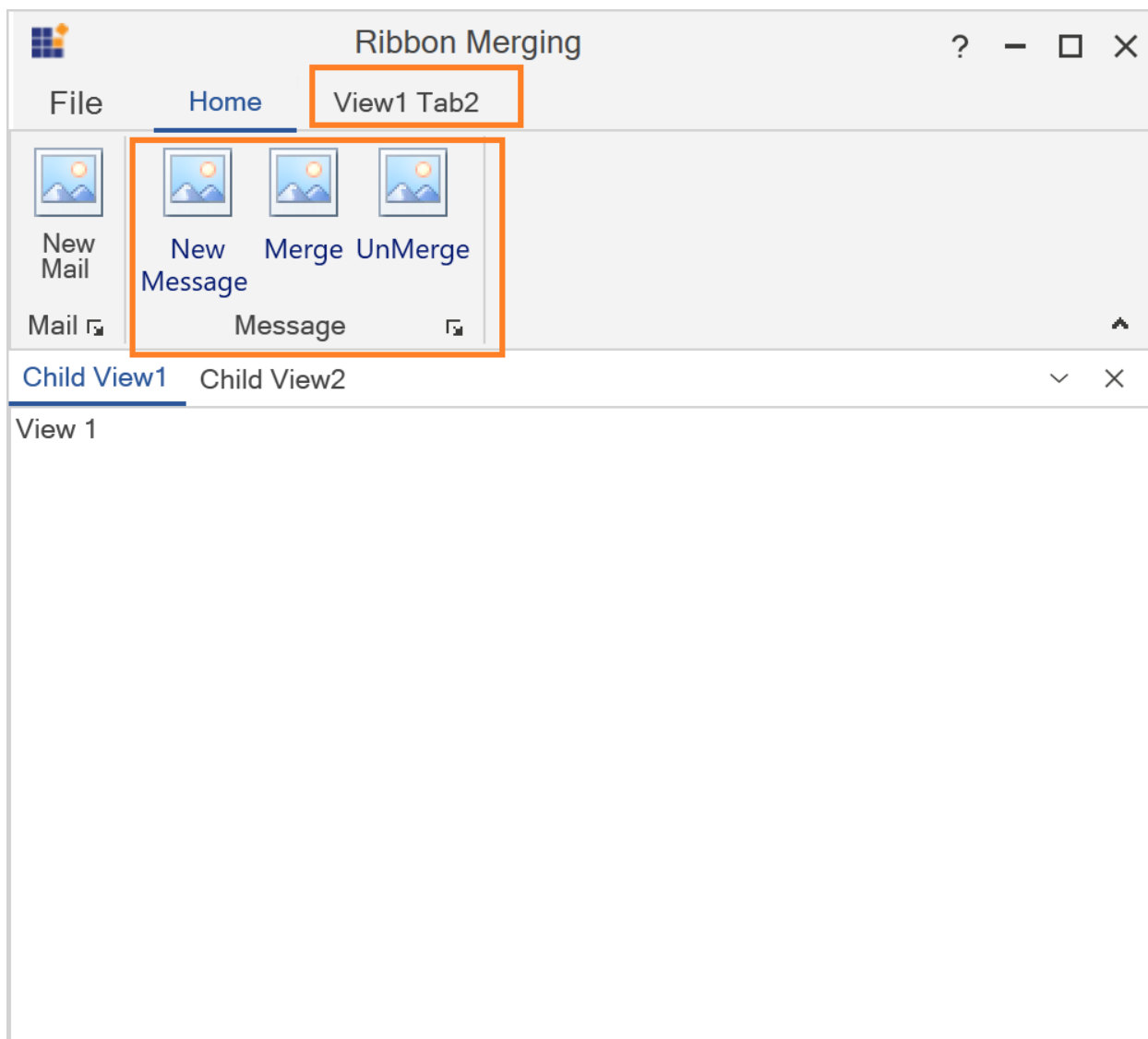
- [TDI](#) - The child view's will be loaded in tabs. In this mode, the child view's ribbon of the active tab will be merged to the MDI parent Ribbon.

In the below example the [DocumentContainer.Mode](#) property for [DocumentContainer](#) is set to [TDI](#).

XML

```
<syncfusion:RibbonWindow
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="Ribbon.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:Ribbon"
mc:Ignorable="d"
WindowStartupLocation="CenterScreen"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="Office2019Colorful"
Title="Ribbon Merging" Height="450" Width="450">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*/>
</Grid.RowDefinitions>
<syncfusion:Ribbon x:Name="parentRibbon" >
<syncfusion:RibbonTab Caption="Home">
<syncfusion:RibbonBar Header="Mail">
<syncfusion:RibbonButton Label="New Mail" SizeForm="Large"
Click="RibbonButton_Click"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
<syncfusion:DocumentContainer x:Name="doc" Grid.Row="1"
MDIParentRibbon="{Binding ElementName=parentRibbon}" Mode="TDI">
<local:ChildView1 x:Name="ChildView1"
syncfusion:DocumentContainer.Header="Child View1"
syncfusion:DocumentContainer.MDIBounds="150,50,300,230"/>
<local:ChildView2 x:Name="ChildView2"
syncfusion:DocumentContainer.Header="Child View2"
syncfusion:DocumentContainer.MDIBounds="20,10,300,230"/>
</syncfusion:DocumentContainer>
</Grid>
</syncfusion:RibbonWindow>
```



In the image notice that **Child View1** tab is selected and elements of selected child view's ribbon are merged into the MDI parent Ribbon.

Note: [View sample in GitHub](#)

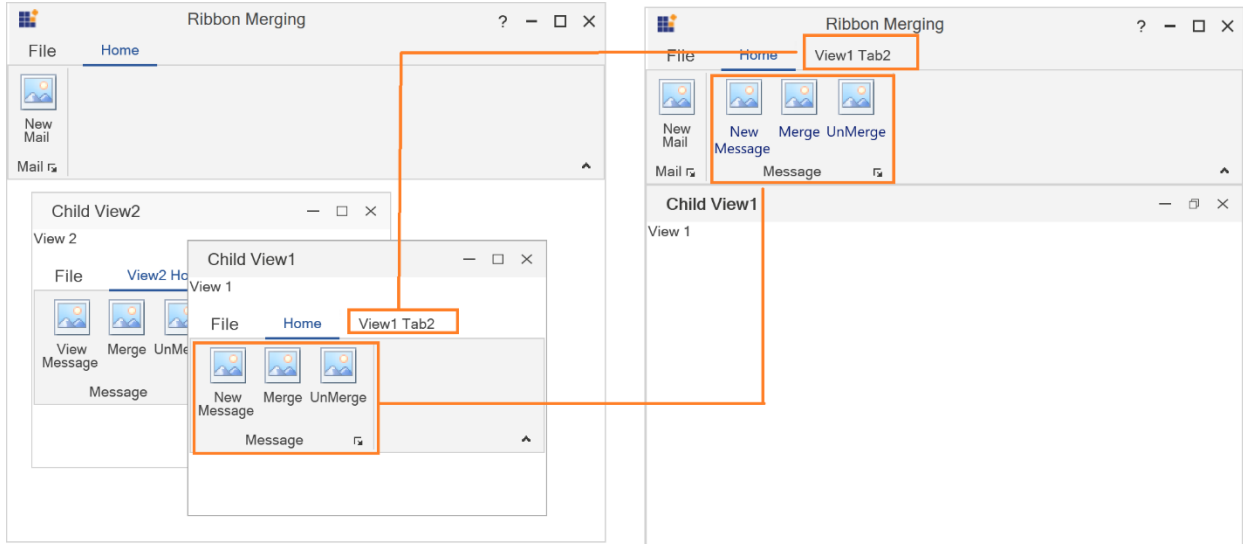
Merge Type

[RibbonTab.MergeType](#) property indicates how the items in child view's ribbon are merged with MDI parent ribbon. [MergeType](#) has following options,

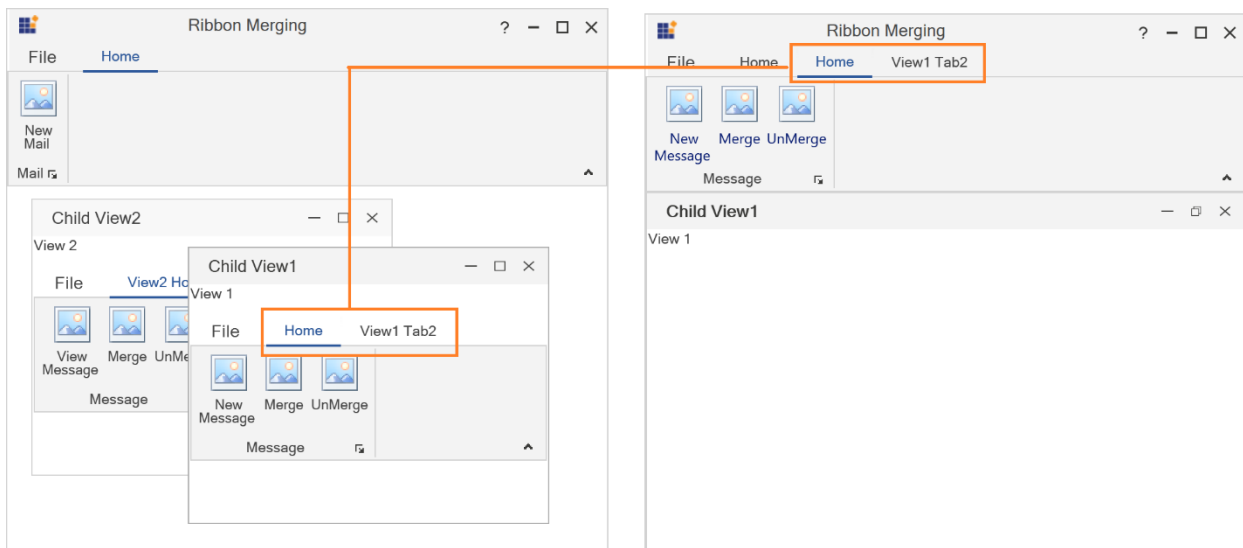
- [Add](#) - Adds the child view's [RibbonTab](#) to the MDI parent ribbon based on [RibbonTab.MergeOrder](#), even if MDI parent has [RibbonTab](#) with same [Caption](#).
- [Merge](#) - The default value of [RibbonTab.MergeType](#). If [RibbonTab.Caption](#) of MDI parent ribbon and [RibbonTab.Caption](#) of child view's ribbon has same name and child view's [RibbonTab.MergeType](#) is [Merge](#), then child view's [RibbonBar](#)'s are merged with MDI parent's tab. If MDI parent ribbon doesn't have tab with same caption, then child view's ribbon tab added to MDI parent ribbon based on [RibbonTab.MergeOrder](#).

Let's look at the example,

In the example both the MDI parent ribbon and child view1's ribbon has ribbon tab with same caption and the default value of [MergeType](#) is [Merge](#). So, when child view1 maximized, bar's in child view's Home tab get are merged to MDI parent ribbon's Home tab.



If you change the [MergeType](#) of ribbon tab in child view1 as [Add](#), then [Home](#) tab will be added as new tab like [View1 Tab2](#)



Note: [View sample in GitHub](#)

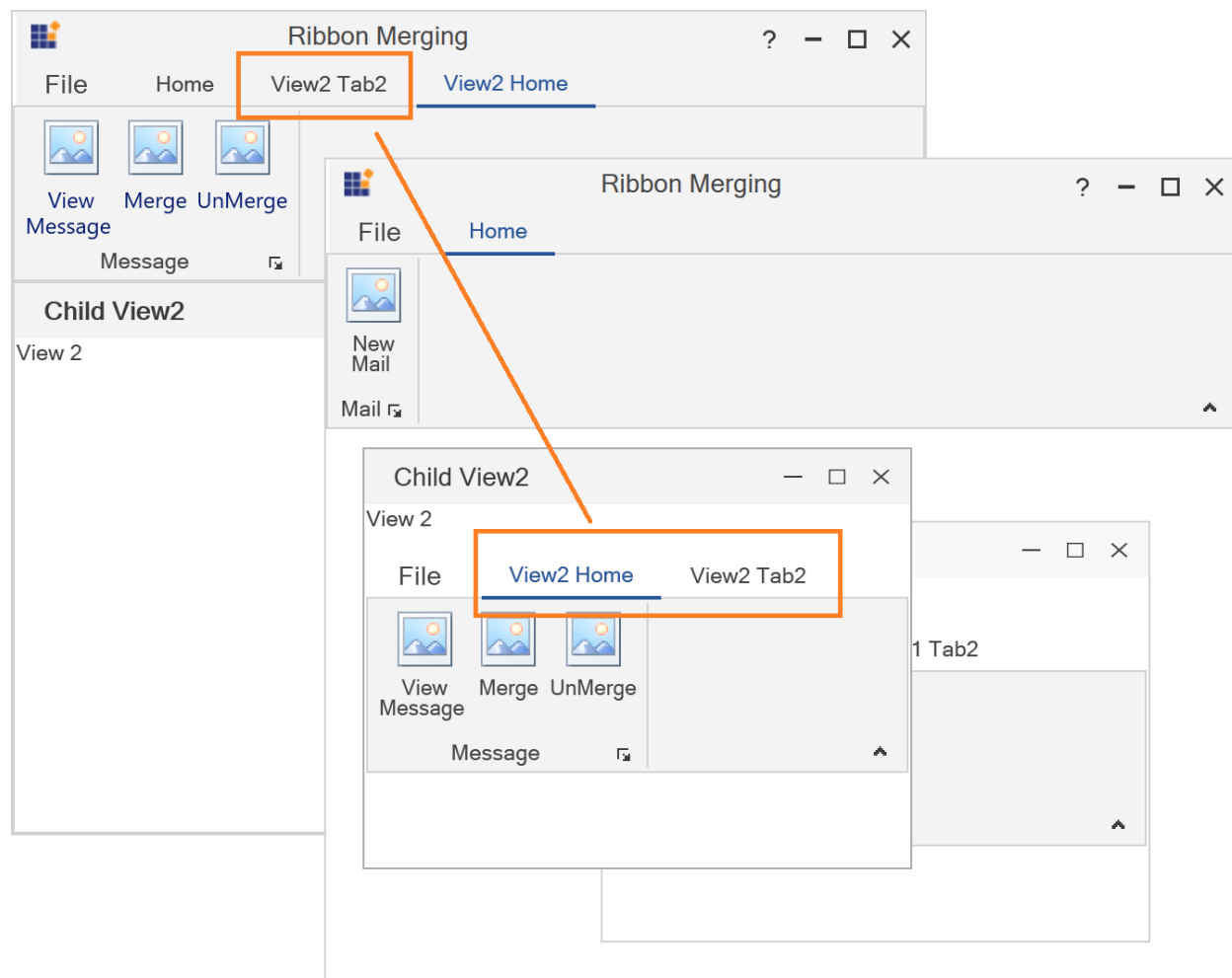
Merge Order

MDI parent ribbon positions the child view's ribbon tabs followed by the tabs of MDI parent ribbon. You can change the position of child view's ribbon using [RibbonTab.MergeOrder](#) property.

In the below code, the [RibbonTab.MergeOrder](#) for [View2 Tab2](#) is set as 1. So, the ribbon tab is positioned at 1st index while merging to MDI parent.

XML

```
<UserControl x:Class="Ribbon.ChildView2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:local="clr-namespace:Ribbon"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>
<ContentPresenter>
<ContentPresenter.Content>
<Grid>
<syncfusion:Ribbon x:Name="childRibbon" >
<syncfusion:RibbonTab Caption="View2 Home" MergeType="AddItems">
<syncfusion:RibbonBar Header="Message">
<syncfusion:RibbonButton Label="View Message" SizeForm="Large"/>
<syncfusion:RibbonButton x:Name="MergeButton" Label="Merge"
SizeForm="Large" />
<syncfusion:RibbonButton x:Name="UnMergeButton" Label="UnMerge"
SizeForm="Large"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="View2 Tab2" MergeOrder="1">
<syncfusion:RibbonBar Header="Folders">
<syncfusion:RibbonButton Label="New Folder" SizeForm="Large"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:Ribbon>
<TextBlock Text="View 2"/>
</Grid>
</ContentPresenter.Content>
</ContentPresenter>
</Grid>
</UserControl>
```

Note: [View sample in GitHub](#)

Merging and Unmerging in code

Ribbon merging can be performed in code by using [Ribbon.Merge](#) method of MDI parent ribbon. Child view's ribbon should be passed as parameter to perform merge operation.

Similarly, the child ribbon can be unmerged from MDI parent ribbon using [Ribbon.UnMerge](#) method of MDI parent ribbon.

C#

```
public MainWindow()
{
    InitializeComponent();
    this.Child1.MergeButton.Click += MergeButton_Click;
    this.Child1.UnMergeButton.Click += UnMergeButton_Click;
}

private void UnMergeButton_Click(object sender, RoutedEventArgs e)
{
    this.parentRibbon.UnMerge(this.Child1.childRibbon);
}

private void MergeButton_Click(object sender, RoutedEventArgs e)
{
}
```

```
this.parentRibbon.Merge(this.Child1.childRibbon);
}
```

VB.NET

```
Public Sub New()
    InitializeComponent()
    Me.Child1.MergeButton.Click += AddressOf MergeButton_Click
    Me.Child1.UnMergeButton.Click += AddressOf UnMergeButton_Click
End Sub
Private Sub UnMergeButton_Click(ByVal sender As Object, ByVal e As
    RoutedEventArgs)
    Me.parentRibbon.UnMerge(Me.Child1.childRibbon)
End Sub
Private Sub MergeButton_Click(ByVal sender As Object, ByVal e As
    RoutedEventArgs)
    Me.parentRibbon.Merge(Me.Child1.childRibbon)
End Sub
```

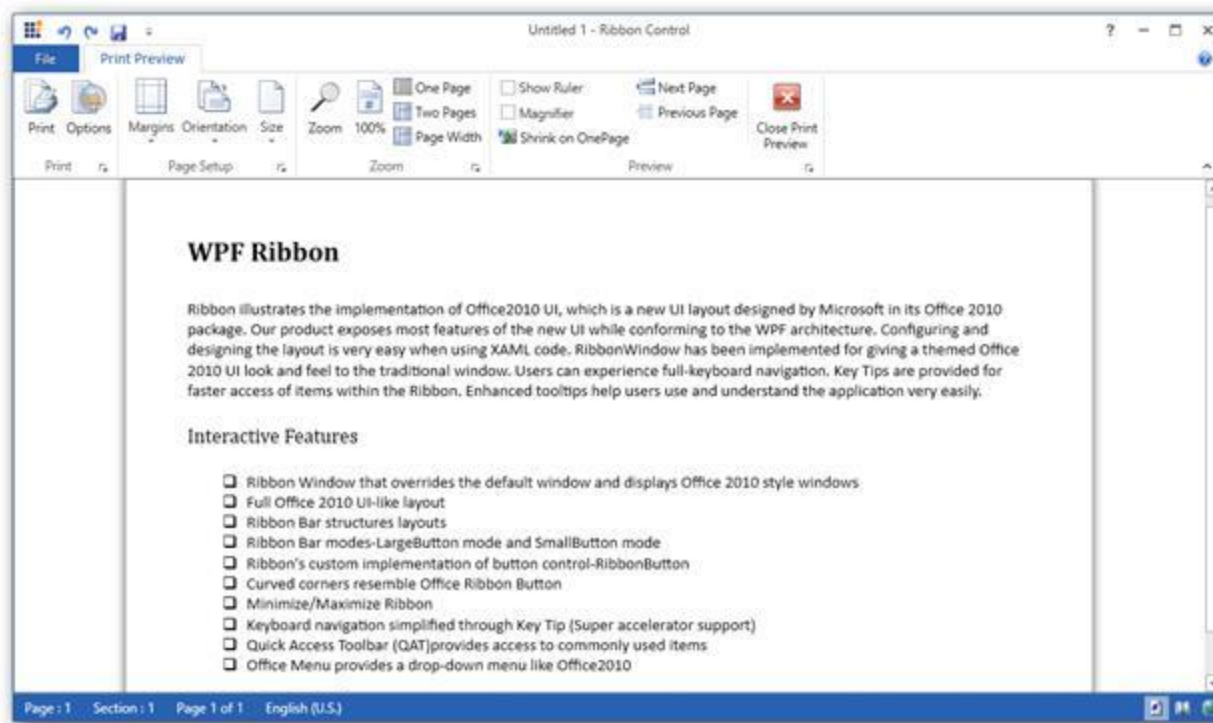
Ribbon ModelTab in WPF Ribbon

Modal Tab in Ribbon Control are used to display a collection of commands other than the commands which are available in the core tabs. At this point, the core tab gets disabled.

Use case scenarios

Print Preview is a Modal Tab which displays Print Preview related commands





Adding modal tabs to an application

To add RibbonTabs as ModalTabs in an application, use **ModalTabCollection** property of the Ribbon.

XML

```
<syncfusion:Ribbon.ModalTabCollection >
<syncfusion:ModalTabCollection >
<syncfusion:RibbonTab Caption="Print Preview" x:Name="_printPreviewTab">
<syncfusion:RibbonBar Header="Sample Bar">
<syncfusion:RibbonButton Label="Close Tab"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:ModalTabCollection>
</syncfusion:Ribbon.ModalTabCollection>
```

Add Modal Tab to the simplified layout

When the simplified layout is enabled, the Model tab can be added, and its items will be displayed in a single line as shown below. To know more about the simplified layout, refer [here](#).

XML

```
<syncfusion:RibbonWindow x:Class="RibbonButton_IconTemp.Window1"
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:RibbonButton_IconTemp" xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
skin:SfSkinManager.VisualStudio="MaterialLight"
Title="Untitled 1 - Ribbon Control" Height="450" Width="800">
```

```

<Grid x:Name="grid">
<syncfusion:Ribbon VerticalAlignment="Top" EnableSimplifiedLayoutMode="True"
LayoutMode="Simplified">
<syncfusion:Ribbon.ModalTabCollection>
<syncfusion:ModalTabCollection>
<syncfusion:RibbonTab x:Name="insertTab"
Caption="Insert"
IsChecked="True">
<syncfusion:RibbonBar
Header="Illustrations"
IsLauncherButtonVisible="True">
<syncfusion:RibbonButton
Label="Picture"
MediumIcon="Resources/Picture20.png"
SizeForm="Large">
</syncfusion:RibbonButton>
<syncfusion:RibbonButton
Label="Comment"
MediumIcon="Resources/New Comment20.png"
SizeForm="Large">
</syncfusion:RibbonButton>
<syncfusion:DropDownButton
Label="Shapes"
MediumIcon="Resources/Insert Shapes20.png"
SizeForm="Small">
</syncfusion:DropDownButton>
<syncfusion:DropDownButton
Label="Chart"
MediumIcon="Resources/Base chart20.png"
SizeForm="Small">
</syncfusion:DropDownButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
</syncfusion:ModalTabCollection>
</syncfusion:Ribbon.ModalTabCollection>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

C#

```

Ribbon ribbon = new Ribbon();
ribbon.VerticalAlignment = VerticalAlignment.Top;
ribbon.EnableSimplifiedLayoutMode = true;
ribbon.LayoutMode = LayoutMode.Simplified;
// Creating new tabs
RibbonTab insertTab = new RibbonTab();
insertTab.Caption = "Insert";
insertTab.IsChecked = true;
// Creating new bar
RibbonBar illustrationsBar = new RibbonBar();
illustrationsBar.Header = "Illustrations";
// Creating items
// Creating items
RibbonButton pictureButton = new RibbonButton();
pictureButton.Label = "Picture";

```

```

pictureButton.SizeForm = SizeForm.Large;
pictureButton.MediumIcon = new BitmapImage(new
Uri(@"Resources/Picture20.png", UriKind.RelativeOrAbsolute));
RibbonButton commentButton = new RibbonButton();
commentButton.Label = "Comment";
commentButton.SizeForm = SizeForm.Large;
commentButton.MediumIcon = new BitmapImage(new Uri(@"Resources/New
Comment20.png", UriKind.RelativeOrAbsolute));
DropDownButton shapesButton = new DropDownButton();
shapesButton.Label = "Shapes";
shapesButton.SizeForm = SizeForm.Small;
shapesButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Insert
Shapes20.png", UriKind.RelativeOrAbsolute));
DropDownButton chartButton = new DropDownButton();
chartButton.Label = "Chart";
chartButton.SizeForm = SizeForm.Small;
chartButton.MediumIcon = new BitmapImage(new Uri(@"Resources/Base
chart20.png", UriKind.RelativeOrAbsolute));
// Adding items to bar
illustrationsBar.Items.Add(pictureButton);
illustrationsBar.Items.Add(commentButton);
illustrationsBar.Items.Add(shapesButton);
illustrationsBar.Items.Add(chartButton);
// Adding bars to the tabs
insertTab.Items.Add(illustrationsBar);
// Adding modal tab
ribbon.ModalTabCollection.Add(insertTab);
insertTab.Name = "insertTab";
grid.Children.Add(ribbon);
//Displaying modal tab
ribbon.ShowModalTab("insertTab");
SfSkinManager.SetVisualStyle(this, VisualStyles.MaterialLight);

```



How to handle modal tabs in ribbon

The `ShowModalTab` and `CloseModalTabs` method handle the visibility of the Modal Tabs in the Ribbon control. Any ModalTab from the ModalTabCollection can be displayed whenever required.

To show specific Modal Tab in ribbon, call ShowModalTab method. This can be done from any event of core Ribbon Tab control.

C#

```

private void ShowModalTabBtn_Click(object sender, RoutedEventArgs e)
{
    this._ribbon.ShowModalTab("_printPreviewTab");
}

```

VB.NET

```
Private Sub ShowModalTabBtn_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
Me._ribbon.ShowModalTab("_printPreviewTab")
End Sub
```

CloseModalTabs method is used to close the currently opened Modal Tab in Ribbon control. This method should be called in any event of currently displaying Modal Tab element. This is illustrated in the code given below.

C#

```
private void CloseModalTabBtn_Click(object sender, RoutedEventArgs e)
{
this._ribbon.CloseModalTabs();
}
```

VB.NET

```
Private Sub CloseModalTabBtn_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
Me._ribbon.CloseModalTabs()
End Sub
```

ToolTip in WPF Ribbon

ToolTip is used to display small information about the specific element while hovering mouse on it and it is implemented as like in MS Outlook.

ScreenTip is an enhanced ToolTip and it is used to show a popup window when the mouse cursor is hovered over **RibbonItem**. The popup is used to provide details of the **RibbonItem**'s function. The difference between ToolTip and ScreenTip is that, ToolTip hosts controls to show a small popup window, whereas ScreenTip includes predefined properties. With these predefined properties, you can create ScreenTip easily.

ScreenTip has four properties.

1. Description - Specifies the description of the ScreenTip. 2. Image - Specifies the image of the ScreenTip. 3. Content - Specifies the content of the ScreenTip. 4. HelpText - Specifies the HelpText of the ScreenTip.

Adding tooltip for ribbon items

Ribbon ToolTip can be set to each Ribbon items by defining ScreenTip inside the ToolTip.

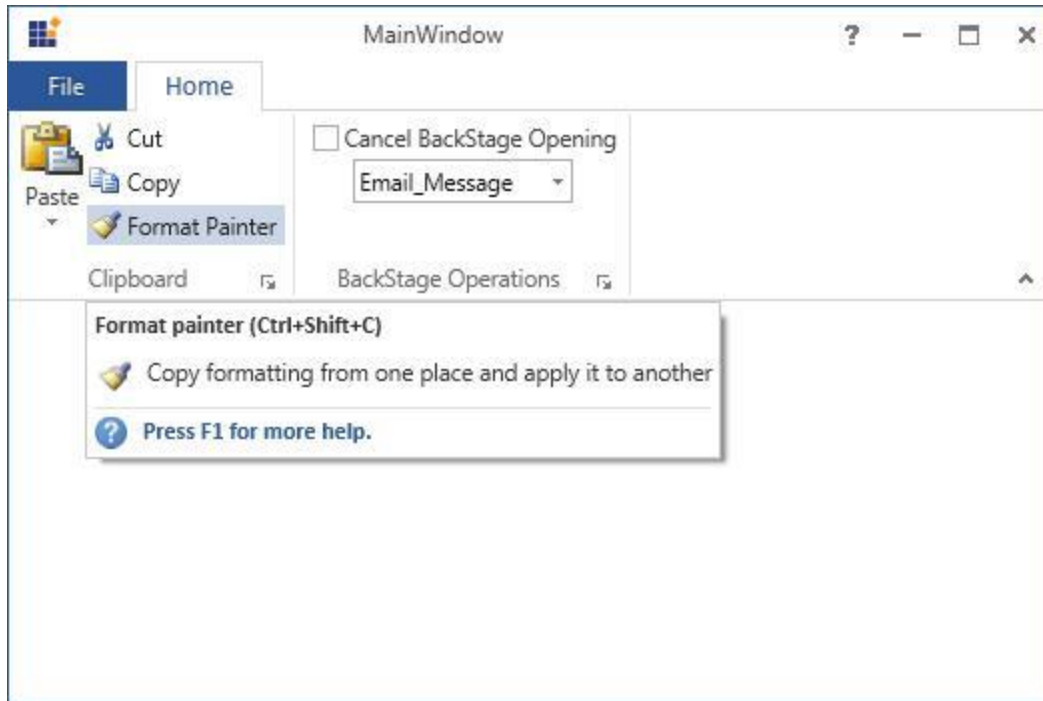
XML

```
<Syncfusion:Ribbon x:Name="_ribbon" VerticalAlignment="Top">
<Syncfusion:RibbonTab Caption="Home" IsChecked="False" >
<Syncfusion:RibbonBar Header="Clipboard">
<Syncfusion:RibbonBar.ToolTip>
<Syncfusion:ScreenTip MinWidth="150" Description="ScreenTip Example"
VerticalOffset="10" Content="Represent a ribbon simple buttons"/>
</Syncfusion:RibbonBar.ToolTip>
<Syncfusion:SplitButton Name="paste" Label="Paste" SizeForm="Large"
LargeIcon="/Resources/Paste16.png">
<Syncfusion:SplitButton.ToolTip>
```

```

<Syncfusion:ScreenTip Description="Paste (Ctrl+V)" Content="Paste the
contents of clipboard."/>
</Syncfusion:SplitButton.ToolTip>
</Syncfusion:SplitButton>
<Syncfusion:RibbonButton Label="Cut" SizeForm="Small"
SmallIcon="/Resources/Cut16.png" >
<Syncfusion:RibbonButton.ToolTip>
<Syncfusion:ScreenTip Description="Cut (Ctrl+X)" Content="Cut the selection
and put it on the clipboard."/>
</Syncfusion:RibbonButton.ToolTip>
</Syncfusion:RibbonButton>
<Syncfusion:RibbonButton Name="copy" Label="Copy" SizeForm="Small"
SmallIcon="/Resources/Copy16.png" >
<Syncfusion:RibbonButton.ToolTip>
<Syncfusion:ScreenTip Content="Copy the selection and put it on the
clipboard" Description="Copy (Ctrl+C)" HelpText="Press F1 for more help."/>
</Syncfusion:RibbonButton.ToolTip>
</Syncfusion:RibbonButton>
<Syncfusion:RibbonButton IsMultiLine="True" Label="Format Painter"
SizeForm="Small" SmallIcon="/Resources/FormatPainter16.png" >
<Syncfusion:RibbonButton.ToolTip>
<Syncfusion:ScreenTip Content="Copy formatting from one place and apply it
to another" Description="Format painter (Ctrl+Shift+C)"
ImageSource="/Resources/FormatPainter16.png" HelpText="Press F1 for more
help."/>
</Syncfusion:RibbonButton.ToolTip>
</Syncfusion:RibbonButton>
</Syncfusion:RibbonBar>
<Syncfusion:RibbonBar Header="BackStage Operations" >
<Syncfusion:RibbonCheckBox Content="Cancel BackStage Opening"
x:Name="BackStageOpeningCheckBox">
<Syncfusion:RibbonCheckBox.ToolTip>
<Syncfusion:ScreenTip Description="Cancel BackStage" Content="Restrict
backstage opening"/>
</Syncfusion:RibbonCheckBox.ToolTip>
</Syncfusion:RibbonCheckBox>
<Syncfusion:RibbonComboBox SelectedIndex="0" FlowDirection="LeftToRight"
Width="110">
<Syncfusion:RibbonComboBox.ToolTip>
<Syncfusion:ScreenTip Description="New Items" Content="Create a new item
such as Meeting or contact."/>
</Syncfusion:RibbonComboBox.ToolTip>
<Syncfusion:RibbonComboBoxItem>Email_Message</Syncfusion:RibbonComboBoxItem>
<Syncfusion:RibbonComboBoxItem>Meeting</Syncfusion:RibbonComboBoxItem>
<Syncfusion:RibbonComboBoxItem>Appointment</Syncfusion:RibbonComboBoxItem>
</Syncfusion:RibbonComboBox>
</Syncfusion:RibbonBar>
</Syncfusion:RibbonTab>
</Syncfusion:Ribbon>

```

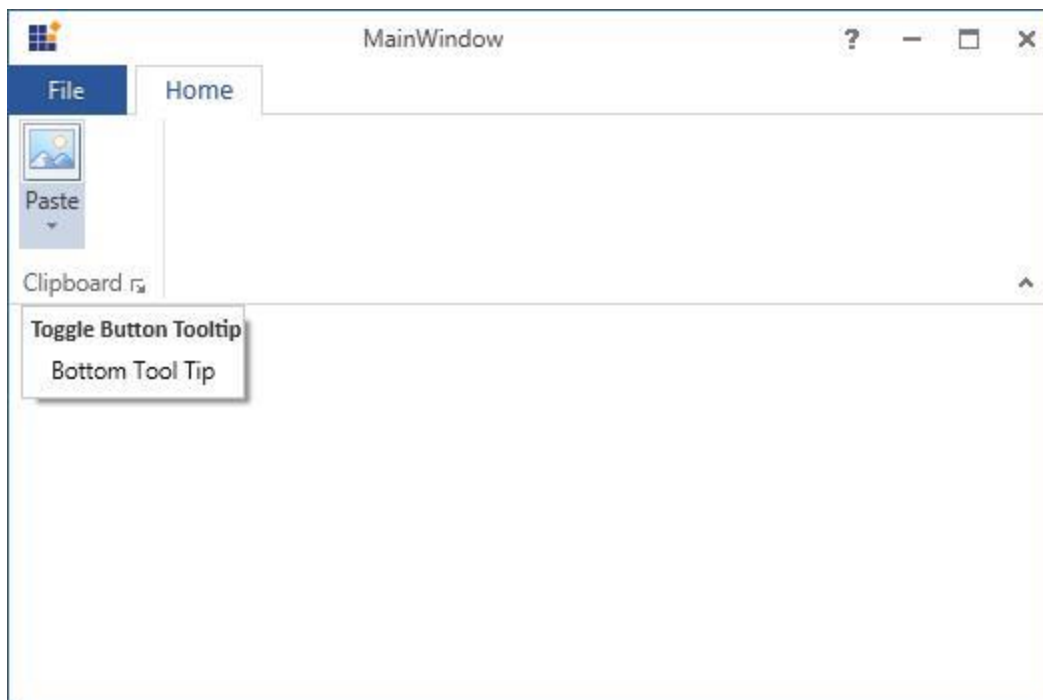


Setting the ToolTip for the upper and lower half of the split button

The ToolTip property is used to set the tooltip for the upper half of the Split Button, while the ToggleButtonToolTip property is used to set the tooltip for the drop-down in the lower half of the split button.

XML

```
<Syncfusion:SplitButton Label="Paste" SizeForm="Large" >
  <Syncfusion:SplitButton.ToolTip>
    <Syncfusion:ScreenTip Description="Split Button Tooltip"
      VerticalOffset="32">
      <TextBlock Text="Top Tool Tip" />
    </Syncfusion:ScreenTip>
  </Syncfusion:SplitButton.ToolTip>
  <Syncfusion:SplitButton.ToggleButtonToolTip>
    <Syncfusion:ScreenTip Description="Toggle Button Tooltip"
      VerticalOffset="29">
      <TextBlock Text="Bottom Tool Tip" />
    </Syncfusion:ScreenTip>
  </Syncfusion:SplitButton.ToggleButtonToolTip>
</Syncfusion:SplitButton>
```

Options for inserting help text in ScreenTip

Essential Tool WPF is enhanced with `HelpText` option. Users can add help text in ScreenTip. A line separator separates the Screen tip information from help text.

XML

```
<Syncfusion:RibbonButton IsMultiLine="True" Label="Format Painter"
SizeForm="Small" SmallIcon="/Resources/FormatPainter16.png" >
<Syncfusion:RibbonButton.ToolTip>
```

```
<Syncfusion:ScreenTip Content="Copy formatting from one place and apply it
to another" Description="Format painter (Ctrl+Shift+C)"
ImageSource="/Resources/FormatPainter16.png" HelpText="Press F1 for more
help."/>
</Syncfusion:RibbonButton.ToolTip>
</Syncfusion:RibbonButton>
```

Note: When no HelpText is set to the HelpText area and line separator get automatically hidden

KeyBoard Support in WPF Ribbon

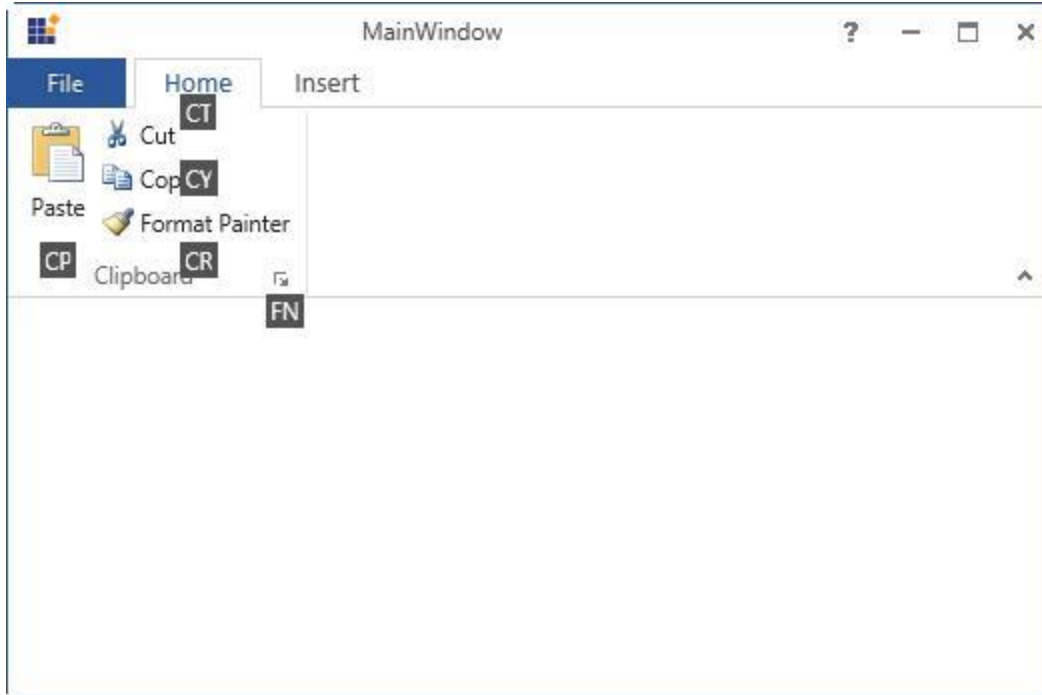
To invoke any command using KeyBoard, Ribbon control provides KeyTip support to access the RibbonItem. On pressing Alt key, KeyTips for the corresponding RibbonItem gets displayed.

Adding KeyTip to the ribbon items

RibbonControl provides with an attached property called **KeyTip** that helps to set the Key tip for the RibbonItem. It also let to set the Key tip for the RibbonTab, Launcher Button, and for the controls that are needed to add in the Ribbon like RibbonButton, SplitButton...etc

XML

```
<syncfusion:Ribbon Name="_ribbon" VerticalAlignment="Top">
<syncfusion:RibbonTab Caption="Home" syncfusion:Ribbon.KeyTip="H" >
<syncfusion:RibbonBar Header="Clipboard" syncfusion:Ribbon.KeyTip="FN" >
<syncfusion:RibbonButton Label="Paste" SizeForm="Large"
LargeIcon="/Resources/Paste32.png"
syncfusion:Ribbon.KeyTip="CP"/>
<syncfusion:RibbonButton Label="Cut" SizeForm="Small"
SmallIcon="/Resources/Cut16.png" syncfusion:Ribbon.KeyTip="CT"/>
<syncfusion:RibbonButton Label="Copy" SizeForm="Small"
SmallIcon="/Resources/Copy16.png"
syncfusion:Ribbon.KeyTip="CY" />
<syncfusion:RibbonButton Label="Format Painter" SizeForm="Small"
SmallIcon="/Resources/FormatPainter16.png"
syncfusion:Ribbon.KeyTip="CR" />
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab Caption="Insert" syncfusion:Ribbon.KeyTip="I"
IsChecked="True" />
</syncfusion:Ribbon>
```



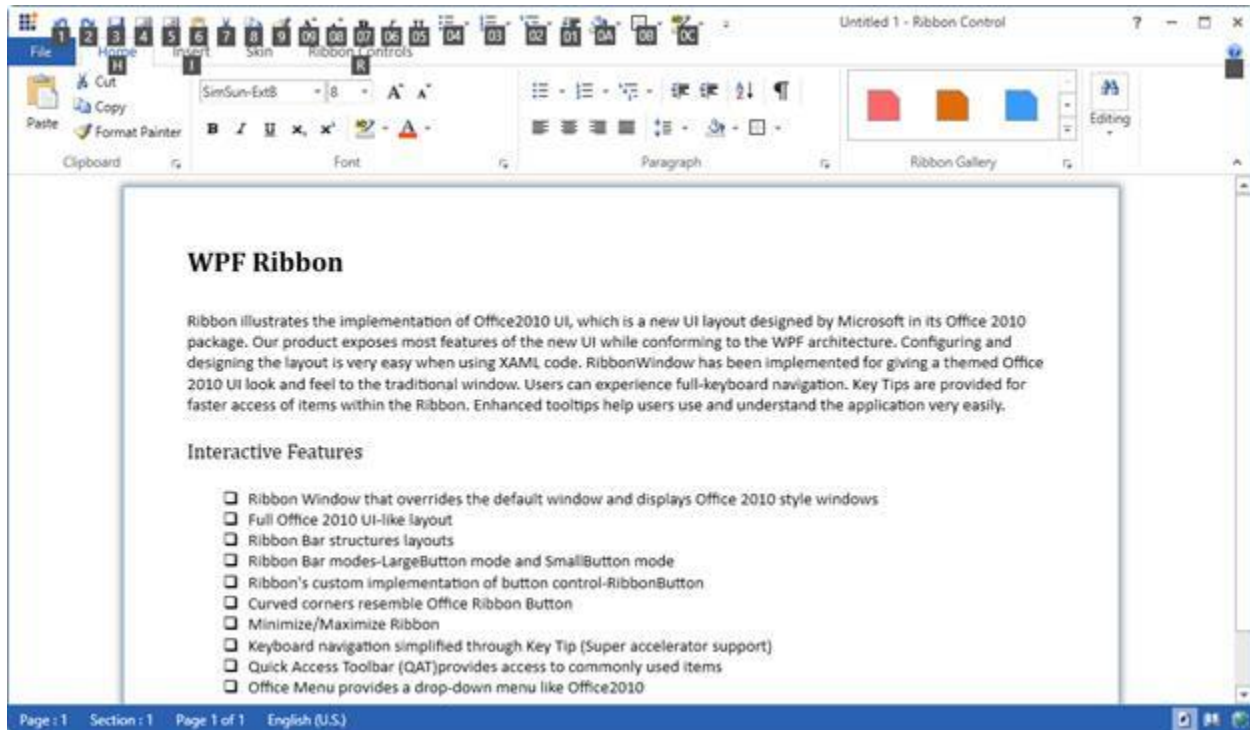
KeyTip for QAT items

KeyTip for the QAT items gets displayed by default by pressing Alt Key. The KeyTips are arranged in the order like...1,2,3,4,5,6,7,8,9,0,09,08,07.....01,0A,0B..... etc.

XML

```
<syncfusion:Ribbon.QuickAccessToolBar>
<syncfusion:QuickAccessToolBar>
<syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Save"
syncfusion:RibbonCommandManager.SynchronizedItem="Save" />
<syncfusion:RibbonButton Label="Quick Print"
syncfusion:RibbonCommandManager.SynchronizedItem="Quick Print"/>
<syncfusion:RibbonButton Label="Print Preview"
syncfusion:RibbonCommandManager.SynchronizedItem="Print Preview"/>
<syncfusion:RibbonButton Label="Undo"
syncfusion:RibbonCommandManager.SynchronizedItem="Undo" />
<syncfusion:RibbonButton Label="Redo"
syncfusion:RibbonCommandManager.SynchronizedItem="Redo" />
<syncfusion:RibbonButton Label="Paste"
syncfusion:RibbonCommandManager.SynchronizedItem="Paste"/>
</syncfusion:QuickAccessToolBar.QATMenuItems>
<syncfusion:RibbonButton Label="Undo" SmallIcon="/Resources/Undo16.png"
SizeForm="ExtraSmall"
ToolTip="Undo" syncfusion:RibbonCommandManager.SynchronizedItem="Undo" />
<syncfusion:RibbonButton Label="Redo" SmallIcon="/Resources/Redo16.png"
SizeForm="ExtraSmall"
ToolTip="Redo" syncfusion:RibbonCommandManager.SynchronizedItem="Redo"/>
<syncfusion:RibbonButton Command="ApplicationCommands.Save"
SizeForm="ExtraSmall"
syncfusion:RibbonCommandManager.SynchronizedItem="Save"
SmallIcon="/Resources/Save16.png"/>
</syncfusion:QuickAccessToolBar>
```

```
</syncfusion:Ribbon.QuickAccessToolBar>
```



How to access particular item in ribbon using KeyTip

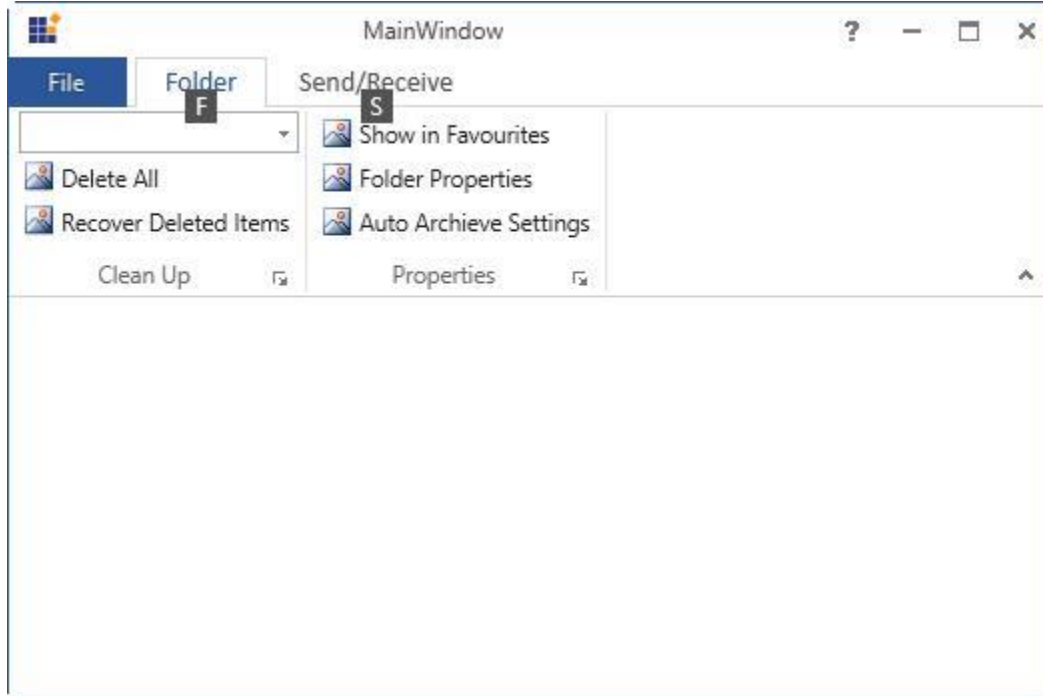
The below example illustrates how to access particular Ribbon item using KeyTip

XML

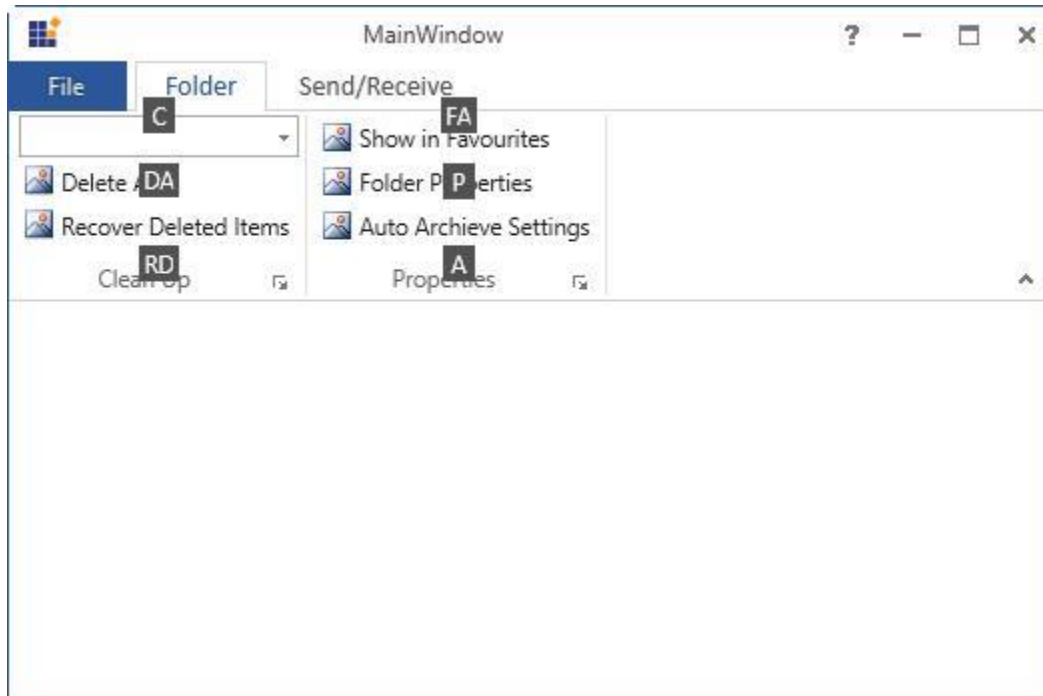
```
<syncfusion:Ribbon VerticalAlignment="Top" >
<syncfusion:RibbonTab syncfusion:Ribbon.KeyTip="F" Caption="Folder"
IsChecked="True" >
<syncfusion:RibbonBar Header="Clean Up" >
<syncfusion:RibbonComboBox syncfusion:Ribbon.KeyTip="C" Label="CleanUp">
<syncfusion:RibbonComboBoxItem Content="Cleanup Folder"/>
<syncfusion:RibbonComboBoxItem Content="Cleanup Folder and Sub Folders"/>
</syncfusion:RibbonComboBox>
<syncfusion:RibbonButton syncfusion:Ribbon.KeyTip="DA" SizeForm="Small"
Label="Delete All"/>
<syncfusion:RibbonButton syncfusion:Ribbon.KeyTip="RD" SizeForm="Small"
Label="Recover Deleted Items"/>
</syncfusion:RibbonBar >
<syncfusion:RibbonBar Name="_ribbonBar" Header="Properties" >
<syncfusion:RibbonButton syncfusion:Ribbon.KeyTip="FA" SizeForm="Small"
Label="Show in Favourites"/>
<syncfusion:RibbonButton syncfusion:Ribbon.KeyTip="P" SizeForm="Small"
Label="Folder Properties"/>
<syncfusion:RibbonButton syncfusion:Ribbon.KeyTip="A" SizeForm="Small"
Label="Auto Archieve Settings"/>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>
<syncfusion:RibbonTab syncfusion:Ribbon.KeyTip="S" Caption="Send/Receive"
IsChecked="False" />
```

```
</syncfusion:Ribbon>
```

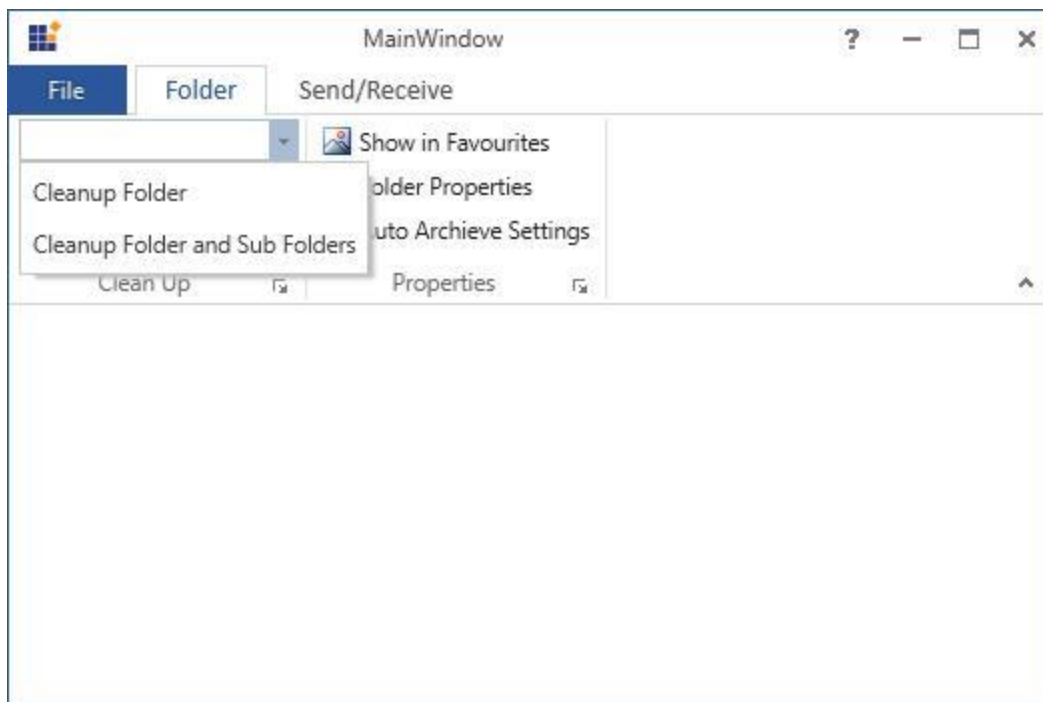
The below screenshot displays the steps involved to access RibbonComboBox. When Alt key is pressed, KeyTip for the RibbonTab gets displayed. Then choose the desired RibbonTab with the KeyTip



In the Next Step, all the KeyTips under RibbonTab gets displayed



Finally, Press the RibbonComboBox KeyTip to access it



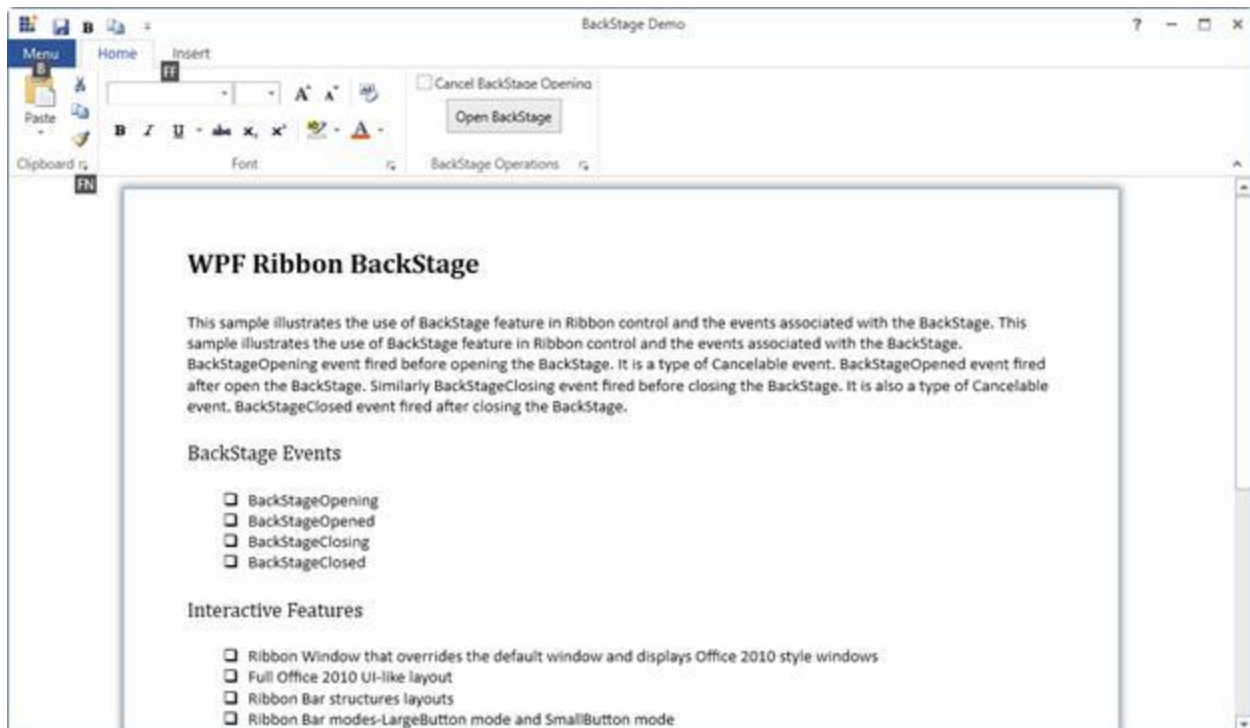
How does BackStage can be accessed using KeyTip

The backstage feature of the Ribbon control provides KeyTip support to display KeyTips on pressing the Alt key. It is also possible to set KeyTip for the BackStage elements which includes BackStageCommandButton, BackStageTabItem

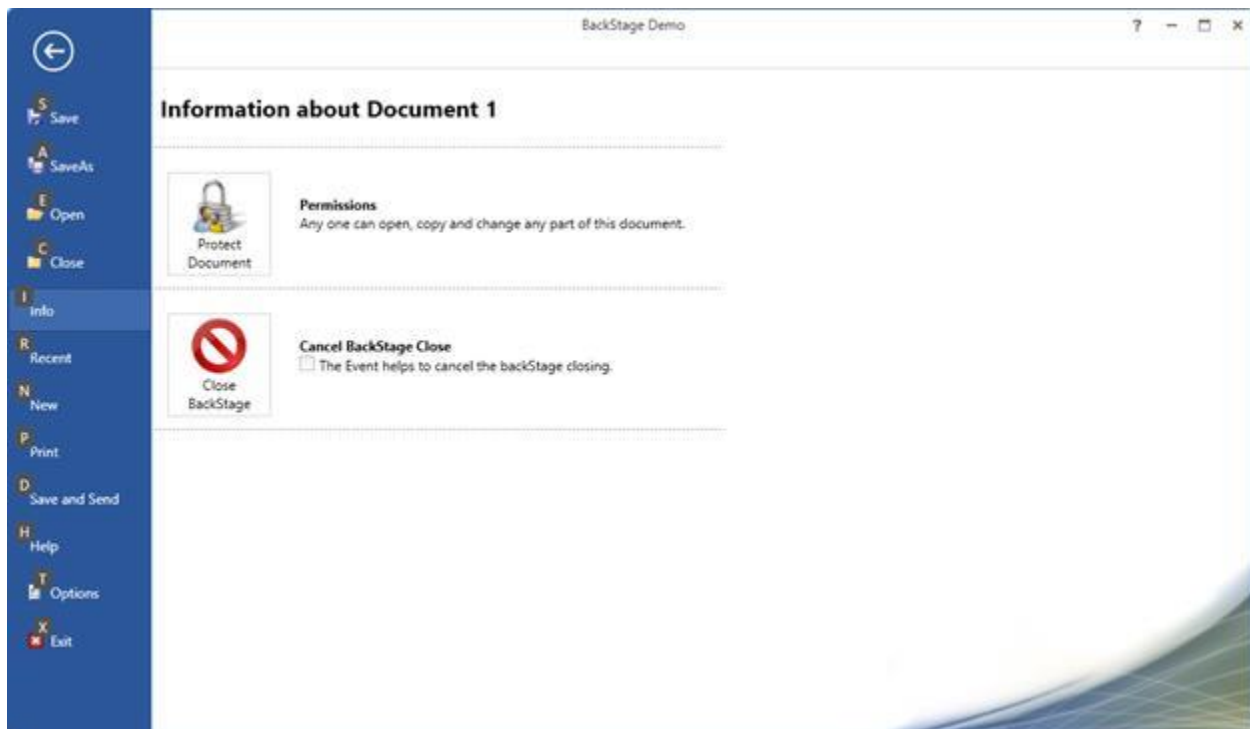
XML

```
<syncfusion:Ribbon.BackStage>
<syncfusion:Backstage x:Name="_ribbonBackStage"
syncfusion:Ribbon.KeyTip="B">
<syncfusion:BackStageCommandButton Header="Save"
syncfusion:Ribbon.KeyTip="S" Command="Save" Icon="/Resources/Save16.png"/>
<syncfusion:BackStageCommandButton Header="Save As"
syncfusion:Ribbon.KeyTip="A" Command="SaveAs"
Icon="/Resources/Save16.png"/>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="I" Header="Info"/>
<syncfusion:BackstageTabItem syncfusion:Ribbon.KeyTip="R" Header="Recent"/>
</syncfusion:Backstage>
</syncfusion:Ribbon.BackStage>
```

In the below screenshot, "B" is set as KeyTip for the Backstage. So when the user press "B", it automatically open the BackStage



The following screenshot display the KeyTip for the BackStageCommand and BackStageTabItem



Touch Support in WPF Ribbon

Ribbon control have touch support and it provides Touch UI which is easy to access the element in Ribbon.

How to enable touch in RibbonWindow

To enable touch in the RibbonWindow, set `EnableTouch` property of the `SkinStorage` as `True`. The following code snippet illustrates this

XML

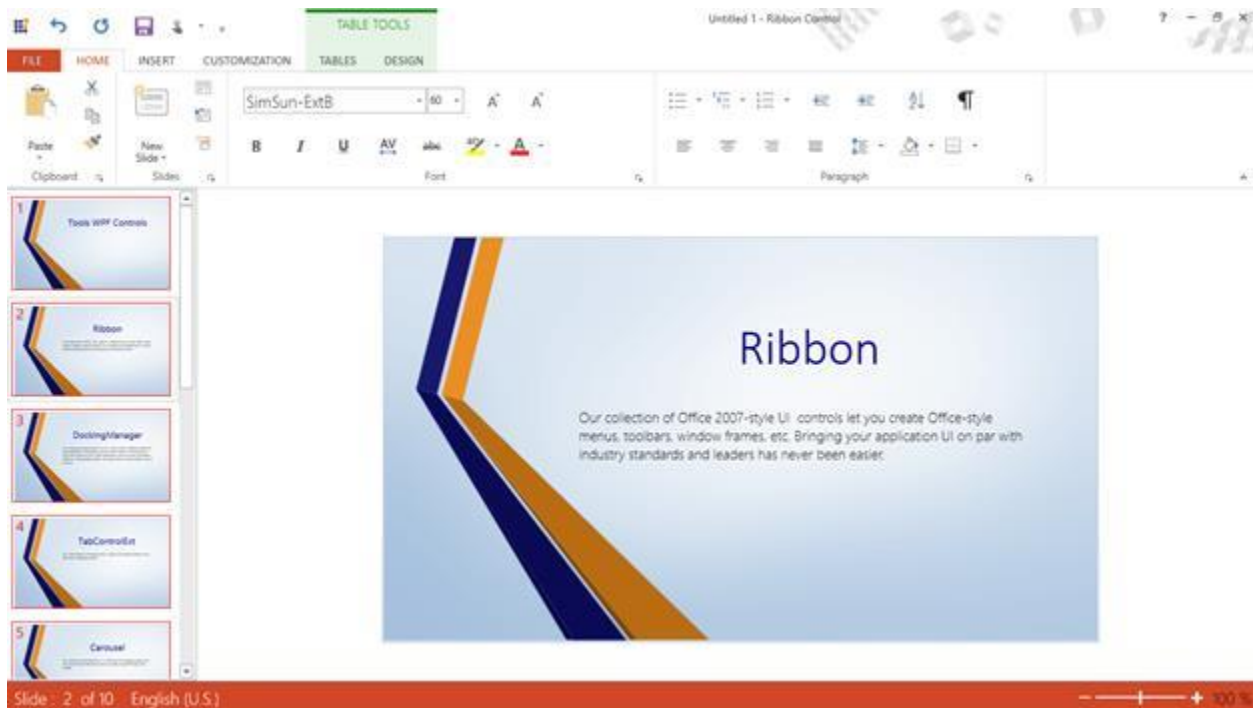
```
<syncfusion:RibbonWindow
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="RibbonButtonPanel.MainWindow"
Title="MainWindow" Height="350" Width="525"
syncfusion:SkinStorage.VisualStyle="Office2013" x:Name="_ribbonWindow"
syncfusion:SkinStorage.EnableTouch="True"/>
```

C#

```
SkinStorage.SetEnableTouch(_ribbonWindow, true);
```

VB.NET

```
SkinStorage.SetEnableTouch(_ribbonWindow, True)
```

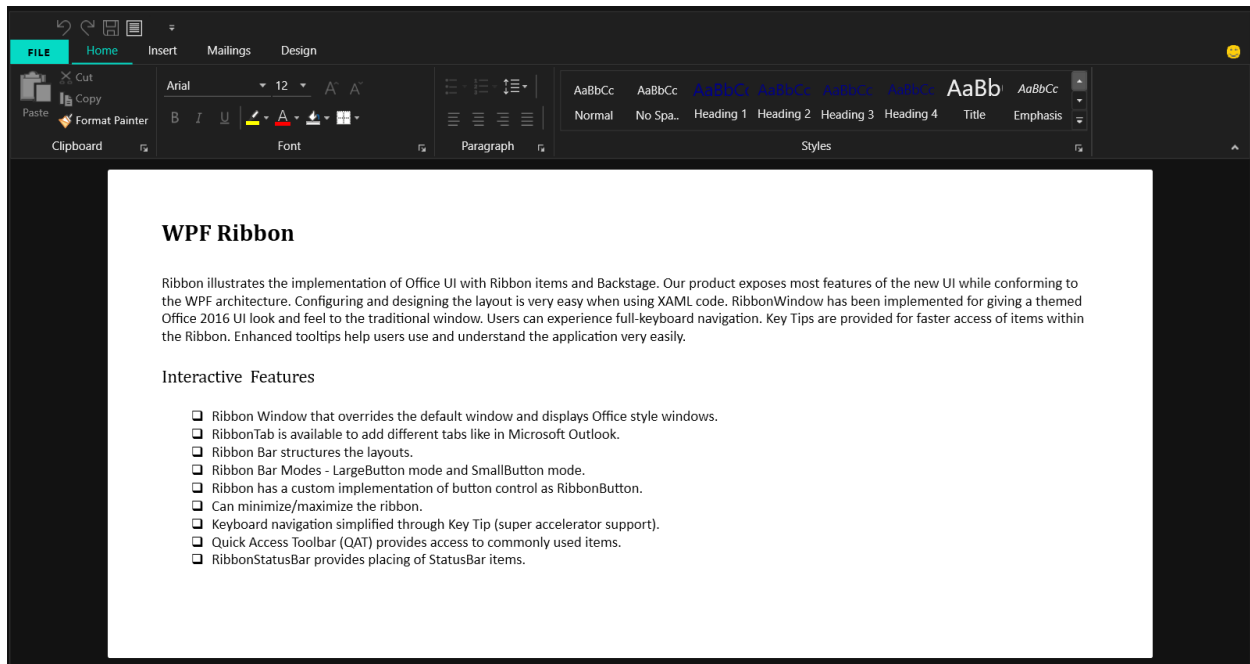


Styling and Templates in WPF Ribbon

Theme

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

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



Patterns and Practices in WPF Ribbon

Ribbon with MVVM

For better control customization MVVM pattern can be followed. The following steps illustrate a simple MVVM pattern with Ribbon control

1.Create new WPF project 2.Add **Model** class for each element which need to be included in Ribbon control. In this sample class has been created for RibbonTab, RibbonBar and RibbonItem

Model

C#

```
public class CustomRibbonTab
{
    public string TabHeader { get; set; }
    public ObservableCollection<CustomRibbonBar> CustomRibbonBars { get; set; }
    public CustomRibbonTab()
    {
        CustomRibbonBars = new ObservableCollection<CustomRibbonBar>();
    }
}

public class CustomRibbonBar
{
    public string BarHeader { get; set; }
    public ObservableCollection<CustomRibbonItem> CustomRibbonItems { get; set; }
}

public CustomRibbonBar()
{
    CustomRibbonItems = new ObservableCollection<CustomRibbonItem>();
}

public class CustomRibbonItem
{
    public CustomRibbonItem()
    {
    }
}
```

```

IsSplitButton = false;
IsBoolean = true;
}
public string ItemHeader
{
    get;
    set;
}
public string Image { get; set; }
public bool IsBoolean { get; set; }
public bool IsLarge { get; set; }
public bool IsSplitButton { get; set; }
}

```

VB.NET

```

Public Class CustomRibbonTab
Public Property TabHeader() As String
Public Property CustomRibbonBars() As ObservableCollection(Of
CustomRibbonBar)
Public Sub New()
CustomRibbonBars = New ObservableCollection(Of CustomRibbonBar) ()
End Sub
End Class
Public Class CustomRibbonBar
Public Property BarHeader() As String
Public Property CustomRibbonItems() As ObservableCollection(Of
CustomRibbonItem)
Public Sub New()
CustomRibbonItems = New ObservableCollection(Of CustomRibbonItem) ()
End Sub
End Class
Public Class CustomRibbonItem
Public Sub New()
IsSplitButton = False
IsBoolean = True
End Sub
Public Property ItemHeader() As String
Public Property Image() As String
Public Property IsBoolean() As Boolean
Public Property IsLarge() As Boolean
Public Property IsSplitButton() As Boolean
End Class

```

3. Create **ViewModel** class where the collection has been declared and the items has been populated to it.

ViewModel

C#

```

public class ViewModel
{
    public ObservableCollection<CustomRibbonTab> CustomRibbonTabs { get; set; }
    public ViewModel()
    {

```

```

CustomRibbonTabs = new ObservableCollection<CustomRibbonTab>();
PopulateRibbonTabs();
}
void PopulateRibbonTabs()
{
    CustomRibbonTab Tab1 = new CustomRibbonTab() { TabHeader = "Home" };
    PopulateRibbonHomeBars(Tab1);
    CustomRibbonTabs.Add(Tab1);
}
//Home Tab
void PopulateRibbonHomeBars(CustomRibbonTab Tab)
{
    CustomRibbonBar Bar1 = new CustomRibbonBar() { BarHeader = "Clipboard" };
    PopulateRibbonNewItems(Bar1);
    CustomRibbonBar Bar2 = new CustomRibbonBar() { BarHeader = "Editing" };
    PopulateRibbonEditingItems(Bar2);
    Tab.CustomRibbonBars.Add(Bar1);
    Tab.CustomRibbonBars.Add(Bar2);
}
void PopulateRibbonNewItems(CustomRibbonBar Bar)
{
    CustomRibbonItem Item1 = new CustomRibbonItem() { ItemHeader = "Paste",
    IsLarge = true, Image = "Paste32.png" };
    CustomRibbonItem Item2 = new CustomRibbonItem() { ItemHeader = "Cut", Image
    = "Cut16.png" };
    CustomRibbonItem Item3 = new CustomRibbonItem() { ItemHeader = "Copy", Image
    = "Copy16.png" };
    CustomRibbonItem Item4 = new CustomRibbonItem() { ItemHeader = "Format
    Painter", Image = "FormatPainter16.png" };
    Bar.CustomRibbonItems.Add(Item1);
    Bar.CustomRibbonItems.Add(Item2);
    Bar.CustomRibbonItems.Add(Item3);
    Bar.CustomRibbonItems.Add(Item4);
}
private void PopulateRibbonEditingItems(CustomRibbonBar Bar)
{
    CustomRibbonItem Item1 = new CustomRibbonItem() { ItemHeader = "Hyperlink",
    IsLarge = true, Image = "hyperlink32.png" };
    CustomRibbonItem Item2 = new CustomRibbonItem() { ItemHeader = "Replace",
    IsLarge = true, Image = "replace_32.png" };
    CustomRibbonItem Item3 = new CustomRibbonItem() { ItemHeader = "Zoom",
    IsLarge = true, Image = "Zoom_32x32.png" };
    Bar.CustomRibbonItems.Add(Item1);
    Bar.CustomRibbonItems.Add(Item2);
    Bar.CustomRibbonItems.Add(Item3);
}
}

```

VB.NET

```

Public Class ViewModel
    Public Property CustomRibbonTabs() As ObservableCollection(Of
    CustomRibbonTab)
    Public Sub New()
        CustomRibbonTabs = New ObservableCollection(Of CustomRibbonTab) ()
        PopulateRibbonTabs()
    End Sub
End Class

```

```
End Sub
Private Sub PopulateRibbonTabs()
Dim Tab1 As New CustomRibbonTab() With {.TabHeader = "Home"}
PopulateRibbonHomeBars(Tab1)
CustomRibbonTabs.Add(Tab1)
End Sub
'Home Tab
Private Sub PopulateRibbonHomeBars(ByVal Tab As CustomRibbonTab)
Dim Bar1 As New CustomRibbonBar() With {.BarHeader = "Clipboard"}
PopulateRibbonNewItem(Bar1)
Dim Bar2 As New CustomRibbonBar() With {.BarHeader = "Editing"}
PopulateRibbonEditingItems(Bar2)
Tab.CustomRibbonBars.Add(Bar1)
Tab.CustomRibbonBars.Add(Bar2)
End Sub
Private Sub PopulateRibbonNewItem(ByVal Bar As CustomRibbonBar)
Dim Item1 As New CustomRibbonItem() With {
.ItemHeader = "Paste",
.IsLarge = True,
.Image = "Paste32.png"
}
Dim Item2 As New CustomRibbonItem() With {
.ItemHeader = "Cut",
.Image = "Cut16.png"
}
Dim Item3 As New CustomRibbonItem() With {
.ItemHeader = "Copy",
.Image = "Copy16.png"
}
Dim Item4 As New CustomRibbonItem() With {
.ItemHeader = "Format Painter",
.Image = "FormatPainter16.png"
}
Bar.CustomRibbonItems.Add(Item1)
Bar.CustomRibbonItems.Add(Item2)
Bar.CustomRibbonItems.Add(Item3)
Bar.CustomRibbonItems.Add(Item4)
End Sub
Private Sub PopulateRibbonEditingItems(ByVal Bar As CustomRibbonBar)
Dim Item1 As New CustomRibbonItem() With {
.ItemHeader = "Hyperlink",
.IsLarge = True,
.Image = "hyperlink32.png"
}
Dim Item2 As New CustomRibbonItem() With {
.ItemHeader = "Replace",
.IsLarge = True,
.Image = "replace_32.png"
}
Dim Item3 As New CustomRibbonItem() With {
.ItemHeader = "Zoom",
.IsLarge = True,
.Image = "Zoom_32x32.png"
}
Bar.CustomRibbonItems.Add(Item1)
Bar.CustomRibbonItems.Add(Item2)
Bar.CustomRibbonItems.Add(Item3)
```

```
End Sub
End Class
```

4. In XAML bind the collection to Ribbon control and use ItemContainerStyle to bind the inner level items like RibbonBar and RibbonItems

MainWindow.xaml

XML

```
<syncfusion:RibbonWindow x:Class="RibbonSample_in_MVVM.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
syncfusion:SkinStorage.VisualStudio="Office2013"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:RibbonSample_in_MVVM"
Title="MainWindow" Height="350" Width="525">
<Window.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.Tools.WPF;component/Framework/Ribbon/Themes/Office2013St
yle.xaml" />
</ResourceDictionary.MergedDictionaries>
<local:BooltoSizeformConverter x:Key="sizeform"/>
<local:ItemDataTemplateSelector x:Key="selector"/>
<local:ImageConverter x:Key="image"/>
<DataTemplate x:Key="Ribbonbutton">
<syncfusion:RibbonButton Label="{Binding ItemHeader}" SizeForm="{Binding
IsLarge, Converter={StaticResource sizeform}}" LargeIcon="{Binding
Image,Converter={StaticResource image}}" SmallIcon="{Binding
Image,Converter={StaticResource image}}" />
</DataTemplate>
<DataTemplate x:Key="Splitbutton">
<syncfusion:SplitButton Label="{Binding ItemHeader}" SizeForm="{Binding
IsLarge, Converter={StaticResource sizeform}}" LargeIcon="{Binding
Image,Converter={StaticResource image}}" SmallIcon="{Binding
Image,Converter={StaticResource image}}" />
</DataTemplate>
</ResourceDictionary>
</Window.Resources>
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:Ribbon
x:Name="_ribbon" VerticalAlignment="Top"
ItemsSource="{Binding CustomRibbonTabs}"
<syncfusion:Ribbon.ItemContainerStyle>
<Style TargetType="{x:Type syncfusion:RibbonTab}">
<Setter Property="Caption" Value="{Binding TabHeader}"></Setter>
<Setter Property="ItemsSource" Value="{Binding CustomRibbonBars}" />
<Setter Property="ItemContainerStyle">
<Setter.Value>
<Style BasedOn="{StaticResource Office2013RibbonBarStyle}"
TargetType="{x:Type syncfusion:RibbonBar}">
<Setter Property="Header" Value="{Binding BarHeader}" />
```

```

<Setter Property="ItemsSource" Value="{Binding CustomRibbonItems}"/>
<Setter Property="ItemTemplateSelector" Value="{StaticResource selector}"/>
</Style>
</Setter.Value>
</Setter>
</Style>
</syncfusion:Ribbon.ItemContainerStyle>
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>

```

5. Converter class is used to set **SizeForm** for the Ribbon items and to set images.

[Converter.cs](#)

C#

```

public class BooltoSizeformConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        if (value.Equals(true))
        {
            return "Large";
        }
        else
        {
            return "Small";
        }
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

public class ItemDataTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject
        container)
    {
        FrameworkElement element = container as FrameworkElement;
        if (element != null && item != null)
        {
            if (item is CustomRibbonItem && (item as CustomRibbonItem).IsSplitButton)
            {
                return element.FindResource("Splitbutton") as DataTemplate;
            }
            else
            {
                return element.FindResource("Ribbonbutton") as DataTemplate;
            }
        }
        return null;
    }
}

```

```
public class ImageConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        if (value != null)
        {
            string str = value.ToString();
            return "../Images/" + str;
        }
        else
        {
            return value;
        }
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

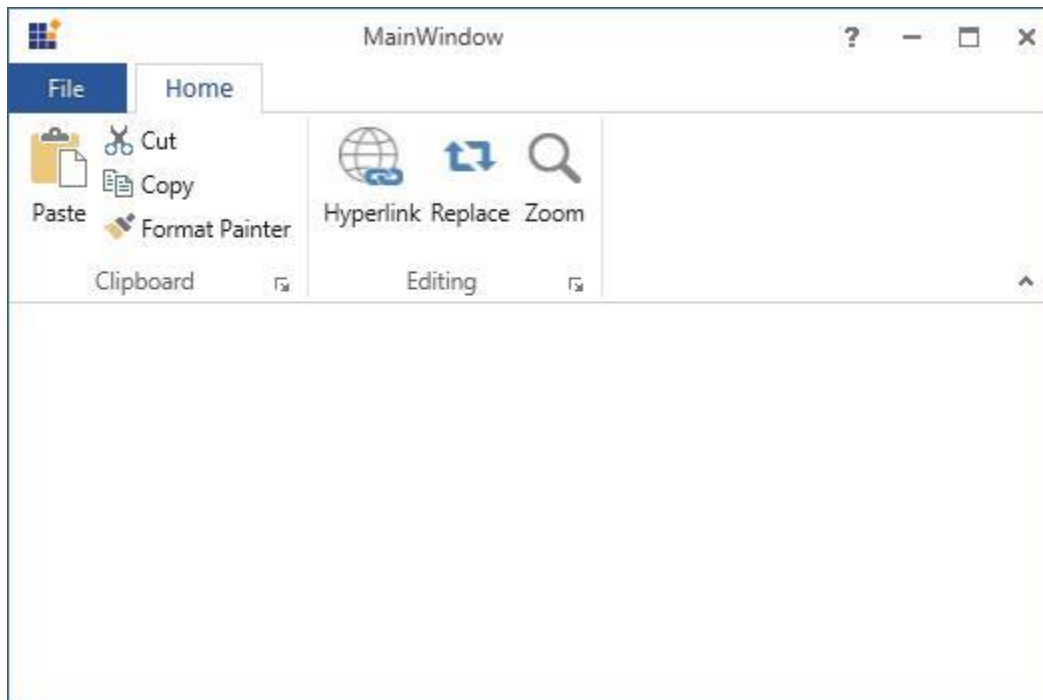
VB.NET

```
Public Class BooltoSizeformConverter
    Implements IValueConverter
    Public Function Convert(ByVal value As Object, ByVal targetType As Type,
        ByVal parameter As Object, ByVal culture As
        System.Globalization.CultureInfo) As Object
        If value.Equals(True) Then
            Return "Large"
        Else
            Return "Small"
        End If
    End Function
    Public Function ConvertBack(ByVal value As Object, ByVal targetType As Type,
        ByVal parameter As Object, ByVal culture As
        System.Globalization.CultureInfo) As Object
        Throw New NotImplementedException()
    End Function
End Class

Public Class ItemDataTemplateSelector
    Inherits DataTemplateSelector
    Public Overrides Function SelectTemplate(ByVal item As Object, ByVal
        container As DependencyObject) As DataTemplate
        Dim element As FrameworkElement = TryCast(container, FrameworkElement)
        If element IsNot Nothing AndAlso item IsNot Nothing Then
            If TypeOf item Is CustomRibbonItem AndAlso (TryCast(item,
                CustomRibbonItem)).IsSplitButton Then
                Return TryCast(element.FindResource("Splitbutton"), DataTemplate)
            Else
                Return TryCast(element.FindResource("Ribbonbutton"), DataTemplate)
            End If
        End If
        Return Nothing
    End Function
End Class
```

```
End Class
Public Class ImageConverter
Implements IValueConverter
Public Function Convert(ByVal value As Object, ByVal targetType As Type,
ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object
If value IsNot Nothing Then
Dim str As String = value.ToString()
Return "../Images/" & str
Else
Return value
End If
End Function
Public Function ConvertBack(ByVal value As Object, ByVal targetType As Type,
ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object
Throw New System.NotImplementedException()
End Function
End Class
```

Now the output displays the Ribbon control with the populated items



Practice with PRISM

Ribbon control provides PRISM support. The following steps explain about creating simple sample project in the PRISM.

1.Create new WPF project and add the following references to the solution project.

Microsoft.Practices.ServiceLocation.dll *Microsoft.Practices.Unity.dll*

Microsoft.Practices.Unity.Configuration.dll *Microsoft.Practices.Unity.RegistrationByConvention.dll*

Prism.dll *Prism.Unity.Wpf.dll* * *Prism.Wpf.dll* 2.Rename MainWindow to Shell in the Project 3.Add new class called Bootstrapper.cs to initialize the prism application.

C#

```
class Bootstrapper : UnityBootstrapper
{
    protected override DependencyObject CreateShell()
    {
        return Container.Resolve<Shell>();
    }
    protected override void InitializeShell()
    {
        base.InitializeShell();
        App.Current.MainWindow = (Shell) this.Shell;
        App.Current.MainWindow.Show();
    }
    protected override void ConfigureModuleCatalog()
    {
        base.ConfigureModuleCatalog();
        ModuleCatalog catalog = (ModuleCatalog) this.ModuleCatalog;
        catalog.AddModule(typeof(HomeTabModule.HomeTabModules));
    }
}
```

VB.NET

```
Friend Class Bootstrapper
Inherits UnityBootstrapper
Protected Overrides Function CreateShell() As DependencyObject
Return Container.Resolve(Of Shell)()
End Function
Protected Overrides Sub InitializeShell()
MyBase.InitializeShell()
App.Current.MainWindow = CType(Me.Shell, Shell)
App.Current.MainWindow.Show()
End Sub
Protected Overrides Sub ConfigureModuleCatalog()
MyBase.ConfigureModuleCatalog()
Dim catalog As ModuleCatalog = CType(Me.ModuleCatalog, ModuleCatalog)
catalog.AddModule(GetType(HomeTabModule.HomeTabModules))
End Sub
End Class
```

4. Override OnStartup method in the App.xaml.cs to execute Bootstrapper when the application starts

C#

```
public partial class App : Application
{
    protected override void OnStartup(StartupEventArgs e)
    {
        base.OnStartup(e);
        Bootstrapper bootstrapper = new Bootstrapper();
        bootstrapper.Run();
    }
}
```

VB.NET

```
Partial Public Class App
Inherits Application
Protected Overrides Sub OnStartup(ByVal e As StartupEventArgs)
MyBase.OnStartup(e)
Dim bootstrapper As New Bootstrapper()
bootstrapper.Run()
End Sub
End Class
```

5.Next step is to create regions in the shell. To do this, first add the following namespace in the shell Window

XML

```
xmlns:Cal="http://www.codeplex.com/CompositeWPF"
```

In the below code, a region called “Tabs” has been created to load RibbonTab Module views

XML

```
<syncfusion:RibbonWindow x:Class="RibbonDemoSample.Shell"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:Cal="http://www.codeplex.com/CompositeWPF"
syncfusion:SkinStorage.VisualStudio="Office2013"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525">
<Grid>
<syncfusion:Ribbon Cal:RegionManager.RegionName="Tabs"
VerticalAlignment="Top">
</syncfusion:Ribbon>
</Grid>
</syncfusion:RibbonWindow>
```

6.Adding Module to the project

Right click the Solution project, point to “Add” and then click “NewProject”. The new Window called AddNewProject gets open. Select “ClassLibrary” from Visual C# then rename the project with desired name and click “Ok”. Now a new Module has been created in the Solution Project

Now add following assemblies to the Module project

- PresentationCore.dll
- PresentationFramework.dll
- WindowsBase.dll

Also add following Prism assemblies

- Microsoft.Practices.ServiceLocation.dll
- Microsoft.Practices.Unity.dll
- Microsoft.Practices.Unity.Configuration.dll

- Microsoft.Practices.Unity
- RegistrationByConvention.dll
- Prism.dll
- Prism.Unity.Wpf.dll
- Prism.WPF.dll 7. In the Shell project, add the reference to the “HomeTabModule” project by registering with ModuleCatalog instance in the GetModuleCatalog method

C#

```
class Bootstrapper : UnityBootstrapper
{
    protected override DependencyObject CreateShell()
    {
        return Container.Resolve<Shell>();
    }
    protected override void InitializeShell()
    {
        base.InitializeShell();
        App.Current.MainWindow = (Shell) this.Shell;
        App.Current.MainWindow.Show();
    }
    protected override void ConfigureModuleCatalog()
    {
        base.ConfigureModuleCatalog();
        ModuleCatalog catalog = (ModuleCatalog) this.ModuleCatalog;
        catalog.AddModule(typeof(HomeTabModule.HomeTabModules));
    }
}
```

VB.NET

```
Friend Class Bootstrapper
Inherits UnityBootstrapper
Protected Overrides Function CreateShell() As DependencyObject
Return Container.Resolve(Of Shell)()
End Function
Protected Overrides Sub InitializeShell()
MyBase.InitializeShell()
App.Current.MainWindow = CType(Me.Shell, Shell)
App.Current.MainWindow.Show()
End Sub
Protected Overrides Sub ConfigureModuleCatalog()
MyBase.ConfigureModuleCatalog()
Dim catalog As ModuleCatalog = CType(Me.ModuleCatalog, ModuleCatalog)
catalog.AddModule(GetType(HomeTabModule.HomeTabModules))
End Sub
End Class
```

8.Adding Views to the Module**XML**

```
<syncfusion:RibbonTab x:Class="HomeTabModule.HomeTab"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
d:DesignHeight="300" d:DesignWidth="300" Caption="Home" IsChecked="True" >
<syncfusion:RibbonBar>
<syncfusion:RibbonBar.Resources>
<ResourceDictionary
Source="/Syncfusion.Tools.WPF;component/Framework/Ribbon/Themes/Office2013St
yle.xaml" />
</syncfusion:RibbonBar.Resources>
<syncfusion:RibbonBar.Style>
<Style BasedOn="{StaticResource Office2013RibbonBarStyle}"
TargetType="{x:Type syncfusion:RibbonBar}">
</Style>
</syncfusion:RibbonBar.Style>
<syncfusion:RibbonButton Width="44"
Margin="3,0,3,3"
Label="Paste"
SizeForm="Large"
syncfusion:Ribbon.KeyTip="CP"
syncfusion:RibbonCommandManager.SynchronizedItem="Paste">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Paste (Ctrl+V)">
<TextBlock Width="130"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
Text="Paste the contents of clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton HorizontalAlignment="Left"
Label="Cut"
SizeForm="Small"
syncfusion:Ribbon.KeyTip="CT">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Cut (Ctrl+X)">
<TextBlock Width="130"
HorizontalAlignment="Left"
Text="Cut the selection and put it on the clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton HorizontalAlignment="Left"
Label="Copy"
SizeForm="Small"
syncfusion:Ribbon.KeyTip="CY">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Copy (Ctrl+C)">
<TextBlock Width="130"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
Text="Copy the selection and put it on the clipboard."
TextWrapping="Wrap" />
</syncfusion:ScreenTip>

```

```

</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
<syncfusion:RibbonButton Label="Format Painter"
SizeForm="Small"
syncfusion:Ribbon.KeyTip="CR">
<syncfusion:RibbonButton.ToolTip>
<syncfusion:ScreenTip Description="Format painter (Ctrl+Shift+C)"
HelpText="Press F1 for more help.">
<TextBlock Width="175"
HorizontalAlignment="Left"
Foreground="#FF4C4C4C"
TextWrapping="Wrap">
<Run Text="Copy formatting from one place and apply it to another." />
<LineBreak />
<LineBreak />
<Run Text="Double-click this button to apply the same formatting to multiple
places in the document." />
</TextBlock>
</syncfusion:ScreenTip>
</syncfusion:RibbonButton.ToolTip>
</syncfusion:RibbonButton>
</syncfusion:RibbonBar>
</syncfusion:RibbonTab>

```

9. Add a region to the shell and after creating View for the Module, register the view as Module using the below code

C#

```

public class HomeTabModules : IModule
{
    private readonly IRegionManager regionManager;
    public HomeTabModules(IRegionManager regionManager)
    {
        this.regionManager = regionManager;
    }
    public void Initialize()
    {
        regionManager.Regions["Tabs"].Add(new HomeTab());
    }
}

```

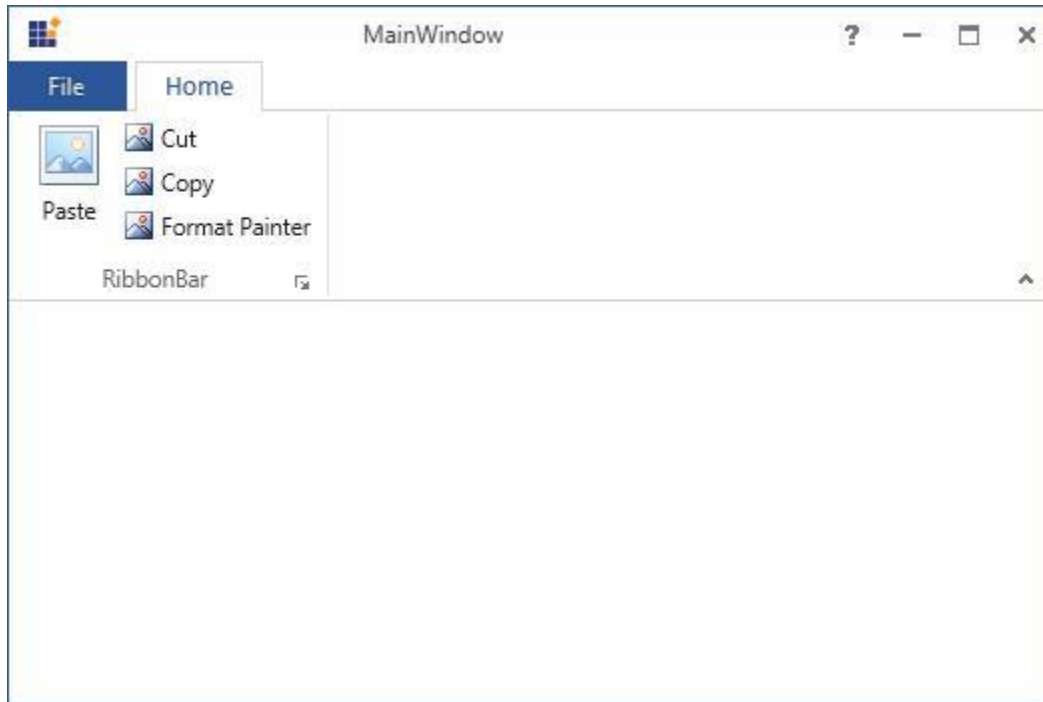
VB.NET

```

Public Class HomeTabModules
Implements IModule
Private ReadOnly regionManager As IRegionManager
Public Sub New(ByVal regionManager As IRegionManager)
Me.regionManager = regionManager
End Sub
Public Sub Initialize()
regionManager.Regions("Tabs").Add(New HomeTab())
End Sub
End Class

```

Now run the project. RibbonTabModule get added as one of the Module in the Shell. Similarly any number of Modules can be added based on the complexity of the project.



How to

Handle Pop-Up Opening and Closing Event in ApplicationMenu

To perform the action based on opening and closing of the ApplicationMenu, make use of `IsPopupOpenChanged` event.

XML

```
<syncfusion:ApplicationMenu Name="_applicationMenu" Width="38" Height="38"
ApplicationButtonImage="Resources/App.ico"
IsPopupOpenChanged="_applicationMenu_IsPopupOpenChanged">
```

C#

```
private void _applicationMenu_IsPopupOpenChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    if((d as ApplicationMenu).IsPopupOpen)
    {
        newMenuButton.IsEnabled = false;
    }
    else
    {
        newMenuButton.IsEnabled = true;
    }
}
```

VB.NET

```

Private Sub _applicationMenu_IsPopupOpenChanged(ByVal d As DependencyObject,
ByVal e As DependencyPropertyChangedEventArgs)
If(TryCast(d, ApplicationMenu)).IsPopupOpen Then
newMenuButton.IsEnabled = False
Else
newMenuButton.IsEnabled = True
End If
End Sub

```

SfRichTextBoxAdv

WPF RichTextBox (SfRichTextBoxAdv) Overview

The SfRichTextBoxAdv control allows you to view, edit and print rich text content including text, images, hyperlink etc. arranged in sections, tables, paragraphs, headers, footers and comments. You can perform all editing operations using keyboard, mouse and touch.



Features

The SfRichTextBoxAdv control supports basic features for editing rich text contents such as

- Viewing and editing rich text, images, tables and comments.
- Importing and exporting word documents (.doc, .docx), rich text format documents (.rtf), HTML documents (.htm, .html), XAML documents (.xaml) and text documents (.txt).
- Printing rendered contents as page by page.
- Supports all image types except Metafile images.
- Supports undo and redo all editing operation such as inserting text, table, images, hyperlink, comments and all formatting operations such as bold, italic etc.
- Supports first page header and footer as well as odd and even page headers and footers.
- Moving or copying portion of the document to clipboard and also pasting rich text contents from clipboard into the document.
- Supports loading encrypted word documents with valid password.

Note: Currently, the SfRichTextBoxAdv cannot edit rich text in headers and footers.

You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Getting Started with WPF RichTextBox (SfRichTextBoxAdv)

This section describes how to get started with [WPF RichTextBox](#) (SfRichTextBoxAdv) control.

Assembly Reference

You can find WPF RichTextBox (SfRichTextBoxAdv) control from the following assembly under the namespace Syncfusion.Windows.Controls.RichTextBoxAdv

- Syncfusion.SfRichTextBoxAdv.WPF

The following assembly references are required to deploy SfRichTextBoxAdv control in your application.

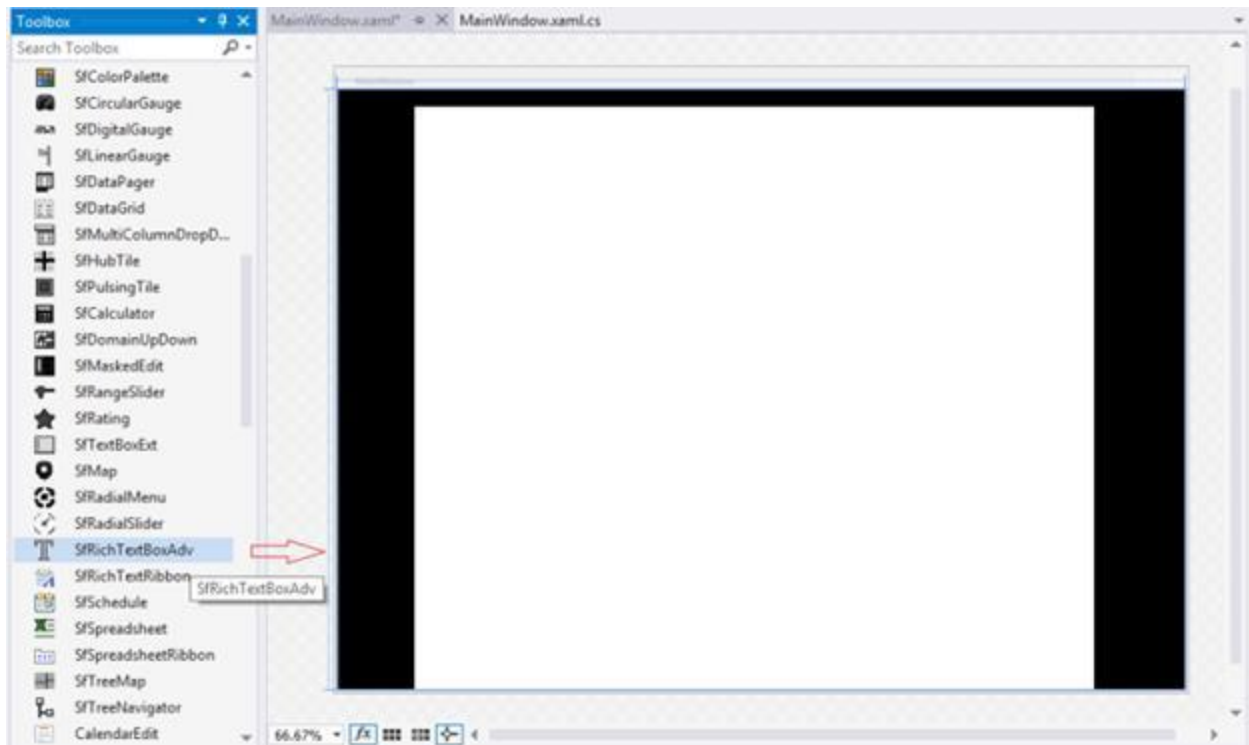
- Syncfusion.Compression.Base
- Syncfusion.OfficeChart.Base
- Syncfusion.Shared.WPF
- For 3.5 and 4.0 Frameworks – Syncfusion.DocIO.ClientProfile
- For 4.5 and higher Frameworks – Syncfusion.DocIO.Base

Note: Starting with v16.2.0.41 (2018 Vol 2), if you reference Syncfusion assemblies from trial setup or from the NuGet feed, you also have to add "Syncfusion.Licensing" assembly reference and 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.

Adding SfRichTextBoxAdv to an application

After adding the aforementioned assembly references to your application, you can add SfRichTextBoxAdv as any normal control.

You can either drag the SfRichTextBoxAdv control from the toolbox window to the Design view or directly define the control in XAML view.



XML

```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv"
xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf" />
```

C#

```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
```

Using SfRichTextBoxAdv as a standard RichTextBox

This section discusses about how to use the SfRichTextBoxAdv control as a standard RichTextBox control with rich text formatting options.

XML

```
<Window x:Class="Sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf">
<Window.Resources>
<RichTextBoxAdv:UnderlineToggleConverter x:Key="UnderlineToggleConverter"/>
<RichTextBoxAdv:LeftAlignmentToggleConverter
x:Key="LeftAlignmentToggleConverter"/>
<RichTextBoxAdv:CenterAlignmentToggleConverter
x:Key="CenterAlignmentToggleConverter"/>
```

```

<RichTextBoxAdv:RightAlignmentToggleConverter
x:Key="RightAlignmentToggleConverter"/>
<RichTextBoxAdv:JustifyAlignmentToggleConverter
x:Key="JustifyAlignmentToggleConverter"/>
<Style TargetType="Button">
<Setter Property="Background" Value="Transparent" />
<Setter Property="Margin" Value="12 4"/>
</Style>
<Style TargetType="ToggleButton">
<Setter Property="Background" Value="Transparent" />
<Setter Property="Margin" Value="12 4"/>
</Style>
</Window.Resources>
<Grid Background="#F1F1F1">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid>
<!-- Defines the data context as RichTextBoxAdv -->
<StackPanel Orientation="Horizontal" DataContext="{Binding
ElementName=richTextBoxAdv}">
<!-- UI option to perform Undo/Redo using command binding -->
<StackPanel Orientation="Horizontal">
<Button Command="RichTextBoxAdv:SfRichTextBoxAdv.UndoCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" Focusable="False">
<Image Source="/Images/Undo.png" Height="40" Width="40" />
</Button>
<Button Command="RichTextBoxAdv:SfRichTextBoxAdv.RedoCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" Focusable="False">
<Image Source="/Images/Redo.png" Height="40" Width="40" />
</Button>
</StackPanel>
<!-- UI option to perform Clipboard operations using command binding -->
<Border Width="2" Height="46" Background="#1F1F1F"/>
<StackPanel Orientation="Horizontal">
<Button Command="RichTextBoxAdv:SfRichTextBoxAdv.CutCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" Focusable="False">
<Image Source="/Images/Cut.png" Height="40" Width="40" />
</Button>
<Button Command="RichTextBoxAdv:SfRichTextBoxAdv.CopyCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" Focusable="False">
<Image Source="/Images/Copy.png" Height="40" Width="40" />
</Button>
<Button Command="RichTextBoxAdv:SfRichTextBoxAdv.PasteCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" Focusable="False">
<Image Source="/Images/Paste.png" Height="40" Width="40" />
</Button>
</StackPanel>
<!-- UI option to apply character formatting using property binding -->
<Border Width="2" Height="46" Background="#1F1F1F"/>
<StackPanel Orientation="Horizontal">
<ToggleButton IsChecked="{Binding Selection.CharacterFormat.Bold}"
Focusable="False">
<Image Source="/Images/Bold.png" Height="40" Width="40" />
</ToggleButton>

```

```

<ToggleButton IsChecked="{Binding Selection.CharacterFormat.Italic}"
Focusable="False">
<Image Source="/Images/Italic.png" Height="40" Width="40" />
</ToggleButton>
<ToggleButton IsChecked="{Binding Selection.CharacterFormat.Underline,
Converter={StaticResource UnderlineToggleConverter}}" Focusable="False">
<Image Source="/Images/Underline.png" Height="40" Width="40" />
</ToggleButton>
</StackPanel>
<Border Width="2" Height="46" Background="#1F1F1F"/>
<!-- UI option to apply paragraph formatting using property binding -->
<StackPanel Orientation="Horizontal">
<ToggleButton IsChecked="{Binding Selection.ParagraphFormat.TextAlignment,
Converter={StaticResource LeftAlignmentToggleConverter}}" Focusable="False">
<Image Source="/Images/Left.png" Height="40" Width="40" />
</ToggleButton>
<ToggleButton IsChecked="{Binding Selection.ParagraphFormat.TextAlignment,
Converter={StaticResource CenterAlignmentToggleConverter}}"
Focusable="False">
<Image Source="/Images/Center.png" Height="40" Width="40" />
</ToggleButton>
<ToggleButton IsChecked="{Binding Selection.ParagraphFormat.TextAlignment,
Converter={StaticResource RightAlignmentToggleConverter}}"
Focusable="False">
<Image Source="/Images/Right.png" Height="40" Width="40" />
</ToggleButton>
<ToggleButton IsChecked="{Binding Selection.ParagraphFormat.TextAlignment,
Converter={StaticResource JustifyAlignmentToggleConverter}}"
Focusable="False">
<Image Source="/Images/Justify.png" Height="40" Width="40" />
</ToggleButton>
</StackPanel>
</StackPanel>
</Grid>
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv" Grid.Row="1" />
</Grid>
</Window>

```

Note: [View Standard RichTextBox example in GitHub](#)

Creating Document editor with Ribbon

This section discusses about how to create document editor with ribbon similar to Microsoft Word. The SfRichTextRibbon is a Ribbon control customized to work with SfRichTextBoxAdv control, which gives you the look and feel of Microsoft Word.

You can find the SfRichTextRibbon control from the following assembly under the namespace Syncfusion.Windows.Controls.RichTextBoxAdv

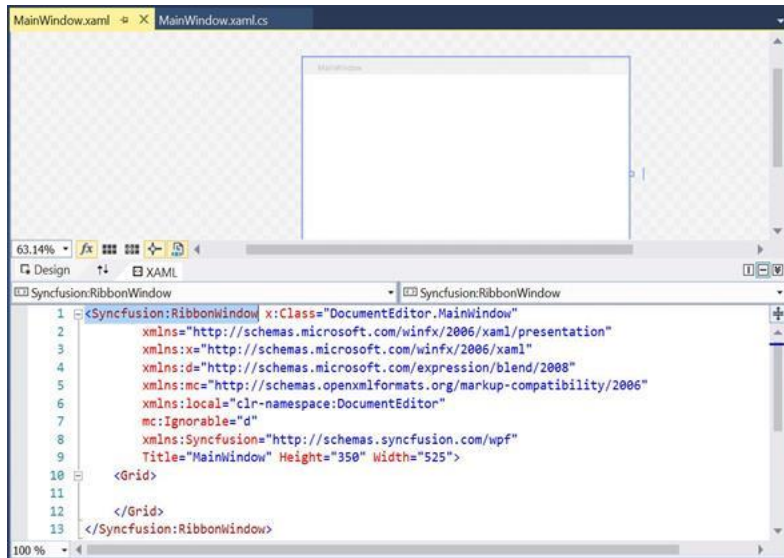
- Syncfusion.SfRichTextRibbon.WPF

The following assembly reference are additionally required to deploy SfRichTextRibbon control along with aforementioned assemblies.

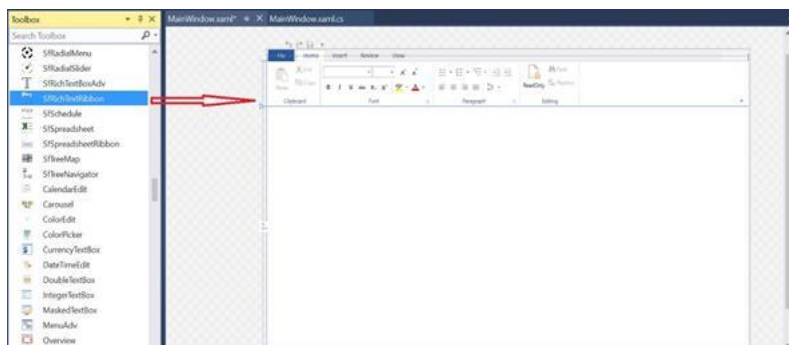
- Syncfusion.Tools.WPF

Adding the Control via Designer

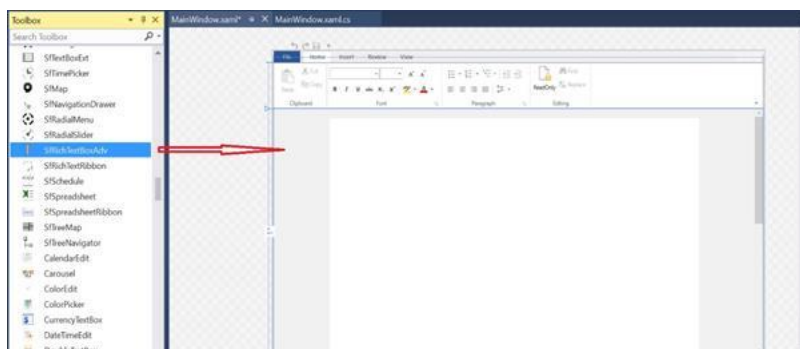
1. Create new WPF application in Visual Studio 2. Use the Syncfusion RibbonWindow instead of Window



3. Drag the SfRichTextRibbon control from the Toolbox window to the Design View. An instance of the SfRichTextRibbon control is created in the Design view



4. Drag the SfRichTextBoxAdv control from the Toolbox window to the Design View. An instance of the SfRichTextBoxAdv control is created in the Design view



5. To make an interaction between SfRichTextRibbon and SfRichTextBoxAdv, bind the SfRichTextBoxAdv as DataContext to the SfRichTextRibbon

XML

```
<Syncfusion:SfRichTextRibbon x:Name="richTextRibbon"
SnapsToDevicePixels="True" DataContext="{Binding
ElementName=richTextBoxAdv}" />
```

Adding Control via code.

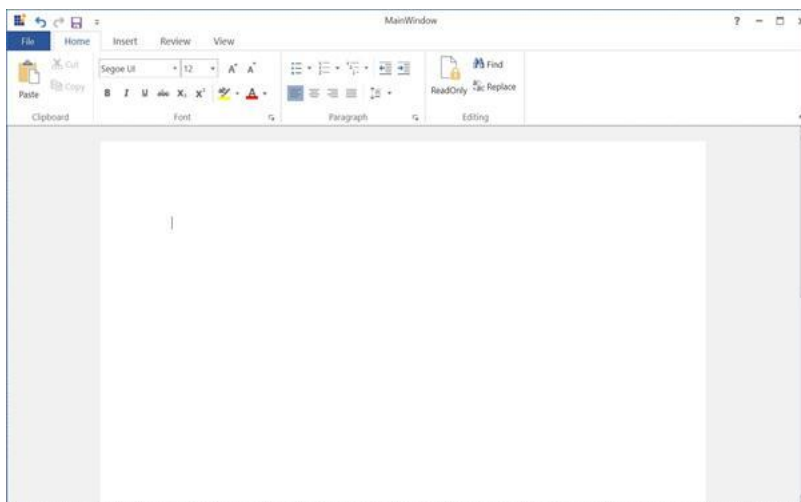
You can also add the SfRichTextRibbon and SfRichTextBoxAdv programmatically by using XAML

XML

```
<Syncfusion:RibbonWindow x:Class="DocumentEditor.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:DocumentEditor"
mc:Ignorable="d"
xmlns:Syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525"
Syncfusion:SkinStorage.VisualStudio="Office2013">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Syncfusion:SfRichTextRibbon x:Name="richTextRibbon"
SnapsToDevicePixels="True" />
<Syncfusion:SfRichTextBoxAdv x:Name="richTextBoxAdv" Background="#F1F1F1"
Grid.Row="1"></Syncfusion:SfRichTextBoxAdv>
</Grid>
</Syncfusion:RibbonWindow>
```

Note: Prefer using SfRichTextRibbon within RibbonWindow in your application, since the backstage of Ribbon will be opened only when the ribbon is loaded under the RibbonWindow

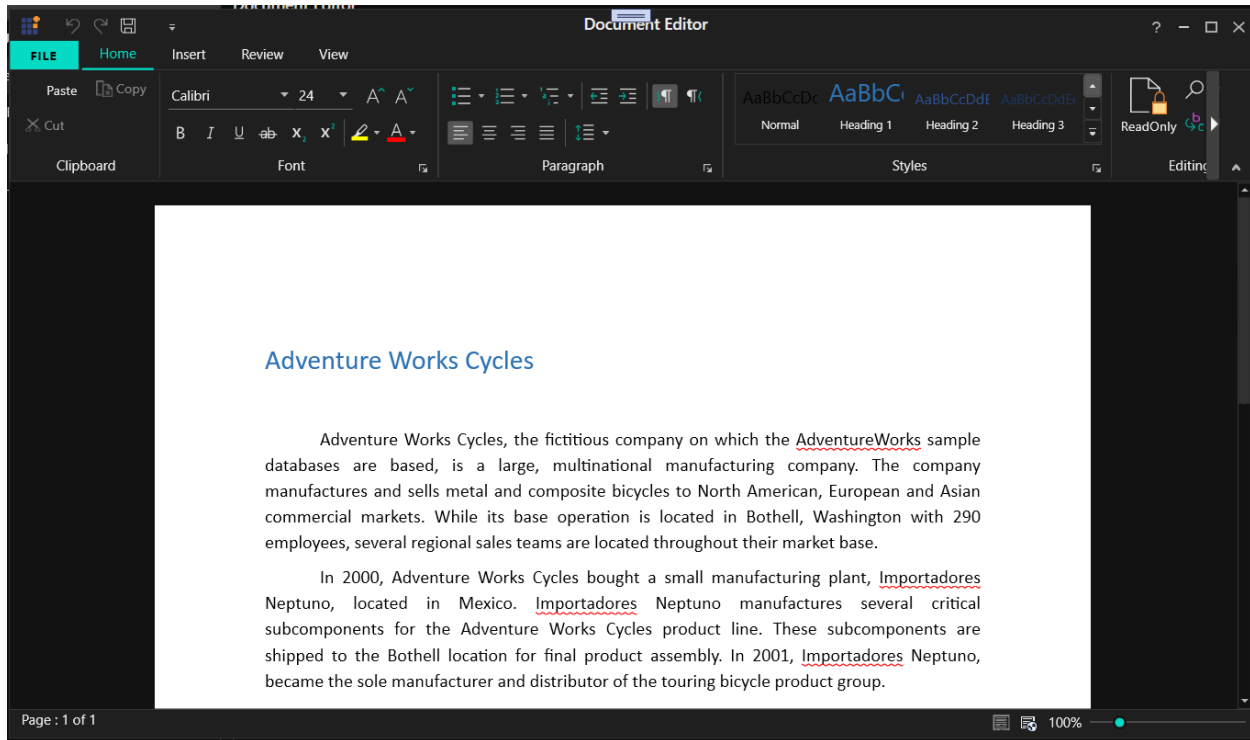
Finally, you can get the following output similar to Microsoft Word on executing the application



Theme

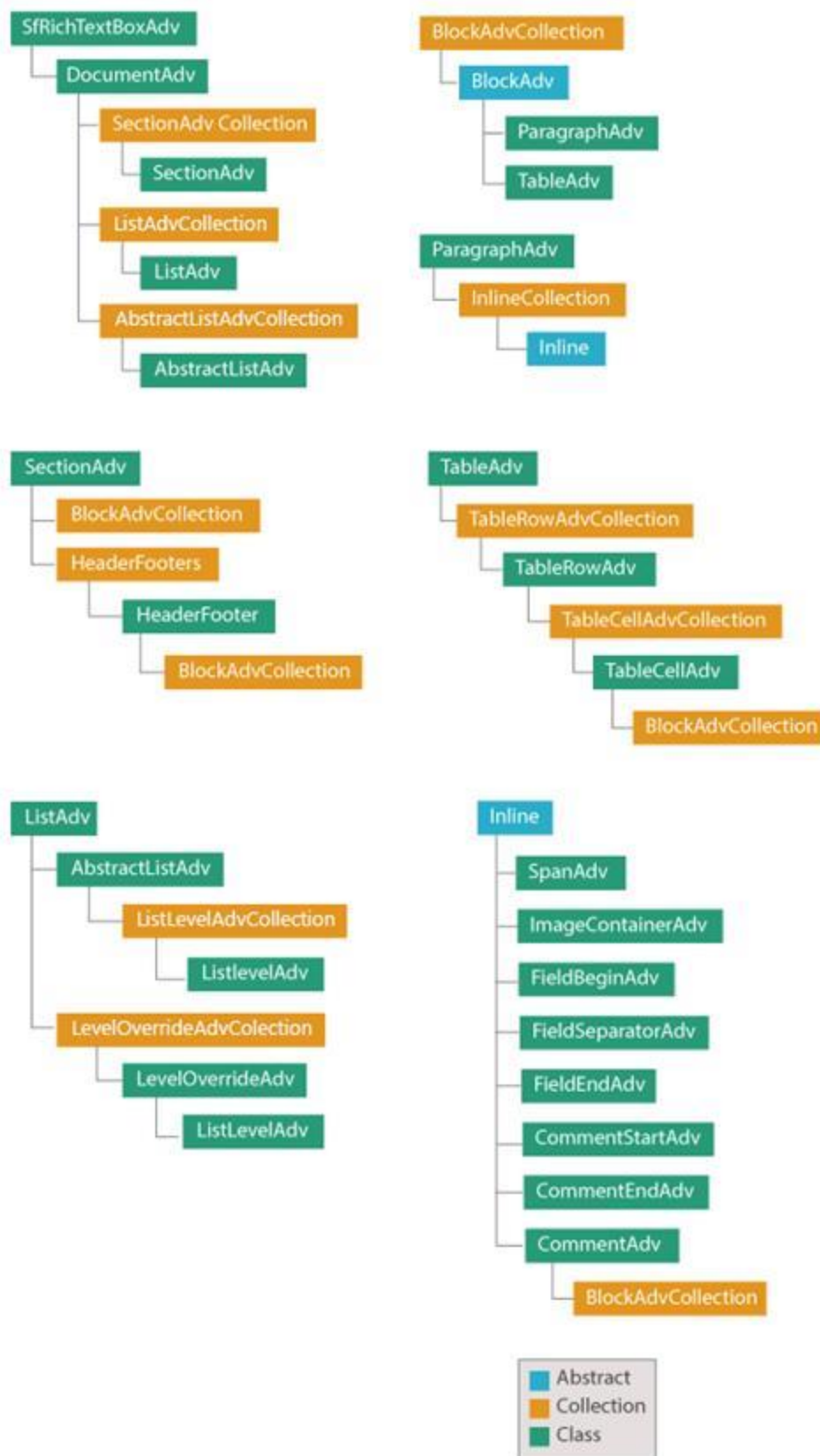
SfRichTextBoxAdv supports various built-in themes. Refer to the below links to apply themes for the SfRichTextBoxAdv,

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



Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Document Structure in WPF RichTextBox (SfRichTextBoxAdv)



Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Document Properties in WPF RichTextBox (SfRichTextBoxAdv)

[WPF RichTextBox](#) keeps tracking the statistics about your documents. These statistics contain information about word count, paragraph count and pages count.

Word Count

RichTextBox automatically counts the number of words in a document while you type. You can get the words count from [WordCount](#) property. The default value of this property is 0.

The following sample code demonstrates how to get the total number of words in the document.

XML

```
<TextBlock Text="{Binding Path=WordCount, Mode=TwoWay}" />
```

C#

```
int wordCount.Text = richTextBoxAdv.WordCount.ToString();
```

VB.NET

```
wordCount.Text = richTextBoxAdv.WordCount.ToString()
```

Paragraph Count

RichTextBox automatically counts the number of paragraphs in a document while you type. You can get the paragraph count from [ParagraphCount](#) property. The default value of this property is 0. Also, it ignores empty paragraphs.

The following sample code demonstrates how to get the total number of paragraphs in the document.

XML

```
<TextBlock Name="ParagraphCount" Text="{Binding Path=ParagraphCount, Mode=TwoWay}" />
```

C#

```
int paragraphCount.Text = richTextBoxAdv.ParagraphCount.ToString();
```

VB.NET

```
paragraphCount.Text = richTextBoxAdv.ParagraphCount.ToString()
```

Page Count

RichTextBox counts the number of pages in a document while you type. You can get the pages count from [PageCount](#) property. The default value of this property is 0.

The following sample code demonstrates how to get the total number of pages in the document.

XML

```
<TextBlock x:Name="PageCount" Grid.Row="0" />
<RichTextBoxAdv:SfRichTextBoxAdv Grid.Row="1" x:Name="richTextBoxAdv"
SelectionChanged="RichTextBoxAdv_SelectionChanged"/>
```

C#

```
private void RichTextBoxAdv_SelectionChanged(object obj,
SelectionChangedEventArgs args)
{
    pageCount.Text = richTextBoxAdv.PageCount.ToString();
}
```

VB.NET

```
Private Sub RichTextBoxAdv_SelectionChanged(ByVal obj As Object, ByVal args
As SelectionChangedEventArgs)
    pageCount.Text = richTextBoxAdv.PageCount.ToString()
End Sub
```

Current Page number

The [CurrentPageNumber](#) property in the RichTextBox control returns the page number where the selection(cursor) is present.

The following sample code demonstrates how to get current page number in the document.

XML

```
<TextBlock x:Name="CurrentPageNumber" Grid.Row="0" />
<RichTextBoxAdv:SfRichTextBoxAdv Grid.Row="1" x:Name="richTextBoxAdv"
SelectionChanged="RichTextBoxAdv_SelectionChanged"/>
```

C#

```
private void RichTextBoxAdv_SelectionChanged(object obj,
SelectionChangedEventArgs args)
{
    currentPageNumber.Text = richTextBoxAdv.CurrentPageNumber.ToString();
}
```

VB.NET

```
Private Sub RichTextBoxAdv_SelectionChanged(ByVal obj As Object, ByVal args
As SelectionChangedEventArgs)
    currentPageNumber.Text = richTextBoxAdv.CurrentPageNumber.ToString()
End Sub
```

Note: The above PageCount and CurrentPageNumber properties are not a dependency property. And it is not notifying for dynamic changes. So, get these properties value in

selection changed event.

You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Layout Types in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv control allows you to choose between the following layout types.

- Pages
- Continuous
- Block

Pages

When using Pages layout type, the rich text document content is rendered sequentially in several pages, similar to the Print layout view of Microsoft Word. The size and margin of each page are defined by Section format properties.

The following code example demonstrates how to define layout type of SfRichTextBoxAdv control as Pages.

XML

```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv" LayoutType="Pages"
xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf"/>
```

C#

```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
// Defines the layout type as Pages
richTextBoxAdv.LayoutType = LayoutType.Pages;
```

VB.NET

```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
' Defines the layout type as Pages
richTextBoxAdv.LayoutType = LayoutType.Pages
```



Continuous

When using Continuous layout type, the entire rich text document content is rendered continuously in a single page, similar to the Web layout view of Microsoft Word. This layout looks like a simple text box with rich-text content and can be used for applications such as forums and blogs.

The following code example demonstrates how to define layout type of SfRichTextBoxAdv control as Continuous.

XML

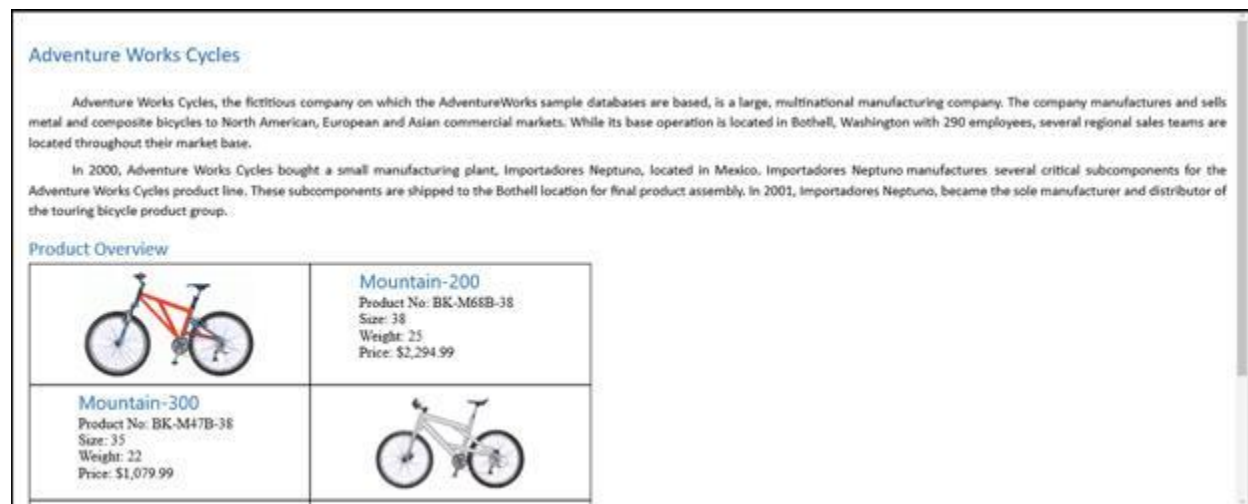
```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv"
LayoutType="Continuous" xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf"/>
```

C#

```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
// Defines the layout type as Continuous.
richTextBoxAdv.LayoutType = LayoutType.Continuous;
```

VB.NET

```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
' Defines the layout type as Continuous.
richTextBoxAdv.LayoutType = LayoutType.Continuous
```



Block

When using Block layout type, the rich text content is rendered continuously in a single page as read only. This layout looks like a simple text block with rich text content such as texts, images, and tables. Block Layout also supports copying contents to the clipboard. This can be used for applications such as forums and blogs in order to display the rich-text contents with same look and feel as in the continuous layout type.

The following code example demonstrates how to define layout type of SfRichTextBoxAdv control as Block.

XML

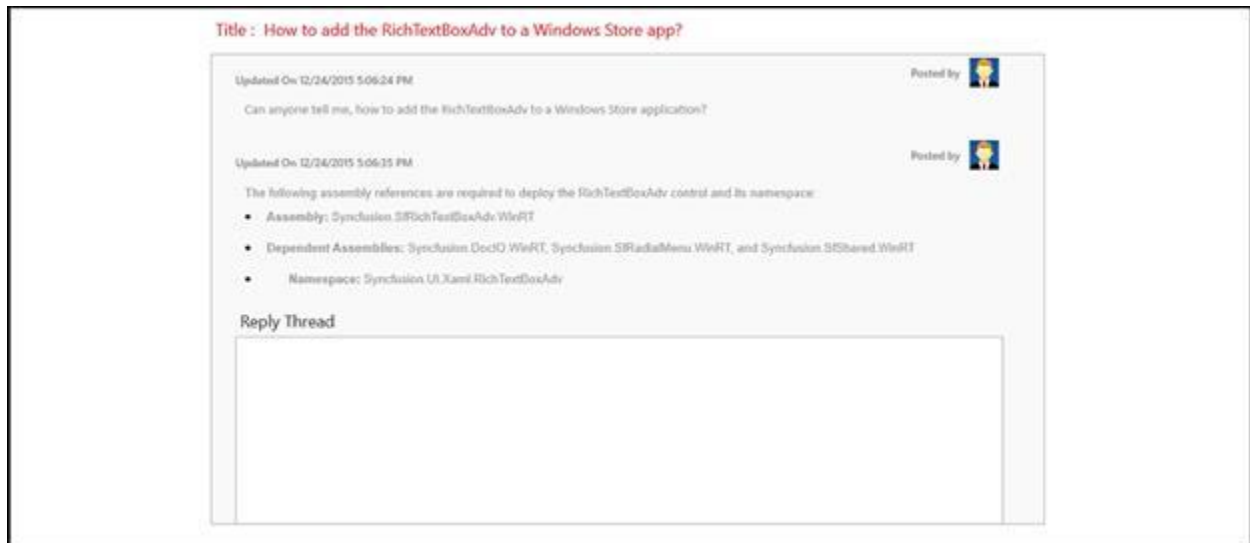
```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv" LayoutType="Block"
xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf"/>
```

C#

```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
// Defines the layout type as Block.
richTextBoxAdv.LayoutType = LayoutType.Block;
```

VB.NET

```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
' Defines the layout type as Block.
richTextBoxAdv.LayoutType = LayoutType.Block
```



Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Commands in WPF RichTextBox (SfRichTextBoxAdv)

Commands are a way to handle user interface (UI) actions. They are a loosely coupled way to bind the UI to the logic that performs the action. The SfRichTextBoxAdv supports commands for mostly used operations which are classified below.

- Character Formatting – Bold, Italic, Underline, Strike through, Baseline alignment, Font family, Font size, Font color and Highlight color.

- Paragraph Formatting – Left indent, Right indent, First line indent, Text alignment, Before spacing, After spacing, Line spacing, Line spacing type, Increase indent, Decrease indent and Change list type.
- Clipboard – Cut, Copy and Paste.
- History – Undo and Redo.
- Import and Export – Open document, Save document and New document.
- Comments – New comment, Delete comment, Delete all comments, Previous comment, Next comment and Show comments.
- Table – Insert table, Insert row, Insert column, Delete table, Delete row, Delete column and Merge selected cells, Select cell, Select column, Select row, Select table and Align cell content.
- UI options – Show hyperlink dialog, Show options pane and Layout type.
- Insert – Insert picture and Insert hyperlink.
- Document Styles – Create, modify, apply and clear style.

UI Command to access character formatting

The following code example demonstrates how to bind commands for applying character format.

XML

```
<!-- Binds button to the BoldCommand -->
<Button Content="Bold" Command="RichTextBoxAdv:SfRichTextBoxAdv.BoldCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ItalicCommand -->
<Button Content="Italic"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ItalicCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
```

The following code example demonstrates how to bind commands with parameter.

XML

```
<Button Content="Textalignment"
Command="Syncfusion:SfRichTextBoxAdv.TextAlignmentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}"
CommandParameter="Right" />
```

List of available Commands

The following table contains the list of available UI Commands in SfRichTextBoxAdv.

Command Name	Description	Command parameter (optional)
AddToDictionaryCommand	Represents the command to add the selected or specified custom word into the custom dictionary.	The custom word to be added.
AfterSpacingCommand	Represents the command to apply after spacing for the selected paragraphs.	The after spacing value to be applied.

Command Name	Description	Command parameter (optional)
AutoFitTableCommand	Represents the command that fits the columns width of selected table based on the specified AutoFitType .	The AutoFitType to fit the columns of the table.
BackSpaceKeyCommand	Represents the command that performs action for "Backspace" key.	NA
BaselineAlignmentCommand	Represents the command to apply BaselineAlignment for the selected text.	The baseline alignment value to be applied.
BeforeSpacingCommand	Represents the command to apply before spacing for the selected paragraphs.	The before spacing value to be applied.
BoldCommand	Represents the command to toggle bold style for the selected text.	NA
CellBottomMarginCommand	Represents the command to apply bottom margin for the selected table cells.	The bottom margin value to be applied.
CellContentAlignmentCommand	Represents the command to apply content alignment for the selected table cells.	The cell content alignment value to be applied.
CellLeftMarginCommand	Represents the command to apply left margin for the selected table cells.	The left margin value to be applied.
CellRightMarginCommand	Represents the command to apply right margin for the selected table cells.	The right margin value to be applied.
CellSpacingCommand	Represents the command to apply cell spacing for the selected table.	The cell spacing value to be applied.
CellTopMarginCommand	Represents the command to apply top margin for the selected table cells.	The top margin value to be applied.
CellVerticalAlignmentCommand	Represents the command to apply vertical	The vertical alignment value to be applied.

Command Name	Description	Command parameter (optional)
	alignment for the selected table cells.	
ChangeAllSpellingCommand	Represents the command that changes all occurrences of a selected misspelled word.	The word to be replaced.
ChangeSpellingCommand	Represents the command that changes the selected misspelled word.	The word to be replaced.
CheckSpellingCommand	Represents the command that checks spelling mistakes in the document of SfRichTextBoxAdv control.	NA
ControlDownKeyCommand	Represents the command that performs navigation for "CTRL + DOWN" key.	NA
ControlEndKeyCommand	Represents the command that performs navigation for "CTRL + END" key.	NA
ControlHomeKeyCommand	Represents the command that performs navigation for "CTRL + HOME" key.	NA
ControlLeftKeyCommand	Represents the command that performs navigation for "CTRL + LEFT" key.	NA
ControlRightKeyCommand	Represents the command that performs navigation for "CTRL + RIGHT" key.	NA
ControlShiftDownKeyCommand	Represents the command that performs selection for "CTRL + SHIFT + DOWN" key.	NA
ControlShiftEndKeyCommand	Represents the command that performs selection for "CTRL + SHIFT + END" key.	NA
ControlShiftHomeKeyCommand	Represents the command that performs selection for	NA

Command Name	Description	Command parameter (optional)
	"CTRL + SHIFT + HOME" key.	
ControlShiftLeftKeyCommand	Represents the command that performs selection for "CTRL + SHIFT + LEFT" key.	NA
ControlShiftRightKeyCommand	Represents the command that performs selection for "CTRL + SHIFT + RIGHT" key.	NA
ControlShiftUpKeyCommand	Represents the command that performs selection for "CTRL + SHIFT + UP" key.	NA
ControlUpKeyCommand	Represents the command that performs navigation for "CTRL + UP" key.	NA
CopyCommand	Represents the command that copies the selected contents to clipboard.	NA
CopyHyperlinkCommand	Represents the command that copies the selected hyperlink to clipboard.	NA
CutCommand	Represents the command that removes the selected contents from SfRichTextBoxAdv control and copies the same to clipboard.	NA
DecreaseFontSizeCommand	Represents the command to decrease font size for the selected text.	The font size value to be applied.
DecreaseIndentCommand	Represents the command to decrease left indent for the selected paragraphs.	NA
DefaultCellBottomMarginCommand	Represents the command to apply bottom margin for the selected table.	The cell bottom margin value to be applied.
DefaultCellLeftMarginCommand	Represents the command to apply left margin for the selected table.	The cell left margin value to be applied.

Command Name	Description	Command parameter (optional)
DefaultCellRightMarginCommand	Represents the command to apply right margin for the selected table.	The cell right margin value to be applied.
DefaultCellTopMarginCommand	Represents the command to apply top margin for the selected table.	The cell top margin value to be applied.
DeleteAllCommentsCommand	Represents the command to delete all the comments in the document.	NA
DeleteColumnCommand	Represents the command to delete the selected column of a table.	NA
DeleteCommentCommand	Represents the command that deletes the selected comment.	NA
DeleteKeyCommand	Represents the command that performs action for "DELETE" key.	NA
DeleteRowCommand	Represents the command to delete the selected row of a table.	NA
DeleteTableCommand	Represents the command to delete the selected table.	NA
DownKeyCommand	Represents the command that performs navigation for "DOWN" key.	NA
EndKeyCommand	Represents the command that performs navigation for "END" key.	NA
EnterKeyCommand	Represents the command that performs action for "ENTER" key.	NA
FirstLineIndentCommand	Represents the command to apply first line indent for the selected paragraphs.	The first line indent value to be applied.
FontColorCommand	Represents the command to apply font color for the selected text.	The font color value to be applied.

Command Name	Description	Command parameter (optional)
FontFamilyCommand	Represents the command to apply font family for the selected text.	The font family value to be applied.
FontSizeCommand	Represents the command to apply font size for the selected text.	The font size value to be applied.
HighlightColorCommand	Represents the command to apply HighlightColor for the selected text.	The highlight color value to be applied.
HomeKeyCommand	Represents the command that performs navigation for "HOME" key.	NA
IgnoreAllSpellingErrorsCommand	Represents the command that ignores all the occurrence of a selected misspelled word.	The misspelled word to be ignored.
IncreaseFontSizeCommand	Represents the command to increase font size for the selected text.	The font size value to be applied.
IncreaseIndentCommand	Represents the command to increase indent for the selected paragraphs.	NA
InsertBreakCommand	Represents the command that inserts a break at selection.	The break type to be inserted
InsertColumnCommand	Represents the command that inserts a column to the selected table.	The ColumnPlacement value to insert column.
InsertHyperlinkCommand	Represents the command that inserts a hyperlink at selection.	The hyperlink to be inserted.
InsertPictureCommand	Represents the command that inserts a picture at selection.	The picture to be inserted.
InsertRowCommand	Represents the command that inserts a row to the selected table.	The RowPlacement value to insert row.
InsertTableCommand	Represents the command that inserts a table at selection.	The row and column count of the table.

Command Name	Description	Command parameter (optional)
ItalicCommand	Represents the command to toggle italic style for the selected text.	NA
LayoutTypeCommand	Represents the command to change layout type of the SfRichTextBoxAdv control.	The layout type to set for the SfRichTextBoxAdv control.
LeftIndentCommand	Represents the command to apply left indent for the selected paragraphs.	The left indent value to be applied.
LeftKeyCommand	Represents the command that performs navigation for "LEFT" key.	NA
LineSpacingCommand	Represents the command to apply line spacing for the selected paragraphs.	The line spacing value to be applied.
LineSpacingTypeCommand	Represents the command to apply LineSpacingType for the selected paragraphs.	The LineSpacingType value to be applied.
MergeSelectedCellsCommand	Represents the command that merges the selected table cells.	NA
NavigateHyperlinkCommand	Represents the command that performs navigation for the selected hyperlink.	NA
NewCommentCommand	Represents the command that adds a comment at the selection.	NA
NewDocumentCommand	Represents the command that creates a new document in SfRichTextBoxAdv control.	NA
NextCommentCommand	Represents the command that performs navigation to next comment in the document.	NA
OpenDocumentCommand	Represents the command that opens an existing document in the	NA

Command Name	Description	Command parameter (optional)
	SfRichTextBoxAdv control.	
PageDownKeyCommand	Represents the command that performs navigation for "PAGEDOWN" key.	NA
PageFitCommand	Represents the command to apply PageFitType for the SfRichTextBoxAdv control.	The PageFitType value to be applied.
PageUpKeyCommand	Represents the command that performs navigation for "PAGEUP" key.	NA
PasteCommand	Represents the command that pastes the content from clipboard to SfRichTextBoxAdv control.	NA
PreviousCommentCommand	Represents the command that performs navigation to previous comment in the document.	NA
PrintDocumentCommand	Represents the command that prints the document of SfRichTextBoxAdv control.	NA
RedoCommand	Represents the command that redo the last undo operation in SfRichTextBoxAdv control.	NA
RemoveHyperlinkCommand	Represents the command that removes the selected hyperlink.	NA
RightIndentCommand	Represents the command to apply right indent for the selected paragraphs.	The right indent value to be applied.
RightKeyCommand	Represents the command that performs navigation for "RIGHT" key.	NA
SaveAsDocumentCommand	Represents the command that saves the document	NA

Command Name	Description	Command parameter (optional)
	of SfRichTextBoxAdv control.	
SaveDocumentCommand	Represents the command that saves the document of SfRichTextBoxAdv control.	NA
SelectAllCommand	Represents the command that selects all the content of SfRichTextBoxAdv control.	NA
SelectCellCommand	Represents the command that selects the table cell.	NA
SelectColumnCommand	Represents the command that selects the entire column of a table.	NA
SelectRowCommand	Represents the command that selects the entire row of a table.	NA
SelectTableCommand	Represents the command that selects the table.	NA
ShiftDownKeyCommand	Represents the command that performs selection for "SHIFT + DOWN" key.	NA
ShiftEndKeyCommand	Represents the command that performs selection for "SHIFT + END" key.	NA
ShiftEnterKeyCommand	Represents the command that performs action [inserts line break] for "SHIFT + ENTER" key.	NA
ShiftHomeKeyCommand	Represents the command that performs selection for "SHIFT + HOME" key.	NA
ShiftLeftKeyCommand	Represents the command that performs selection for "SHIFT + LEFT" key.	NA
ShiftRightKeyCommand	Represents the command that performs selection for "SHIFT + RIGHT" key.	NA
ShiftTabKeyCommand	Represents the command that performs selection for "SHIFT + TAB" key.	NA

Command Name	Description	Command parameter (optional)
ShiftUpKeyCommand	Represents the command that performs selection for "SHIFT + UP" key.	NA
ShowBordersAndShadingDialogCommand	Represents the command that shows the borders and shading dialog.	NA
ShowCellOptionsDialogCommand	Represents the command that shows the cell options dialog.	NA
ShowCommentsCommand	Represents the command to show or hide comments in the SfRichTextBoxAdv control.	NA
ShowEncryptDocumentDialogCommand	Represents the command that shows dialog to specify password for opening encrypted Word document.	NA
ShowFindAndReplaceDialogCommand	Represents the command that shows the find and replace dialog.	The Boolean value to show find or replace tab.
ShowFontDialogCommand	Represents the command that shows the font dialog.	NA
ShowHyperlinkDialogCommand	Represents the command that shows the hyperlink dialog.	NA
ShowInsertTableDialogCommand	Represents the command that shows the insert table dialog.	NA
ShowListDialogCommand	Represents the command that shows the list dialog.	NA
ShowOptionsPaneCommand	Represents the command that shows the options pane.	NA
ShowParagraphDialogCommand	Represents the command that shows the paragraph dialog.	NA
ShowSpellingPaneCommand	Represents the command that shows the spelling pane.	NA

Command Name	Description	Command parameter (optional)
ShowTableDialogCommand	Represents the command that shows the table dialog.	NA
ShowTableOptionsDialogCommand	Represents the command that shows the table options dialog.	NA
SpaceKeyCommand	Represents the command that performs action [inserts space] for "SPACE" key.	NA
StrikeThroughCommand	Represents the command to apply StrikeThrough for the selected text.	The strike through value to be applied.
TabKeyCommand	Represents the command that performs action [inserts tab] for "TAB" key.	NA
TableAlignmentCommand	Represents the command to apply table alignment for the selected table.	The table alignment value to be applied.
TableLeftIndentCommand	Represents the command to apply left indent for the selected table.	The left indent value to be applied.
TextAlignmentCommand	Represents the command to apply text alignment for the selected paragraphs.	The TextAlignment value to be applied.
ToggleBaselineAlignmentCommand	Represents the command that toggles baseline alignment for the selected text.	The baseline alignment value to be applied.
ToggleBeforeSpacingCommand	Represents the command that toggles before spacing for the selected paragraphs.	NA
ToggleHighlightColorCommand	Represents the command that toggles highlight color for the selected text.	NA
ToggleUnderlineCommand	Represents the command that toggles underline for the selected text.	The Underline value to be applied.

Command Name	Description	Command parameter (optional)
UnderlineCommand	Represents the command to apply underline for the selected text.	The Underline value to be applied.
UndoCommand	Represents the command that undo the last edit operation in SfRichTextBoxAdv control.	NA
UpKeyCommand	Represents the command that performs navigation for "UP" key.	NA
ShowStyleDialogCommand	Represents the command that shows the create style dialog.	NA
ShowStylesDialogCommand	Represents the command that shows the modify style dialog.	NA
ApplyStyleCommand	Represents the command, which requests to apply style for the selected paragraph.	Name of the Style
ClearFormattingCommand	Represents the command, which requests to remove formatting from the selected paragraph.	NA

Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Selection in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv supports selecting a portion of the document either through UI interactions by using mouse, touch, keyboard or through supported APIs.

The following sample code demonstrates how to retrieve text position from document using paragraph instance and offset value.

C#

```
// Gets the first section of the document.
SectionAdv sectionAdv = richTextBoxAdv.Document.Sections[0];
```

VB.NET

```
'Gets the first section of the document
Dim sectionAdv As SectionAdv = richTextBoxAdv.Document.Sections(0)
```

C#

```
// Gets the first block from the section, which is a paragraph.
ParagraphAdv paragraphAdv = sectionAdv.Blocks[0] as ParagraphAdv;
// Gets the text position of the specified paragraph at offset 24.
// TextPosition is returned as null, if no such paragraph or offset exists
in the document.
TextPosition startPosition =
richTextBoxAdv.Document.GetTextPosition(paragraphAdv, 24);
```

VB.NET

```
' Gets the first block from the section, which is a paragraph.
Dim paragraphAdv As ParagraphAdv = TryCast(sectionAdv.Blocks(0),
ParagraphAdv)
' Gets the text position of the specified paragraph at offset 24.
' TextPosition is returned as null, if no such paragraph or offset exists in
the document.
Dim startPosition As TextPosition =
richTextBoxAdv.Document.GetTextPosition(paragraphAdv, 24)
```

C#

```
// Gets the third block of a section, which is table.
TableAdv tableAdv = sectionAdv.Blocks[2] as TableAdv;
// Gets the third block from second row second cell of the table, which is
paragraph.
ParagraphAdv paragraphAdv = tableAdv.Rows[1].Cells[1].Blocks[2] as
ParagraphAdv;
TextPosition position =
richTextBoxAdv.Document.GetTextPosition(paragraphAdv, 12);
```

VB.NET

```
' Gets the third block of a section, which is table.
Dim tableAdv As TableAdv = TryCast(sectionAdv.Blocks(2), TableAdv)
' Gets the third block from second row second cell of the table, which is
paragraph.
Dim paragraphAdv As ParagraphAdv =
TryCast(tableAdv.Rows(1).Cells(1).Blocks(2), ParagraphAdv)
Dim position As TextPosition =
richTextBoxAdv.Document.GetTextPosition(paragraphAdv, 12)
```

The following sample code demonstrates how to retrieve text position from document using hierarchical index.

C#

```
/*
The hierarchical index should be given as "section-index;block-index;offset-
in-paragraph"
If block in the index is paragraph, then next value is considered as offset
and position is retrieved.
```

```

For example "0;0;1" gets text position of Paragraph (first block of first
section) at the offset=1.
If block in the index is table, then next value is considered as row index
and following value as cell index.
The value after cell is again a block index. Then the same process
continues.
"table-index;row-index;cell-index;block-index;"
For example "0;2;1;1;0;21" gets text position of Paragraph at first block of
second cell of second row of table (at third block of first section) at the
offset=21.
If offset value is followed by "C" which stands for comment, then the
comment is retrieved which is followed by block index in comment. Then the
process of retrieving paragraph from block index continues.
paragraph-index;offset-in-paragraph;C;block-index;offset-in-paragraph
For example "0;3;16;C;2;6", gets text position of Paragraph at third block
of comment (present at offset = 16 in paragraph at third block of first
section) at the offset=6.
*/
// Gets the text position from the document based on hierarchical index.
TextPosition position = richTextBoxAdv.Document.GetTextPosition("0;0;24");
/* Here text position returned should be first section's first block (which
is paragraph) and offset=24. */

```

VB.NET

```

'
'The hierarchical index should be given as "section-index;block-
index;offset-in-paragraph"
'If block in the index is paragraph, then next value is considered as offset
and position is retrieved.
'For example "0;0;1" gets text position of Paragraph (first block of first
section) at the offset=1.
'If block in the index is table, then next value is considered as row index
and following value as cell index.
'The value after cell is again a block index. Then the same process
continues.
'"table-index;row-index;cell-index;block-index;"
'For example "0;2;1;1;0;21" gets text position of Paragraph at first block of
second cell of second row of table (at third block of first section) at
the offset=21.
'If offset value is followed by "C" which stands for comment, then the
comment is retrieved which is followed by block index in comment. Then the
process of retrieving paragraph from block index continues.
'paragraph-index;offset-in-paragraph;C;block-index;offset-in-paragraph
'For example "0;3;16;C;2;6", gets text position of Paragraph at third block
of comment (present at offset = 16 in paragraph at third block of first
section) at the offset=6.
'
' Gets the text position from the document based on hierarchical index.
Dim position As TextPosition =
richTextBoxAdv.Document.GetTextPosition("0;0;24")
' Here text position returned should be first section's first block (which
is paragraph) and offset=24.

```

The following sample code demonstrates how to move selection start and selection end both at a specific text position.

C#

```
// Makes an empty selection at the specific text position.  
richTextBoxAdv.Selection.Select(position, position);
```

VB.NET

```
' Makes an empty selection at the specific text position.  
richTextBoxAdv.Selection.[Select](position, position)
```

The following sample code demonstrates how to select a portion of document.

C#

```
// Retrieves the position of the first paragraph start.  
TextPosition startPosition =  
richTextBoxAdv.Document.GetTextPosition("0;0;0");  
// Retrieves the position of the first paragraph at offset=20.  
TextPosition endPosition =  
richTextBoxAdv.Document.GetTextPosition("0;0;20");  
// Selects the text positions in forward direction.  
richTextBoxAdv.Selection.Select(startPosition, endPosition);
```

VB.NET

```
' Retrieves the position of the first paragraph start.  
Dim startPosition As TextPosition =  
richTextBoxAdv.Document.GetTextPosition("0;0;0")  
' Retrieves the position of the first paragraph at offset=20.  
Dim endPosition As TextPosition =  
richTextBoxAdv.Document.GetTextPosition("0;0;20")  
' Selects the text positions in forward direction.  
richTextBoxAdv.Selection.[Select](startPosition, endPosition)
```

C#

```
// Selects the text positions in reverse direction.  
richTextBoxAdv.Selection.Select(endPosition, startPosition);
```

VB.NET

```
' Selects the text positions in reverse direction.  
richTextBoxAdv.Selection.[Select](endPosition, startPosition)
```

Note: You can select a text position within a comment with another text position within the same comment only. It is not possible to select a text position within comment with a text position that exists outside of that comment.

Multi Selection

The SfRichTextBoxAdv also supports selecting different portions of the document at a time. The following code example demonstrates how to perform multi selection in SfRichTextBoxAdv.

C#

```
// Retrieves the position of the first paragraph start.
TextPosition startPosition1 =
richTextBoxAdv.Document.GetTextPosition("0;0;0");
// Retrieves the position of the first paragraph at offset=20.
TextPosition endPosition1 =
richTextBoxAdv.Document.GetTextPosition("0;0;20");
// Retrieves the position of the third paragraph start.
TextPosition startPosition2 =
richTextBoxAdv.Document.GetTextPosition("0;2;0");
// Retrieves the position of the third paragraph at offset=20.
TextPosition endPosition2 =
richTextBoxAdv.Document.GetTextPosition("0;2;20");
// Selects the first paragraph and third paragraph at a time, leaving second
paragraph.
richTextBoxAdv.Selection.SelectionRanges.Add(startPosition1, endPosition1);
richTextBoxAdv.Selection.SelectionRanges.Add(startPosition2, endPosition2);
```

VB.NET

```
' Retrieves the position of the first paragraph start.
Dim startPosition1 As TextPosition =
richTextBoxAdv.Document.GetTextPosition("0;0;0")
' Retrieves the position of the first paragraph at offset=20.
Dim endPosition1 As TextPosition =
richTextBoxAdv.Document.GetTextPosition("0;0;20")
' Retrieves the position of the third paragraph start.
Dim startPosition2 As TextPosition =
richTextBoxAdv.Document.GetTextPosition("0;2;0")
' Retrieves the position of the third paragraph at offset=20.
Dim endPosition2 As TextPosition =
richTextBoxAdv.Document.GetTextPosition("0;2;20")
' Selects the first paragraph and third paragraph at a time, leaving second
paragraph.
richTextBoxAdv.Selection.SelectionRanges.Add(startPosition1, endPosition1)
richTextBoxAdv.Selection.SelectionRanges.Add(startPosition2, endPosition2)
```

Apply Formatting for selection

The SfRichTextBoxAdv supports the following format properties that can be applied for selection contents.

Character Format-bold, italic, font size, font family, font color, highlight color, underline, strikethrough, subscript, and superscript.

Paragraph Format-before and after spacing, first line, left and right indenting, text justification, line spacing, and multilevel list.

Selection Format-page size and page margin.

The following sample code demonstrates how to apply subscript format for the selected content.

C#

```
// Applies subscript format for the selected text contents.  
richTextBoxAdv.Selection.CharacterFormat.BaselineAlignment =  
Syncfusion.Windows.Controls.RichTextBoxAdv.BaselineAlignment.Subscript;
```

VB.NET

```
' Applies subscript format for the selected text contents.  
richTextBoxAdv.Selection.CharacterFormat.BaselineAlignment =  
Syncfusion.Windows.Controls.RichTextBoxAdv.BaselineAlignment.Subscript
```

The following sample code demonstrates how to apply after spacing for the selected paragraphs.

C#

```
// Applies after spacing for the selected paragraphs.  
richTextBoxAdv.Selection.ParagraphFormat.AfterSpacing = 24;
```

VB.NET

```
' Applies after spacing for the selected paragraphs.  
richTextBoxAdv.Selection.ParagraphFormat.AfterSpacing = 24
```

The following sample code demonstrates how to apply page margin for the selected sections.

C#

```
// Applies page margin for the selected sections.  
richTextBoxAdv.Selection.SectionFormat.PageMargin = new Thickness(96, 48,  
96, 48);
```

VB.NET

```
' Applies page margin for the selected sections.  
richTextBoxAdv.Selection.SectionFormat.PageMargin = New Thickness(96, 48,  
96, 48)
```

Binding Selection format properties

The SfRichTextBoxAdv provides support to bind the rich-text format options of selection content.

The following code sample demonstrates how to bind the bold format option of SfRichTextBoxAdv.

XML

```
<!-- Binds the toggle button to Selection bold character format -->  
<ToggleButton x:Name="toggleButton" Content="Bold" IsChecked="{Binding  
Path=Selection.CharacterFormat.Bold, Mode=TwoWay,  
ElementName=richTextBoxAdv}" />
```

C#

```
// Initializes the new binding for toggle bold.
```

```
Binding binding = new Binding() { Source = richTextBoxAdv, Path = new
PropertyPath("Selection.CharacterFormat.Bold"), Mode = BindingMode.TwoWay };
// Binds the IsChecked property to Selection.CharacterFormat.Bold property
of RichTextBoxAdv.
toggleButton.SetBinding(ToggleButton.IsCheckedProperty, binding);
```

VB.NET

```
' Initializes the new binding for toggle bold.
Dim binding As New Binding() With { _
Key .Source = richTextBoxAdv, _
Key .Path = New PropertyPath("Selection.CharacterFormat.Bold"), _
Key .Mode = BindingMode.TwoWay _
}
' Binds the IsChecked property to Selection.CharacterFormat.Bold property of
RichTextBoxAdv.
toggleButton.SetBinding(ToggleButton.IsCheckedProperty, binding)
```

The following code sample demonstrates how to bind the bold format option of SfRichTextBoxAdv.

XML

```
<!--Binds IsChecked property of toggle button to Selection text alignment
paragraph format -->
<ToggleButton x:Name="toggleButton" Content="Left" IsChecked="{Binding
ElementName=richTextBoxAdv,Path=Selection.ParagraphFormat.TextAlignment,Conv
erter={StaticResource LeftAlignmentToggleConverter}}"/>
```

C#

```
//Initializes the new binding for toggle text alignment property as left.
Binding binding = new Binding() { Source = richTextBoxAdv, Path = new
PropertyPath("Selection.ParagraphFormat.TextAlignment"), Mode =
BindingMode.TwoWay, Converter = new LeftAlignmentToggleConverter() };
//Binds the IsChecked property to Selection.ParagraphFormat.TextAlignment
property of RichTextBoxAdv.
toggleButton.SetBinding(ToggleButton.IsCheckedProperty, binding);
```

VB.NET

```
'Initializes the new binding for toggle text alignment property as left.
Dim binding As New Binding() With { _
Key .Source = richTextBoxAdv, _
Key .Path = New PropertyPath("Selection.ParagraphFormat.TextAlignment"), _
Key .Mode = BindingMode.TwoWay, _
Key .Converter = New LeftAlignmentToggleConverter() _
}
' Binds the IsChecked property to Selection.ParagraphFormat.TextAlignment
property of RichTextBoxAdv.
toggleButton.SetBinding(ToggleButton.IsCheckedProperty, binding)
```

Keyboard shortcuts to perform selection

The following keyboard shortcuts are supported by SfRichTextBoxAdv for navigation.

Navigation Shortcut	Description
Right Arrow	Navigates to one position forward.
Left Arrow	Navigates to one position backward.
Down Arrow	Navigates to the same position at next line.
Up Arrow	Navigates to the same position at previous line.
Home	Navigates to start of the current line.
End	Navigates to end of the current line.
CTRL + Home	Navigates to the document start position.
CTRL + End	Navigates to the document end position.
CTRL + Right	Navigates to the next word start position.
CTRL + Left	Navigates to the current word start position.
CTRL + Down	Navigates to the next paragraph start position.
CTRL + Up	Navigates to the current paragraph start position.

The following keyboard shortcuts are supported by SfRichTextBoxAdv for selection.

Selection Shortcut	Description
CTRL + Right Arrow	Extends selection to one position forward.
CTRL + Left Arrow	Extends selection to one position backward.
CTRL + Down Arrow	Extends selection to the same position at next line.
CTRL + Up Arrow	Extends selection to the same position at previous line.
SHIFT + Home	Extends selection to start of the current line.
SHIFT + End	Extends selection to end of the current line.
CTRL + SHIFT + Home	Extends selection to the document start position.
CTRL + SHIFT + End	Extends selection to the document end position.
CTRL + SHIFT + Right	Extends selection to the current word end position.
CTRL + SHIFT + Left	Extends selection to the current word start position.
CTRL + SHIFT + Down	Extends selection to the current paragraph end position.
CTRL + SHIFT + Up	Extends selection to the current paragraph start position.
CTRL + A	Selects the entire document.

How to show blinking cursor and selection highlight even when the control lost focus

The SfRichTextBoxAdv control allows you to show the blinking cursor and selection highlight even when the control doesn't have focus. You can choose any one of the following selection visibility options:

- **None** - Don't display neither caret nor selection highlight when the RichTextBox control doesn't have focus.
- **ShowCaret** - Displays the caret (blinking cursor) when the RichTextBox control doesn't have focus.
- **ShowSelection** - Displays the selection highlight when the RichTextBox control doesn't have focus.

The following code example demonstrates how to display the selection highlight even when the RichTextBox control doesn't have focus.

XML

```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv"
LostFocusBehavior="ShowSelection" xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf" />
```

C#

```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
// Displays the selection highlight when the RichTextBox control doesn't
have focus.
richTextBoxAdv.LostFocusBehavior = LostFocusBehavior.ShowSelection;
```

VB.NET

```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
' Displays the selection highlight when the RichTextBox control doesn't have
focus.
richTextBoxAdv.LostFocusBehavior = LostFocusBehavior.ShowSelection
```

Note: This API is supported starting from release version v17.4.0.X.

How to determine the editing context type

The SfRichTextBoxAdv control allows you to know the editing context type based on the selected content. The following are the editing context types:

- **Text** - Denotes that the editing context is text
- **Image** - Denotes that the editing context is image
- **Table** - Denotes that the editing context is table

The following code example demonstrates how to determine the editing context type based on the selection.

C#

```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
if (richTextBoxAdv.Selection EditingContext.Type == EditingContextType.Text)
{
}
```

VB.NET

```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
If (richTextBoxAdv.Selection EditingContext.Type == EditingContextType.Text)
Then
End If
```

How to delete the selected content

The SfRichTextBoxAdv supports deleting the selected portion of the document either through UI command, keyboard or through supported APIs.

The following code sample demonstrates how to delete the selected portion of the document using the DeleteKeyCommand.

XML

```
<!-- Binds button to the DeleteKeyCommand -->
<Button Content="Delete"
Command="RichTextBoxAdv:SfRichTextBoxAdv.DeleteKeyCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}"/>
```

The following code sample demonstrates how to delete the selected portion of the document using the Delete method. This method is valid only when the selection is non-empty, and it returns true if the selected content is deleted. Otherwise false.

C#

```
//Deletes the selected content in SfRichTextBoxAdv control.
bool isDeleted = richTextBoxAdv.Selection.Delete();
```

VB.NET

```
Dim isDeleted As Boolean = richTextBoxAdv.Selection.Delete()
```

Note: This API is supported starting from release version v18.2.0.X.

You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Find and Replace in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv control supports searching text contents in the document. This when used in combination with selection becomes a powerful tool enabling scenarios like highlighting specific parts of the document, applying formatting such as bold or replacing text. You can extend your search by using regular expression to find particular pattern of text in the document.

The following code example explains how to find the first occurrence of a particular text in the document and apply bold formatting.

C#

```
// Finds the first occurrence of specified text that matches case in
RichTextBoxAdv document from current selection.
TextSearchResult textSearchResult = richTextBoxAdv.Find("Panda",
FindOptions.CaseSensitive);
// Selects the text search result.
richTextBoxAdv.Selection.Select(textSearchResult.Start,
textSearchResult.End);
// Applies Bold formatting for the current selection.
richTextBoxAdv.Selection.CharacterFormat.Bold = true;
```

VB.NET

```
' Finds the first occurrence of specified text that matches case in
RichTextBoxAdv document from current selection.
Dim textSearchResult As TextSearchResult = richTextBoxAdv.Find("Panda",
FindOptions.CaseSensitive)
' Selects the text search result.
richTextBoxAdv.Selection.[Select](textSearchResult.Start,
textSearchResult.[End])
' Applies Bold formatting for the current selection.
richTextBoxAdv.Selection.CharacterFormat.Bold = True
```

The following code example demonstrates how to find all occurrences of a particular pattern of text in the document and highlighting the result.

C#

```
// Finds all the words that starts with 'S' in RichTextBoxAdv document.
TextSearchResults textSearchResults = richTextBoxAdv.FindAll(new
Regex(@"\bS\S*"), FindOptions.None);
// If any text search results found.
if (textSearchResults != null)
{
for (int j = 0; j < textSearchResults.Count; j++)
{
TextSearchResult textSearchResult = textSearchResults[j];
// Adds the search result text positions to the selection.
richTextBoxAdv.Selection.SelectionRanges.Add(textSearchResult.Start,
textSearchResult.End);
}
// Apply highlight color for the selection.
richTextBoxAdv.Selection.CharacterFormat.HighlightColor =
HighlightColor.Yellow;
}
```

VB.NET

```
' Finds all the words that starts with 'S' in RichTextBoxAdv document.
Dim textSearchResults As TextSearchResults = richTextBoxAdv.FindAll(New
Regex(@"\bS\S*"), FindOptions.None)
' If any text search results found.
```

```
If textSearchResults IsNot Nothing Then
For j As Integer = 0 To textSearchResults.Count - 1
Dim textSearchResult As TextSearchResult = textSearchResults(j)
' Adds the search result text positions to the selection.
richTextBoxAdv.Selection.SelectionRanges.Add(textSearchResult.Start,
textSearchResult.[End])
Next
' Apply highlight color for the selection.
richTextBoxAdv.Selection.CharacterFormat.HighlightColor =
HighlightColor.Yellow
End If
```

Replacing existing text

You can replace a single occurrence or all occurrences of a particular text or pattern of text in a document with another text by performing search operation. This helps you to modify the contents easily.

The following code example demonstrates how to replace a single occurrence of a text with another text in SfRichTextBoxAdv.

C#

```
// Finds the text "colour" that matches whole word in the document.
TextSearchResult textSearchResult = richTextBoxAdv.Find("colour",
FindOptions.WholeWord);
// If any text search result found, replace it with the text "color".
if (textSearchResult != null)
textSearchResult.Replace("color");
```

VB.NET

```
' Finds the text "colour" that matches whole word in the document.
Dim textSearchResult As TextSearchResult = richTextBoxAdv.Find("colour",
FindOptions.WholeWord)
' If any text search result found, replace it with the text "color".
If textSearchResult IsNot Nothing Then
textSearchResult.Replace("color")
End If
```

The following code example demonstrates how to replace all occurrences of a particular text with another text in SfRichTextBoxAdv.

C#

```
// Finds the text "analyse" that matches whole word in the document.
TextSearchResults textSearchResults = richTextBoxAdv.FindAll("analyse",
FindOptions.WholeWord);
// If any text search results found, replace all occurrences with the text
"analyze".
if(textSearchResults != null)
textSearchResults.ReplaceAll("analyze");
```

VB.NET

```
' Finds the text "analyse" that matches whole word in the document.
Dim textSearchResults As TextSearchResults =
richTextBoxAdv.FindAll("analyse", FindOptions.WholeWord)
' If any text search results found, replace all occurrences with the text
"analyze".
If textSearchResults IsNot Nothing Then
textSearchResults.ReplaceAll("analyse")
End If
```

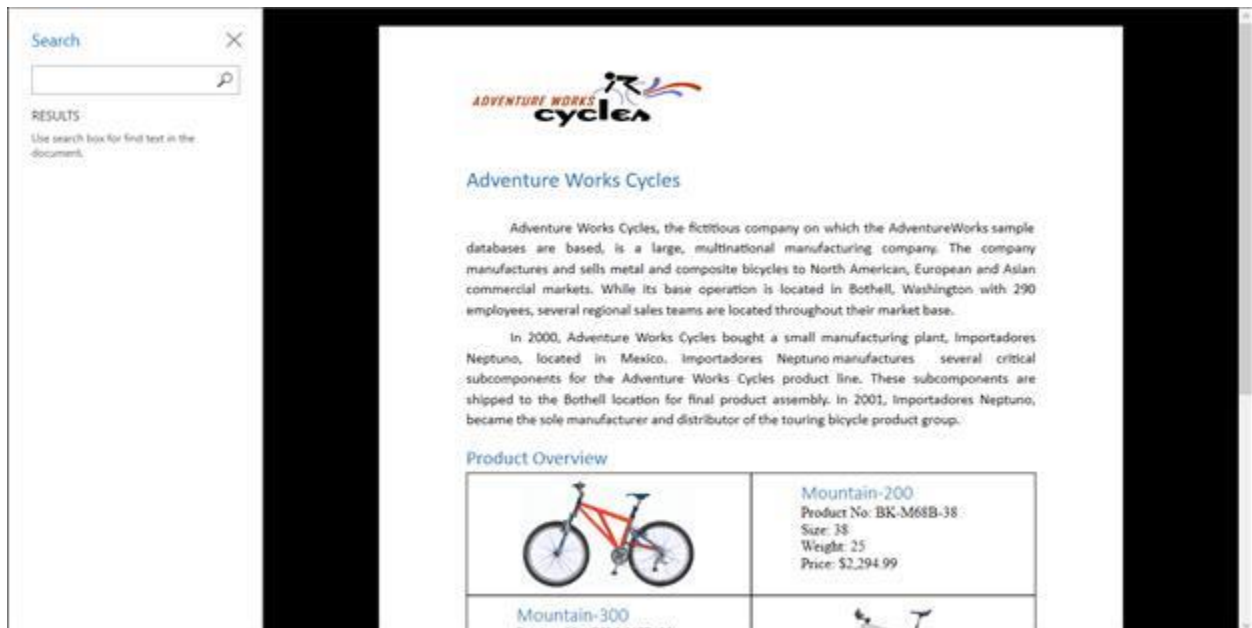
Options Pane

The SfRichTextBoxAdv provides the built-in options pane support to find the text and navigate to text results similar to Microsoft Word application.

The following code example demonstrates how to show the options pane in SfRichTextBoxAdv through command binding.

XML

```
<!-- Binding Button to UI Command that shows the options pane -->
<Button Content="Show Options Pane"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowOptionsPaneCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
```



Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Spell Check in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv provides support for checking spelling mistakes in the rich text document content. It also supports enabling the following spell checking options.

- Ignore words in UPPERCASE.

- Ignore words that contain numbers.
- Ignore URIs.

The following sample code demonstrates how to enable spell checker in SfRichTextBoxAdv.

XML

```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv"
xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf">
<RichTextBoxAdv:SfRichTextBoxAdv.SpellChecker>
<RichTextBoxAdv:SpellChecker IsEnabled="True" IgnoreURIs="true"
IgnoreAlphaNumericWords="true" IgnoreUppercaseWords="true"/>
</RichTextBoxAdv:SfRichTextBoxAdv.SpellChecker>
</RichTextBoxAdv:SfRichTextBoxAdv>
```

C#

```
// Enables spell checker in RichTextBoxAdv.
richTextBoxAdv.SpellChecker.IsEnabled = true;
// Ignores alpha numeric words while spell check.
richTextBoxAdv.SpellChecker.IgnoreAlphaNumericWords = true;
// Ignores upper case words while spell check.
richTextBoxAdv.SpellChecker.IgnoreUppercaseWords = true;
// Ignores URIs while spell check.
richTextBoxAdv.SpellChecker.IgnoreURIs = true;
```

VB.NET

```
' Enables spell checker in RichTextBoxAdv.
richTextBoxAdv.SpellChecker.IsEnabled = True
' Ignores alpha numeric words while spell check.
richTextBoxAdv.SpellChecker.IgnoreAlphaNumericWords = True
' Ignores upper case words while spell check.
richTextBoxAdv.SpellChecker.IgnoreUppercaseWords = True
' Ignores URIs while spell check.
richTextBoxAdv.SpellChecker.IgnoreURIs = True
```

Adding Custom Dictionaries

The SfRichTextBoxAdv also supports defining custom dictionaries that can be referred while check spelling mistakes. The SfRichTextBoxAdv ignores words that are defined in the referenced custom dictionaries. The SfRichTextBoxAdv supports option for adding misspelled word to dictionary. This option will be enabled only when at least one custom dictionary is defined. The misspelled words will be added to first item in the custom dictionary collection.

The following code example demonstrates how to define custom dictionaries for spell checking.

XML

```
<!-- xmlns:System="clr-namespace:System;assembly=mscorlib" -->
<!-- xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf" -->
```

```
<RichTextBoxAdv:SpellChecker IsEnabled="True" IgnoreURIs="true"
IgnoreAlphaNumericWords="true" IgnoreUppercaseWords="true">
<RichTextBoxAdv:SpellChecker.CustomDictionaries>
<System:String>../../../../Assets/DefaultDictionary.dic</System:String>
<System:String>../../../../Assets/CustomDictionary.dic</System:String>
</RichTextBoxAdv:SpellChecker.CustomDictionaries>
</RichTextBoxAdv:SpellChecker>
```

Multilingual Spell Check Support

The SfRichTextBoxAdv provides support for check spelling mistakes based on multi languages. You can do it so by defining language property for SfRichTextBoxAdv control.

The following code example demonstrates how to enable spell checking based on language.

XML

```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv"
xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf" Language="fr">
<RichTextBoxAdv:SfRichTextBoxAdv.SpellChecker>
<RichTextBoxAdv:SpellChecker IsEnabled="True" IgnoreURIs="true"
IgnoreAlphaNumericWords="true" IgnoreUppercaseWords="true"/>
</RichTextBoxAdv:SfRichTextBoxAdv.SpellChecker>
</RichTextBoxAdv:SfRichTextBoxAdv>
```

Note: In order to enable spell checking functionality based on particular language, language pack for .NET Framework should be installed in the machine.

You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Clipboard in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv takes advantage of the clipboard support and allows you to copy or paste contents to and from the clipboard in the following formats.

- Rich text format.
- Text.
- Image.

UI Command to access clipboard operations

The following code example demonstrates how to bind commands for accessing clipboard operations.

XML

```
<!-- Binds button to the CutCommand -->
<Button Content="Cut" Command="RichTextBoxAdv:SfRichTextBoxAdv.CutCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the CopyCommand -->
<Button Content="Copy" Command="RichTextBoxAdv:SfRichTextBoxAdv.CopyCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the PasteCommand -->
```

```
<Button Content="Paste"
Command="RichTextBoxAdv:SfRichTextBoxAdv.PasteCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
```

Note: In order to cut, copy or paste, the standard keyboard shortcuts such as CTRL + X, CTRL + C, CTRL + V can also be used.

You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Undo Redo in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv provides history preservation support, which means each editing operation performed against its document content will be preserved in history. You can easily undo any editing action with ease. The undone actions will also be preserved in a separate stack enabling you to redo the action again.

Note: Currently, the number of actions that can be preserved in both undo and redo stacks is limited to 500.

UI Command to perform undo/redo operations

The following code example demonstrates how to bind commands for performing undo and redo operations.

XML

```
<!-- Binds button to the UndoCommand -->
<Button Content="Undo" Command="RichTextBoxAdv:SfRichTextBoxAdv.UndoCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the RedoCommand -->
<Button Content="Redo" Command="RichTextBoxAdv:SfRichTextBoxAdv.RedoCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
```

Note: In order to perform undo/redo, the standard keyboard shortcuts such as CTRL + Z, CTRL + Y can also be used.

Enable/Disable Undo Redo

Disable Undo for all the editing actions

In SfRichTextBoxAdv, the Undo functionality is enabled by default. You can disable it by using the `IsUndoEnabled` property of the `EditorSettings` class.

The following code example demonstrates how to disable the Undo functionality in SfRichTextBoxAdv.

XML

```
<Syncfusion:SfRichTextBoxAdv x:Name="richTextBoxAdv">
<Syncfusion:SfRichTextBoxAdv.EditorSettings>
<Syncfusion:EditorSettings IsUndoEnabled="False"/>
</Syncfusion:SfRichTextBoxAdv.EditorSettings>
</Syncfusion:SfRichTextBoxAdv>
```

C#

```
// Defines the SfRichTextBoxAdv control with undo operation disabled.
```



```
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
richTextBoxAdv.EditorSettings.IsUndoEnabled = false;
```

VB.NET

```
' Defines the SfRichTextBoxAdv control with undo operation disabled.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
richTextBoxAdv.EditorSettings.IsUndoEnabled = false
```

Note: This API is supported starting from release version v18.1.0.X.

Disable Undo for style modification actions

In SfRichTextBoxAdv, the Undo functionality is enabled by default for all the editing operations. If you want to disable the Undo functionality for modifying an existing style alone. You can disable it by using the `CanUndoStyle` property of the `EditorSettings` class.

The following code example demonstrates how to disable the Undo support for modifying an existing style in SfRichTextBoxAdv.

XML

```
<Syncfusion:SfRichTextBoxAdv x:Name="richTextBoxAdv">
  <Syncfusion:SfRichTextBoxAdv.EditorSettings>
    <Syncfusion:EditorSettings CanUndoStyle="False"/>
  </Syncfusion:SfRichTextBoxAdv.EditorSettings>
</Syncfusion:SfRichTextBoxAdv>
```

C#

```
// Defines the SfRichTextBoxAdv control with document style undo operation disabled.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
richTextBoxAdv.EditorSettings.CanUndoStyle = false;
```

VB.NET

```
' Defines the SfRichTextBoxAdv control with document style undo operation disabled.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
richTextBoxAdv.EditorSettings.CanUndoStyle = false
```

Note: This API is supported starting from release version v18.1.0.X.

You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

List in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv supports both the single-level and multilevel lists similar to the Microsoft Word. Lists are used to organize data as step-by-step instructions in documents for easy understanding of key points.

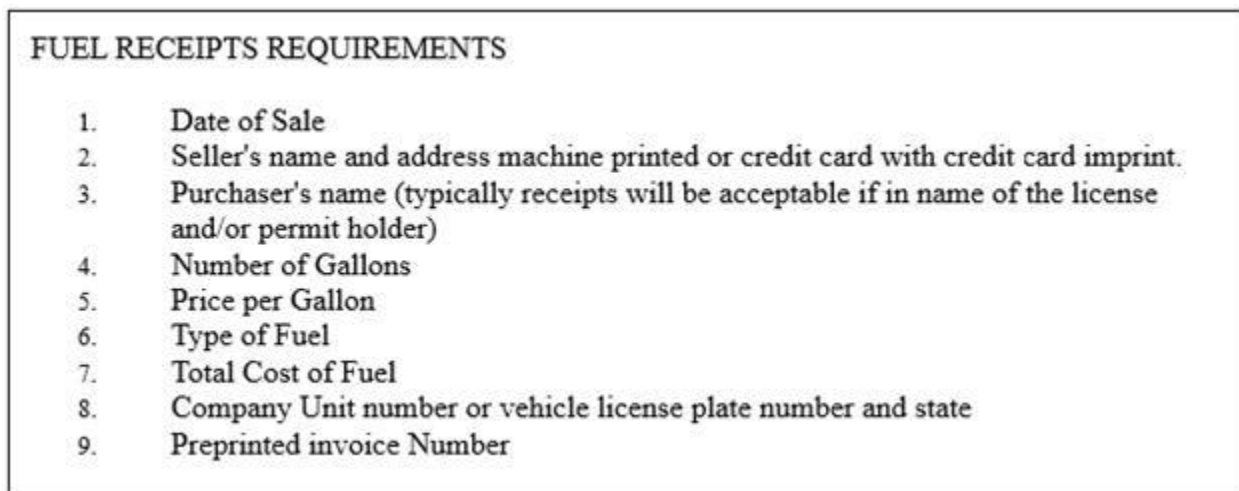
Single Level List

Single level means that all the items in the list have the same hierarchy and indentation. It can be a numbered or a bulleted list.

The following screenshot shows single level bulleted list.



The following screenshot shows single level numbered list.



Multilevel List

Multilevel means defining a list within a list where up to nine levels can be defined similar to the Microsoft Word. A multilevel list can be bulleted or numbered and also mixed with numbers, letters, and bullets. For example, one level can be bulleted and next level can be a numbered list inside it.

The following screenshot shows multilevel list.

Image Formats

1. Image File Compression
 - 1.1. Lossless Compression
 - 1.2. Lossy Compression
2. Exif – Exchangable Image File Format those using JPEG Compression
3. Major graphic file formats
 - 3.1. Raster formats
 - 3.1.1.JPEG
 - 3.1.2.TIF
 - 3.1.3.RAW

Adding List

Each list in the document can contain reference to any one of the abstract lists in the document. Both abstract list and list should be defined a unique Id. List should refer the abstract list with the abstract list's Id. List format for a paragraph should refer to the list with the list's Id.

The following code example demonstrates how to define a single level numbered list for a document and how it is applied to a paragraph.

XML

```
<RichTextBoxAdv:DocumentAdv>
<RichTextBoxAdv:DocumentAdv.AbstractLists>
<RichTextBoxAdv:AbstractListAdv AbstractListId="1">
<RichTextBoxAdv:AbstractListAdv.Levels>
<RichTextBoxAdv:ListLevelAdv ListLevelPattern="LowLetter" NumberFormat="%1."
StartAt="1" FollowCharacter="Tab" RestartLevel="0">
<RichTextBoxAdv:ListLevelAdv.ParagraphFormat>
<RichTextBoxAdv:ParagraphFormat LeftIndent="48" FirstLineIndent="24"/>
</RichTextBoxAdv:ListLevelAdv.ParagraphFormat>
</RichTextBoxAdv:ListLevelAdv>
</RichTextBoxAdv:AbstractListAdv.Levels>
</RichTextBoxAdv:AbstractListAdv>
</RichTextBoxAdv:DocumentAdv.AbstractLists>
<RichTextBoxAdv:DocumentAdv.Lists>
<RichTextBoxAdv:ListAdv AbstractListId="1" ListId="1">
</RichTextBoxAdv:ListAdv>
</RichTextBoxAdv:DocumentAdv.Lists>
<RichTextBoxAdv:SectionAdv>
<RichTextBoxAdv:ParagraphAdv>
<RichTextBoxAdv:ParagraphAdv.ParagraphFormat>
<RichTextBoxAdv:ParagraphFormat>
<RichTextBoxAdv:ParagraphFormat.ListFormat>
<RichTextBoxAdv:ListFormat ListId="1" ListLevelNumber="0"/>
</RichTextBoxAdv:ParagraphFormat.ListFormat>
</RichTextBoxAdv:ParagraphFormat>
```

```

</RichTextBoxAdv:ParagraphAdv.ParagraphFormat>
<RichTextBoxAdv:SpanAdv>List Item 1</RichTextBoxAdv:SpanAdv>
</RichTextBoxAdv:ParagraphAdv>
</RichTextBoxAdv:SectionAdv>
</RichTextBoxAdv:DocumentAdv>

```

C#

```

// Initializes a new abstract list instance.
AbstractListAdv abstractListAdv = new AbstractListAdv(null);
abstractListAdv.AbstractListId = 1;
// Defines new ListLevel instance.
ListLevelAdv listLevel = new ListLevelAdv(abstractListAdv);
listLevel.ParagraphFormat.LeftIndent = 48d;
listLevel.ParagraphFormat.FirstLineIndent = 24d;
listLevel.FollowCharacter = FollowCharacterType.Tab;
listLevel.ListLevelPattern = ListLevelPattern.LowLetter;
listLevel.NumberFormat = "%1.";
listLevel.RestartLevel = 0;
listLevel.StartAt = 1;
// Adds list level to abstract list.
abstractListAdv.Levels.Add(listLevel);
// Adds abstract list to the document.
richTextBoxAdv.Document.AbstractLists.Add(abstractListAdv);
// Creates a new list instance.
ListAdv listAdv = new ListAdv(null);
listAdv.ListId = 1;
// Sets the abstract list Id for this list.
listAdv.AbstractListId = 1;
// Adds list to the document.
richTextBoxAdv.Document.Lists.Add(listAdv);
// Add list item 1
ParagraphAdv paragraphAdv = new ParagraphAdv();
SpanAdv spanAdv = new SpanAdv() { Text = "List Item 1" };
paragraphAdv.Inlines.Add(spanAdv);
richTextBoxAdv.Document.Sections[0].Blocks.Add(paragraphAdv);
// Defines the list format for the paragraph.
paragraphAdv.ParagraphFormat.ListFormat.ListId = 1;
paragraphAdv.ParagraphFormat.ListFormat.ListLevelNumber = 0;

```

VB.NET

```

' Initializes a new abstract list instance.
Dim abstractListAdv As New AbstractListAdv(Nothing)
abstractListAdv.AbstractListId = 1
' Defines new ListLevel instance.
Dim listLevel As New ListLevelAdv(abstractListAdv)
listLevel.ParagraphFormat.LeftIndent = 48.0
listLevel.ParagraphFormat.FirstLineIndent = 24.0
listLevel.FollowCharacter = FollowCharacterType.Tab
listLevel.ListLevelPattern = ListLevelPattern.LowLetter
listLevel.NumberFormat = "%1."
listLevel.RestartLevel = 0
listLevel.StartAt = 1
' Adds list level to abstract list.
abstractListAdv.Levels.Add(listLevel)

```

```

' Adds abstract list to the document.
richTextBoxAdv.Document.AbstractLists.Add(abstractListAdv)
' Creates a new list instance.
Dim listAdv As New ListAdv(Nothing)
listAdv.ListId = 1
' Sets the abstract list Id for this list.
listAdv.AbstractListId = 1
' Adds list to the document.
richTextBoxAdv.Document.Lists.Add(listAdv)
' Add list item 1
Dim paragraphAdv As ParagraphAdv = New ParagraphAdv()
Dim spanAdv As SpanAdv = New SpanAdv()
spanAdv.Text = "List Item 1"
paragraphAdv.Inlines.Add(spanAdv)
richTextBoxAdv.Document.Sections(0).Blocks.Add(paragraphAdv)
' Defines the list format for the paragraph.
paragraphAdv.ParagraphFormat.ListFormat.ListId = 1
paragraphAdv.ParagraphFormat.ListFormat.ListLevelNumber = 0

```

The following code example demonstrates how to define number format for numbered list in the SfRichTextBoxAdv control.

C#

```

// Defines the number format for the list level.
/* Note
 * The percent sign (%) followed by any number from 1 through 9 represents
 the number style from the respective list level.
 * For example, if you wanted the format for the first level to be "Article
 I." "Article II," and so on, the string for the NumberFormat property would
 be "Article %1." and the ListLevelPattern property would be set to
 ListLevelPattern.UpRoman.
 */
listLevel.NumberFormat = "Article %1.";
listLevel.ListLevelPattern = ListLevelPattern.UpRoman;

```

VB.NET

```

' Defines the number format for the list level.
' Note
' * The percent sign (%) followed by any number from 1 through 9 represents
 the number style from the respective list level.
' * For example, if you wanted the format for the first level to be "Article
 I." "Article II," and so on, the string for the NumberFormat property would
 be "Article %1." and the ListLevelPattern property would be set to
 ListLevelPattern.UpRoman.
'
listLevel.NumberFormat = "Article %1."
listLevel.ListLevelPattern = ListLevelPattern.UpRoman

```

You can define bulleted list by setting list level pattern as Bullet. You can define various bullets by defining the bullet character. The following code sample demonstrates how to define dot, square and arrow bullets in the SfRichTextBoxAdv control.

C#

```
// Defines Bulleted List.
listLevel.ListLevelPattern = ListLevelPattern.Bullet;
// Defining Dot Bullet
listLevel.NumberFormat = "\uf0b7";
listLevel.CharacterFormat.FontFamily = new FontFamily("Symbol");
// Defines Square bullet.
listLevel.NumberFormat = "\uf0a7";
listLevel.CharacterFormat.FontFamily = new FontFamily("Wingdings");
// Defines Arrow Bullet.
listLevel.NumberFormat = "\u27a4";
listLevel.CharacterFormat.FontFamily = new FontFamily("Symbol");
```

VB.NET

```
' Defines Bulleted List.
listLevel.ListLevelPattern = ListLevelPattern.Bullet
' Defining Dot Bullet
listLevel.NumberFormat = "\uf0b7"
listLevel.CharacterFormat.FontFamily = New FontFamily("Symbol")
' Defines Square bullet.
listLevel.NumberFormat = "\uf0a7"
listLevel.CharacterFormat.FontFamily = New FontFamily("Wingdings")
' Defines Arrow Bullet.
listLevel.NumberFormat = "\u27a4"
listLevel.CharacterFormat.FontFamily = New FontFamily("Symbol")
```

Level overrides

The list levels for a list are defined in the abstract list to which it refers to. Additionally you can define level overrides for any list level. The SfRichTextBoxAdv supports two types of level overrides.

1. Start at override – Only start value for the list is overridden and other properties are referred to list level defined in abstract list.
2. Level override – The list level is completely overridden.

The following code example demonstrates how to override the start at value for an existing list level in the SfRichTextBoxAdv control.

XML

```
<RichTextBoxAdv:ListAdv AbstractListId="1" ListId="1">
  <RichTextBoxAdv:ListAdv.LevelOverrides>
    <RichTextBoxAdv:LevelOverrideAdv StartAt="2" LevelNumber="0"/>
  </RichTextBoxAdv:ListAdv.LevelOverrides>
</RichTextBoxAdv:ListAdv>
```

C#

```
// Adds StartAtOverride for the list at first level.
// LevelNumber ranges from 0 to 8.
LevelOverrideAdv levelOverride = new LevelOverrideAdv(listAdv);
levelOverride.LevelNumber = 0;
levelOverride.StartAt = 2;
listAdv.LevelOverrides.Add(levelOverride);
```

VB.NET

```
' Adds StartAtOverride for the list at first level.
' LevelNumber ranges from 0 to 8.
Dim levelOverride As New LevelOverrideAdv(listAdv)
levelOverride.LevelNumber = 0
levelOverride.StartAt = 2
listAdv.LevelOverrides.Add(levelOverride)
```

The following code example demonstrates how to add level override for any existing list level in the SfRichTextBoxAdv control.

XML

```
<RichTextBoxAdv:ListAdv AbstractListId="1" ListId="1">
  <RichTextBoxAdv:ListAdv.LevelOverrides>
    <!-- Overrides fourth list level-->
    <RichTextBoxAdv:LevelOverrideAdv LevelNumber="3">
      <RichTextBoxAdv:LevelOverrideAdv.OverrideListLevel>
        <RichTextBoxAdv:ListLevelAdv ListLevelPattern="UpRoman" StartAt="3"
        NumberFormat="%1) " />
      </RichTextBoxAdv:LevelOverrideAdv.OverrideListLevel>
    </RichTextBoxAdv:LevelOverrideAdv>
  </RichTextBoxAdv:ListAdv.LevelOverrides>
</RichTextBoxAdv:ListAdv>
```

C#

```
// Adds ListLevel override for the list at fourth level.
// LevelNumber ranges from 0 to 8.
LevelOverrideAdv levelOverride = new LevelOverrideAdv(listAdv);
levelOverride.LevelNumber = 3;
levelOverride.OverrideListLevel = new ListLevelAdv(levelOverride);
levelOverride.OverrideListLevel.ListLevelPattern = ListLevelPattern.UpRoman;
levelOverride.OverrideListLevel.NumberFormat = "%1) ";
levelOverride.OverrideListLevel.StartAt = 3;
listAdv.LevelOverrides.Add(levelOverride);
```

VB.NET

```
' Adds ListLevel override for the list at fourth level.
' LevelNumber ranges from 0 to 8.
Dim levelOverride As New LevelOverrideAdv(listAdv)
levelOverride.LevelNumber = 3
levelOverride.OverrideListLevel = New ListLevelAdv(levelOverride)
levelOverride.OverrideListLevel.ListLevelPattern = ListLevelPattern.UpRoman
levelOverride.OverrideListLevel.NumberFormat = "%1) "
levelOverride.OverrideListLevel.StartAt = 3
listAdv.LevelOverrides.Add(levelOverride)
```

Editing list

You can retrieve the list applied for the current selection. By doing so, you can edit the list according to your requirement. After editing the list, you need to set it for the current selection in order to make the changes effective.

The following code sample demonstrates how to apply the list to the selection content in the SfRichTextBoxAdv control.

C#

```
// Gets the current list for the selection content.
ListAdv listAdv = richTextBoxAdv.Selection.ParagraphFormat.GetList();
```

VB.NET

```
' Gets the current list for the selection content.
Dim listAdv As ListAdv = richTextBoxAdv.Selection.ParagraphFormat.GetList()
```

The following code example demonstrates how to apply a list for the selection content in the SfRichTextBoxAdv control. When the selection content has a list, then it gets modified with that list. Otherwise the list is added to the document and applied to the selection content.

C#

```
// Applies list for the Selection content.
richTextBoxAdv.Selection.ParagraphFormat.SetList(listAdv);
richTextBoxAdv.Selection.ParagraphFormat.ListLevelNumber = 0;
```

VB.NET

```
' Applies list for the Selection content.
richTextBoxAdv.Selection.ParagraphFormat.SetList(listAdv)
richTextBoxAdv.Selection.ParagraphFormat.ListLevelNumber = 0
```

Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Hyperlink in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv supports hyperlink field similar to the Microsoft Word. You can link part of the document content to Internet or file location, mail address or any text.

The following code example illustrates how to insert a hyperlink field.

XML

```
<RichTextBoxAdv:ParagraphAdv>
<RichTextBoxAdv:FieldBeginAdv></RichTextBoxAdv:FieldBeginAdv>
<RichTextBoxAdv:SpanAdv Text=" HYPERLINK &#34;http://www.syncfusion.com&#34;
"></RichTextBoxAdv:SpanAdv>
<RichTextBoxAdv:FieldSeparatorAdv></RichTextBoxAdv:FieldSeparatorAdv>
<RichTextBoxAdv:SpanAdv Text="Syncfusion">
<RichTextBoxAdv:SpanAdv.CharacterFormat>
<RichTextBoxAdv:CharacterFormat Underline="Single" FontColor="#ff0563c1"/>
```



```

</RichTextBoxAdv:SpanAdv.CharacterFormat>
</RichTextBoxAdv:SpanAdv>
<RichTextBoxAdv:FieldEndAdv></RichTextBoxAdv:FieldEndAdv>
</RichTextBoxAdv:ParagraphAdv>

```

C#

```

// Appends the field start.
paragraphAdv.Inlines.Add(new FieldBeginAdv());
// Appends the field code.
SpanAdv fieldCode = new SpanAdv();
string url = "www.syncfusion.com";
fieldCode.Text = " HYPERLINK \"" + url + "\" ";
paragraphAdv.Inlines.Add(fieldCode);
// Appends the field separator
paragraphAdv.Inlines.Add(new FieldSeparatorAdv());
// Appends the field result.
SpanAdv fieldResult = new SpanAdv();
fieldResult.Text = "Syncfusion";
fieldResult.CharacterFormat.Underline = Underline.Single;
fieldResult.CharacterFormat.FontColor = Color.FromArgb(0xff, 0x05, 0x63,
0xc1);
paragraphAdv.Inlines.Add(fieldResult);
// Appends the field end.
paragraphAdv.Inlines.Add(new FieldEndAdv());

```

VB.NET

```

' Appends the field start.
paragraphAdv.Inlines.Add(New FieldBeginAdv())
' Appends the field code.
Dim fieldCode As New SpanAdv()
Dim url As String = "www.syncfusion.com"
fieldCode.Text = (Convert.ToString(" HYPERLINK """) & url) + "" "
paragraphAdv.Inlines.Add(fieldCode)
' Appends the field separator
paragraphAdv.Inlines.Add(New FieldSeparatorAdv())
' Appends the field result.
Dim fieldResult As New SpanAdv()
fieldResult.Text = "Syncfusion"
fieldResult.CharacterFormat.Underline = Underline.[Single]
fieldResult.CharacterFormat.FontColor = Color.FromArgb(&Hff, &H5, &H63,
&Hc1)
paragraphAdv.Inlines.Add(fieldResult)
' Appends the field end.
paragraphAdv.Inlines.Add(New FieldEndAdv())

```

The following code example illustrates how to insert hyperlink field into SfRichTextBoxAdv Document through UI command.

XML

```



<Button Content="Insert Hyperlink"
Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertHyperlinkCommand"

```

```
CommandTarget="{Binding ElementName=richTextBoxAdv}"
CommandParameter="www.google.com"/>
```

Hyperlink ScreenTip

In RichTextBox control ToolTip (ScreenTip) shows some information or navigation link, when the mouse hovers over that hyperlink and it disappears when the mouse is moved away from that hyperlink. By default, it shows navigation link of that hyperlink and you can set the text you want to use for your ScreenTip.

Without ScreenTipText	With ScreenTipText
	

The following code example illustrates how to insert a hyperlink field with ScreenTip.

XML

```
<RichTextBoxAdv:ParagraphAdv>
<RichTextBoxAdv:FieldBeginAdv></RichTextBoxAdv:FieldBeginAdv>
<RichTextBoxAdv:SpanAdv Text=" HYPERLINK
&#34;\http://www.syncfusion.com&#34; \o SfRichTextBox
"></RichTextBoxAdv:SpanAdv>
<RichTextBoxAdv:FieldSeparatorAdv></RichTextBoxAdv:FieldSeparatorAdv>
<RichTextBoxAdv:SpanAdv Text="SfRichTextBoxAdv">
<RichTextBoxAdv:SpanAdv.CharacterFormat>
<RichTextBoxAdv:CharacterFormat Underline="Single" FontColor="#ff0563c1"/>
</RichTextBoxAdv:SpanAdv.CharacterFormat>
</RichTextBoxAdv:SpanAdv>
<RichTextBoxAdv:FieldEndAdv></RichTextBoxAdv:FieldEndAdv>
</RichTextBoxAdv:ParagraphAdv>
```

C#

```
ParagraphAdv paragraphAdv = new ParagraphAdv();
// Appends the field start.
paragraphAdv.Inlines.Add(new FieldBeginAdv());
// Appends the field code.
SpanAdv fieldCode = new SpanAdv();
string url = "www.syncfusion.com";
string screenTip = "SfRichTextBox";
fieldCode.Text = " HYPERLINK \"" + url + "\" \o \"" + screenTip + "\" ";
paragraphAdv.Inlines.Add(fieldCode);
// Appends the field separator
paragraphAdv.Inlines.Add(new FieldSeparatorAdv());
// Appends the field result.
SpanAdv fieldResult = new SpanAdv();
fieldResult.Text = "SfRichTextBoxAdv";
fieldResult.CharacterFormat.Underline = Underline.Single;
fieldResult.CharacterFormat.FontColor = Color.FromArgb(0xff, 0x05, 0x63,
0xc1);
paragraphAdv.Inlines.Add(fieldResult);
// Appends the field end.
paragraphAdv.Inlines.Add(new FieldEndAdv());
```

```
richTextBoxAdv.Document.Sections[0].Blocks.Add(paragraphAdv);
```

VB.NET

```
' Appends the field start.
paragraphAdv.Inlines.Add(New FieldBeginAdv())
' Appends the field code.
Dim fieldCode As New SpanAdv()
Dim url As String = "www.syncfusion.com"
Dim screenTip As String = "SfRichTextBox"
fieldCode.Text = (Convert.ToString(" HYPERLINK """) & url) + "" "" "" "" "" +
(screenTip) + "" "" "" "" ""
paragraphAdv.Inlines.Add(fieldCode)
' Appends the field separator
paragraphAdv.Inlines.Add(New FieldSeparatorAdv())
' Appends the field result.
Dim fieldResult As New SpanAdv()
fieldResult.Text = "SfRichTextBoxAdv"
fieldResult.CharacterFormat.Underline = Underline.[Single]
fieldResult.CharacterFormat.FontColor = Color.FromArgb(&Hff, &H5, &H63,
&Hcl)
paragraphAdv.Inlines.Add(fieldResult)
' Appends the field end.
paragraphAdv.Inlines.Add(New FieldEndAdv())
```

The following code example illustrates how to insert hyperlink field with ScreenTip into RichTextBox Document through UI command.

C#

```
SfRichTextBoxAdv.InsertHyperlinkCommand.Execute(new string[3] {
"www.syncfusion.com", "SfRichTextBoxAdv", "SfRichTextBox" },
richTextBoxAdv);
```

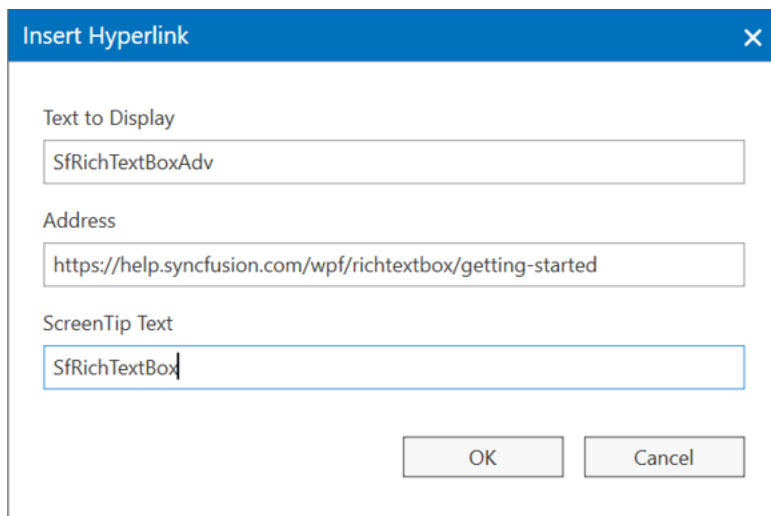
The following section illustrates how to insert hyperlink field with ScreenTip in RichTextBox Document through built-in hyperlink dialog UI command like Microsoft Word application.

1. Open insert hyperlink dialog.

XML

```
<!-- Binds button to the ShowHyperlinkDialogCommand -->
<Button Content="Hyperlink"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowHyperlinkDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
```

2. Enter the display text, URL link and ScreenTip text. 3. Click on OK to close the dialog box.



Note: ScreenTip option is supported from V18.4.0.30

Hyperlink Navigation

The SfRichTextBoxAdv supports event to identify whenever hyperlink navigation is requested. This allows you to easily customize the hyperlink navigation functionality.

The following code example demonstrates how to customize hyperlink navigation functionality for the SfRichTextBoxAdv instance.

C#

```
// Hooks the event handler for RequestNavigate event.
richTextBoxAdv.RequestNavigate += RichTextBoxAdv_RequestNavigate;
/// <summary>
/// Handles the RequestNavigate event of the richTextBoxAdv control.
/// </summary>
/// <param name="obj">The source of the event.</param>
/// <param name="args">The <see cref="RequestNavigateEventArgs"/> instance
/// containing the event data.</param>
private void RichTextBoxAdv_RequestNavigate(object obj,
Syncfusion.Windows.Controls.RichTextBoxAdv.RequestNavigateEventArgs args)
{
    if (args.Hyperlink.LinkType == HyperlinkType.Webpage ||
args.Hyperlink.LinkType == HyperlinkType.Email)
        Process.Start(new ProcessStartInfo(new
Uri(args.Hyperlink.NavigationLink).AbsoluteUri));
    else if (args.Hyperlink.LinkType == HyperlinkType.File &&
File.Exists(args.Hyperlink.NavigationLink))
        Process.Start(args.Hyperlink.NavigationLink);
}
// Unhooks the event handler for RequestNavigate event.
richTextBoxAdv.RequestNavigate -= RichTextBoxAdv_RequestNavigate;
```

VB.NET

```
' Hooks the event handler for RequestNavigate event.
AddHandler richTextBoxAdv.RequestNavigate, AddressOf
RichTextBoxAdv_RequestNavigate
''' <summary>
```

```

''' Handles the RequestNavigate event of the richTextBoxAdv control.
''' </summary>
''' <param name="obj">The source of the event.</param>
''' <param name="args">The <see cref="RequestNavigateEventArgs"/> instance
    containing the event data.</param>
Private Sub RichTextBoxAdv_RequestNavigate(obj As Object, args As
Syncfusion.Windows.Controls.RichTextBoxAdv.RequestNavigateEventArgs)
If args.Hyperlink.LinkType = HyperlinkType.Webpage OrElse
args.Hyperlink.LinkType = HyperlinkType.Email Then
Process.Start(New ProcessStartInfo(New
Uri(args.Hyperlink.NavigationLink).AbsoluteUri))
ElseIf args.Hyperlink.LinkType = HyperlinkType.File AndAlso
File.Exists(args.Hyperlink.NavigationLink) Then
Process.Start(args.Hyperlink.NavigationLink)
End If
End Sub
' Unhooks the event handler for RequestNavigate event.
RemoveHandler richTextBoxAdv.RequestNavigate, AddressOf
RichTextBoxAdv_RequestNavigate

```

Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Image in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv allows you to insert images of various formats such as bitmap images (.bmp), JPEG (.jpg, .jpeg), PNG (.png) except Metafile images.

The following code example illustrates how to insert picture into the SfRichTextBoxAdv document through UI Command.

XML

```

<Button Content="Insert Picture"
Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertPictureCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />

```

Image Resizer

The SfRichTextBoxAdv also supports built-in image resizer to resize the images present in the document to your wish. The image resizer accepts both touch and mouse interactions.



Text wrapping style

Text wrapping refers to how images are fit with surrounding text in a document. Please [refer to this page](#) for more information about text wrapping styles available in Word documents.

Positioning the image

Starting from v19.1.0.x, RichTextBox preserves the position properties of the image and displays the image based on position properties. It does not support modifying the position properties. Whereas the image will be automatically moved along with text edited if it is positioned relative to the line or paragraph.

Note: At present, the image with text wrapping style **In-Line with Text** can only be dragged and dropped anywhere in the document.

You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

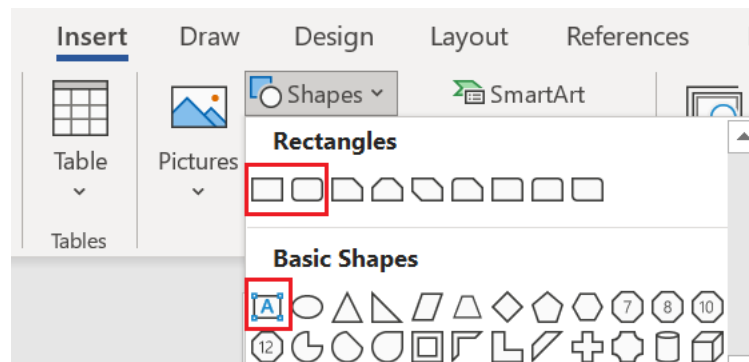
Shapes in WPF RichTextBox (SfRichTextBoxAdv)

Shapes are drawing objects that include a text box, rectangles, lines, curves, circles, etc. It can be preset or custom geometry. At present, RichTextBox does not have support to insert shapes. However, if the document contains a shape while importing, it will be preserved properly.

Note: Starting from v18.3.0.x, the shape preservation is supported.

Supported shapes

The RichTextBox has preservation support for Text box and Rectangle shapes.



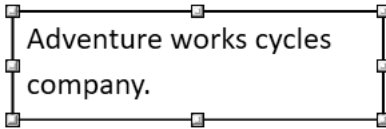
Text box Shape

A text box is a rectangular area on the document where you can enter text. When you click in a text box, a flashing cursor will display indicating that you can begin typing. It allows you to enter multiple lines of text with all text formatting.

Adventure Works Cycles, the fictitious company on which the Adventure Works sample databases are based, is a large, multinational manufacturing company. The company manufactures and sells metal Adventure works cycles and composite bicycles to North American, European and and company. Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales teams are located throughout their market base.

Shape Resizer

The RichTextBox also supports a built-in shape resizer to resize the shapes present in the document. The shape resizer accepts both touch and mouse interactions.



Text wrapping style

Text wrapping refers to how shapes fit with surrounding text in a document. Please [refer to this page](#) for more information about text wrapping styles available in Word documents.

Positioning the shape

Starting from v19.1.0.x, RichTextBox preserves the position properties of the shape and displays the shape based on position properties. It does not support modifying the position properties. Whereas the shape will be automatically moved along with text edited if it is positioned relative to the line or paragraph.

Note: At present, the shape with text wrapping style **In-Line with Text** can only be dragged and dropped anywhere in the document.

You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Table in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv allows you to add tables into the rich text document. You can insert any rows or columns to the existing table and also can delete existing rows and columns. The SfRichTextBoxAdv also allows you to merge the selected cells into one (both vertically and horizontally).

The following code example illustrates how to add tables into the rich text document.

XML

```
<RichTextBoxAdv:DocumentAdv>
<RichTextBoxAdv:SectionAdv>
<RichTextBoxAdv:TableAdv>
<RichTextBoxAdv:TableRowAdv>
<RichTextBoxAdv:TableCellAdv>
<RichTextBoxAdv:TableCellAdv.CellFormat>
<RichTextBoxAdv:CellFormat CellWidth="240"/>
</RichTextBoxAdv:TableCellAdv.CellFormat>
<RichTextBoxAdv:ParagraphAdv>
<RichTextBoxAdv:SpanAdv>Cell 1</RichTextBoxAdv:SpanAdv>
</RichTextBoxAdv:ParagraphAdv>
</RichTextBoxAdv:TableCellAdv>
<RichTextBoxAdv:TableCellAdv>
<RichTextBoxAdv:TableCellAdv.CellFormat>
<RichTextBoxAdv:CellFormat CellWidth="240"/>
</RichTextBoxAdv:TableCellAdv.CellFormat>
<RichTextBoxAdv:ParagraphAdv>
<RichTextBoxAdv:SpanAdv>Cell 2</RichTextBoxAdv:SpanAdv>
</RichTextBoxAdv:ParagraphAdv>
</RichTextBoxAdv:TableCellAdv>
</RichTextBoxAdv:TableRowAdv>
</RichTextBoxAdv:TableAdv>
<RichTextBoxAdv:ParagraphAdv/>
</RichTextBoxAdv:SectionAdv>
```

```
</RichTextBoxAdv:DocumentAdv>
```

C#

```
// Initializes a document.
DocumentAdv document = new DocumentAdv();
// Initialize a section.
SectionAdv section = new SectionAdv();
// Initialize a table.
TableAdv tableAdv = new TableAdv();
// Initialize a row.
TableRowAdv tableRowAdv = new TableRowAdv();
// Initialize a table cell.
TableCellAdv tableCellAdv = new TableCellAdv();
tableCellAdv.CellFormat.CellWidth = 240;
// Initializes a paragraph.
ParagraphAdv paragraphAdv = new ParagraphAdv();
SpanAdv spanAdv = new SpanAdv();
spanAdv.Text = "Cell 1";
paragraphAdv.Inlines.Add(spanAdv);
tableCellAdv.Blocks.Add(paragraphAdv);
// Initialize and add any number of blocks to the cell here.
tableRowAdv.Cells.Add(tableCellAdv);
// Initialize and add any number of cells to the row here.
tableAdv.Rows.Add(tableRowAdv);
// Initialize and add any number of rows to the table here.
section.Blocks.Add(tableAdv);
// Initialize and add any number of blocks to the section here.
document.Sections.Add(section);
// Initialize and add any number of sections to the document here.
// Assign the document to the RichTextBoxAdv instance.
richTextBoxAdv.Document = document;
```

VB.NET

```
' Initializes a document.
Dim document As New DocumentAdv()
' Initialize a section.
Dim section As New SectionAdv()
' Initialize a table.
Dim tableAdv As New TableAdv()
' Initialize a row.
Dim tableRowAdv As New TableRowAdv()
' Initialize a table cell.
Dim tableCellAdv As New TableCellAdv()
tableCellAdv.CellFormat.CellWidth = 240
' Initializes a paragraph.
Dim paragraphAdv As New ParagraphAdv()
Dim spanAdv As New SpanAdv()
spanAdv.Text = "Cell 1"
paragraphAdv.Inlines.Add(spanAdv)
tableCellAdv.Blocks.Add(paragraphAdv)
' Initialize and add any number of blocks to the cell here.
tableRowAdv.Cells.Add(tableCellAdv)
' Initialize and add any number of cells to the row here.
tableAdv.Rows.Add(tableRowAdv)
```



```
' Initialize and add any number of rows to the table here.
section.Blocks.Add(tableAdv)
' Initialize and add any number of blocks to the section here.
document.Sections.Add(section)
' Initialize and add any number of sections to the document here.
' Assign the document to the RichTextBoxAdv instance.
richTextBoxAdv.Document = document
```

UI Commands for accessing table

The following code example illustrates how to bind the Button UI Command for inserting a table.

XML

```
<!-- Inserts the table with default size of one row and two columns -->
<Button Content="Insert Table"
Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertTableCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Inserts the table with the size of two rows and three columns -->
<Button Content="Insert Table"
Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertTableCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}"
CommandParameter="2,3"/>
```

C#

```
// InsertTableCommand accepting string parameter.
SfRichTextBoxAdv.InsertTableCommand.Execute("2,3", richTextBoxAdv);
// InsertTableCommand accepting int[] with row, column size as parameter
SfRichTextBoxAdv.InsertTableCommand.Execute(new int[] { 2, 3 },
richTextBoxAdv);
```

VB.NET

```
' InsertTableCommand accepting string parameter.
SfRichTextBoxAdv.InsertTableCommand.Execute("2,3", richTextBoxAdv)
' InsertTableCommand accepting int[] with row, column size as parameter
SfRichTextBoxAdv.InsertTableCommand.Execute(New Integer() { 2, 3 },
richTextBoxAdv)
```

The following code example illustrates how to bind the Button UI Command for inserting rows and columns.

XML

```
<!-- Inserts one row above to the current row -->
<!-- Command parameter can be either Above or Below -->
<Button Content="Insert Row"
Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertRowCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}"
CommandParameter="Above"/>
<!-- Inserts one column to the right of current column -->
<!-- Command parameter can be either Left or Right -->
<Button Content="Insert Column"
Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertColumnCommand"
```

```
CommandTarget="{Binding ElementName=richTextBoxAdv}"
CommandParameter="Right"/>
```

The following code example illustrates how to bind the Button UI Command for selecting cell, row, column and table.

XML

```
<!--Selects the Cell-->
<Button Content="Select Cell"
Command="RichTextBoxAdv:SfRichTextBoxAdv.SelectCellCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!--Selects the Column-->
<Button Content="Select Column"
Command="RichTextBoxAdv:SfRichTextBoxAdv.SelectColumnCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!--Selects the Row-->
<Button Content="Select Row"
Command="RichTextBoxAdv:SfRichTextBoxAdv.SelectRowCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!--Selects the Table-->
<Button Content="Select Table"
Command="RichTextBoxAdv:SfRichTextBoxAdv.SelectTableCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
```

The following code example illustrates how to bind the Button UI Command for merging selected cells.

XML

```
<!-- Merges the selected cells -->
<Button Content="Merge Cells"
Command="RichTextBoxAdv:SfRichTextBoxAdv.MergeSelectedCellsCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
```

The following code example illustrates how to bind the Button UI Command to change content alignment of the selected cells.

XML

```
<!--Change cell content alignment with command parameter as comma
separated(vertical alignment and text alignment)-->
<Button Content="Cell Content Alignment"
Command="RichTextBoxAdv:SfRichTextBoxAdv.CellContentAlignmentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}"
CommandParameter="Top,Left" />
<!--or-->
<!--Change cell content alignment with command parameter single sting
(vertical alignment and text alignment)-->
<Button Content="Cell Content Alignment"
Command="RichTextBoxAdv:SfRichTextBoxAdv.CellContentAlignmentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}"
CommandParameter="CenterRight"/>
<!--or-->
<!--Change cell content alignment with command parameter as string
array(vertical alignment and text alignment string order respectively)-->
```

```
<Button Content="Cell Content Alignment"
Command="RichTextBoxAdv:SfRichTextBoxAdv.CellContentAlignmentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}">
<Button.CommandParameter>
<x:Array Type="sys:String"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:sys="clr-
namespace:System;assembly=mscorlib">
<sys:String>Bottom</sys:String>
<sys:String>Left</sys:String>
</x:Array>
</Button.CommandParameter>
</Button>
```

The following code example illustrates how to Button UI Command for deleting a row, deleting a column and deleting an entire table.

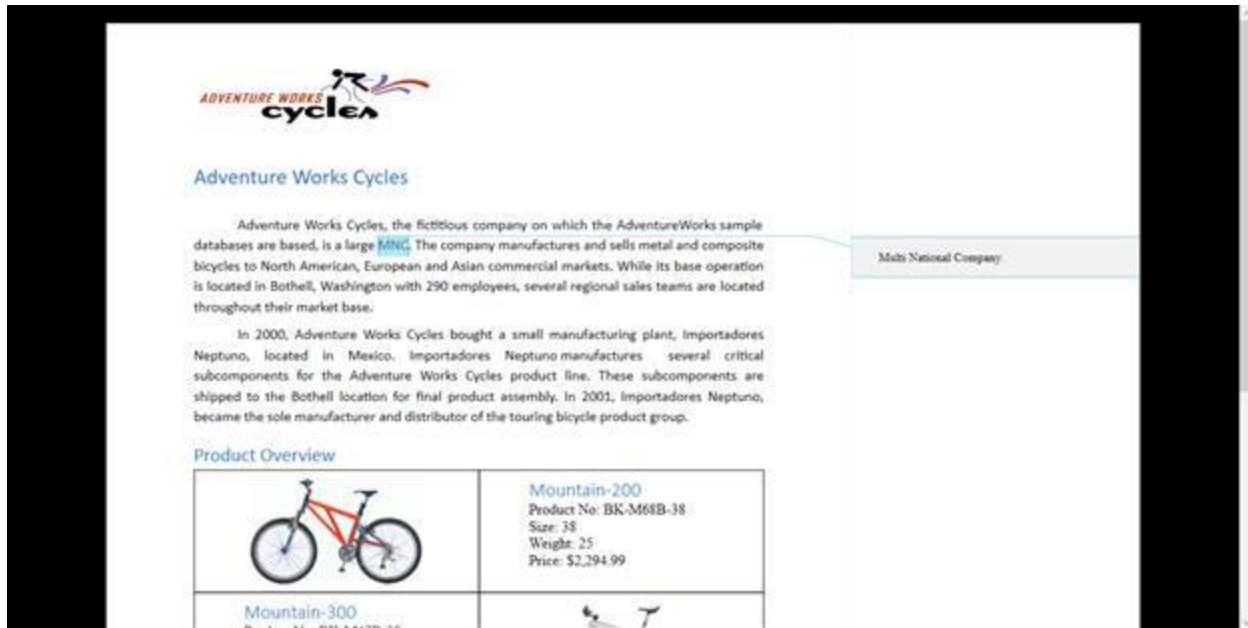
XML

```
<!-- Deletes the column -->
<Button Content="Delete Column"
Command="RichTextBoxAdv:SfRichTextBoxAdv.DeleteColumnCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Deletes the row -->
<Button Content="Delete Row"
Command="RichTextBoxAdv:SfRichTextBoxAdv.DeleteRowCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Deletes the table -->
<Button Content="Delete Table"
Command="RichTextBoxAdv:SfRichTextBoxAdv.DeleteTableCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
```

Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Comment in WPF RichTextBox (SfRichTextBoxAdv)

A Comment is a note or annotation that an author or reviewer can add to the document. The SfRichTextBoxAdv control supports viewing and editing the comments in the document. It renders the comments present in the document in review pane, similar to the Microsoft Word.



Note: Currently, the SfRichTextBoxAdv shows review pane only with Pages layout type.

UI Commands for accessing comment

The following operations can be performed through command binding in SfRichTextBoxAdv control:

- Insert a new comment.
- Delete an existing comment or delete all the existing comments.
- Navigate to the next comment.
- Navigate to the previous comment.
- Show/Hide review pane.

The following code example demonstrates how to bind commands for accessing comment in SfRichTextBoxAdv document.

XML

```
<!-- Binds button to the ShowCommentsCommand -->
<Button Content="Show Comments"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowCommentsCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the NewCommentCommand -->
<Button Content="New Comment"
Command="RichTextBoxAdv:SfRichTextBoxAdv.NewCommentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the DeleteCommentCommand -->
<Button Content="Delete Comment"
Command="RichTextBoxAdv:SfRichTextBoxAdv.DeleteCommentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the DeleteAllCommentsCommand -->
<Button Content="Delete All Comments"
Command="RichTextBoxAdv:SfRichTextBoxAdv.DeleteAllCommentsCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the PreviousCommentCommand -->
```

```

<Button Content="Previous Comment"
Command="RichTextBoxAdv:SfRichTextBoxAdv.PreviousCommentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the NextCommentCommand -->
<Button Content="Next Comment"
Command="RichTextBoxAdv:SfRichTextBoxAdv.NextCommentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />

```

Customizing comment visual style

The SfRichTextBoxAdv provides event support to notify whenever a comment is added to the document. With the help of it, you can customize the visual style for the comment. You can also set the author and initial of the comment.

The following code example demonstrates how to customize comment visual style using event.

C#

```

// Hooks the CommentAdding event of RichTextBoxAdv.
richTextBoxAdv.CommentAdding += RichTextBoxAdv_CommentAdding;
// Unhooks the CommentAdding event of RichTextBoxAdv.
richTextBoxAdv.CommentAdding -= RichTextBoxAdv_CommentAdding;
// Handles the CommentAdding event of the richTextBoxAdv control.
private void RichTextBoxAdv_CommentAdding(object obj, CommentAddingEventArgs args)
{
    if (!isFileLoading)
    {
        //Defines the author and initial for the comment.
        args.Comment.Author = "Peter";
        args.Comment.Initial = "Franken";
    }
    // Defines the background brush for the comment.
    argsVisualStyle.BackgroundBrush = new SolidColorBrush(Color.FromArgb(0xff, 0xff, 0xa8, 0xa8));
    // Defines the border brush for the comment.
    argsVisualStyle.BorderBrush = new SolidColorBrush(Color.FromArgb(0xff, 0xFF, 0x01, 0x01));
    // Defines the highlight color for the commented content.
    argsVisualStyle.HighlightColor = Color.FromArgb(0xff, 0xFF, 0xa8, 0x8);
}

```

VB.NET

```

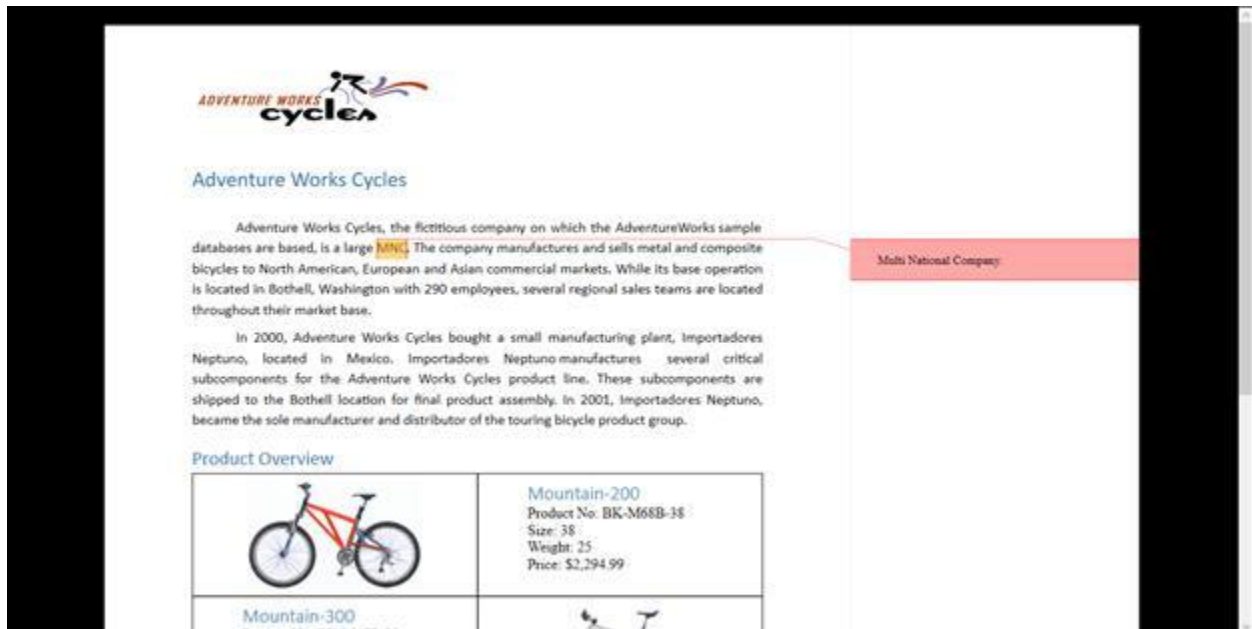
' Hooks the CommentAdding event of RichTextBoxAdv.
AddHandler richTextBoxAdv.CommentAdding, AddressOf
RichTextBoxAdv_CommentAdding
' Unhooks the CommentAdding event of RichTextBoxAdv.
RemoveHandler richTextBoxAdv.CommentAdding, AddressOf
RichTextBoxAdv_CommentAdding
' Handles the CommentAdding event of the richTextBoxAdv control.
Private Sub RichTextBoxAdv_CommentAdding(obj As Object, args As
CommentAddingEventArgs)
If Not isFileLoading Then
'Defines the author and initial for the comment.
args.Comment.Author = "Peter"

```

```

args.Comment.Initial = "Franken"
End If
' Defines the background brush for the comment.
argsVisualStyle.BackgroundBrush = New SolidColorBrush(Color.FromArgb(&Hff,
&Hff, &Ha8, &Ha8))
' Defines the border brush for the comment.
argsVisualStyle.BorderBrush = New SolidColorBrush(Color.FromArgb(&Hff,
&Hff, &H1, &H1))
' Defines the highlight color for the commented content.
argsVisualStyle.HighlightColor = Color.FromArgb(&Hff, &Hff, &Ha8, &H8)
End Sub

```



Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Document Styles in WPF RichTextBox (SfRichTextBoxAdv)

A style is a predefined set of table, numbering, paragraph, and character formatting properties that can be applied to regions within a document.

In RichTextBoxAdv, styles are created and added to a document programmatically or using the built-in Styles dialog.

A style in a document should have the following properties:

Name

Name of the style. All styles in a document have a unique name, which is used as an identifier when applying the style.

Type

Specifies the document elements that the style will target. For example, paragraph or character.

Next

Specifies the style that will be automatically applied to a new paragraph, which is created following a paragraph with the parent paragraph style applied.

Link

Provides a relation between the paragraph and character style.

CharacterFormat

Specifies the properties of paragraph and character style.

ParagraphFormat

Specifies the properties of paragraph style.

BasedOn

Specifies that the current style inherits the style set to this property. This is how hierarchical styles are defined. It can be optional.

Note: The style type should match the inherited style type. For example, it is not possible to have a character style inherit a paragraph style.

Default style

The default style for span and paragraph properties is normal. It internally inherits the default style of the document loaded or RichTextBoxAdv control.

Style hierarchy

Each style initially checks its local value for the property that is being evaluated and turns to the style it is based on. If no local value is found, it turns to its default style.

Style inheritance of different styles are listed as follows:

Character style

Character styles are based only on other character styles, and character properties are inherited from the base character style.

Paragraph style

Paragraph styles are based on other paragraph styles or on linked styles.

When a paragraph style is based on another paragraph style, the inheritance of the properties is as follows:

- Paragraph properties are inherited from the base paragraph style.
- Span properties are inherited from the base paragraph style.

When a paragraph style is based on a linked style, the inheritance of the properties is as follows:

- Paragraph properties are inherited from the paragraph style part in its base linked style.
- Span properties are inherited from the span style part in its base linked style.

Linked style

Linked styles are composite styles and their components are paragraph and character styles with link between them. To apply paragraph properties, take the properties from the linked paragraph style. Similarly, to apply character properties, take the properties from linked character style.

Linked styles are based on other linked styles or on paragraph styles.

When a linked style is based on a paragraph style, the hierarchy of the properties is as follows:

- Paragraph properties are inherited from its base paragraph style.
- Character properties are inherited from its base character style.

When a linked style is based on another linked style, the hierarchy of the properties is as follows:

- Paragraph properties are inherited from the paragraph style part in its base linked style.
- Span properties are inherited from the character style part in its base linked style.

SfRichTextBoxAdv provides the following functionalities related with styles:

- Create new style
- Modify an existing style
- Apply style
- Clear formatting

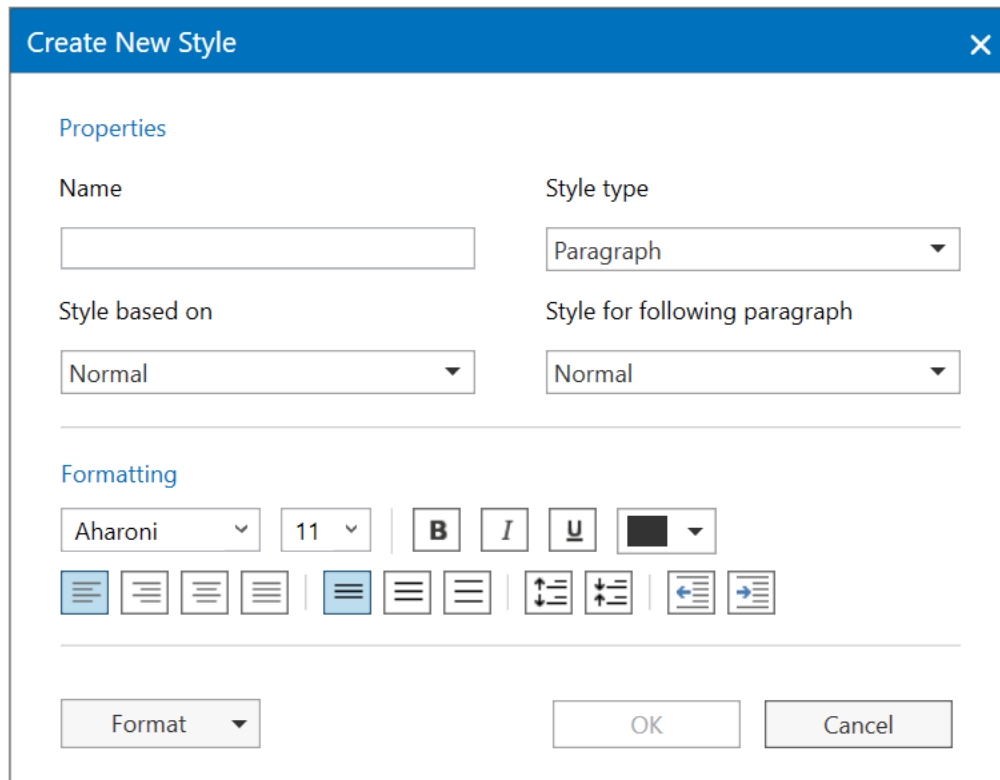
Create new style

New styles are created and added to the style collection of the document. Here, you can create character, paragraph and linked type styles.

The following code example explains how to create new style dialog through command binding.

XML

```
<Button Content="Create style"
Command="Syncfusion:SfRichTextBoxAdv.ShowStyleDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" ></Button>
```

The image shows a 'Create New Style' dialog box with a blue header bar containing a close button. The dialog is divided into two main sections: 'Properties' and 'Formatting'. In the 'Properties' section, there is a 'Name' text box, a 'Style type' dropdown menu set to 'Paragraph', a 'Style based on' dropdown menu set to 'Normal', and a 'Style for following paragraph' dropdown menu set to 'Normal'. The 'Formatting' section contains a row of controls: a font family dropdown set to 'Aharoni', a font size dropdown set to '11', and buttons for bold (B), italic (I), underline (U), and a color selection box. Below these are two rows of alignment and indentation icons. At the bottom of the dialog are three buttons: 'Format' with a dropdown arrow, 'OK', and 'Cancel'.

Modify an existing style

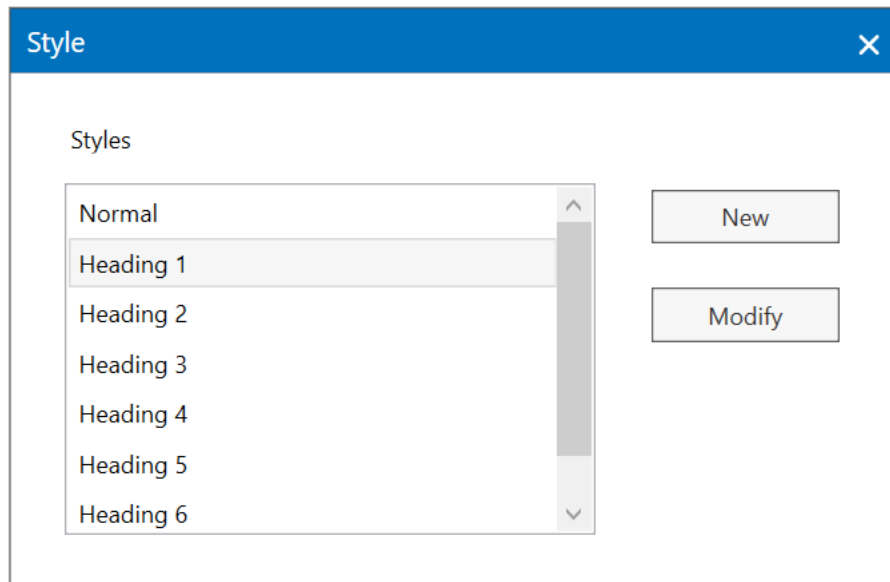
You can modify a style directly using the `ShowStylesDialogCommand` in `SfRichTextBoxAdv`.

The following code example explains how to modify the style dialog through command binding.

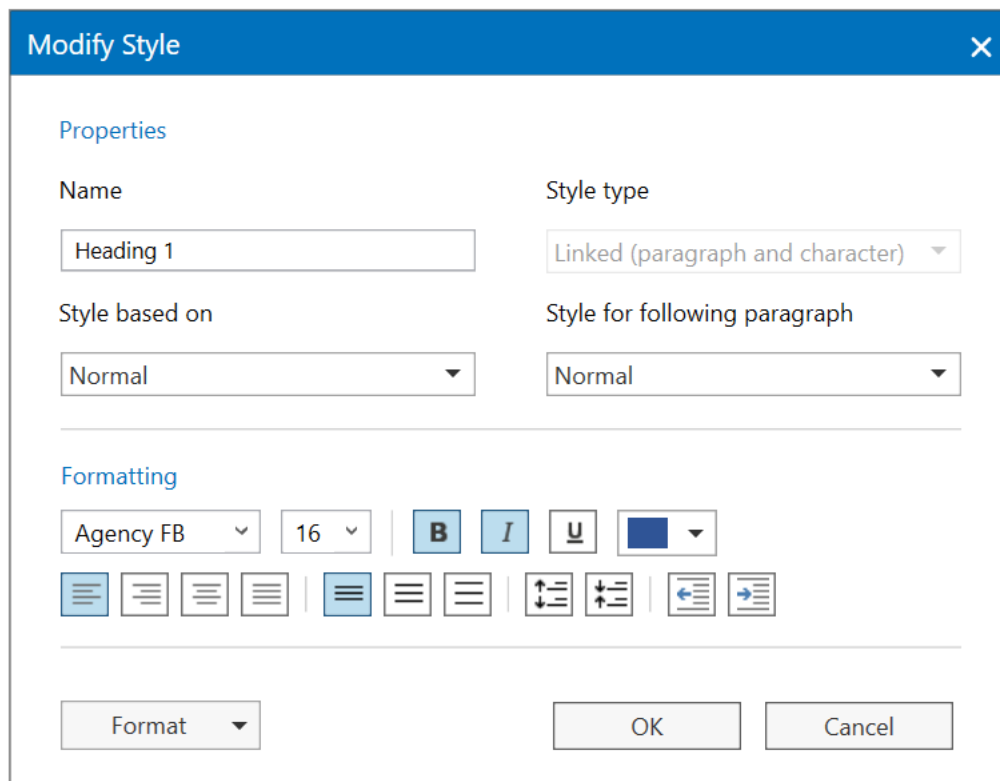
XML

```
<Button Content="Modify style"
Command="Syncfusion:SfRichTextBoxAdv.ShowStylesDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}"></Button>
```

From the styles list select a style you want to modify.



In the Formatting section, make any formatting changes you want, such as font style, size, or color, alignment, line spacing, or indentation.



Apply style

The styles are applied using the `ApplyStyleCommand` in `SfRichTextBoxAdv`. The parameter should be passed is the Name of the Style.

The styles of the Character type are applied to the currently selected part of the document. If there is no selection, the values that will be applied to the word at caret position. The styles of Paragraph type follow the same logic and are applied to all paragraphs in the selection or the current paragraph.

When there is no selection, styles of Linked type will change the values of the paragraph and apply both the Paragraph and Character properties. When there is selection, Linked Style changes only the character properties of the selected text.

The following code example explains how to apply style through command binding.

XML

```
<Button Content="Apply style"
Command="Syncfusion:SfRichTextBoxAdv.ApplyStyleCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}"
CommandParameter="Heading 1"></Button>
```

Clear formatting

It will remove all the formatting from the selection leaving only the normal unformatted text. But when there is no selection, it will remove only the formatting inherited from style.

The following code example explains how to clear the formatting of text through command binding.

XML

```
<Button Content="Clear formatting"
Command="Syncfusion:SfRichTextBoxAdv.ClearFormattingCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}"></Button>
```

Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Import and Export in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv allows you to import/export word documents (.docx, .doc), rich text format documents (.rtf), HTML documents (.htm, .html), XAML documents (.xaml) and text documents (.txt).

The following sample code demonstrates how to import contents from a file into SfRichTextBoxAdv.

C#

```
// Imports the document.
void ImportDocument ()
{
    // Initializes the open file dialog.
    OpenFileDialog openFileDialog = new OpenFileDialog()
    {
        Filter = "All supported files (*.docx;*.doc;*.htm;*.html;*.rtf;*.txt;*.xaml)|*.docx;*.doc;*.htm;*.html;*.rtf;*.txt;*.xaml|Word Document (*.docx)|*.docx|Word 97 - 2003 Document (*.doc)|*.doc|Web Page (*.htm;*.html)|*.htm;*.html|Rich Text File (*.rtf)|*.rtf|Text File (*.txt)|*.txt|Xaml File (*.xaml)|*.xaml",
        FilterIndex = 1,
        Multiselect = false
    };
    if ((bool)openFileDialog.ShowDialog())
    {
        // Loads the file into RichTextBoxAdv.
        richTextBoxAdv.Load(openFileDialog.FileName);
        // Sets the File name as Document Title.
    }
}
```

```
richTextBoxAdv.DocumentTitle =
openFileDialog.FileName.Remove(openFileDialog.FileName.LastIndexOf("."));
}
}
```

VB.NET

```
' Imports the document.
Private Sub ImportDocument()
' Initializes the open file dialog.
Dim openFileDialog As New OpenFileDialog() With { _
Key .Filter = "All supported files
(*.docx;*.doc;*.htm;*.html;*.rtf;*.txt;*.xaml)|*.docx;*.doc;*.htm;*.html;*.r
tf;*.txt;*.xaml|Word Document (*.docx)|*.docx|Word 97 - 2003 Document
(*.doc)|*.doc|Web Page (*.htm;*.html)|*.htm;*.html|Rich Text File
(*.rtf)|*.rtf|Text File (*.txt)|*.txt|Xaml File (*.xaml)|*.xaml", _
Key .FilterIndex = 1, _
Key .Multiselect = False _
}
If CBool(openFileDialog.ShowDialog()) Then
' Loads the file into RichTextBoxAdv.
richTextBoxAdv.Load(openFileDialog.FileName)
' Sets the File name as Document Title.
richTextBoxAdv.DocumentTitle =
openFileDialog.FileName.Remove(openFileDialog.FileName.LastIndexOf("."))
End If
End Sub
```

The following code example demonstrates how to export SfRichTextBoxAdv contents into a file.

C#

```
// Exports the document.
void ExportDocument ()
{
// Initializes the file save picker.
SaveFileDialog saveFileDialog = new SaveFileDialog()
{
Filter = "Word Document (*.docx)|*.docx|Word 97 - 2003 Document
(*.doc)|*.doc|Web Page (*.htm;*.html)|*.htm;*.html|Rich Text File
(*.rtf)|*.rtf|Text File (*.txt)|*.txt|Xaml File (*.xaml)|*.xaml",
FilterIndex = 1
};
if ((bool)saveFileDialog.ShowDialog())
{
// Saves the document content into a file.
richTextBoxAdv.Save(saveFileDialog.FileName);
}
}
```

VB.NET

```
' Exports the document.
Private Sub ExportDocument()
' Initializes the file save picker.
```

```

Dim saveFileDialog As New SaveFileDialog() With { _
Key .Filter = "Word Document (*.docx)|*.docx|Word 97 - 2003 Document (*.doc)|*.doc|Web Page (*.htm,*.html)|*.htm;*.html|Rich Text File (*.rtf)|*.rtf|Text File (*.txt)|*.txt|Xaml File (*.xaml)|*.xaml", _
Key .FilterIndex = 1 _
}
If CBool(saveFileDialog.ShowDialog()) Then
' Saves the document content into a file.
richTextBoxAdv.Save(saveFileDialog.FileName)
End If
End Sub

```

Note: When the SfRichTextBoxAdv control encounters an unsupported element, it does not render the element, instead, it continues to the next supported element and render it. Examples of unsupported elements are AutoShapes, watermarks, charts, SmartArt, WordArt, equations, document structure tags, styles, wrapping styles, fields other than hyperlinks, absolutely positioned tables, and absolutely positioned images.

UI Commands for importing/exporting documents

The following code example demonstrates how to bind commands for performing importing and exporting documents.

XML

```

<!-- Binds button to the OpenDocumentCommand -->
<Button Content="Open"
Command="RichTextBoxAdv:SfRichTextBoxAdv.OpenDocumentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the SaveDocumentCommand -->
<Button Content="Save"
Command="RichTextBoxAdv:SfRichTextBoxAdv.SaveDocumentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />

```

Note: In order to perform import/export documents, the standard keyboard shortcuts such as CTRL + O, CTRL + S can also be used.

Asynchronous import settings

Show or hide the loading page number

The SfRichTextBoxAdv control shows the current loading page number by default at the bottom right corner of the control while loading the document asynchronously. You can hide this loading page number by using the ShowPageNumber property of LoadAsyncSettings class.

The following code example demonstrates how to hide the loading page number in SfRichTextBoxAdv control.

XML

```

<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv">
<RichTextBoxAdv:SfRichTextBoxAdv.LoadAsyncSettings>
<RichTextBoxAdv:LoadAsyncSettings ShowPageNumber="False"/>
</RichTextBoxAdv:SfRichTextBoxAdv.LoadAsyncSettings>
</RichTextBoxAdv:SfRichTextBoxAdv>

```

C#

```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
////Hides the loading page number.
richTextBoxAdv.LoadAsyncSettings.ShowPageNumber = false;
```

VB.NET

```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
' Hides the loading page number.
richTextBoxAdv.LoadAsyncSettings.ShowPageNumber = false
```

Note: This API is supported starting from release version v17.4.0.X.

Events to notify document starts and completes loading and saving

SfRichTextBoxAdv control also provides below events to notify document starts and completes loading and saving.

Events Table

Event	Description
DocumentChanging	This event is triggered when the document starts loading.
DocumentChanged	This event is triggered after the document is successfully loaded.
DocumentSaving	This event is triggered when the document starts saving.
DocumentSaved	This event is triggered after the document is successfully saved.

Note: This API is supported starting from release version v18.2.0.X.

You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Printing Contents in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv supports API to print the rich text content rendered as pages using the print dialog.

The following sample code demonstrates how to print the document content as pages.

C#

```
// Displays the Print Dialog to perform printing of document content as pages.
richTextBoxAdv.PrintDocument();
```

VB.NET

```
' Displays the Print Dialog to perform printing of document content as pages.
richTextBoxAdv.PrintDocument()
```

The SfRichTextBoxAdv also supports event to notify whenever the printing operation is completed. The following code example demonstrates how to handle for the print completed event.

C#

```
// Hooks the print completed event.
richTextBoxAdv.PrintCompleted += RichTextBoxAdv_PrintCompleted;
// Called whenever the print completed event is fired.
private void RichTextBoxAdv_PrintCompleted(object obj,
PrintCompletedEventArgs args)
{
    // Handle your code here.
}
// Unhooks the print completed event.
richTextBoxAdv.PrintCompleted -= RichTextBoxAdv_PrintCompleted;
```

VB.NET

```
' Hooks the print completed event.
AddHandler richTextBoxAdv.PrintCompleted, AddressOf
RichTextBoxAdv_PrintCompleted
' Called whenever the print completed event is fired.
Private Sub RichTextBoxAdv_PrintCompleted(obj As Object, args As
PrintCompletedEventArgs)
' Handle your code here.
End Sub
' Unhooks the print completed event.
RemoveHandler richTextBoxAdv.PrintCompleted, AddressOf
RichTextBoxAdv_PrintCompleted
```

UI Command for printing

The following code example demonstrates how to bind UI Command to invoke printing in SfRichTextBoxAdv.

XML

```
<!-- Binds button to the PrintDocumentCommand -->
<Button Content="Print"
Command="RichTextBoxAdv:SfRichTextBoxAdv.PrintDocumentCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
```

Note: In order to invoke printing, the standard keyboard shortcut CTRL + P can also be used.

You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Mini Toolbar in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv supports built-in mini toolbar to provide rich text formatting options such as Bold, Italic etc. The following screenshot shows built-in mini toolbar of SfRichTextBoxAdv control.



Enable/Disable Mini Toolbar

In SfRichTextBoxAdv, the built-in mini toolbar is enabled by default. It is possible to enable/disable the built-in mini toolbar. The following code example demonstrates how to disable the built-in mini toolbar in SfRichTextBoxAdv.

XML

```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv"
EnableMiniToolBar="False" xmlns:RichTextBoxAdv="clr-
namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfR
ichTextBoxAdv.Wpf" />
```

C#

```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
//Disables the built-in mini tool bar in SfRichTextBoxAdv.
richTextBoxAdv.EnableMiniToolBar = false;
```

VB.NET

```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
'Disables the built-in mini tool bar in SfRichTextBoxAdv.
richTextBoxAdv.EnableMiniToolBar = False
```

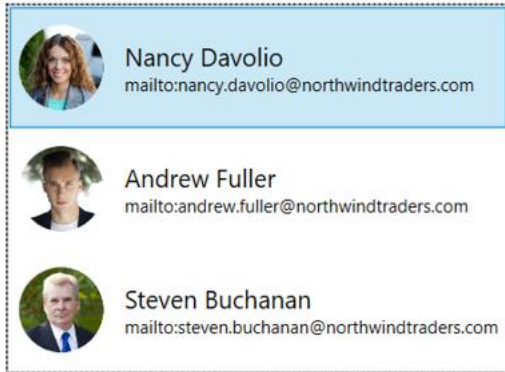
Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Automatic Suggestion in WPF RichTextBox (SfRichTextBoxAdv)

Automatic Suggestion functionality for using @mentions

[WPF RichTextBox](#) control shows an inline dropdown with a list of suggested names while type the mention character (@ symbol). The list of names will filter as you type more letters. You can use up or down arrow key to move selection and Tab or Enter key to insert selected item in keyboard or use mouse to click any option in the list. The selected item from the suggestion list will be inserted as hyperlink with the display text and its respective link.

Hi @an



The following sample code demonstrates how to use @mentions in RichTextBox.

XML

```
<Window.Resources>
<x:Array Type="{x:Type RichTextBoxAdv:NameSuggestionItem}"
x:Key="suggestionItems">
<RichTextBoxAdv:NameSuggestionItem Name = "Nancy Davolio"
Link="mailto:nancy.davolio@northwindtraders.com"
ImageSource="/Assets/People_Circle0.png" />
<RichTextBoxAdv:NameSuggestionItem Name = "Andrew Fuller"
Link="mailto:andrew.fuller@northwindtraders.com"
ImageSource="/Assets/People_Circle5.png"/>
<RichTextBoxAdv:NameSuggestionItem Name = "Steven Buchanan"
Link="mailto:steven.buchanan@northwindtraders.com"
ImageSource="/Assets/People_Circle18.png"/>
</x:Array>
</Window.Resources>
<Grid>
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextboxadv"
LayoutType="Continuous">
<RichTextBoxAdv:SfRichTextBoxAdv.SuggestionSettings>
<RichTextBoxAdv:SuggestionSettings>
<RichTextBoxAdv:SuggestionSettings.SuggestionProviders>
<RichTextBoxAdv:NameSuggestionProvider ItemsSource="{StaticResource
suggestionItems}">
</RichTextBoxAdv:NameSuggestionProvider>
</RichTextBoxAdv:SuggestionSettings.SuggestionProviders>
</RichTextBoxAdv:SuggestionSettings>
</RichTextBoxAdv:SfRichTextBoxAdv.SuggestionSettings>
</RichTextBoxAdv:SfRichTextBoxAdv>
</Grid>
```

C#

```
ISuggestionProvider suggestionProvider = new NameSuggestionProvider();
List<NameSuggestionItem> suggestionItems = new List<NameSuggestionItem>();
NameSuggestionItem suggestionItem = new NameSuggestionItem();
suggestionItem.Name = "Nancy Davolio";
suggestionItem.Link = "mailto:nancy.davolio@northwindtraders.com";
BitmapImage bitmapImage = new BitmapImage(new Uri(new
DirectoryInfo(@"..\..\Assets\People_Circle0.png").FullName));
suggestionItem.ImageSource = bitmapImage;
```

```

suggestionItems.Add(suggestionItem);
suggestionItem = new NameSuggestionItem();
suggestionItem.Name = "Andrew Fuller";
suggestionItem.Link = "mailto:andrew.fuller@northwindtraders.com";
bitmapImage = new BitmapImage(new Uri(new
DirectoryInfo(@"..\..\Assets\People_Circle5.png").FullName));
suggestionItem.ImageSource = bitmapImage;
suggestionItems.Add(suggestionItem);
suggestionItem = new NameSuggestionItem();
suggestionItem.Name = "Steven Buchanan";
suggestionItem.Link = "mailto:steven.buchanan@northwindtraders.com";
bitmapImage = new BitmapImage(new Uri(new
DirectoryInfo(@"..\..\Assets\People_Circle18.png").FullName));
suggestionItem.ImageSource = bitmapImage;
suggestionItems.Add(suggestionItem);
(suggestionProvider as NameSuggestionProvider).ItemsSource =
suggestionItems;
richTextboxadv.SuggestionSettings.SuggestionProviders.Add(suggestionProvider
);

```

VB.NET

```

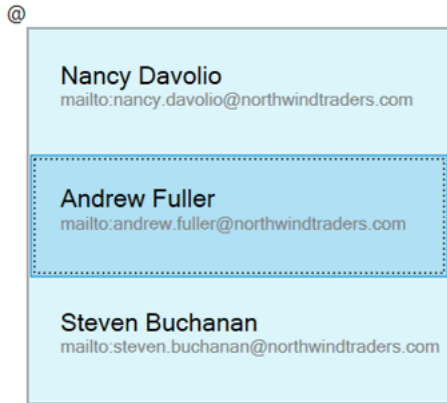
Dim suggestionProvider As ISuggestionProvider = New NameSuggestionProvider()
Dim suggestionItems As List<NameSuggestionItem> = New
List<NameSuggestionItem>()
Dim suggestionItem As NameSuggestionItem = New NameSuggestionItem()
suggestionItem.Name = "Nancy Davolio"
suggestionItem.Link = "mailto:nancy.davolio@northwindtraders.com"
Dim bitmapImage As BitmapImage = New BitmapImage(New Uri(New
DirectoryInfo(@"..\..\Assets\People_Circle0.png").FullName))
suggestionItem.ImageSource = bitmapImage
suggestionItems.Add(suggestionItem)
suggestionItem = New NameSuggestionItem()
suggestionItem.Name = "Andrew Fuller"
suggestionItem.Link = "mailto:andrew.fuller@northwindtraders.com"
bitmapImage = New BitmapImage(New Uri(New
DirectoryInfo(@"..\..\Assets\People_Circle5.png").FullName))
suggestionItem.ImageSource = bitmapImage
suggestionItems.Add(suggestionItem)
suggestionItem = New NameSuggestionItem()
suggestionItem.Name = "Steven Buchanan"
suggestionItem.Link = "mailto:steven.buchanan@northwindtraders.com"
bitmapImage = New BitmapImage(New Uri(New
DirectoryInfo(@"..\..\Assets\People_Circle18.png").FullName))
suggestionItem.ImageSource = bitmapImage
suggestionItems.Add(suggestionItem)
TryCast(suggestionProvider, NameSuggestionProvider).ItemsSource =
suggestionItems
richTextboxadv.SuggestionSettings.SuggestionProviders.Add(suggestionProvider
)

```

[View example in GitHub](#)

Customize the SuggestionBox ItemTemplate and Style

By default, the drop-down window lists the filtered items as an image, display text and link. If you want to remove the image or link. You can write your own item Template.



The following sample code demonstrates how to modify the suggestion box item template and style.

XML

```
<Window.Resources>
<x:Array Type="{x:Type RichTextBoxAdv:NameSuggestionItem}"
x:Key="suggestionItems">
<RichTextBoxAdv:NameSuggestionItem Name = "Nancy Davolio"
Link="mailto:nancy.davolio@northwindtraders.com" />
<RichTextBoxAdv:NameSuggestionItem Name = "Andrew Fuller"
Link="mailto:andrew.fuller@northwindtraders.com" />
<RichTextBoxAdv:NameSuggestionItem Name = "Steven Buchanan"
Link="mailto:steven.buchanan@northwindtraders.com" />
</x:Array>
<Style x:Key="SuggestionBoxStyle" TargetType="ListBox">
<Setter Property="MinWidth" Value="300" />
<Setter Property="MinHeight" Value="250" />
<Setter Property="Background" Value="#FFDBF5FB"/>
<Setter Property="ItemTemplate">
<Setter.Value>
<DataTemplate DataType="local:NameSuggestionItem">
<StackPanel Orientation="Vertical" Height="50" VerticalAlignment="Center"
Margin="12,15,0,0">
<TextBlock Text="{Binding Name}" FontFamily="microsoft sans serif"
FontSize="14" />
<TextBlock Text="{Binding Link}" FontFamily="microsoft sans serif"
Foreground="Gray" FontSize="10" />
</StackPanel>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
<Grid>
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextboxadv"
LayoutType="Continuous">
<RichTextBoxAdv:SfRichTextBoxAdv.SuggestionSettings>
<RichTextBoxAdv:SuggestionSettings>
<RichTextBoxAdv:SuggestionSettings.SuggestionProviders>
```

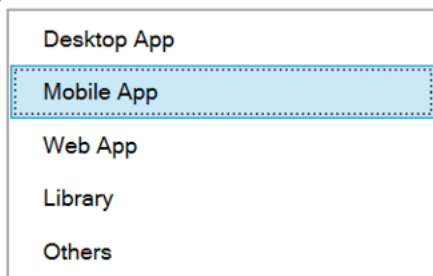
```
<RichTextBoxAdv:NameSuggestionProvider ItemsSource="{StaticResource
suggestionItems}"
SuggestionBoxStyle="{StaticResource SuggestionBoxStyle}">
</RichTextBoxAdv:NameSuggestionProvider>
</RichTextBoxAdv:SuggestionSettings.SuggestionProviders>
</RichTextBoxAdv:SuggestionSettings>
</RichTextBoxAdv:SfRichTextBoxAdv.SuggestionSettings>
</RichTextBoxAdv:SfRichTextBoxAdv>
</Grid>
```

Custom mention character

Any character can be used as mention character, default value is @.

#Desktop App

#



The following sample code demonstrates how to use '#' as mention character.

XML

```
<Grid>
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextboxadv"
LayoutType="Continuous">
<RichTextBoxAdv:SfRichTextBoxAdv.SuggestionSettings>
<RichTextBoxAdv:SuggestionSettings>
<RichTextBoxAdv:SuggestionSettings.SuggestionProviders>
<RichTextBoxAdv:NameSuggestionProvider MentionCharacter="#"
ItemsSource="{StaticResource suggestionItems}">
</RichTextBoxAdv:NameSuggestionProvider>
</RichTextBoxAdv:SuggestionSettings.SuggestionProviders>
</RichTextBoxAdv:SuggestionSettings>
</RichTextBoxAdv:SfRichTextBoxAdv.SuggestionSettings>
</RichTextBoxAdv:SfRichTextBoxAdv>
</Grid>
```

C#

```
ISuggestionProvider suggestionProvider = new NameSuggestionProvider();
suggestionProvider.MentionCharacter = '#';
richTextboxadv.SuggestionSettings.SuggestionProviders.Add(suggestionProvider
);
```

Multiple Suggestion provider

Two or more suggestion providers can be used at a time but, each suggestion provider should have different mention character. And each suggestion provider can have different item source and suggestion box style.



The following sample code demonstrates how to use two suggestion providers. Here we have used '@' and '#' as mention characters.

XML

```
<Window.Resources>
<Style x:Key="SuggestionBoxStyle" TargetType="ListBox">
<Setter Property="MinWidth" Value="300" />
<Setter Property="MinHeight" Value="250" />
<Setter Property="ItemTemplate">
<Setter.Value>
<DataTemplate DataType="local:NameSuggestionItem">
<StackPanel Orientation="Vertical" Height="50" Width="200"
VerticalAlignment="Center" Margin="12,15,0,0">
<TextBlock Text="{Binding Name}" FontFamily="microsoft sans serif"
FontSize="14" />
<TextBlock Text="{Binding Link}" FontFamily="microsoft sans serif"
Foreground="Gray" FontSize="10" />
</StackPanel>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
<x:Array Type="{x:Type RichTextBoxAdv:NameSuggestionItem}"
x:Key="suggestionItems">
<RichTextBoxAdv:NameSuggestionItem Name = "Nancy Davolio"
Link="mailto:nancy.davolio@northwindtraders.com"
ImageSource="/Assets/People_Circle0.png" />
<RichTextBoxAdv:NameSuggestionItem Name = "Andrew Fuller"
Link="mailto:andrew.fuller@northwindtraders.com"
ImageSource="/Assets/People_Circle5.png"/>
<RichTextBoxAdv:NameSuggestionItem Name = "Steven Buchanan"
Link="mailto:steven.buchanan@northwindtraders.com"
ImageSource="/Assets/People_Circle18.png"/>
</x:Array>
<x:Array Type="{x:Type RichTextBoxAdv:NameSuggestionItem}"
x:Key="suggestionItems01">
<RichTextBoxAdv:NameSuggestionItem Name = "Desktop App" Link="10 queries"
/>
<RichTextBoxAdv:NameSuggestionItem Name = "Mobile App" Link="13 queries" />
<RichTextBoxAdv:NameSuggestionItem Name = "Web App" Link="15 queries"/>
</x:Array>
</Window.Resources>
<Grid>
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextboxadv"
LayoutType="Continuous">
<RichTextBoxAdv:SfRichTextBoxAdv.SuggestionSettings>
```

```

<RichTextBoxAdv:SuggestionSettings>
<RichTextBoxAdv:SuggestionSettings.SuggestionProviders>
<RichTextBoxAdv:NameSuggestionProvider
ItemsSource="{StaticResource suggestionItems}">
</RichTextBoxAdv:NameSuggestionProvider>
<RichTextBoxAdv:NameSuggestionProvider MentionCharacter="#"
ItemsSource="{StaticResource suggestionItems01}"
SuggestionBoxStyle="{StaticResource SuggestionBoxStyle}">
</RichTextBoxAdv:NameSuggestionProvider>
</RichTextBoxAdv:SuggestionSettings.SuggestionProviders>
</RichTextBoxAdv:SuggestionSettings>
</RichTextBoxAdv:SfRichTextBoxAdv.SuggestionSettings>
</RichTextBoxAdv:SfRichTextBoxAdv>
</Grid>

```

C#

```

ISuggestionProvider suggestionProvider = new NameSuggestionProvider();
List<NameSuggestionItem> suggestionItems = new List<NameSuggestionItem>();
NameSuggestionItem suggestionItem1 = new NameSuggestionItem();
suggestionItem1.Name = "Nancy Davolio";
suggestionItem1.Link = "mailto:nancy.davolio@northwindtraders.com";
BitmapImage bitmapImage = new BitmapImage(new Uri(new
DirectoryInfo(@"..\..\Assets\People_Circle0.png").FullName));
suggestionItem1.ImageSource = bitmapImage;
suggestionItems.Add(suggestionItem1);
NameSuggestionItem suggestionItem2 = new NameSuggestionItem();
suggestionItem2.Name = "Andrew Fuller";
suggestionItem2.Link = "mailto:andrew.fuller@northwindtraders.com";
bitmapImage = new BitmapImage(new Uri(new
DirectoryInfo(@"..\..\Assets\People_Circle5.png").FullName));
suggestionItem2.ImageSource = bitmapImage;
suggestionItems.Add(suggestionItem2);
NameSuggestionItem suggestionItem3 = new NameSuggestionItem();
suggestionItem3.Name = "Steven Buchanan";
suggestionItem3.Link = "mailto:steven.buchanan@northwindtraders.com";
bitmapImage = new BitmapImage(new Uri(new
DirectoryInfo(@"..\..\Assets\People_Circle18.png").FullName));
suggestionItem3.ImageSource = bitmapImage;
suggestionItems.Add(suggestionItem3);
(suggestionProvider as NameSuggestionProvider).ItemsSource =
suggestionItems;
richTextboxadv.SuggestionSettings.SuggestionProviders.Add(suggestionProvider
);
ISuggestionProvider suggestionProviderAppType = new
NameSuggestionProvider();
suggestionProviderAppType.SuggestionBoxStyle =
this.Resources["SuggestionBoxStyle"] as System.Windows.Style;
suggestionProviderAppType.MentionCharacter = '#';
List<NameSuggestionItem> appTypes = new List<NameSuggestionItem>();
NameSuggestionItem desktopApp = new NameSuggestionItem();
desktopApp.Name = "Desktop App";
desktopApp.Link = "10 queries";
desktopApp.ImageSource = bitmapImage;
appTypes.Add(desktopApp);
NameSuggestionItem mobileApp = new NameSuggestionItem();

```

```
mobileApp.Name = "Mobile App";
mobileApp.Link = "13 queries";
mobileApp.ImageSource = bitmapImage;
appTypes.Add(mobileApp);
NameSuggestionItem webApp = new NameSuggestionItem();
webApp.Name = "Web App";
webApp.Link = "15 queries";
webApp.ImageSource = bitmapImage;
appTypes.Add(webApp);
(suggestionProviderAppType as NameSuggestionProvider).ItemsSource =
appTypes;
richTextboxadv.SuggestionSettings.SuggestionProviders.Add(suggestionProvider
AppType);
```

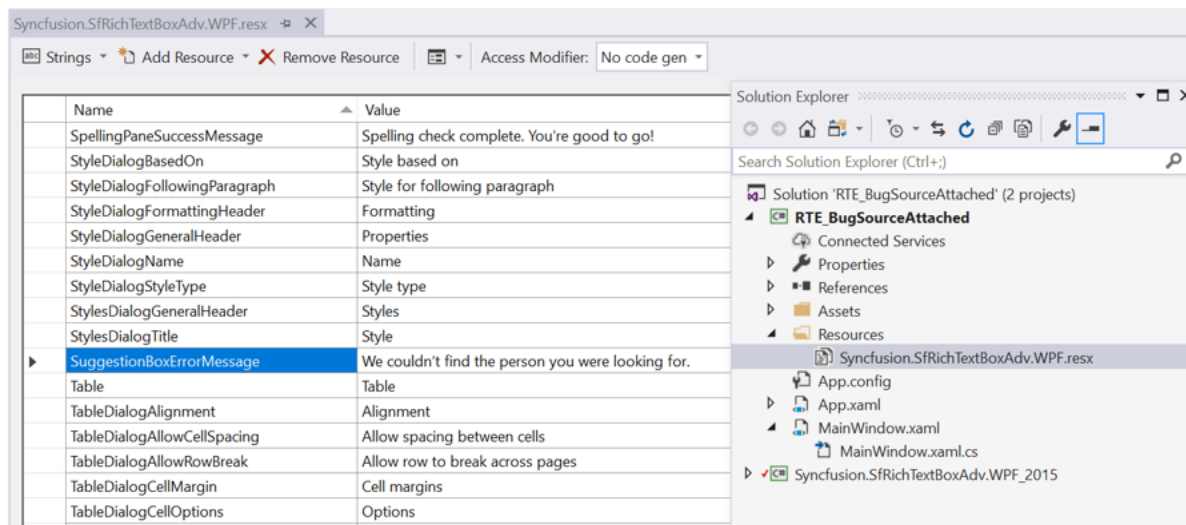
[View example in GitHub](#)

Display error message when suggestions are empty

When the entered item is not in the suggestion list, suggestion box displays a text indicating that “We couldn’t find the person you were looking for.”. The text to be displayed for this can be customized using the `SuggestionBoxErrorMessage` property in resource file (.resx).

- Right click your project and add new folder named Resources.
- Add [default resource file](#) of RichTextBox control into Resources folder.
- Modify the value of resource key `SuggestionBoxErrorMessage` in resource file.

@z|
We couldn't find the person you were looking for.



Custom suggestion provider

By default, we have implemented [NameSuggestionProvider](#) as suggestion provider. And you can implement your own suggestion provider, inheriting from [ISuggestionProvider](#). Which helps you to customize the search and insert selected item functionalities.

The following sample code demonstrates how to create your own suggestion provider inherited from ISuggestionProvider.

C#

```
internal class AppTypeSuggestionProvider : DependencyObject,
ISuggestionProvider
{
    #region Property
    public char MentionCharacter
    {
        get
        {
            return (char)GetValue(MentionCharacterProperty);
        }
        set
        {
            SetValue(MentionCharacterProperty, value);
        }
    }
    public Style SuggestionBoxStyle
    {
        get
        {
            return (Style)GetValue(SuggestionBoxStyleProperty);
        }
        set
        {
            SetValue(SuggestionBoxStyleProperty, value);
        }
    }
    public IEnumerable ItemsSource
    {
        get
        {
            return (IEnumerable)GetValue(ItemsSourceProperty);
        }
        set
        {
            SetValue(ItemsSourceProperty, value);
        }
    }
    public static DependencyProperty MentionCharacterProperty
    {
        get
        {
            return mentionCharacterProperty;
        }
    }
    public static DependencyProperty ItemsSourceProperty
    {
        get
        {
            return itemsSourceProperty;
        }
    }
    public static DependencyProperty SuggestionBoxStyleProperty
```



```

{
    get
    {
        return suggestionBoxStyleProperty;
    }
}
#endregion
#region Static Dependency Properties
/// <summary>
/// Identifies the MentionCharacter dependency property.
/// </summary>
private static DependencyProperty mentionCharacterProperty =
DependencyProperty.Register("MentionCharacter", typeof(char),
typeof(NameSuggestionProvider), new PropertyMetadata('@'));
/// <summary>
/// Identifies the ItemSource dependency property.
/// </summary>
private static DependencyProperty itemsSourceProperty =
DependencyProperty.Register("ItemsSource", typeof(IEnumerable),
typeof(NameSuggestionProvider), new PropertyMetadata(null));
/// <summary>
/// Identifies the SuggestionBoxStyle dependency property.
/// </summary>
private static DependencyProperty suggestionBoxStyleProperty =
DependencyProperty.Register("SuggestionBoxStyle", typeof(Style),
typeof(NameSuggestionProvider), new PropertyMetadata(null));
#endregion
public void Dispose()
{
    ClearValue(mentionCharacterProperty);
    if (ItemsSource != null)
    {
        foreach (NameSuggestionItem itemSource in ItemsSource)
        {
            itemSource.Dispose();
        }
    }
    ClearValue(itemsSourceProperty);
    ClearValue(suggestionBoxStyleProperty);
}
public void InsertSelectedItem(SfRichTextBoxAdv richTextBoxAdv, object
selectedItem)
{
    NameSuggestionItem nameSuggestionItem = selectedItem as NameSuggestionItem;
    richTextBoxAdv.Selection.InsertText(MentionCharacter +
nameSuggestionItem.Name);
}
public List<object> Search(string searchText)
{
    List<object> matchedItems = new List<object>();
    foreach (NameSuggestionItem item in ItemsSource)
    {
        if (item.Name.ToUpperInvariant().StartsWith(searchText.ToUpperInvariant()))
        {
            matchedItems.Add(item);
        }
    }
}

```

```
return matchedItems;
}
}
```

[View example in GitHub](#)

Customize search

In default searching, it list the items which contains the typed text. And you can modify the searching logic like list the items starts or ends with typed text, by implementing your own suggestion provider and overriding the Search method.

Search "€" contains	Search "€" starts with
	

The following sample code demonstrates how to override search operation in your suggestion provider.

C#

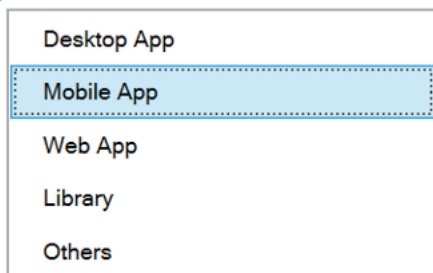
```
public List<object> Search(string searchText)
{
    List<object> matchedItems = new List<object>();
    foreach (NameSuggestionItem item in ItemsSource)
    {
        if (item.Name.ToUpperInvariant().StartsWith(searchText.ToUpperInvariant()))
        {
            matchedItems.Add(item);
        }
    }
    return matchedItems;
}
```

Customize insert item

By default, the selected item from the suggestions list is inserted as hyperlink. And you can insert it as plain text or without link, by implementing your own suggestion provider and overriding the "InsertSelectedItem" method.

#Desktop App

#



The following sample code demonstrates how to override insert selected item operation in your suggestion provider.

C#

```
public void InsertSelectedItem(SfRichTextBoxAdv richTextBoxAdv, object
selectedItem)
{
    NameSuggestionItem nameSuggestionItem = selectedItem as NameSuggestionItem;
    richTextBoxAdv.Selection.InsertText (MentionCharacter +
    nameSuggestionItem.Name) ;
}
```

Note: This feature is supported from V18.4.0.30.

[View example in GitHub](#)

Note: You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Dialogs in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv provides support for the following built-in dialogs similar to Microsoft Word application.

- Font Dialog
- Paragraph Dialog
- List Dialog
- Insert Table Dialog
- Insert Hyperlink Dialog
- Find and Replace Dialog
- Password Dialog
- Table Properties Dialog
- Table Options Dialog
- Cell Options Dialog
- Borders and Shading Dialog
- Styles Dialog

UI Commands for accessing dialogs

The following code example demonstrates how to show the built-in dialogs in SfRichTextBoxAdv through command binding.

XML

```
<!-- Binds button to the ShowFontDialogCommand -->
<Button Content="Font"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowFontDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowParagraphDialogCommand -->
<Button Content="Paragraph"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowParagraphDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowListDialogCommand -->
<Button Content="List"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowListDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowInsertTableDialogCommand -->
```

```

<Button Content="Insert Table"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowInsertTableDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowHyperlinkDialogCommand -->
<Button Content="Hyperlink"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowHyperlinkDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowFindAndReplaceDialogCommand -->
<Button Content="Find and Replace"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowFindAndReplaceDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowEncryptDocumentDialogCommand -->
<Button Content="Encrypt Document"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowEncryptDocumentDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowTableDialogCommand -->
<Button Content="Table Properties"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowTableDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowTableOptionsDialogCommand -->
<Button Content="Table Options"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowTableOptionsDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowCellOptionsDialogCommand -->
<Button Content="Cell Options"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowCellOptionsDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowBordersAndShadingDialogCommand -->
<Button Content="Borders and Shading"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowBordersAndShadingDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowStyleDialogCommand -->
<Button Content="Create style"
Command="Syncfusion:SfRichTextBoxAdv.ShowStyleDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />
<!-- Binds button to the ShowStylesDialogCommand -->
<Button Content="Modify style"
Command="Syncfusion:SfRichTextBoxAdv.ShowStylesDialogCommand"
CommandTarget="{Binding ElementName=richTextBoxAdv}" />

```

Customizing dialogs

This section describes how to create custom window for the dialogs of SfRichTextBoxAdv.

The following code example demonstrates how to design custom window for the font dialog of SfRichTextBoxAdv.

XML

```

<Window x:Class="DocumentEditor.FontWindow"
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:DocumentEditor"
mc:Ignorable="d"
xmlns:Syncfusion="http://schemas.syncfusion.com/wpf"

```

```

xmlns:System="clr-namespace:System;assembly=mscorlib"
Title="FontWindow" Height="530" Width="500">
<Grid>
<Syncfusion:FontDialog Name="fontDialog"></Syncfusion:FontDialog>
</Grid>
</Window>

```

C#

```

public partial class FontWindow : Window
{
    public FontWindow(SfRichTextBoxAdv rte)
    {
        InitializeComponent();
        fontDialog.DataContext = rte;
        this.Loaded += FontWindow_Loaded;
    }
    /// <summary>
    /// Called when the font window is loaded.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void FontWindow_Loaded(object sender, RoutedEventArgs e)
    {
        //Hooks the event for cancel button of font dialog.
        FontDialog font = VisualUtils.FindDescendant(sender as FontWindow,
        typeof(FontDialog)) as FontDialog;
        Button cancelButton = font.Template.FindName("PART_CancelButton", font as
        FrameworkElement) as Button;
        if (cancelButton != null)
        {
            cancelButton.Click += CancelButton_Click;
        }
        Button applybutton = font.Template.FindName("PART_ApplyFontFormatButton",
        font as FrameworkElement) as Button;
        if (applybutton != null)
        {
            applybutton.Click += Applybutton_Click;
        }
    }
    private void Applybutton_Click(object sender, RoutedEventArgs e)
    {
        this.Hide(); //Hides the window.
    }
    protected override void OnClosing(CancelEventArgs e)
    {
        //Hiding the window on close button.
        base.OnClosing(e);
        Hide();
        e.Cancel = true;
    }
    /// <summary>
    /// Called when the cancel button is clicked.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>

```

```
private void CancelButton_Click(object sender, RoutedEventArgs e)
{
    this.Hide();//hides the window.
}
}
```

VB.NET

```
Public Partial Class FontWindow
    Inherits Window
    Public Sub New(rte As SfRichTextBoxAdv)
        InitializeComponent()
        fontDialog.DataContext = rte
        AddHandler this.Loaded, AddressOf FontWindow_Loaded
    End Sub
    ''' <summary>
    ''' Called when the font window is loaded.
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    Private Sub FontWindow_Loaded(sender As Object, e As RoutedEventArgs)
        'Hooks the event for cancel button of font dialog.
        Dim font As FontDialog = TryCast(VisualUtils.FindDescendant(TryCast(sender,
        FontWindow), GetType(FontDialog)), FontDialog)
        Dim cancelButton As Button =
        TryCast(font.Template.FindName("PART_CancelButton", TryCast(font,
        FrameworkElement)), Button)
        If cancelButton IsNot Nothing Then
            AddHandler cancelButton.Click, AddressOf CancelButton_Click
        End If
        Dim applybutton As Button =
        TryCast(font.Template.FindName("PART_ApplyFontFormatButton", TryCast(font,
        FrameworkElement)), Button)
        If applybutton IsNot Nothing Then
            Addhandler applybutton.Click, AddressOf Applybutton_Click
        End If
    End Sub
    Private Sub Applybutton_Click(sender As Object, e As RoutedEventArgs)
        Me.Hide()
        'Hides the window.
    End Sub
    Protected Overrides Sub OnClosing(e As CancelEventArgs)
        'Hiding the window on close button.
        MyBase.OnClosing(e)
        Hide()
        e.Cancel = True
    End Sub
    ''' <summary>
    ''' Called when the cancel button is clicked.
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    Private Sub CancelButton_Click(sender As Object, e As RoutedEventArgs)
        Me.Hide()
        'hides the window.
    End Sub
```

End Class

Note: After creating custom dialog window, you have to set the SfRichTextBoxAdv instance as DataContext of the dialog.

The following code example demonstrates how to deploy the created font dialog window in your application.

XML

```
<Syncfusion:RibbonWindow x:Class="DocumentEditor.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:DocumentEditor"
mc:Ignorable="d"
xmlns:Syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:System="clr-namespace:System;assembly=mscorlib"
xmlns:RichTextBoxAdv="clr-namespace:Syncfusion.Windows.Controls.RichTextBoxAdv;assembly=Syncfusion.SfRichTextBoxAdv.WPF"
Title="MainWindow" Width="1087" Height="635"
Syncfusion:SkinStorage.VisualStudio="Office2013">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*/>
</Grid.RowDefinitions>
<Button Content="Font Dialog" Height="50" Width="200"
Click="Button_Click"></Button>
<Syncfusion:SfRichTextBoxAdv x:Name="richTextBoxAdv" Background="#F1F1F1"
DocumentTitle="RichTextBoxAdv" LayoutType="Pages" AcceptsTab="True"
Grid.Row="1"></Syncfusion:SfRichTextBoxAdv>
</Grid>
</Syncfusion:RibbonWindow>
```

C#

```
public partial class MainWindow : RibbonWindow
{
    public FontWindow fontWindow { get; set; }
    public MainWindow()
    {
        InitializeComponent();
        fontWindow = new FontWindow(richTextBoxAdv);
        this.Unloaded += MainWindow_Unloaded;
    }
    private void MainWindow_Unloaded(object sender, RoutedEventArgs e)
    {
        fontWindow.Close(); //closing the font window.
    }
    private void Button_Click(object sender, RoutedEventArgs e)
    {
        fontWindow.Show(); //showing the font window.
    }
}
```

}
VB.NET

```
Public Partial Class MainWindow
Inherits RibbonWindow
Public Property fontWindow() As FontWindow
Get
Return m_fontWindow
End Get
Set
m_fontWindow = Value
End Set
End Property
Private m_fontWindow As FontWindow
Public Sub New()
InitializeComponent()
fontWindow = New FontWindow(richTextBoxAdv)
AddHandler this.Unloaded, AddressOf MainWindow_Unloaded
End Sub
Private Sub MainWindow_Unloaded(sender As Object, e As RoutedEventArgs)
fontWindow.Close()
'closing the font window.
End Sub
Private Sub Button_Click(sender As Object, e As RoutedEventArgs)
fontWindow.Show()
'showing the font window.
End Sub
End Class
```

Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Localization in WPF RichTextBox (SfRichTextBoxAdv)

Localization is the process of configuring the application to a specific language. SfRichTextBoxAdv provides support to localize all the static text in ribbon and all its dialogs. Localization can be done by adding resource file (Resx) and setting the specific culture in the application.

Setting Current UI Culture

For localizing your application to specific culture, you have to set the 'CurrentUICulture' as required before invoking the InitializeComponent() method.

The following code example demonstrates how to set culture information for localizing an application.

C#

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


VB.NET

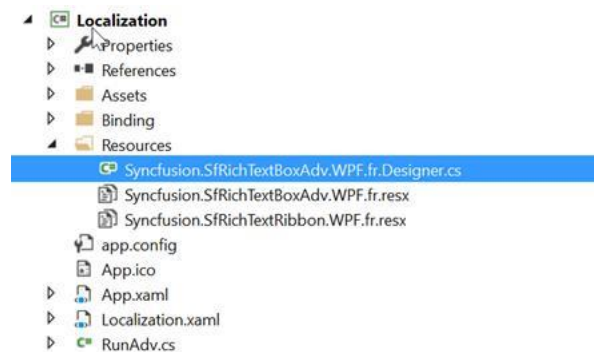
```

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

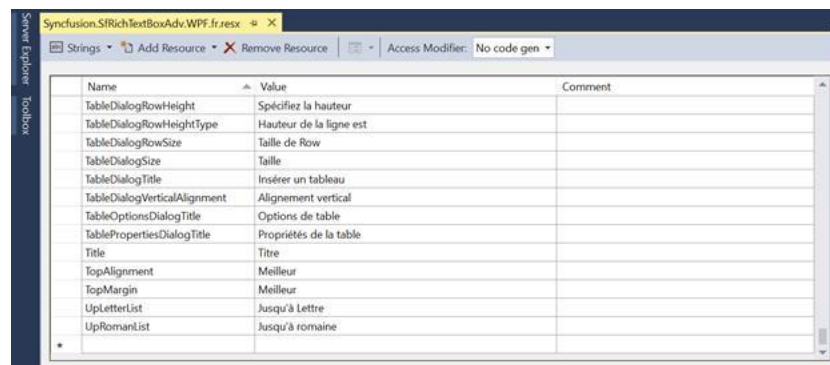
```

Adding Resource file

- Create a folder with name 'Resources' in your application.
- Add default English("en-US") [Resx](#) (resource) file of SfRichTextBoxAdv and SfRichTextRibbon in the 'Resources' folder named as Syncfusion.SfRichTextBoxAdv.WPF.resx and Syncfusion.SfRichTextRibbon.WPF.resx respectively
- Create Resx (resource) files and named as Syncfusion.SfRichTextBoxAdv.WPF. [Culture name].resx and Syncfusion.SfRichTextRibbon.WPF. [Culture name].resx. For example, Syncfusion.SfRichTextBoxAdv.WPF.fr.resx and Syncfusion.SfRichTextRibbon.WPF.fr.resx for French culture. For your reference, French("fr-FR") [Resx](#) file.

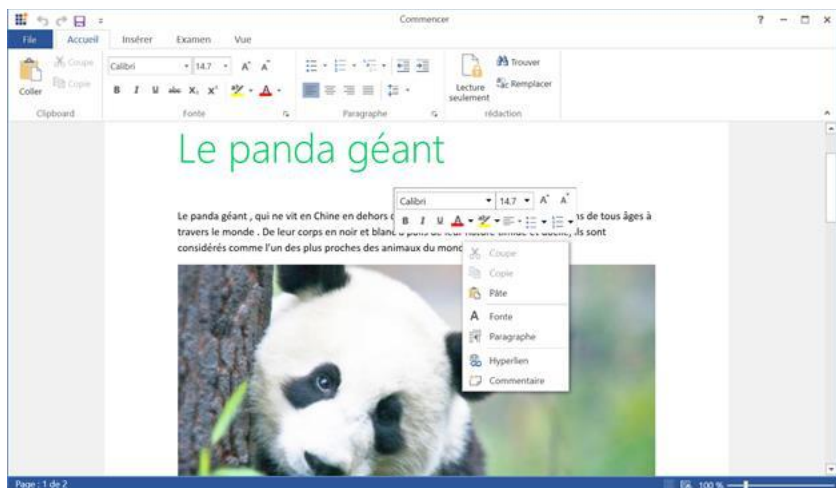


- Add the resource key such as name and its corresponding localized value in Resource Designer of Syncfusion.SfRichTextBoxAdv.WPF.fr.resx and Syncfusion.SfRichTextRibbon.WPF.fr.resx file.



Note: If you have not used SfRichTextRibbon in your application, you can skip Syncfusion.SfRichTextRibbon.WPF.[Culture name].resx file mentioned above.

The following screenshot shows the localization in SfRichTextBoxAdv and SfRichTextRibbon



Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

MVVM in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv control can be used with Model-View-View Model (MVVM) pattern. This section will demonstrate how to use the SfRichTextBoxAdv control with MVVM pattern.

Creating a View Model

The following code example demonstrates how to implement a view model class that contains properties to preserve the description about some of the animals and the animal that is selected for discussion. Whenever the animal chosen for discussion is changed, previously chosen animal description is updated to the database and newly chosen animal description is updated to the text property.

C#

```
/// <summary>
/// Represents the view model class.
/// </summary>
public class ViewModel : INotifyPropertyChanged
{
    #region Field
    private string animal;
    private string text;
    Dictionary<string, string> animals = null;
    bool skipUpdating = false;
    #endregion
    #region Properties
    /// <summary>
    /// Gets or sets the animal.
    /// </summary>
    /// <value>
    /// The document title.
    /// </value>
    public string Animal
    {
        get
        {
            return animal;
        }
    }
}
```

```

    }
    set
    {
        animal = value;
        NotifyPropertyChanged("Animal");
    }
}

/// <summary>
/// Gets the animals.
/// </summary>
/// <value>
/// The animals.
/// </value>
public ICollection<string> Animals
{
    get
    {
        return animals.Keys;
    }
}

/// <summary>
/// Gets or sets the Text.
/// </summary>
/// <value>
/// The document.
/// </value>
public string Text
{
    get
    {
        return text;
    }
    set
    {
        text = value;
        NotifyPropertyChanged("Text");
    }
}
#endregion
#region Event
public event PropertyChangedEventHandler PropertyChanged;
#endregion
#region Constructor
/// <summary>
/// Initializes a new instance of the <see cref="ViewModel"/> class.
/// </summary>
public ViewModel()
{
    Initialize();
}
#endregion
#region Implementation
/// <summary>
/// Handles initialization.
/// </summary>
private void Initialize()
{

```

```

animals = new Dictionary<string, string>();
animals.Add("Tiger", "The tiger is the largest cat species, reaching a total
body length of up to 3.38 m over curves and exceptionally weighing up to
388.7 kg in the wild.");
animals.Add("Lion", "The lion is one of the strongest animal. It is also
known as the king of jungles.");
animals.Add("Panda", "The giant panda, also known as panda bear or simply
panda, is a bear native to south central China. It is easily recognized by
the large, distinctive black patches around its eyes, over the ears, and
across its round body.");
animals.Add("Bear", "Bears are mammals and are classified as dog like
carnivorous.");
animals.Add("Deer", "Deer are the ruminant mammals. Species in the family
include the white-tailed deer, mule deer, elk, moose, red deer, reindeer,
fallow deer, roe deer.");
Animal = "Lion";
}
/// <summary>
/// Notifies the property changed.
/// </summary>
/// <param name="propertyName">Name of the property.</param>
private void NotifyPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    // Updates the text when the animal changes (reflects the view).
    if (propertyName == "Animal")
    {
        skipUpdating = true;
        Text = animals[animal];
        skipUpdating = false;
    }
    // Updates the document content, when changes done in view.
    if (propertyName == "Text" && !skipUpdating)
        animals[Animal] = Text;
    }
#endregion
}

```

VB.NET

```

''' <summary>
''' Represents the view model class.
''' </summary>
Public Class ViewModel
    Implements INotifyPropertyChanged
    #Region "Field"
    Private m_animal As String
    Private m_text As String
    Private m_animals As Dictionary(Of String, String) = Nothing
    Private skipUpdating As Boolean = False
    #End Region
    #Region "Properties"
    ''' <summary>
    ''' Gets or sets the animal.
    ''' </summary>

```

```

''' <value>
''' The document title.
''' </value>
Public Property Animal() As String
Get
Return m_animal
End Get
Set
m_animal = value
NotifyPropertyChanged("Animal")
End Set
End Property
''' <summary>
''' Gets the animals.
''' </summary>
''' <value>
''' The animals.
''' </value>
Public ReadOnly Property Animals() As ICollection(Of String)
Get
Return m_animals.Keys
End Get
End Property
''' <summary>
''' Gets or sets the Text.
''' </summary>
''' <value>
''' The document.
''' </value>
Public Property Text() As String
Get
Return m_text
End Get
Set
m_text = value
NotifyPropertyChanged("Text")
End Set
End Property
#End Region
#Region "Event"
Public Event PropertyChanged As PropertyChangedEventHandler
#End Region
#Region "Constructor"
''' <summary>
''' Initializes a new instance of the <see cref="ViewModel"/> class.
''' </summary>
Public Sub New()
Initialize()
End Sub
#End Region
#Region "Implementation"
''' <summary>
''' Handles initialization.
''' </summary>
Private Sub Initialize()
m_animals = New Dictionary(Of String, String)()

```

```

m_animals.Add("Tiger", "The tiger is the largest cat species, reaching a
total body length of up to 3.38 m over curves and exceptionally weighing up
to 388.7 kg in the wild.");
m_animals.Add("Lion", "The lion is one of the strongest animal. It is also
known as the king of jungles.");
m_animals.Add("Panda", "The giant panda, also known as panda bear or simply
panda, is a bear native to south central China. It is easily recognized by
the large, distinctive black patches around its eyes, over the ears, and
across its round body.");
m_animals.Add("Bear", "Bears are mammals and are classified as dog like
carnivorous.");
m_animals.Add("Deer", "Deer are the ruminant mammals. Species in the family
include the white-tailed deer, mule deer, elk, moose, red deer, reindeer,
fallow deer, roe deer.");
Animal = "Lion"
End Sub
''' <summary>
''' Notifies the property changed.
''' </summary>
''' <param name="propertyName">Name of the property.</param>
Private Sub NotifyPropertyChanged(propertyName As String)
RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(propertyName))
' Updates the text when the animal changes (reflects the view).
If propertyName = "Animal" Then
skipUpdating = True
Text = m_animals(m_animal)
skipUpdating = False
End If
' Updates the document content, when changes done in view.
If propertyName = "Text" AndAlso Not skipUpdating Then
m_animals(Animal) = Text
End If
End Sub
#End Region
End Class

```

Implementing extension class for SfRichTextBoxAdv

The following code example demonstrates how to implement an extension class for SfRichTextBoxAdv with dependency property that supports two way binding.

C#

```

/// <summary>
/// Represents the extension class for SfRichTextBoxAdv.
/// </summary>
public class SfRichTextBoxAdvExtension : SfRichTextBoxAdv
{
    #region Fields
    bool skipUpdating = false;
    #endregion
    #region Properties
    /// <summary>
    /// Gets or Sets the text.
    /// </summary>
    public string Text
    {

```

```

get
{
    return (string)GetValue(TextProperty);
}
set
{
    SetValue(TextProperty, value);
}
}
#endregion
#region Constructor
/// <summary>
/// Initializes the instance of SfRichTextBoxAdvExtension class.
/// </summary>
public SfRichTextBoxAdvExtension()
{
    // Wires the ContentChanged event.
    this.ContentChanged += RicTextBoxAdv_ContentChanged;
}
#endregion
#region Static Dependency Properties
/// <summary>
/// Using as a backing store for Text dependency property to enable styling,
/// animation etc.
/// </summary>
public static readonly DependencyProperty TextProperty =
    DependencyProperty.Register("Text", typeof(string),
        typeof(SfRichTextBoxAdvExtension), new PropertyMetadata(string.Empty, new
            PropertyChangedCallback(OnTextChanged)));
#endregion
#region Static Events
/// <summary>
/// Called when text changed.
/// </summary>
/// <param name="obj"></param>
/// <param name="e"></param>
private static void OnTextChanged(DependencyObject obj,
    DependencyPropertyChangedEventArgs e)
{
    SfRichTextBoxAdvExtension richTextBox = (SfRichTextBoxAdvExtension)obj;
    //Update the document with the Text.
    richTextBox.UpdateDocument((string)e.NewValue);
}
#endregion
#region Events
/// <summary>
/// Called when content changes in SfRichTextBoxAdv.
/// </summary>
/// <param name="obj"></param>
/// <param name="args"></param>
void RicTextBoxAdv_ContentChanged(object obj, ContentChangedEventArgs args)
{
    if (this.Document != null)
    {
        // To skip internal updation of document on setting Text property.
        skipUpdating = true;
        Stream stream = new MemoryStream();
    }
}

```

```

// Saves the document's text into a Stream.
this.Save(stream, FormatType.Txt);
stream.Position = 0;
// Reads the text from the stream.
using (StreamReader reader = new StreamReader(stream))
{
    this.Text = reader.ReadToEnd();
}
skipUpdating = false;
}
}
#endregion
#region Implementation
/// <summary>
/// Updates the document.
/// </summary>
/// <param name="text"></param>
private void UpdateDocument(string text)
{
    // If text property is set internally means, skip updating the document.
    if (!skipUpdating && !string.IsNullOrEmpty(text))
    {
        Stream stream = new MemoryStream();
        // Convert the text to byte array.
        byte[] bytes = Encoding.UTF8.GetBytes(text);
        // Writes the byte array to stream.
        stream.Write(bytes, 0, bytes.Length);
        stream.Position = 0;
        //Load the stream.
        Load(stream, FormatType.Txt);
    }
}
/// <summary>
/// Disposes the instance.
/// </summary>
public new void Dispose()
{
    this.ContentChanged -= RicTextBoxAdv_ContentChanged;
    ClearValue(TextProperty);
    base.Dispose();
}
#endregion
}

```

VB.NET

```

''' <summary>
''' Represents the extension class for SfRichTextBoxAdv.
''' </summary>
Public Class SfRichTextBoxAdvExtension
Inherits SfRichTextBoxAdv
#Region "Fields"
Private skipUpdating As Boolean = False
#End Region
#Region "Properties"
''' <summary>

```



```

''' Gets or Sets the text.
''' </summary>
Public Property Text() As String
Get
Return DirectCast(GetValue(TextProperty), String)
End Get
Set
SetValue(TextProperty, value)
End Set
End Property
#End Region
#Region "Constructor"
''' <summary>
''' Initializes the instance of SfRichTextBoxAdvExtension class.
''' </summary>
Public Sub New()
' Wires the ContentChanged event.
AddHandler this.ContentChanged, AddressOf RicTextBoxAdv_ContentChanged
End Sub
#End Region
#Region "Static Dependency Properties"
''' <summary>
''' Using as a backing store for Text dependency property to enable styling,
animation etc.
''' </summary>
Public Shared ReadOnly TextProperty As DependencyProperty =
DependencyProperty.Register("Text", GetType(String),
GetType(SfRichTextBoxAdvExtension), New PropertyMetadata(String.Empty, New
PropertyChangedCallback(OnTextChanged)))
#End Region
#Region "Static Events"
''' <summary>
''' Called when text changed.
''' </summary>
''' <param name="obj"></param>
''' <param name="e"></param>
Private Shared Sub OnTextChanged(obj As DependencyObject, e As
DependencyPropertyChangedEventArgs)
Dim richTextBox As SfRichTextBoxAdvExtension = DirectCast(obj,
SfRichTextBoxAdvExtension)
'Update the document with the Text.
richTextBox.UpdateDocument(DirectCast(e.NewValue, String))
End Sub
#End Region
#Region "Events"
''' <summary>
''' Called when content changes in SfRichTextBoxAdv.
''' </summary>
''' <param name="obj"></param>
''' <param name="args"></param>
Private Sub RicTextBoxAdv_ContentChanged(obj As Object, args As
ContentChangedEventArgs)
If Me.Document IsNot Nothing Then
' To skip internal updation of document on setting Text property.
skipUpdating = True
Dim stream As Stream = New MemoryStream()
' Saves the document's text into a Stream.

```

```

Me.Save(stream, FormatType.Txt)
stream.Position = 0
' Reads the text from the stream.
Using reader As New StreamReader(stream)
Me.Text = reader.ReadToEnd()
End Using
skipUpdating = False
End If
End Sub
#End Region
#Region "Implementation"
''' <summary>
''' Updates the document.
''' </summary>
''' <param name="text"></param>
Private Sub UpdateDocument(text As String)
' If text property is set internally means, skip updating the document.
If Not skipUpdating AndAlso Not String.IsNullOrEmpty(text) Then
Dim stream As Stream = New MemoryStream()
' Convert the text to byte array.
Dim bytes As Byte() = Encoding.UTF8.GetBytes(text)
' Writes the byte array to stream.
stream.Write(bytes, 0, bytes.Length)
stream.Position = 0
'Load the stream.
Load(stream, FormatType.Txt)
End If
End Sub
''' <summary>
''' Disposes the instance.
''' </summary>
Public Shadows Sub Dispose()
RemoveHandler this.ContentChanged, AddressOf RicTextBoxAdv_ContentChanged
ClearValue(TextProperty)
MyBase.Dispose()
End Sub
#End Region
End Class

```

Creating XAML View

The following code example demonstrates how to create XAML view with SfRichTextBoxAdv and UI properties bound to view model properties.

XML

```

<Window x:Class="Sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:Sample">
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Border>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>

```

```

<RowDefinition Height="*" />
</Grid.RowDefinitions>
<StackPanel Orientation="Horizontal" Margin="4">
<TextBlock Text="Animal :"/>
<ComboBox IsTabStop="False" ItemsSource="{Binding Animals}"
SelectedValue="{Binding Animal, Mode=TwoWay}" />
</StackPanel>
<Grid Margin="10" Grid.Row="1">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<TextBlock Text="Description" />
<Border BorderThickness="1" BorderBrush="#A3A3A3">
<local:SfRichTextBoxAdvExtension Grid.Row="1" Text="{Binding
Path=Text,Mode=TwoWay}" LayoutType="Continuous" />
</Border>
</Grid>
</Grid>
</Border>
</Window>

```

Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Virtualization in WPF RichTextBox (SfRichTextBoxAdv)

The SfRichTextBoxAdv control supports UI Virtualization. UI elements are created only for the contents that are visible in the viewer. The UI elements are created for the contents that become visible while scrolling the viewer. This reduces the main memory utilization and also improves UI performance.

Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Styles and Templates in WPF RichTextBox (SfRichTextBoxAdv)

This section describes the styles and templates for the SfRichTextBoxAdv control. The Template defines the structure of the SfRichTextBoxAdv control and the Style defines its visual appearance. You can modify the default Control template to define a unique appearance for the control.

The following XAML shows the default style and template for the SfRichTextBoxAdv control.

XAML

```

<RichTextBoxAdv:SfRichTextBoxAdvResourceWrapper
x:Key="SfRichTextBoxAdvResourceWrapper" />
<SolidColorBrush x:Key="RichTextBoxAdvBackgroundBrush" Color="#000000" />
<SolidColorBrush x:Key="TextSelectionBrush" Color="#FF808080" />
<SolidColorBrush x:Key="ContextMenuForegroundBrush" Color="#333333" />
<SolidColorBrush x:Key="ContextMenuBorderBrush" Color="#D7D7D7" />
<SolidColorBrush x:Key="ContextMenuBackgroundBrush" Color="#FFFFFF" />
<SolidColorBrush x:Key="ContextMenuMouseOverBackgroundBrush"
Color="#D5DDEB" />
<SolidColorBrush x:Key="MiniToolBarButtonMouseOverBrush" Color="#CAD5E5" />
<SolidColorBrush x:Key="MiniToolBarButtonSelectedBrush" Color="#AABBD6" />

```

```

<SolidColorBrush x:Key="MiniToolBarBorderBrush" Color="LightGray"/>
<SolidColorBrush x:Key="MiniToolBarBackgroundBrush" Color="#FFFFFF"/>
<SolidColorBrush x:Key="PaneBackgroundBrush" Color="#FFFFFF"/>
<SolidColorBrush x:Key="SearchResultItemBackgroundBrush" Color="#FFFFFF"/>
<SolidColorBrush x:Key="SearchResultItemForegroundBrush" Color="#333333"/>
<SolidColorBrush x:Key="SearchResultItemSelectedBorderBrush"
Color="#959595"/>
<SolidColorBrush x:Key="SearchResultItemMouseOverBorderBrush"
Color="#959595"/>
<SolidColorBrush x:Key="SearchResultItemSeparatorForegroundBrush"
Color="#666666"/>
<SolidColorBrush x:Key="SuggestionItemSelectedBrush" Color="#FFCBCFE5"/>
<SolidColorBrush x:Key="SuggestionItemMouseOverBrush" Color="#FFD5E1F2"/>
<SolidColorBrush x:Key="DialogHeaderForegroundBrush" Color="#FFFFFF"/>
<SolidColorBrush x:Key="DialogHeaderBackgroundBrush" Color="#0071BC"/>
<SolidColorBrush x:Key="DialogBorderBrush" Color="#8E8E8E"/>
<SolidColorBrush x:Key="DialogBackgroundBrush" Color="#FFFFFF"/>
<SolidColorBrush x:Key="HeaderLabelForegroundBrush" Color="#0071BC"/>
<SolidColorBrush x:Key="TextBoxForegroundBrush" Color="#333333"/>
<SolidColorBrush x:Key="TextBoxLightForegroundBrush" Color="#666666"/>
<SolidColorBrush x:Key="TextBoxBackgroundBrush" Color="#FFFFFF"/>
<SolidColorBrush x:Key="TextBoxBorderBrush" Color="#A5A5A5"/>
<SolidColorBrush x:Key="ButtonForegroundBrush" Color="#333333"/>
<SolidColorBrush x:Key="ButtonBackgroundBrush" Color="#F7F7F7"/>
<SolidColorBrush x:Key="ButtonBorderBrush" Color="#606060"/>
<SolidColorBrush x:Key="ButtonMouseOverBackgroundBrush" Color="#E5F0FC"/>
<SolidColorBrush x:Key="ButtonMouseOverForegroundBrush" Color="#6BA5C6" />
<SolidColorBrush x:Key="ButtonMouseOverBorderBrush" Color="#569DE5"/>
<SolidColorBrush x:Key="ButtonPressedForegroundBrush" Color="#FFFFFF"/>
<SolidColorBrush x:Key="ButtonPressedBackgroundBrush" Color="#0071BC"/>
<SolidColorBrush x:Key="ButtonPressedBorderBrush" Color="#0071BC"/>
<SolidColorBrush x:Key="ButtonDisabledForegroundBrush" Color="#A5A5A5"/>
<SolidColorBrush x:Key="ButtonDisabledBorderBrush" Color="#A5A5A5"/>
<SolidColorBrush x:Key="ButtonDisabledBackgroundBrush" Color="#FFFFFF"/>
<ContextMenu x:Key="ContextMenuStyle">
<ContextMenu.Resources>
<Style TargetType="MenuItem">
<Setter Property="Height" Value="26" />
<Setter Property="FontFamily" Value="Segoe UI"/>
<Setter Property="FontSize" Value="12"/>
<Setter Property="SnapsToDevicePixels" Value="True"/>
<Setter Property="Foreground" Value="{StaticResource
ContextMenuForegroundBrush}"/>
<Style.Resources>
<Style TargetType="Image">
<Setter Property="Height" Value="16" />
<Setter Property="Width" Value="16" />
</Style>
</Style.Resources>
</Style>
<Style TargetType="{x:Type ContextMenu}">
<Setter Property="OverridesDefaultStyle" Value="True"/>
<Setter Property="SnapsToDevicePixels" Value="True"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type ContextMenu}">

```

```

<Border Margin="10,10,10,10" Background="{StaticResource
ContextMenuBackgroundBrush}" BorderThickness="1"
BorderBrush="{StaticResource ContextMenuBorderBrush}">
<StackPanel Orientation="Vertical" IsItemsHost="True" Margin="1,1,1,1"/>
<Border.Effect>
<DropShadowEffect Color="Black" BlurRadius="7" Direction="315"
Opacity="0.5" ShadowDepth="2"/>
</Border.Effect>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<ControlTemplate x:Key="{x:Static MenuItem.SubmenuHeaderTemplateKey}"
TargetType="MenuItem">
<Border x:Name="Border" Background="{StaticResource
ContextMenuBackgroundBrush}">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="30"/>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="30"/>
</Grid.ColumnDefinitions>
<ContentPresenter x:Name="Icon" Margin="0" VerticalAlignment="Center"
HorizontalAlignment="Center" ContentSource="Icon"
SnapsToDevicePixels="True"/>
<ContentPresenter x:Name="HeaderHost" Grid.Column="1" ContentSource="Header"
VerticalAlignment="Center" HorizontalAlignment="Stretch" Margin="4,0,0,0"
SnapsToDevicePixels="True"/>
<Path Grid.Column="3" HorizontalAlignment="Right" VerticalAlignment="Center"
Data="M 0 0 L 0 7 L 4 3.5 Z" Width="12" Height="12" Margin="0 6 0 0"
Fill="{StaticResource ContextMenuForegroundBrush}"
SnapsToDevicePixels="True"/>
<Popup Name="Popup" Placement="Left" HorizontalOffset="-2"
IsOpen="{TemplateBinding IsSubmenuOpen}" PopupAnimation="Fade">
<Border Name="SubmenuBorder" Background="{StaticResource
ContextMenuBackgroundBrush}" BorderBrush="{StaticResource
ContextMenuBorderBrush}" BorderThickness="1">
<StackPanel Orientation="Vertical" IsItemsHost="True" Margin="1,1,1,1"/>
<Border.Effect>
<DropShadowEffect Color="Black" BlurRadius="30" Direction="315"
Opacity="0.5" ShadowDepth="2"/>
</Border.Effect>
</Border>
</Popup>
</Grid>
</Border>
<ControlTemplate.Triggers>
<Trigger Property="IsHighlighted" Value="true">
<Setter TargetName="Border" Property="Background" Value="{StaticResource
ContextMenuMouseOverBackgroundBrush}"/>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
<ControlTemplate x:Key="{x:Static MenuItem.SubmenuItemTemplateKey}"
TargetType="{x:Type MenuItem}">

```

```

<Border x:Name="Border" Background="{StaticResource
ContextMenuBackgroundBrush}">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="30"/>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="30" />
</Grid.ColumnDefinitions>
<ContentPresenter x:Name="Icon" Grid.Column="0" Margin="0"
HorizontalAlignment="Center" VerticalAlignment="Center" ContentSource="Icon"
SnapsToDevicePixels="True"/>
<ContentPresenter x:Name="HeaderHost" Grid.Column="1" ContentSource="Header"
HorizontalAlignment="Stretch" VerticalAlignment="Center" Margin="4,0,0,0"
SnapsToDevicePixels="True"/>
</Grid>
</Border>
<ControlTemplate.Triggers>
<Trigger Property="IsHighlighted" Value="true">
<Setter TargetName="Border" Property="Background" Value="{StaticResource
ContextMenuMouseOverBackgroundBrush}" />
</Trigger>
<Trigger Property="IsEnabled" Value="False">
<Setter TargetName="Icon" Property="Opacity" Value="0.5" />
<Setter TargetName="HeaderHost" Property="Opacity" Value="0.5" />
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</ContextMenu.Resources>
<Separator/>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper},Path=ContextMenuIgnoreAll}"
Visibility="Collapsed"
Command="RichTextBoxAdv:SfRichTextBoxAdv.IgnoreAllSpellingErrorsCommand"/>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper},Path=ContextMenuAddToDictionary}"
Visibility="Collapsed"
Command="RichTextBoxAdv:SfRichTextBoxAdv.AddToDictionaryCommand"/>
<Separator/>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuCut}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.CutCommand" >
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Cut_Icon.png"/>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuCopy}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.CopyCommand" >
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Copy_Icon.png"/>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuPaste}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.PasteCommand" >

```

```

<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Paste_Icon.png"/>
</MenuItem.Icon>
</MenuItem>
<Separator/>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuEditHyperlink}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowHyperlinkDialogCommand"
CommandParameter="EditHyperlink">
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Hyperlink_Icon.png
"/>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuOpenHyperlink}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.NavigateHyperlinkCommand" />
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuCopyHyperlink}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.CopyHyperlinkCommand" />
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuRemoveHyperlink}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.RemoveHyperlinkCommand" >
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/RemoveHyperlink_Ic
on.png"/>
</MenuItem.Icon>
</MenuItem>
<Separator/>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuFontDialog}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowFontDialogCommand" >
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Font_Icon.png"/>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuParagraphDialog}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowParagraphDialogCommand" >
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Paragraph_Icon.png
"/>
</MenuItem.Icon>
</MenuItem>
<Separator/>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuInsertTable}"
Visibility="Collapsed">
<MenuItem.Items>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuInsertLeftColumn}"

```

```

Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertColumnCommand"
CommandParameter="Left">
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/InsertColumnBefore_
_Icon.png"/>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuInsertRightColumn}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertColumnCommand"
CommandParameter="Right">
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/InsertColumnAfter_
Icon.png"/>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuInsertRowAbove}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertRowCommand"
CommandParameter="Above">
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/InsertRowBefore_Ic
on.png"/>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuInsertRowBelow}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.InsertRowCommand"
CommandParameter="Below">
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/InsertRowAfter_Ico
n.png"/>
</MenuItem.Icon>
</MenuItem>
</MenuItem.Items>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuMergeCell}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.MergeSelectedCellsCommand">
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/MergeCells_Icon.pn
g"/>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuHyperlink}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ShowHyperlinkDialogCommand"
CommandParameter="Hyperlink">
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Hyperlink_Icon.png
"/>

```



```

</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuComment}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.NewCommentCommand">
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Comment_Icon.png"/
>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuDeleteComment}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.DeleteCommentCommand">
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Comment_Icon.png"/
>
</MenuItem.Icon>
</MenuItem>
</ContextMenu>
<Style TargetType="RichTextBoxAdv:SfRichTextBoxAdv">
<Setter Property="Background" Value="{StaticResource
RichTextBoxAdvBackgroundBrush}"/>
<Setter Property="SelectionBrush" Value="{StaticResource
TextSelectionBrush}"/>
<Setter Property="ContextMenu" Value="{StaticResource ContextMenuStyle}"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="RichTextBoxAdv:SfRichTextBoxAdv">
<Border BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
Background="{TemplateBinding Background}">
<Grid>
<Grid.Resources>
<Style x:Key="PaneCloseButtonStyle" TargetType="{x:Type Button}">
<Setter Property="Background" Value="Transparent" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="5"/>
<RowDefinition Height="2*"/>
<RowDefinition Height="5"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="4"/>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="4"/>
</Grid.ColumnDefinitions>
<Path x:Name="pathFill" Grid.Row="1" Grid.Column="1" Fill="{StaticResource
ButtonForegroundBrush}" Height="16" Width="16" Stretch="Fill"
Data="F1M306,369.225L15.3,74.7L0,90L0,120.401L279.225,396L0,671.599L0,702L15
.3,717.3L306,422.775L596.7,717.3L612,702L612,671.599L332.775,396L612,120.401
L612,90L596.7,74.7z"/>

```

```

<Rectangle Grid.Row="0" Grid.RowSpan="3" Grid.Column="0" Grid.ColumnSpan="3"
x:Name="border" Fill="{StaticResource ButtonForegroundBrush}"/>
</Grid>
<ControlTemplate.Triggers>
<Trigger Property="IsEnabled" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0"/>
</Trigger>
<Trigger Property="IsMouseOver" Value="true">
<Setter Property="Fill" TargetName="pathFill" Value="{StaticResource
ButtonMouseOverForegroundBrush}"/>
<Setter Property="Opacity" TargetName="border" Value="0"/>
</Trigger>
<Trigger Property="IsPressed" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0"/>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Grid.Resources>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<Grid x:Name="OptionsPane" Background="{StaticResource PaneBackgroundBrush}"
Width="280" Visibility="Collapsed">
<Grid.Resources>
<Style x:Key="SearchTextBoxStyle" TargetType="{x:Type TextBox}">
<Setter Property="Foreground" Value="{StaticResource
TextBoxForegroundBrush}"/>
<Setter Property="Background" Value="{StaticResource
TextBoxBackgroundBrush}"/>
<Setter Property="BorderBrush" Value="{StaticResource TextBoxBorderBrush}"/>
<Setter Property="BorderThickness" Value="1"/>
<Setter Property="FontFamily" Value="Segoe UI"/>
<Setter Property="FontSize" Value="12"/>
<Setter Property="HorizontalAlignment" Value="Stretch"/>
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
Value="Hidden"/>
<Setter Property="ScrollViewer.VerticalScrollBarVisibility" Value="Hidden"/>
<Setter Property="ScrollViewer.IsDeferredScrollingEnabled" Value="False"/>
<Setter Property="Padding" Value="3 6 3 6"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="TextBox">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="30"/>
</Grid.ColumnDefinitions>
<Border x:Name="BackgroundElement" Background="{TemplateBinding Background}"
Grid.ColumnSpan="2" Margin="{TemplateBinding BorderThickness}"/>
<Border x:Name="BorderElement" BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}" Grid.ColumnSpan="2"/>

```

```

<ScrollViewer x:Name="PART_ContentHost"
HorizontalScrollBarVisibility="{TemplateBinding
ScrollViewer.HorizontalScrollBarVisibility}" IsTabStop="False"
IsDeferredScrollingEnabled="{TemplateBinding
ScrollViewer.IsDeferredScrollingEnabled}" Margin="{TemplateBinding
BorderThickness}" Padding="{TemplateBinding Padding}"
VerticalScrollBarVisibility="{TemplateBinding
ScrollViewer.VerticalScrollBarVisibility}"/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style x:Key="SearchResultTextBoxStyle" TargetType="{x:Type TextBox}">
<Setter Property="Background" Value="Transparent"/>
<Setter Property="Foreground" Value="{StaticResource
TextBoxLightForegroundBrush}"/>
<Setter Property="BorderThickness" Value="0"/>
<Setter Property="KeyboardNavigation.TabNavigation" Value="None"/>
<Setter Property="HorizontalContentAlignment" Value="Left"/>
<Setter Property="FocusVisualStyle" Value="{x:Null}"/>
<Setter Property="AllowDrop" Value="true"/>
<Setter Property="Stylus.IsFlicksEnabled" Value="False"/>
<Setter Property="FontFamily" Value="Segoe UI"/>
<Setter Property="FontSize" Value="12"/>
<Setter Property="HorizontalAlignment" Value="Left"/>
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
Value="Hidden"/>
<Setter Property="ScrollViewer.VerticalScrollBarVisibility" Value="Hidden"/>
<Setter Property="ScrollViewer.IsDeferredScrollingEnabled" Value="False"/>
<Setter Property="Padding" Value="0,6,0,6"/>
<Setter Property="MinHeight" Value="30"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type TextBox}">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="30" />
</Grid.ColumnDefinitions>
<Border x:Name="BackgroundElement" Background="{TemplateBinding Background}"
Grid.ColumnSpan="2" Margin="{TemplateBinding BorderThickness}"/>
<Border x:Name="BorderElement" BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}" Grid.ColumnSpan="2"/>
<ScrollViewer x:Name="PART_ContentHost" Focusable="false"
HorizontalScrollBarVisibility="Hidden"
VerticalScrollBarVisibility="Hidden"/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style x:Key="SearchButtonStyle" TargetType="{x:Type Button}">
<Setter Property="Background" Value="{StaticResource
TextBoxBackgroundBrush}" />
<Setter Property="Template">

```

```

<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="6"/>
<RowDefinition Height="*/>
<RowDefinition Height="6"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="7.25"/>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="7.25"/>
</Grid.ColumnDefinitions>
<Path x:Name="white" Grid.Row="1" Grid.Column="1"
Data="F1M401.625,472.5C306,472.5,229.5,396,229.5,300.375C229.5,204.75,306,12
8.25,401.625,128.25C497.25,128.25,573.75,204.75,573.75,300.375C573.75,396,49
7.25,472.5,401.625,472.5
M401.625,90C286.875,90,191.25,185.625,191.25,300.375C191.25,346.275,206.55,3
84.525,229.5,418.95L19.125,629.325C7.64999999999998,640.8,7.64999999999998,6
56.1,19.125,663.75L38.25,682.875C49.725,694.35,65.025,694.35,72.675,682.875L
283.05,472.5C317.475,495.45,359.55,510.75,401.625,510.75C516.375,510.75,612,
415.125,612,300.375C612,185.625,516.375,90,401.625,90" Fill="{StaticResource
ButtonForegroundBrush}" Stretch="Fill" Width="16" Height="16"/>
<Rectangle Grid.Row="0" Grid.RowSpan="3" Grid.Column="0" Grid.ColumnSpan="3"
Canvas.ZIndex="-1" x:Name="border" Fill="{StaticResource
ButtonForegroundBrush}" />
</Grid>
<ControlTemplate.Triggers>
<Trigger Property="IsEnabled" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0"/>
</Trigger>
<Trigger Property="IsMouseOver" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0.1"/>
</Trigger>
<Trigger Property="IsPressed" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0.16"/>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style x:Key="NavigatePreviousButtonStyle" TargetType="{x:Type Button}">
<Setter Property="Background" Value="Transparent" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="2"/>
<RowDefinition Height="2*/>
<RowDefinition Height="2"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="1"/>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="1"/>

```

```

</Grid.ColumnDefinitions>
<Path x:Name="pathFill" Grid.Row="1" Grid.Column="1" Width="12"
Stretch="Uniform" Fill="{StaticResource ButtonForegroundBrush}"
Data="F1M512.55,327.15L340.425,151.2C332.775,143.55,321.3,143.55,313.65,151.
2L137.7,327.15L164.475,353.925L306,212.4L306,663.75L344.25,663.75L344.25,212
.4L485.775,353.925z" />
<Rectangle Grid.Row="0" Grid.RowSpan="3" Grid.Column="0" Grid.ColumnSpan="3"
x:Name="border" Fill="{StaticResource ButtonForegroundBrush}" />
</Grid>
<ControlTemplate.Triggers>
<Trigger Property="IsEnabled" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0" />
</Trigger>
<Trigger Property="IsMouseOver" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0" />
<Setter Property="Fill" TargetName="pathFill" Value="{StaticResource
ButtonMouseOverForegroundBrush}" />
</Trigger>
<Trigger Property="IsPressed" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0" />
<Setter Property="Fill" TargetName="pathFill" Value="{StaticResource
ButtonPressedBackgroundBrush}" />
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style x:Key="NavigateNextButtonStyle" TargetType="{x:Type Button}">
<Setter Property="Background" Value="Transparent" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="2" />
<RowDefinition Height="2*" />
<RowDefinition Height="2" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="1" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="1" />
</Grid.ColumnDefinitions>
<Path x:Name="pathFill" Grid.Row="1" Grid.Column="1" Width="12"
Stretch="Uniform" Fill="{StaticResource ButtonForegroundBrush}"
Data="F1M512.55,327.15L340.425,151.2C332.775,143.55,321.3,143.55,313.65,151.
2L137.7,327.15L164.475,353.925L306,212.4L306,663.75L344.25,663.75L344.25,212
.4L485.775,353.925z" RenderTransformOrigin="0.5,0.5">
<Path.RenderTransform>
<RotateTransform Angle="180" />
</Path.RenderTransform>
</Path>
<Rectangle Grid.Row="0" Grid.RowSpan="3" Grid.Column="0" Grid.ColumnSpan="3"
x:Name="border" Fill="{StaticResource ButtonForegroundBrush}" />
</Grid>
<ControlTemplate.Triggers>

```

```

<Trigger Property="IsEnabled" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0"/>
</Trigger>
<Trigger Property="IsMouseOver" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0"/>
<Setter Property="Fill" TargetName="pathFill" Value="{StaticResource
ButtonMouseOverForegroundBrush}"/>
</Trigger>
<Trigger Property="IsPressed" Value="true">
<Setter Property="Opacity" TargetName="border" Value="0"/>
<Setter Property="Fill" TargetName="pathFill" Value="{StaticResource
ButtonPressedBackgroundBrush}"/>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style x:Key="SearchResultItemStyle" TargetType="{x:Type ListBoxItem}" >
<Setter Property="Padding" Value="6 3" />
<Setter Property="FontFamily" Value="Segoe UI"/>
<Setter Property="FontSize" Value="12"/>
<Setter Property="HorizontalContentAlignment" Value="Left" />
<Setter Property="VerticalContentAlignment" Value="Top" />
<Setter Property="Background" Value="{StaticResource
SearchResultItemBackgroundBrush}"/>
<Setter Property="Foreground" Value="{StaticResource
SearchResultItemForegroundBrush}"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="ListBoxItem">
<Grid Background="{TemplateBinding Background}">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height ="Auto"/>
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Border x:Name="SelectedBorder" BorderBrush="{StaticResource
SearchResultItemSelectedBorderBrush}" BorderThickness="2" Opacity="0" />
<Border x:Name="ItemBorder" BorderBrush="{StaticResource
SearchResultItemMouseOverBorderBrush}" BorderThickness="1" Opacity="0" />
<ContentPresenter x:Name="contentPresenter" Content="{TemplateBinding
Content}" ContentTemplate="{TemplateBinding ContentTemplate}"
VerticalAlignment="Center" HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}" Margin="{TemplateBinding Padding}"/>
<Rectangle Margin="0,4,0,4" Height="1" Opacity="0.2" Grid.Row="1"
Fill="{StaticResource SearchResultItemSeparatorForegroundBrush}"/>
</Grid>
</Grid>
<ControlTemplate.Triggers>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="IsMouseOver" Value="True"/>
<Condition Property="IsSelected" Value="False"/>
</MultiTrigger.Conditions>
<Setter Property="Opacity" TargetName="ItemBorder" Value="1"/>
</MultiTrigger>

```

```

<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="IsMouseOver" Value="True"/>
<Condition Property="IsSelected" Value="True"/>
</MultiTrigger.Conditions>
<Setter Property="Opacity" TargetName="SelectedBorder" Value="1"/>
</MultiTrigger>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="Selector.IsSelectionActive" Value="False"/>
<Condition Property="IsSelected" Value="True"/>
</MultiTrigger.Conditions>
<Setter Property="Opacity" TargetName="SelectedBorder" Value="1"/>
</MultiTrigger>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="Selector.IsSelectionActive" Value="True"/>
<Condition Property="IsSelected" Value="True"/>
</MultiTrigger.Conditions>
<Setter Property="Opacity" TargetName="SelectedBorder" Value="1"/>
</MultiTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style x:Key="SearchResultsListBoxStyle" TargetType="{x:Type ListBox}">
<Setter Property="Padding" Value="1"/>
<Setter Property="Background" Value="Transparent"/>
<Setter Property="BorderThickness" Value="0"/>
<Setter Property="Foreground" Value="{StaticResource
TextBoxForegroundBrush}"/>
<Setter Property="HorizontalContentAlignment" Value="Left" />
<Setter Property="VerticalContentAlignment" Value="Top" />
<Setter Property="IsTabStop" Value="False" />
<Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
Value="Disabled"/>
<Setter Property="ScrollViewer.VerticalScrollBarVisibility" Value="Auto"/>
<Setter Property="ItemContainerStyle" Value="{StaticResource
SearchResultItemStyle}"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type ListBox}">
<Grid>
<Border CornerRadius="2" BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}">
<ScrollViewer x:Name="ScrollViewer" Padding="{TemplateBinding Padding}"
Background="{TemplateBinding Background}" BorderBrush="Transparent"
BorderThickness="0">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="20"/>
</Grid.ColumnDefinitions>
<ItemsPresenter />
</Grid>
</ScrollViewer>

```



```

</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Grid.Resources>
<Grid.RowDefinitions>
<RowDefinition Height="20"/>
<RowDefinition Height="30"/>
<RowDefinition Height="12"/>
<RowDefinition Height="34"/>
<RowDefinition Height="12"/>
<RowDefinition Height="18"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="8"/>
<RowDefinition Height="*/>
<RowDefinition Height="20"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="26"/>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="26"/>
</Grid.ColumnDefinitions>
<Rectangle Grid.Column="2" Grid.RowSpan="10" Width="2"
HorizontalAlignment="Right" Opacity="0.5" Fill="{StaticResource
ButtonBorderBrush}"/>
<TextBlock x:Name="PaneHeaderBlock" Grid.Row="1" Grid.Column="1"
Text="{Binding Source={StaticResource SfRichTextBoxAdvResourceWrapper},
Path=OptionsPaneSearch}" FontFamily="Segoe UI" FontSize="18"
Foreground="{StaticResource HeaderLabelForegroundBrush}"
VerticalAlignment="Center" HorizontalAlignment="Stretch"/>
<Button x:Name="ClosePaneButton" Grid.Row="1" Grid.Column="1"
Grid.ColumnSpan="2" Style="{StaticResource PaneCloseButtonStyle}"
HorizontalAlignment="Right" Margin="0 0 20 0"/>
<Grid Grid.Row="3" Grid.Column="1">
<Grid.ColumnDefinitions >
<ColumnDefinition Width="5*/>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="*/>
</Grid.ColumnDefinitions>
<TextBox x:Name="SearchTextBox" MinHeight="32" Grid.ColumnSpan="3"
Style="{StaticResource SearchTextBoxStyle}"/>
<Button x:Name="SearchButton" Grid.Column="2" Style="{StaticResource
SearchButtonStyle}" ToolTip="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=SearchButtonToolTip}"/>
</Grid>
<Grid Grid.Row="5" Grid.Column="1">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="12"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<TextBlock Grid.Column="0" Text="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=OptionsPaneResultsHeader}"
FontFamily="Segoe UI" FontSize="14" Foreground="{StaticResource

```



```

TextBoxForegroundBrush}" HorizontalAlignment="Stretch"
VerticalAlignment="Center"/>
<Button x:Name="NavigatePreviousResultButton" Grid.Column="1"
Style="{StaticResource NavigatePreviousButtonStyle}"
Visibility="Collapsed"/>
<Button x:Name="NavigateNextResultButton" Grid.Column="3"
Style="{StaticResource NavigateNextButtonStyle}" Visibility="Collapsed"/>
</Grid>
<TextBox x:Name="SearchResultsCountTextBox" Grid.Row="6" Grid.Column="1"
Style="{StaticResource SearchResultTextBoxStyle}" Visibility="Collapsed"
IsReadOnly="True" Margin="-2 0 0 0" IsEnabled="False"
HorizontalAlignment="Left" VerticalAlignment="Top"
VerticalContentAlignment="Top"/>
<TextBox x:Name="SearchResultsTextBox" Grid.Row="8" Grid.Column="1"
Text="{Binding Source={StaticResource SfRichTextBoxAdvResourceWrapper},
Path=OptionsPaneDescription, Mode=OneWay}" Style="{StaticResource
SearchResultTextBoxStyle}" Margin="-2 0 0 0" Padding="0"
HorizontalAlignment="Stretch" VerticalAlignment="Top" TextWrapping="Wrap"
IsEnabled="False"/>
<ListBox x:Name="SearchResultsListBox" Grid.Row="8" Grid.Column="1"
Style="{StaticResource SearchResultsListBoxStyle}" Margin="-4 0 0 0"
SelectionMode="Single" Visibility="Collapsed"/>
</Grid>
<Grid Grid.Column="1">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<ContentControl x:Name="content" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Grid.Row="0" Grid.Column="0" />
<ScrollBar x:Name="HorizontalScrollBar" Grid.Column="0" Height="16"
Visibility="Collapsed" IsTabStop="False" Minimum="0"
Orientation="Horizontal" Grid.Row="1" />
<ScrollBar x:Name="VerticalScrollBar" Grid.Column="1" IsTabStop="False"
Visibility="Collapsed" Minimum="0" Orientation="Vertical" Grid.Row="0"
Width="16" />
</Grid>
<Grid x:Name="SpellingPane" Grid.Column="2" Background="{StaticResource
PaneBackgroundBrush}" Width="300" Visibility="Collapsed"
SnapsToDevicePixels="False">
<Grid.Resources>
<Style x:Key="SpellingPaneButtonStyle" TargetType="{x:Type Button}">
<Setter Property="Background" Value="{StaticResource
ButtonBackgroundBrush}" />
<Setter Property="Foreground" Value="{StaticResource
ButtonForegroundBrush}" />
<Setter Property="FontWeight" Value="Normal" />
<Setter Property="BorderBrush" Value="{StaticResource ButtonBorderBrush}" />
<Setter Property="BorderThickness" Value="1" />
<Setter Property="Padding" Value="10 1 10 1" />
<Setter Property="HorizontalAlignment" Value="Left" />
<Setter Property="VerticalAlignment" Value="Center" />
<Setter Property="FontFamily" Value="Segoe UI" />

```

```

<Setter Property="FontSize" Value="14"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
<Grid>
<Border x:Name="Border" BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
Background="{TemplateBinding Background}" SnapsToDevicePixels="True">
<ContentPresenter x:Name="ContentPresenter"
ContentTemplate="{TemplateBinding ContentTemplate}"
Content="{TemplateBinding Content}" HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}" Margin="{TemplateBinding Padding}"
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>
</Border>
</Grid>
<ControlTemplate.Triggers>
<Trigger Property="IsMouseOver" Value="true">
<Setter Property="BorderBrush" TargetName="Border" Value="{StaticResource
ButtonMouseOverBorderBrush}"/>
<Setter Property="Background" Value="{StaticResource
ButtonMouseOverBackgroundBrush}"/>
</Trigger>
<Trigger Property="IsPressed" Value="true">
<Setter Property="Background" Value="{StaticResource
ButtonPressedBackgroundBrush}"/>
<Setter Property="Foreground" Value="{StaticResource
ButtonPressedForegroundBrush}"/>
</Trigger>
<Trigger Property="IsEnabled" Value="false">
<Setter Property="Foreground" Value="{StaticResource
ButtonDisabledForegroundBrush}"/>
<Setter Property="Background" Value="{StaticResource
ButtonDisabledBackgroundBrush}"/>
<Setter Property="BorderBrush" TargetName="Border" Value="{StaticResource
ButtonDisabledBorderBrush}"/>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style x:Key="SpellingSuggestionItemStyle" TargetType="{x:Type ListBoxItem}"
>
<Setter Property="Padding" Value="6 3" />
<Setter Property="FontFamily" Value="Segoe UI"/>
<Setter Property="FontSize" Value="14"/>
<Setter Property="HorizontalContentAlignment" Value="Left" />
<Setter Property="VerticalContentAlignment" Value="Top" />
<Setter Property="Height" Value="30"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="ListBoxItem">
<Grid Background="{TemplateBinding Background}" SnapsToDevicePixels="False">
<ContentPresenter x:Name="contentPresenter" Content="{TemplateBinding
Content}" ContentTemplate="{TemplateBinding ContentTemplate}"
VerticalAlignment="Center" HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}" Margin="{TemplateBinding Padding}"/>

```

```

</Grid>
<ControlTemplate.Triggers>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="IsMouseOver" Value="True"/>
<Condition Property="IsSelected" Value="False"/>
</MultiTrigger.Conditions>
<Setter Property="Background" Value="{StaticResource
SuggestionItemMouseOverBrush}"/>
</MultiTrigger>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="IsMouseOver" Value="True"/>
<Condition Property="IsSelected" Value="True"/>
</MultiTrigger.Conditions>
<Setter Property="Background" Value="{StaticResource
SuggestionItemSelectedBrush}"/>
</MultiTrigger>
<Trigger Property="IsSelected" Value="True">
<Setter Property="Background" Value="{StaticResource
SuggestionItemSelectedBrush}"/>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style x:Key="SpellingSuggestionListBoxStyle" TargetType="{x:Type ListBox}">
<Setter Property="Foreground" Value="{StaticResource
TextBoxForegroundBrush}"/>
<Setter Property="HorizontalContentAlignment" Value="Left" />
<Setter Property="VerticalContentAlignment" Value="Top" />
<Setter Property="IsTabStop" Value="True" />
<Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
Value="Disabled"/>
<Setter Property="ScrollViewer.VerticalScrollBarVisibility" Value="Auto"/>
<Setter Property="ItemContainerStyle" Value="{StaticResource
SpellingSuggestionItemStyle}"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type ListBox}">
<Grid SnapsToDevicePixels="False">
<Border BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}">
<ScrollViewer x:Name="ScrollViewer" Padding="{TemplateBinding Padding}"
Background="{TemplateBinding Background}" BorderBrush="Transparent" >
<Grid>
<ItemsPresenter />
</Grid>
</ScrollViewer>
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Grid.Resources>

```

```

<Grid.RowDefinitions>
<RowDefinition Height="41"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="27"/>
<RowDefinition Height="25"/>
<RowDefinition Height="17"/>
<RowDefinition Height="25"/>
<RowDefinition Height="15"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="15"/>
<RowDefinition Height="25"/>
<RowDefinition Height="*/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="13"/>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="13"/>
</Grid.ColumnDefinitions>
<TextBlock x:Name="SpellingPaneHeaderBlock" Grid.Row="0" Grid.Column="1"
Text="{Binding Source={StaticResource SfRichTextBoxAdvResourceWrapper},
Path=SpellingPaneSpelling}" FontFamily="Segoe UI" FontSize="20"
Foreground="{StaticResource HeaderLabelForegroundBrush}"
VerticalAlignment="Center" HorizontalAlignment="Stretch"/>
<Button x:Name="SpellingPaneCloseButton" Grid.Row="0" Grid.Column="1"
Style="{StaticResource PaneCloseButtonStyle}" HorizontalAlignment="Right"
VerticalAlignment="Center"/>
<Separator Grid.Row="1" Grid.Column="1" Opacity="0.5" IsEnabled="False"
VerticalAlignment="Bottom" >/Separator>
<TextBlock x:Name="SpellingPaneMisspelledWordBlock" Grid.Row="3"
Grid.Column="1" FontFamily="Segoe UI" FontSize="14"
Foreground="{StaticResource HeaderLabelForegroundBrush}"
FontWeight="SemiBold" HorizontalAlignment="Stretch"
VerticalAlignment="Center" Margin="0"/>
<Grid Grid.Row="5" Grid.Column="1">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto">/ColumnDefinition>
<ColumnDefinition Width="Auto">/ColumnDefinition>
<ColumnDefinition Width="Auto">/ColumnDefinition>
</Grid.ColumnDefinitions>
<Button x:Name="SpellingPaneResumeButton" Content="{Binding
Source={StaticResource SfRichTextBoxAdvResourceWrapper},
Path=SpellingPaneResume}" Style="{StaticResource SpellingPaneButtonStyle}"
Visibility="Collapsed"/>
<Button x:Name="SpellingPaneIgnoreAllButton" Grid.Column="1"
Command="RichTextBoxAdv:SfRichTextBoxAdv.IgnoreAllSpellingErrorsCommand"
CommandParameter="{Binding
Text,ElementName=SpellingPaneMisspelledWordBlock}" Content="{Binding
Source={StaticResource SfRichTextBoxAdvResourceWrapper},
Path=SpellingPaneIgnoreAll}" Style="{StaticResource
SpellingPaneButtonStyle}" Margin="0 0 0 0"/>
<Button x:Name="SpellingPaneAddToDictionaryButton" Grid.Column="2"
Command="RichTextBoxAdv:SfRichTextBoxAdv.AddToDictionaryCommand"
CommandParameter="{Binding
Text,ElementName=SpellingPaneMisspelledWordBlock}" Content="{Binding
Source={StaticResource SfRichTextBoxAdvResourceWrapper},
Path=SpellingPaneAddToDictionary}" Style="{StaticResource
SpellingPaneButtonStyle}" Margin="15 0 0 0"/>

```

```

</Grid>
<ListBox x:Name="SpellingPaneSuggestionListBox" Grid.Row="7" Grid.Column="1"
Height="152" SelectionMode="Single" Style="{StaticResource
SpellingSuggestionListBoxStyle}"/>
<Grid Grid.Row="9" Grid.Column="1">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"></ColumnDefinition>
<ColumnDefinition Width="Auto"></ColumnDefinition>
</Grid.ColumnDefinitions>
<Button x:Name="SpellingPaneChangeButton"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ChangeSpellingCommand"
CommandParameter="{Binding ElementName=SpellingPaneSuggestionListBox,
Path=SelectedValue}" Content="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=SpellingPaneChange}"
Style="{StaticResource SpellingPaneButtonStyle}"/>
<Button x:Name="SpellingPaneChangeAllButton" Grid.Column="1"
Command="RichTextBoxAdv:SfRichTextBoxAdv.ChangeAllSpellingCommand"
CommandParameter="{Binding ElementName=SpellingPaneSuggestionListBox,
Path=SelectedValue}" Content="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=SpellingPaneChangeAll}"
Style="{StaticResource SpellingPaneButtonStyle}" Margin="15 0 0 0"/>
</Grid>
</Grid>
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

Note: In the control template, you are allowed to reorder the template parts and to add your own elements. However, when changing the control template you should be careful to include all required parts. Usually required parts are marked with Name attribute. Omission of required parts may impact some of the functionality.

Styling the SfRichTextBoxAdv

You can define custom style for the SfRichTextBoxAdv control either by creating empty style and set it up on your own or by copying the default style and modifying it.

The following example demonstrates how to customize the style for SfRichTextBoxAdv control.

XML

```

<RichTextBoxAdv:SfRichTextBoxAdvResourceWrapper
x:Key="SfRichTextBoxAdvResourceWrapper"/>
<SolidColorBrush x:Key="RichTextBoxAdvBackgroundBrush" Color="#000000"/>
<SolidColorBrush x:Key="TextSelectionBrush" Color="#FF808080"/>
<ContextMenu x:Key="ContextMenuStyle">
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuCut}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.CutCommand" >
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Cut_Icon.png"/>
</MenuItem.Icon>
</MenuItem>

```

```

<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuCopy}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.CopyCommand" >
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Copy_Icon.png"/>
</MenuItem.Icon>
</MenuItem>
<MenuItem Header="{Binding Source={StaticResource
SfRichTextBoxAdvResourceWrapper}, Path=ContextMenuPaste}"
Command="RichTextBoxAdv:SfRichTextBoxAdv.PasteCommand" >
<MenuItem.Icon>
<Image
Source="/Syncfusion.SfRichTextBoxAdv.WPF;component/Images/Paste_Icon.png"/>
</MenuItem.Icon>
</MenuItem>
</ContextMenu>
<Style TargetType="RichTextBoxAdv:SfRichTextBoxAdv"
x:Key="RichTextBoxAdvCustomStyle" >
<Setter Property="Background" Value="{StaticResource
RichTextBoxAdvBackgroundBrush}"/>
<Setter Property="SelectionBrush" Value="{StaticResource
TextSelectionBrush}"/>
<Setter Property="ContextMenu" Value="{StaticResource ContextMenuStyle}"/>
<Setter Property="EnableMiniToolBar" Value="False"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="RichTextBoxAdv:SfRichTextBoxAdv">
<Border BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
Background="{TemplateBinding Background}">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<ContentControl x:Name="content" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Grid.Row="0" Grid.Column="0" />
<ScrollBar x:Name="HorizontalScrollBar" Grid.Column="0" Height="16"
Visibility="Collapsed" IsTabStop="False" Minimum="0"
Orientation="Horizontal" Grid.Row="1" />
<ScrollBar x:Name="VerticalScrollBar" Grid.Column="1" IsTabStop="False"
Visibility="Collapsed" Minimum="0" Orientation="Vertical" Grid.Row="0"
Width="16" />
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The following code example demonstrates how to apply the custom style for SfRichTextBoxAdv control. Applying this style will result in a SfRichTextBoxAdv control with no options pane and no spelling pane.

XML

```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv"
Style="{StaticResource RichTextBoxAdvCustomStyle}" />
```

Note: You can refer to our [WPF RichTextBox](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

Setting Background for WPF RichTextBox

The [WPF RichTextBox](#) control allows you to change background color of the control. A background of a control is represented by **Background** property of **SfRichTextBoxAdv** class. The default value of this property is black.

The following code illustrates how to apply color as background to the document.

XML

```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv"
Background="#6699cc" />
```

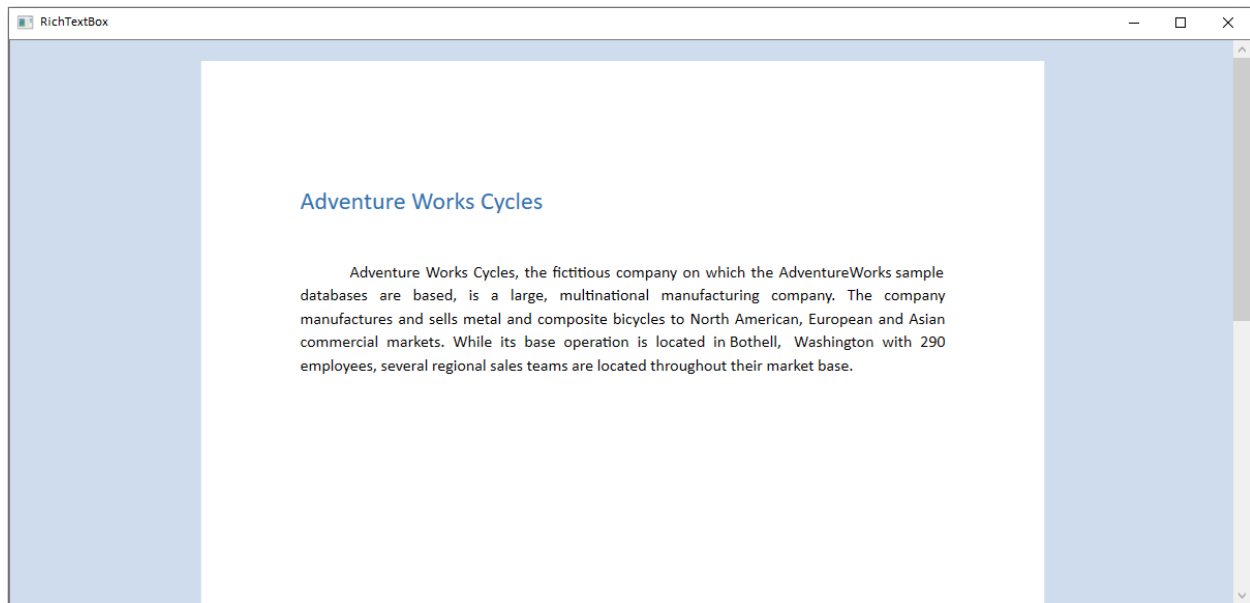
C#

```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
// Sets the control background color
richTextBoxAdv.Background = new SolidColorBrush(Color.FromRgb(102, 153,
204));
```

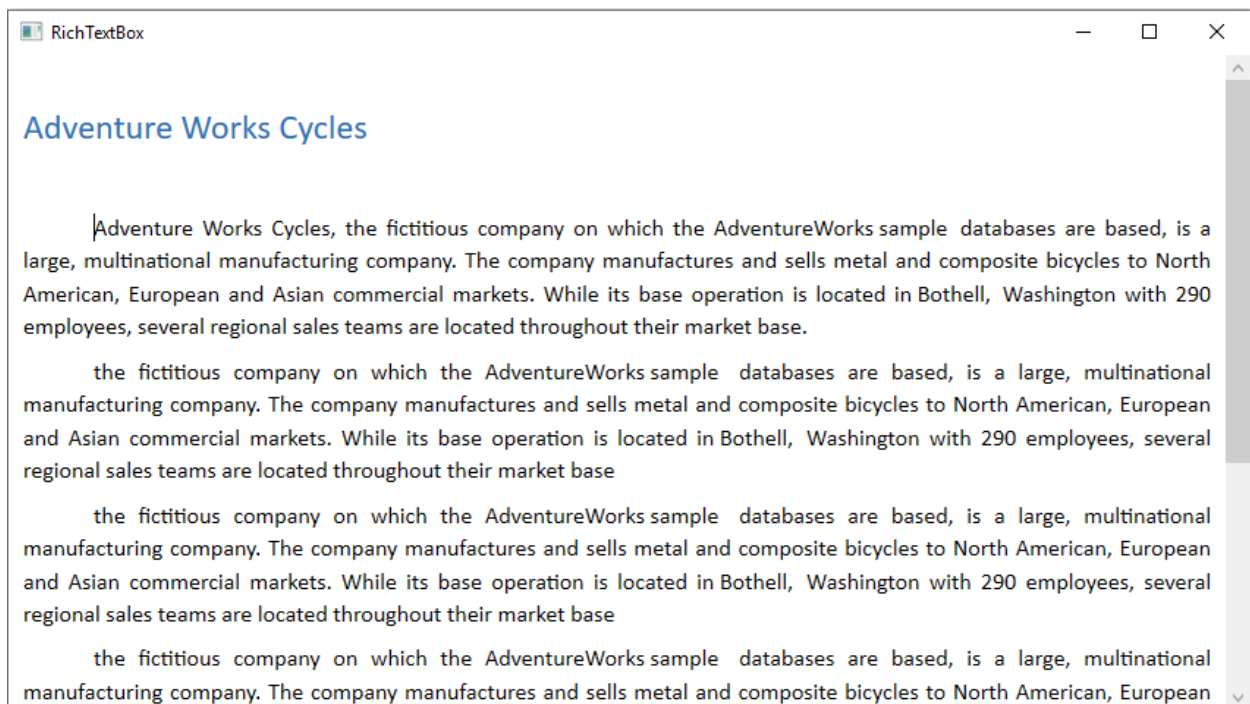
VB.NET

```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
' Sets the control background color.
richTextBoxAdv.Background = new SolidColorBrush(Color.FromRgb(102, 153,
204))
```

Pages layout

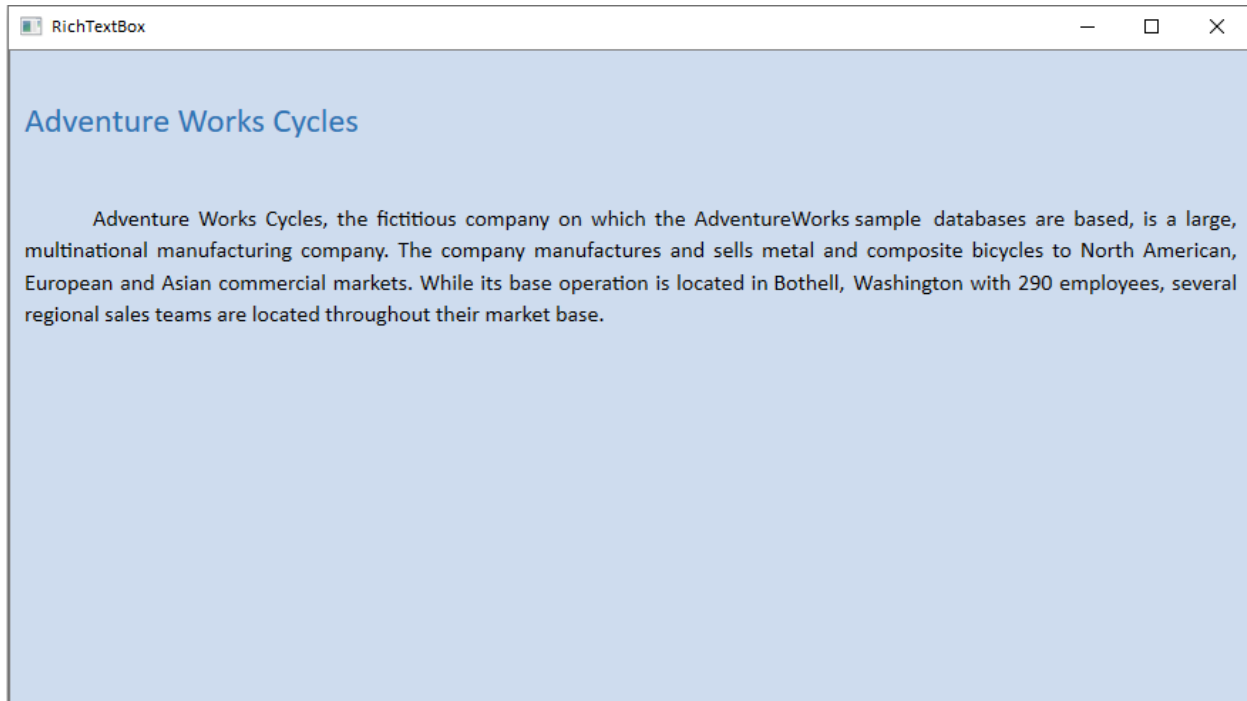


Continuous layout



Block layout

The block layout always inherits the control background color.



How to override the document background in continuous layout type?

By default, the document background properties will be applied when the `LayoutType` is continuous. You can suppress the document background and apply the control background by setting `OverridesDocumentBackground` property to true. The default value of this property is false.

Note: This property is valid only when the `LayoutType` is continuous.

The following code illustrates how to override the document background color.

XML

```
<RichTextBoxAdv:SfRichTextBoxAdv x:Name="richTextBoxAdv"
LayoutType="Continuous" Background="#6699cc"
OverridesDocumentBackground="True" />
```

C#

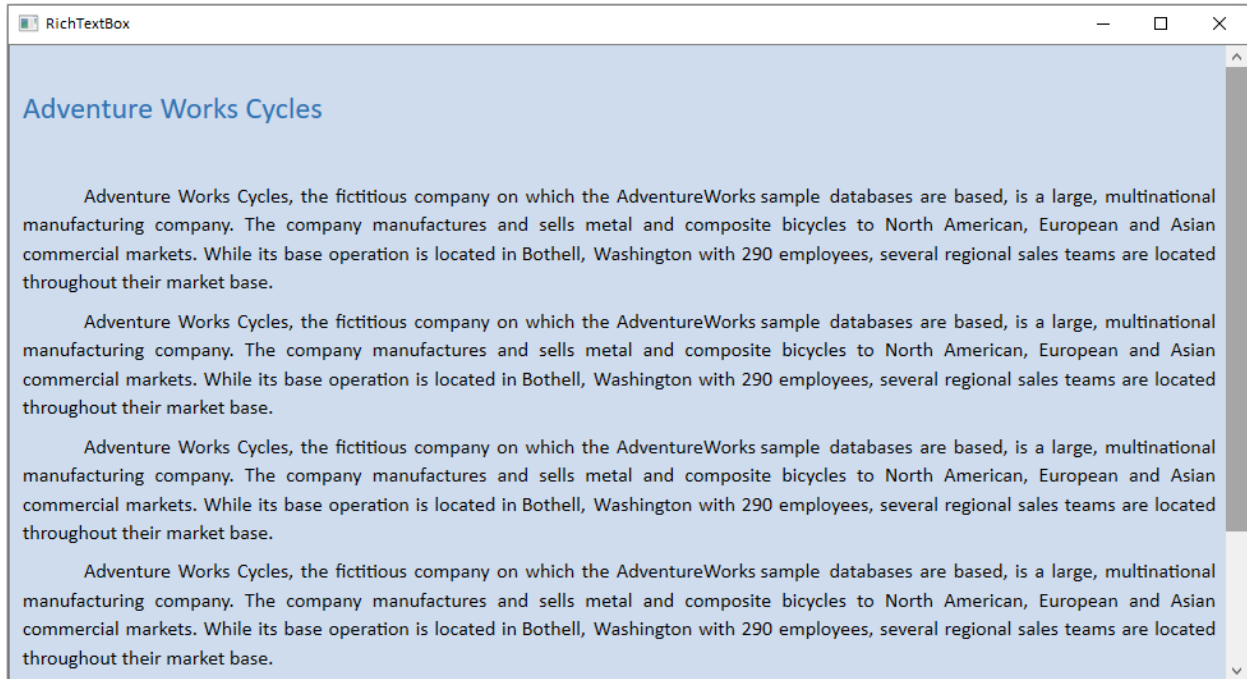
```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
// Sets the control background color
richTextBoxAdv.Background = new SolidColorBrush(Color.FromRgb(102, 153,
204));
// Sets the layout type as continuous
richTextBoxAdv.LayoutType = LayoutType.Continuous;
//Enable the OverridesDocumentBackground property
richTextBoxAdv.OverridesDocumentBackground = true;
```

VB.NET

```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
' Sets the control background color.
```

```
richTextBoxAdv.Background = new SolidColorBrush(Color.FromRgb(102, 153, 204))
' Sets the layout type as continuous
richTextBoxAdv.LayoutType = LayoutType.Continuous
' Enable the OverridesDocumentBackground property
richTextBoxAdv.OverridesDocumentBackground = true
```

Continuous layout:



Setting Background for Document Pages

The RichTextBox control allows you to change background color of the document pages. A background of a document is represented by **Background** property of **DocumentAdv** class. The default value of this property is white.

Note: 1. This property is independent for a document. So the background will change when the document is changed.

2. To maintain same background for all documents, you can reset this property in DocumentChanged event.

The following code illustrates how to apply color as background to the document pages.

C#

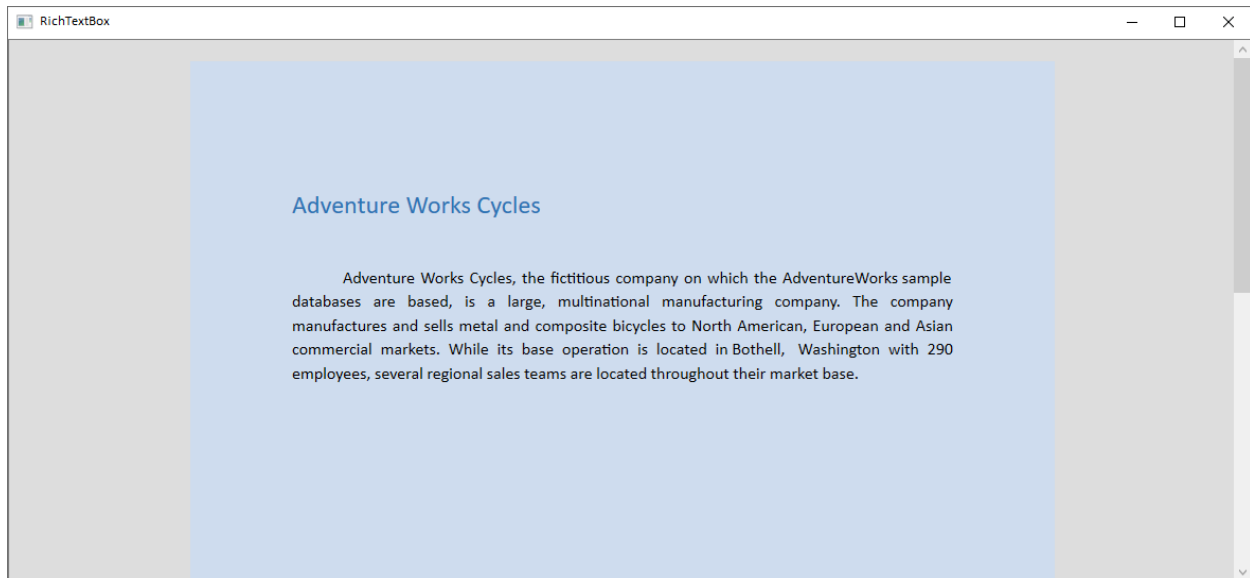
```
// Initializes a new instance of RichTextBoxAdv.
SfRichTextBoxAdv richTextBoxAdv = new SfRichTextBoxAdv();
// Sets the document background color
richTextBoxAdv.Document.Background.Color = Color.FromRgb(102, 153, 204);
```

VB.NET

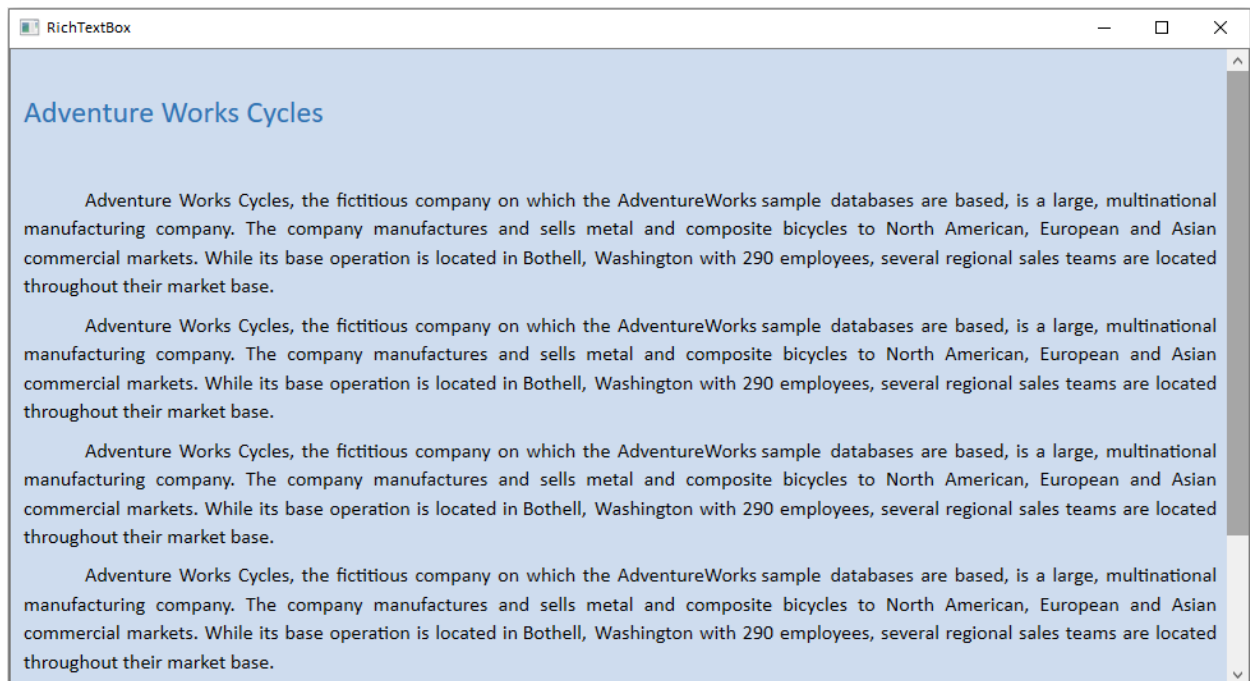
```
' Initializes a new instance of RichTextBoxAdv.
Dim richTextBoxAdv As New SfRichTextBoxAdv()
```

```
' Sets the document background color.  
richTextBoxAdv.Document.Background.Color = Color.FromRgb(102, 153, 204)
```

Pages layout:



Continuous layout:



Note: This API is supported starting from release version v17.4.0.X.

You can also explore our [WPF RichTextBox example](#) to know how to render and configure the editing tools.

FAQ Section

Opening large size documents in WPF SfRichTextBoxAdv control

This page explains Why does out of memory exception throw on opening large size documents in WPF SfRichTextBoxAdv control and more details.

Why does out of memory exception throw on opening large size documents in WPF RichTextBox (SfRichTextBoxAdv)

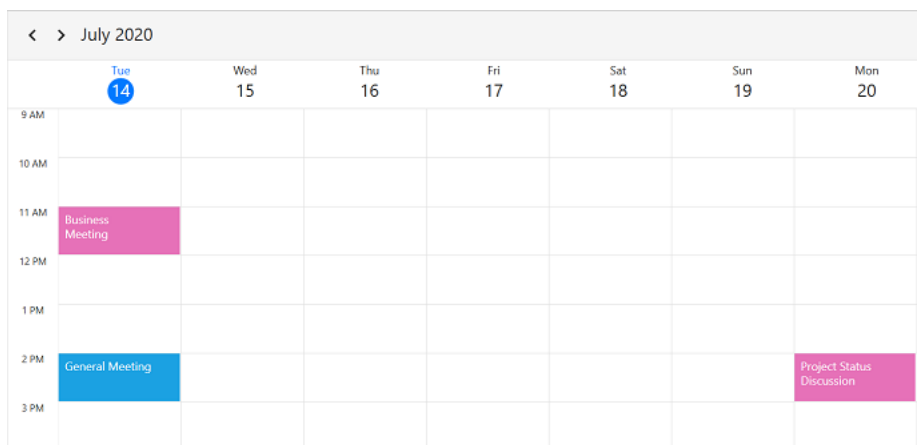
The SfRichTextBoxAdv control keeps the entire rich text content (text, images, tables, and all the other supported elements along with its formatting) of the document and its corresponding information needed for rendering in main memory. In case of opening a DOCX file, you may think that the file size is small and SfRichTextBoxAdv utilizes a very large memory; whereas it is a zip archive file with extension “docx” and SfRichTextBoxAdv control internally decompresses it and populates the content in the document object model by utilizing remarkable amount of main memory.

The SfRichTextBoxAdv control supports UI Virtualization. UI elements are created only for the contents that are visible in the viewer. The UI elements are created for the contents that become visible while scrolling the viewer. This reduces the main memory utilization and also improves UI performance. Even though UI Virtualization is handled, the main memory utilization increases with respect to the increase of content and its complexity. The main memory utilized by an instance will not be released until the instance is removed from the document. So, there is a chance for out of memory exception when the memory utilization exceeds the maximum level as the content of the document increases. In that case, split the contents to several documents or use high configuration machines with extended RAM so as to create or open large size documents comfortably.

SfScheduler

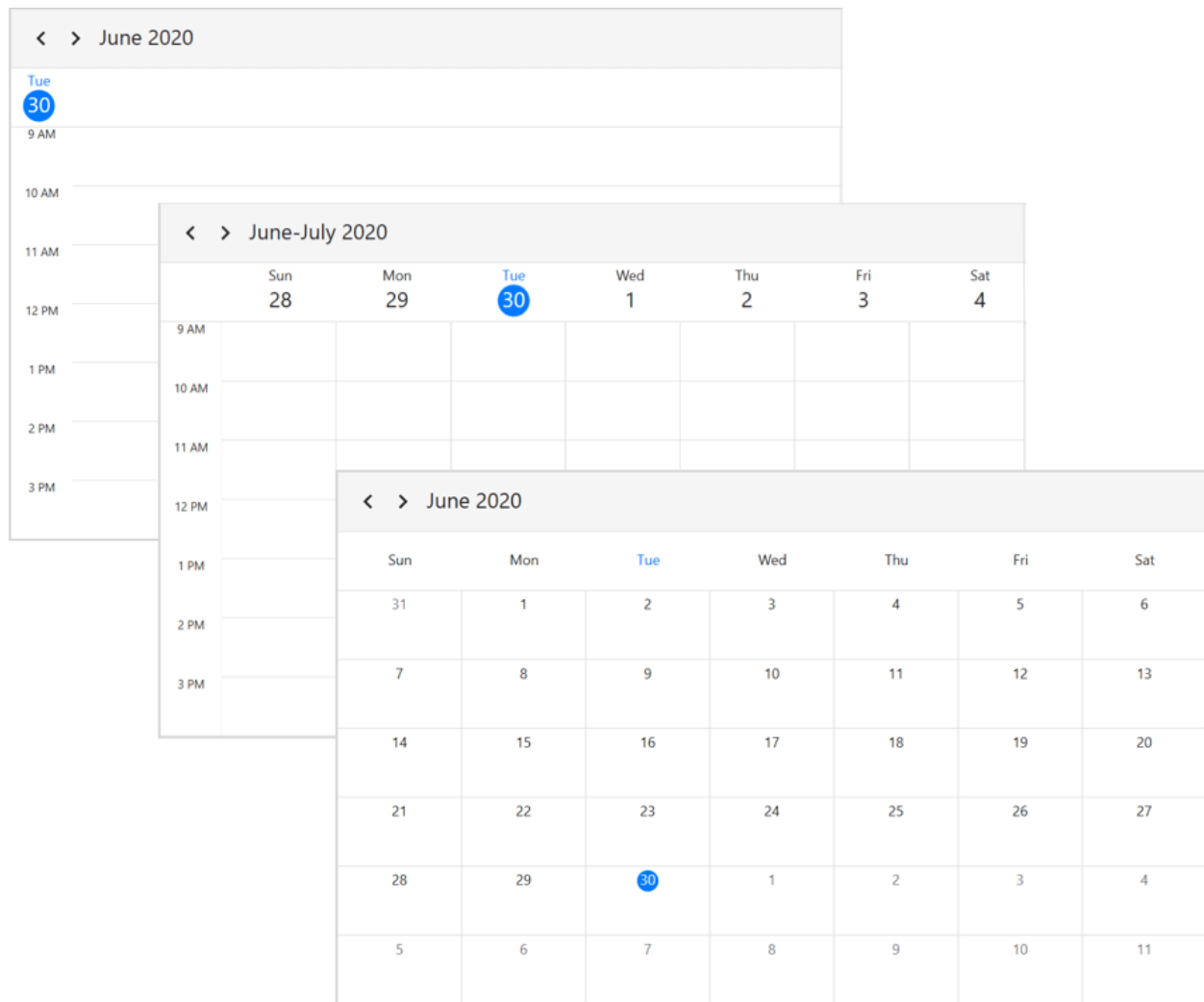
WPF Scheduler (SfScheduler) Overview

The [WPF Scheduler](#) control is used to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.



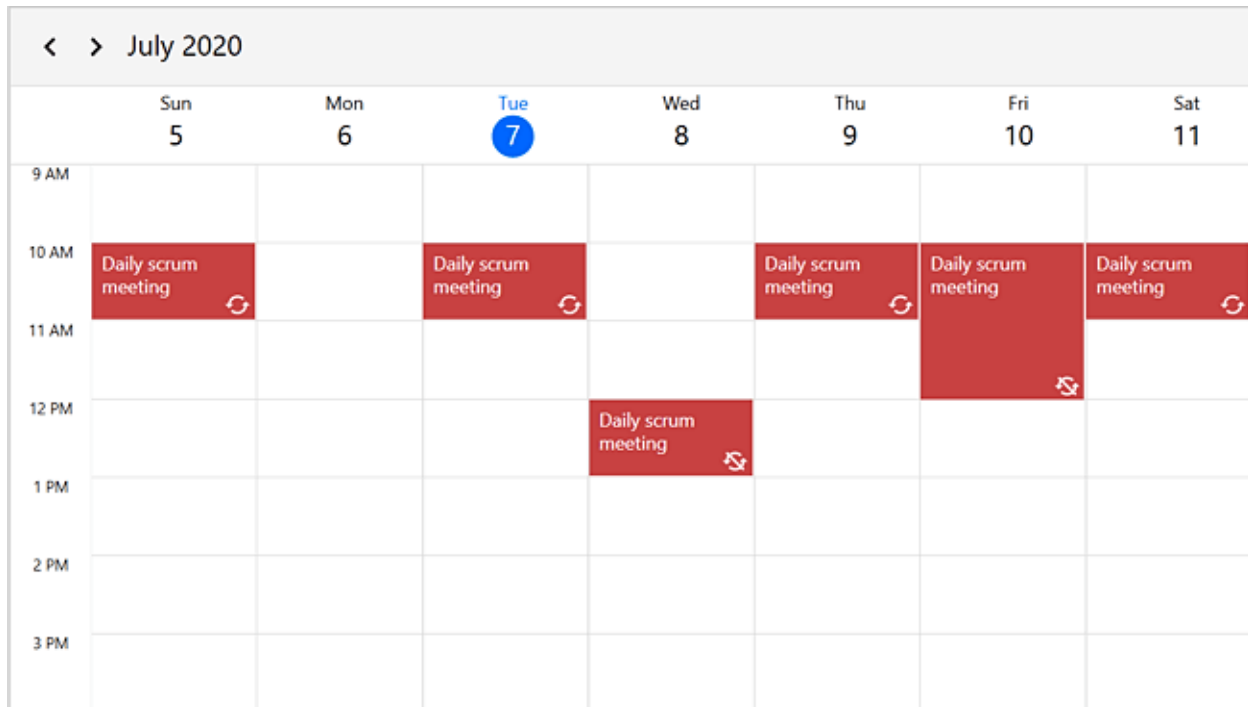
Key features

Built-in Views - The SfScheduler provides eight different types of views such as Day, WorkWeek, Week, TimelineDay, TimelineWeek, TimelineWorkWeek, TimelineMonth, and Month.



Events - Appointments contain information on events scheduled at specific times. In addition to the default appointments, the users can use their own collections to connect a business entity to an appointment by mapping their fields, such as start time, end time, subject, notes, and recurrence.

Recurrence Events — Easily configure recurring events on a daily, weekly, monthly, or yearly basis. Also skip or change the occurrence of a recurring appointment.



Appointment Mapping - SfScheduler control supports to map any collection that implements the IEnumerable interface to populate appointments.

Appointment Editor - Create, edit, or delete appointments using the built-in appointment editor.

Timezone - Display appointments created in various time zones in the system time zone. Appointment start and end times are also automatically adjusted and displayed based on the daylight savings time.

First day of the week - Customize the first day of the week as needed. The default first day is Sunday.

Flexible working days - Customize the workdays in a workweek so that the remaining days will be hidden from view.

Appearance Customization — Provide a unique look to the scheduler with the event appearance customization.

Localization — Display the current date and time by following the globalized date and time formats, and localize all available static texts in the SfScheduler.

Reminder - Use the reminders to organize the appointments in the scheduler. The Scheduler reminds about the appointment at the specified time.

LoadOnDemand - The SfScheduler supports loading appointments on-demand with loading indicator and it improves the loading performance when there are appointments range for multiple years.

Note: You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Getting Started with WPF Scheduler (SfScheduler)

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

Assembly deployment

Refer to the section on [control dependencies](#) for a list of assemblies or NuGet Packages to be used as a guide for using control in any application. Further information on installing the NuGet package can be found in the following link in a WPF application: [How to install nuget packages](#) . Use [Syncfusion Reference Manager](#) to refer the scheduler's dependent assemblies.

Create simple application with SfScheduler

In this section, create WPF application with WPF Scheduler (SfScheduler) control.

- [Creating project](#)
- [Adding control via Designer](#)
- [Adding control manually in XAML](#)
- [Adding control manually in C#](#)

Creating project

In Visual Studio, create a new WPF project to show the features of the WPF Scheduler (SfScheduler) control and add the following namespace to the added assemblies.

Assembly: `Syncfusion.SfScheduler.WPF`

Namespace: `Syncfusion.UI.Xaml.Scheduler`

Adding control via Designer

[SfScheduler](#) control can be added to the application by dragging it from Toolbox and dropping it in a Designer view. The required assembly references will be added automatically.

Adding control manually in XAML

To add the control manually in XAML page, follow the given steps:

1. Add the `Syncfusion.SfScheduler.WPF` assembly reference to the project.
2. Import WPF schema `http://schemas.syncfusion.com/wpf` in the XAML page.
3. Declare the `SfScheduler` control in XAML page.

XML

```
<Window
x:Class="GettingStarted.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:system="clr-namespace:System;assembly=mscorlib"
WindowStartupLocation="CenterScreen">
<Grid>
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Month"/>
</Grid>
</Window>
```

Adding control manually in C#

To add the control manually in C# page, add the `Syncfusion.SfScheduler.WPF` assembly reference to the project.

C#

```
using Syncfusion.UI.Xaml.Scheduler;
namespace GettingStarted
{
    public partial class MainWindow : Window
    {
        SfScheduler schedule = new SfScheduler();
        this.Content = schedule;
    }
}
```

Change different SfScheduler Views

The [WPF Scheduler](#) (SfScheduler) control provides five different types of views to display dates and it can be assigned to the control by using [ViewType](#) property. By default the control is assigned with `MonthView`. Current date will be displayed initially for all the Schedule views.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Month">
```

C#

```
Schedule.ViewType =SchedulerViewType.Month;
```


< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Appointments

[SfScheduler](#) has a built-in capability to handle the appointment arrangement internally based on the [ScheduleAppointment](#) collections. Allocate the collection generated to [Appointments](#) property.

Adding Appointments

The [ScheduleAppointment](#) is a class that includes the specific scheduled appointment. It has some basic properties such as [StartTime](#), [EndTime](#), [Subject](#), and some additional information about the appointment can be added with [Notes](#), [Location](#), and [IsAllDay](#) properties.

XML

```
<schedule:SfSchedule x:Name="schedule" ViewType="Month"/>
```

C#

```
//Creating new event
ScheduleAppointmentCollection appointmentCollection = new
ScheduleAppointmentCollection();
//Creating new event
ScheduleAppointment clientMeeting = new ScheduleAppointment();
DateTime currentDate = DateTime.Now;
DateTime startTime = new DateTime(currentDate.Year, currentDate.Month,
currentDate.Day, 10, 0, 0);
DateTime endTime = new DateTime(currentDate.Year, currentDate.Month,
currentDate.Day, 12, 0, 0);
clientMeeting.StartTime = startTime;
clientMeeting.EndTime = endTime;
clientMeeting.Subject = "ClientMeeting";
appointmentCollection.Add(clientMeeting);
Schedule.ItemsSource = appointmentCollection;
```

Download the entire source code of this demo for WPF from

here [SchedulerGettingStarted](#)

Events/Appointments data mapping

Map the custom appointments data to the scheduler.

Here are the steps to render meetings using [SfScheduler](#) control with respective custom data properties created in a class [Meeting](#).

- [Create an event Data Model](#)
- [Create view model](#)
- [Bind to SfScheduler appointment](#)
- [Bind item source for Scheduler](#)

Create an event Data Model

Create a custom class **Meeting** with mandatory fields **From**, **To** and **EventName** that is used to map the information of the appointment.

C#

```
public class Meeting : INotifyPropertyChanged
{
    DateTime from, to;
    string eventName;
    bool isAllDay;
    string startTimeZone, endTimeZone;
    Brush color;
    public Meeting()
    {
    }
    public DateTime From
    {
        get { return from; }
        set
        {
            from = value;
            RaisePropertyChanged("From");
        }
    }
    public DateTime To
    {
        get { return to; }
        set
        {
            to = value;
            RaisePropertyChanged("To");
        }
    }
    public bool IsAllDay
    {
        get { return isAllDay; }
        set
        {
            isAllDay = value;
            RaisePropertyChanged("IsAllDay");
        }
    }
    public string EventName
    {
        get { return eventName; }
        set
        {
            eventName = value;
            RaisePropertyChanged("EventName");
        }
    }
}
```

```
}  
public string StartTimeZone  
{  
    get { return startTimeZone; }  
    set  
    {  
        startTimeZone = value;  
        RaisePropertyChanged("StartTimeZone");  
    }  
}  
public string EndTimeZone  
{  
    get { return endTimeZone; }  
    set  
    {  
        endTimeZone = value;  
        RaisePropertyChanged("EndTimeZone");  
    }  
}  
public Brush Color  
{  
    get { return color; }  
    set  
    {  
        color = value;  
        RaisePropertyChanged("Color");  
    }  
}  
public event PropertyChangedEventHandler PropertyChanged;  
protected virtual void RaisePropertyChanged(string propertyName, object  
    oldValue = null)  
{  
    this.PropertyChanged?.Invoke(this, new  
        PropertyChangedEventArgs(propertyName));  
}  
}
```

Create view model

By setting **From** and **To** of the **Meeting** class, schedule the meetings for a specific day. Change the subject and color of the appointment using **EventName** and **Color** property. Define the list of custom appointments in a separate class of **ViewModel**.

C#

```
public class ScheduleViewModel  
{  
    private List<string> currentDayMeetings;  
    private List<string> minTimeMeetings;  
    private List<Brush> colorCollection;  
    public ScheduleViewModel()  
    {  
        this.Events = new ObservableCollection<Meeting>();  
        this.InitializeDataForBookings();  
        this.IntializeAppoitments();  
    }  
    public ObservableCollection<Meeting> Events
```

```

{
    get;
    set;
}
private List<Point> GettingTimeRanges()
{
    List<Point> randomTimeCollection = new List<Point>();
    randomTimeCollection.Add(new Point(9, 11));
    randomTimeCollection.Add(new Point(12, 14));
    randomTimeCollection.Add(new Point(15, 17));
    return randomTimeCollection;
}
private void InitializeDataForBookings()
{
    this.currentDayMeetings = new List<string>();
    this.currentDayMeetings.Add("General Meeting");
    this.currentDayMeetings.Add("Plan Execution");
    this.minTimeMeetings = new List<string>();
    this.minTimeMeetings.Add("Client Metting");
    this.minTimeMeetings.Add("Birthday wish alert");
    this.colorCollection = new List<Brush>();
    this.colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF339933")));
    this.colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF00ABA9")));
    this.colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFE671B8")));
    this.colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF1BA1E2")));
    this.colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFD80073")));
}
private void IntializeAppoitments()
{
    Random randomTime = new Random();
    List<Point> randomTimeCollection = this.GettingTimeRanges();
    DateTime date;
    DateTime dateFrom = DateTime.Now.AddDays(-100);
    DateTime dateTo = DateTime.Now.AddDays(100);
    var random = new Random();
    var dateCount = random.Next(4);
    DateTime dateRangeStart = DateTime.Now.AddDays(0);
    DateTime dateRangeEnd = DateTime.Now.AddDays(1);
    for (date = dateFrom; date < dateTo; date = date.AddDays(1))
    {
        if (date.Day % 7 != 0)
        {
            for (int additionalAppointmentIndex = 0; additionalAppointmentIndex < 1;
additionalAppointmentIndex++)
            {
                Meeting meeting = new Meeting();
                int hour =
randomTime.Next((int)randomTimeCollection[additionalAppointmentIndex].X,
(int)randomTimeCollection[additionalAppointmentIndex].Y);
                meeting.From = new DateTime(date.Year, date.Month, date.Day, hour, 0, 0);
                meeting.To = meeting.From.AddHours(1);
                meeting.EventName = this.currentDayMeetings[randomTime.Next(2)];
            }
        }
    }
}

```

```

meeting.Color = this.colorCollection[randomTime.Next(2)];
meeting.IsAllDay = false;
meeting.StartTimeZone = string.Empty;
meeting.EndTimeZone = string.Empty;
this.Events.Add(meeting);
}
}
else
{
Meeting meeting = new Meeting();
meeting.From = new DateTime(date.Year, date.Month, date.Day,
randomTime.Next(9, 11), 0, 0);
meeting.To = meeting.From.AddDays(2).AddHours(1);
meeting.EventName = this.currentDayMeetings[randomTime.Next(2)];
meeting.Color = this.colorCollection[randomTime.Next(2)];
meeting.IsAllDay = true;
meeting.StartTimeZone = string.Empty;
meeting.EndTimeZone = string.Empty;
this.Events.Add(meeting);
}
}
DateTime minDate;
DateTime minDateFrom = DateTime.Now.AddDays(-2);
DateTime minDateTo = DateTime.Now.AddDays(2);
for (minDate = minDateFrom; minDate < minDateTo; minDate =
minDate.AddDays(1))
{
Meeting meeting = new Meeting();
meeting.From = new DateTime(minDate.Year, minDate.Month, minDate.Day,
randomTime.Next(9, 18), 30, 0);
meeting.To = meeting.From;
meeting.EventName = this.minTimeMeetings[randomTime.Next(0, 1)];
meeting.Color = this.colorCollection[randomTime.Next(0, 2)];
meeting.StartTimeZone = string.Empty;
meeting.EndTimeZone = string.Empty;
this.Events.Add(meeting);
}
}
}
}
}

```

Bind to SfScheduler appointment

Map those properties of the **Meeting** class with our WPF Scheduler (SfScheduler) control by using the [AppointmentMapping](#) property.

XML

```

<syncfusion:SfScheduler x:Name="Schedule">
<syncfusion:SfScheduler.AppointmentMapping>
<syncfusion:AppointmentMapping
Subject="EventName"
StartTime="From"
EndTime="To"
AppointmentBackground="Color"
IsAllDay="IsAllDay"
StartTimeZone="StartTimeZone"

```

```
EndTimeZone="EndTimeZone"/>
</syncfusion:SfScheduler.AppointmentMapping>
</syncfusion:SfScheduler>
```

C#

```
AppointmentMapping appointmentMapping = new AppointmentMapping();
appointmentMapping.IsAllDay = "AllDay";
appointmentMapping.StartTime = "From";
appointmentMapping.EndTime = "To";
appointmentMapping.Subject = "Event name";
appointmentMapping.AppointmentBackground = "color";
appointmentMapping.StartTimeZone = "StartTimeZone";
appointmentMapping.EndTimeZone = "EndTimeZone";
Schedule.AppointmentMapping = appointmentMapping;
```

Bind item source for SfScheduler

Create meetings of type `ObservableCollection<Events>` and assign those appointments collection `Events` to the `ItemsSource` property of SfScheduler.

XML

```
<Window.DataContext>
<local:ScheduleViewModel/>
</Window.DataContext>
<syncfusion:SfScheduler x:Name="Schedule"
ItemsSource="{Binding Events}"
ViewType="Month">
</syncfusion:SfScheduler>
```

C#

```
ScheduleViewModel viewModel = new ScheduleViewModel();
Schedule.ItemsSource = viewModel.Events;
```

Note: [View sample in GitHub](#)

Change first day of week

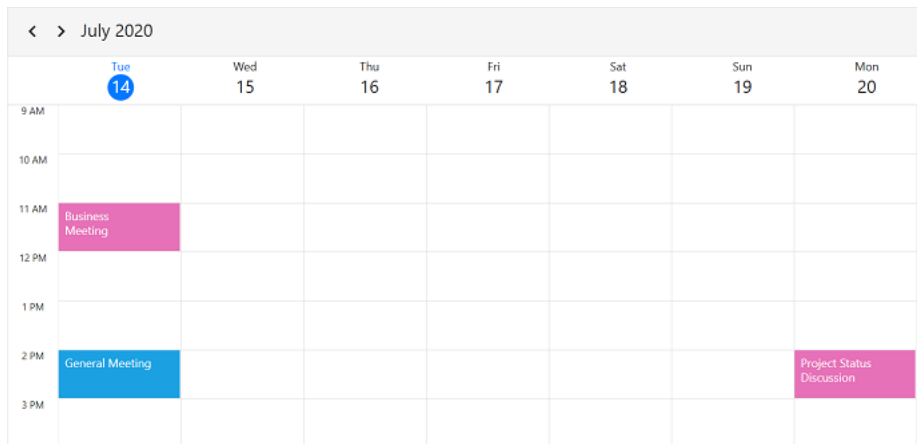
WPF Scheduler (SfScheduler) control will be rendered with `Sunday` as the first day of the week, but it can be customized to any day by using `FirstDayOfWeek` property of `SfScheduler`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" FirstDayOfWeek="Tuesday"/>
```

C#

```
//setting first day of the week
Schedule.FirstDayOfWeek = DayOfWeek.Tuesday;
```

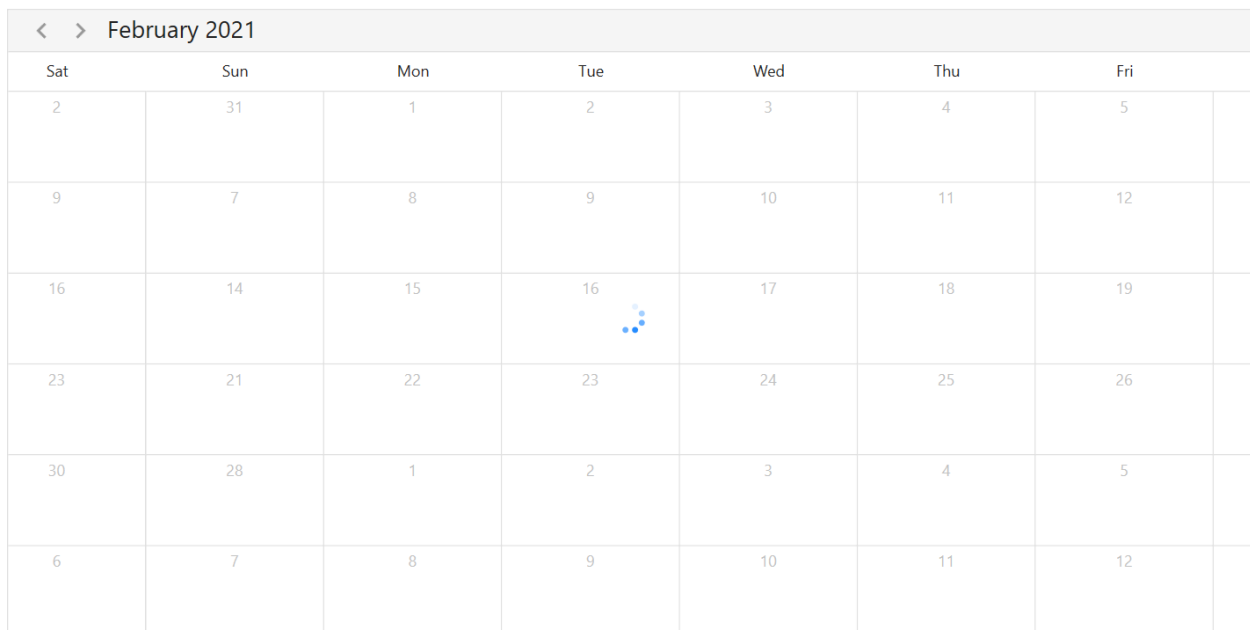


Show busy indicator

The Scheduler supports to show the busy indicator by using the [ShowBusyIndicator](#) property. The default value is set to false, if the value is set to true then the busy indicator will be loaded on view or visible date changed.

XML

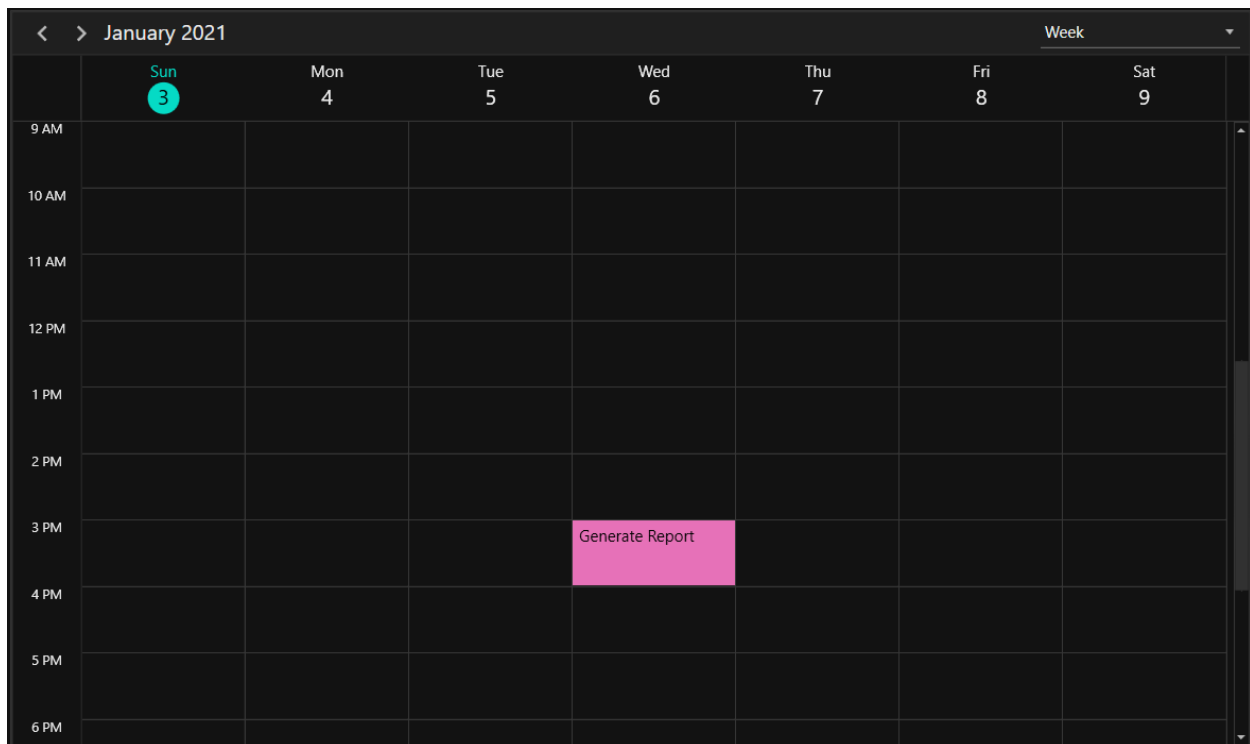
```
<syncfusion:SfScheduler x:Name="Schedule"
ShowBusyIndicator="True"
ViewType="Month">
</syncfusion:SfScheduler>
```



Theme

WPF Scheduler (SfScheduler) supports various built-in themes. Refer to the below links to apply themes for the SfScheduler,

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



Note: You can refer to our [WPF Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Day and Week Views in WPF Scheduler (SfScheduler)

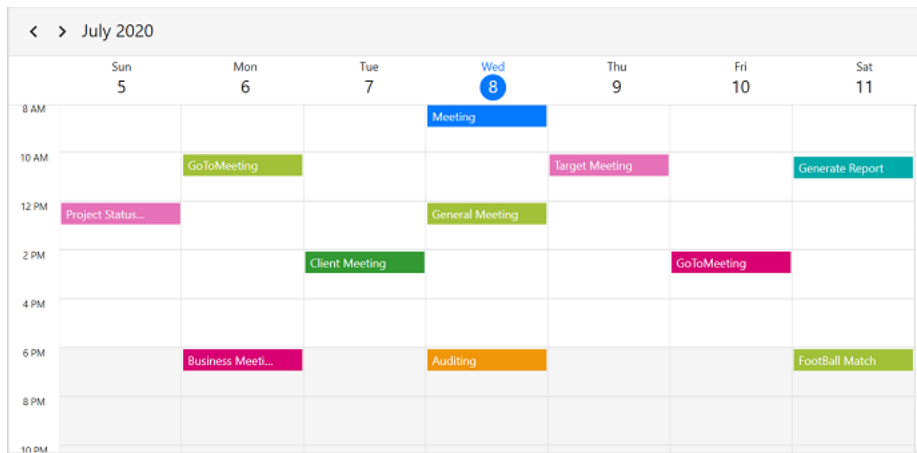
[Scheduler](#) supports to display the day, week, work week views and current day will be visible by default. Appointments on a specific day will be arranged in respective timeslots based on its duration.

Change time interval

Customize the interval of timeslots in all the day, week, work week views by using the [TimeInterval](#) property of [DaysViewSettings](#).

C#

```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.TimeInterval = new System.TimeSpan(0, 120, 0);
```

Note: If the `timeInterval` value is modified (in minutes), the time labels format needs to be changed by setting the `timeFormat` value to `hh:mm`.

Change time interval height

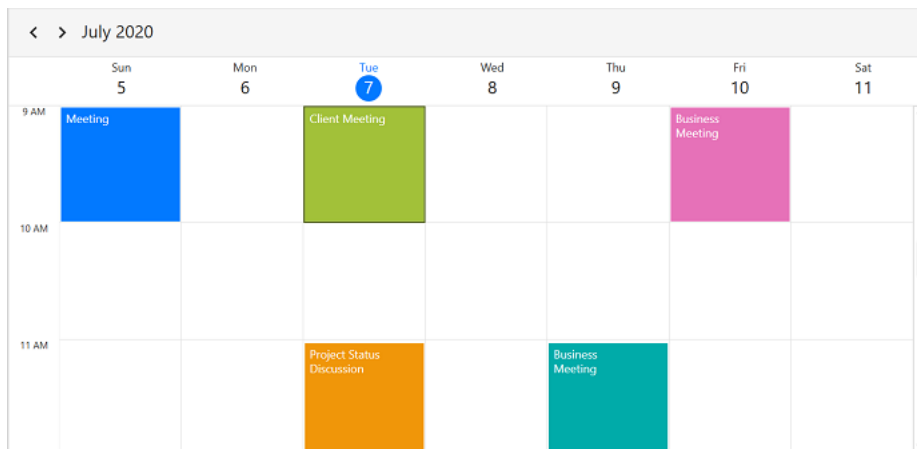
Customize the interval height of timeslots in day, week, work week views by setting `TimeIntervalSize` property of `DaysViewSettings`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
  <syncfusion:SfScheduler.DaysViewSettings>
    <syncfusion:DaysViewSettings
      TimeIntervalSize="120"/>
    </syncfusion:DaysViewSettings>
  </syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.TimeIntervalSize = 120;
```



Flexible working days and working hours

The default values for `StartHour` and `EndHour` are 0 and 24 to show all the timeslots in day, week and work week views. Set the `StartHour` and `EndHour` properties of `DaysViewSettings` to show only the

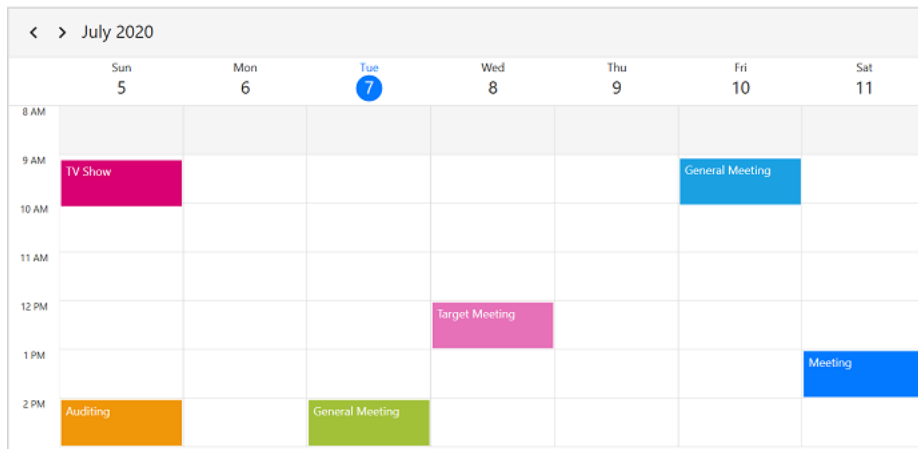
required time duration for users. Set **StartHour** and **EndHour** in time duration to show the required time duration in minutes.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
  <syncfusion:SfScheduler.DaysViewSettings>
    <syncfusion:DaysViewSettings
      StartHour="8"
      EndHour="15" />
    </syncfusion:SfScheduler.DaysViewSettings>
  </syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.StartHour = 8;
Schedule.DaysViewSettings.EndHour = 15;
```



Note:

- The [NonWorkingDays](#) property will be applicable only for **workWeek** and **TimelineWorkWeek** views only, and not applicable for the remaining views.
- Scheduler Appointments UI, which does not fall within the **StartHour** and **EndHour** will not be visible and if it falls partially, it will be clipped.
- No need to specify the decimal point values for **StartHour** and **EndHour**, if you don't want to set the minutes.
- The number of time slots will be calculated based on total minutes of a day and time interval (total minutes of a day ((start hour - end hour) * 60) / time interval).
- If custom **timeInterval** is given, then the number of time slots calculated based on the given **TimeInterval** should result in integer value (total minutes % **timeInterval** = 0), otherwise next immediate time interval that result in integer value when divided by total minutes of a day will be considered. For example, if **TimeInterval** = 2 Hours 15 minutes and total minutes = 1440 (24 Hours per day), then **TimeInterval** will be changed to '144' (1440%144=0) by considering (total minutes % **TimeInterval** = 0), it will return integer value for time slots rendering.

- If the custom `StartHour` and `EndHour` are given, then the number of time slots calculated based on the given `StartHour` and `EndHour` should result in integer value, otherwise next immediate `TimeInterval` will be considered until the result is integer value. For example, if `StartHour` is 9 (09:00AM), `EndHour` is 18.25 (06:15 PM), `TimeInterval` is 30 minutes, and total minutes = 555 ((18.25-9)*60), then the `TimeInterval` will be changed to '37 minutes' (555%37=0) by considering (total minutes % timeInterval = 0). It will return integer value for time slots rendering.

Special time regions

Restrict the user interaction such as selection and highlights specific regions of time in day, week, work week views by adding the `SpecialTimeRegions` property of `SfScheduler`. Set the `StartTime` and `EndTime` properties of `SpecialTimeRegion` to create a `SpecialTimeRegion`, use the `timeZone` property to set the specific timezone for Start and end time of `SpecialTimeRegion`. The `SpecialTimeRegion` will display the text or image on it that set to the text or icon property of `SpecialTimeRegion`.

Enable merges adjacent region of `SpecialTimeRegion` and show them as a single region instead of showing them separately for each day using the `CanMergeAdjacentRegions` property of `SpecialTimeRegion` in the week and workweek views. By default, its value is false.

Selection restriction in timeslots

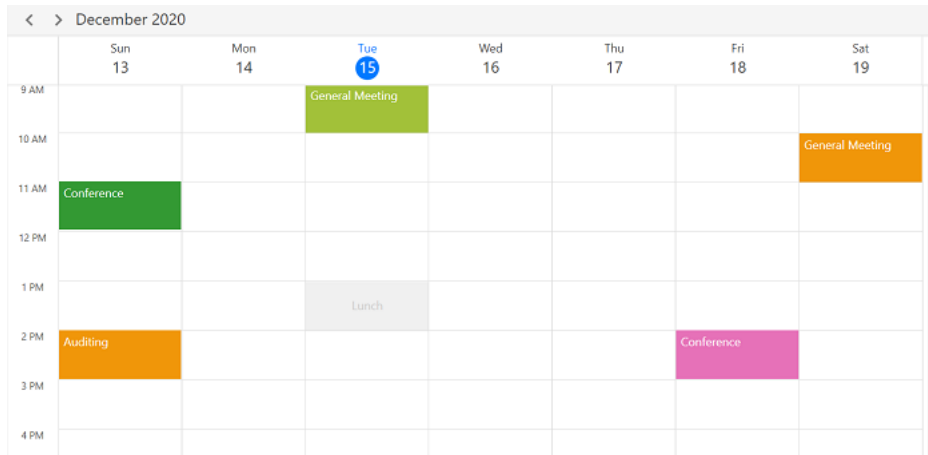
Enable or disable the touch interaction of `SpecialTimeRegion` using the `CanEdit` property of `SpecialTimeRegion`. By default, its value is true.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week" >
  <syncfusion:SfScheduler.DaysViewSettings>
    <syncfusion:DaysViewSettings>
      <syncfusion:DaysViewSettings.SpecialTimeRegions>
        <syncfusion:SpecialTimeRegion
          StartTime="2020/12/15 13:0:0"
          EndTime="2020/12/15 14:0:0"
          CanEdit="False"
          Text="Lunch"
          Background="Gray"
          Foreground="White"/>
      </syncfusion:DaysViewSettings.SpecialTimeRegions>
    </syncfusion:DaysViewSettings>
  </syncfusion:SfScheduler.DaysViewSettings>
</syncfusion:SfScheduler>
```

C#

```
Schedule.DaysViewSettings.SpecialTimeRegions.Add(new SpecialTimeRegion
{
    StartTime = new System.DateTime(2020, 12, 15, 13, 0, 0),
    EndTime = new System.DateTime(2020, 12, 15, 14, 0, 0),
    Text = "Lunch",
    CanEdit = false,
    Background = Brushes.Black,
    Foreground = Brushes.White
});
```

**Note: NOTE**

This property only restricts the interaction on region and it does not restrict the following:

- Programmatic selection (if the user updates the selected date value dynamically)
- Does not clear the selection when the user selects the region and dynamically change

the `CanEdit` property to false

- It does not restrict appointment interaction when the appointment placed

in the region

- It does not restrict the appointment rendering on a region, when appointments are loaded from data services or adding programmatically.

Recurring time region

The recurring time region on a daily, weekly, monthly, or yearly interval. The recurring special time regions can be created by setting the [RecurrenceRule](#) property in [SpecialTimeRegion](#).

Enable merges adjacent region of `SpecialTimeRegion` and show them as a single region instead of showing them separately for each day using the [CanMergeAdjacentRegions](#) property of [SpecialTimeRegion](#) in the week and workweek views. By default, its value is false.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
  <syncfusion:SfScheduler.DaysViewSettings>
    <syncfusion:DaysViewSettings>
      <syncfusion:DaysViewSettings.SpecialTimeRegions>
        <syncfusion:SpecialTimeRegion
          StartTime="2020/12/13 13:0:0"
          EndTime="2020/12/13 14:0:0"
          CanEdit="False"
          Text="Lunch"
          CanMergeAdjacentRegions="True"
          RecurrenceRule="FREQ=DAILY;INTERVAL=1"/>
        </syncfusion:DaysViewSettings.SpecialTimeRegions>
      </syncfusion:DaysViewSettings>
    </syncfusion:SfScheduler.DaysViewSettings>
  </syncfusion:SfScheduler>
```

```

</syncfusion:DaysViewSettings>
</syncfusion:SfScheduler.DaysViewSettings>
</syncfusion:SfScheduler>

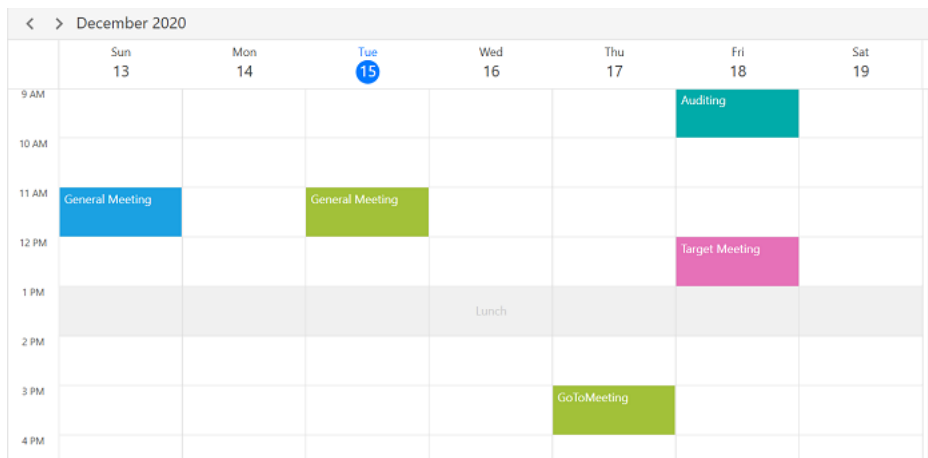
```

C#

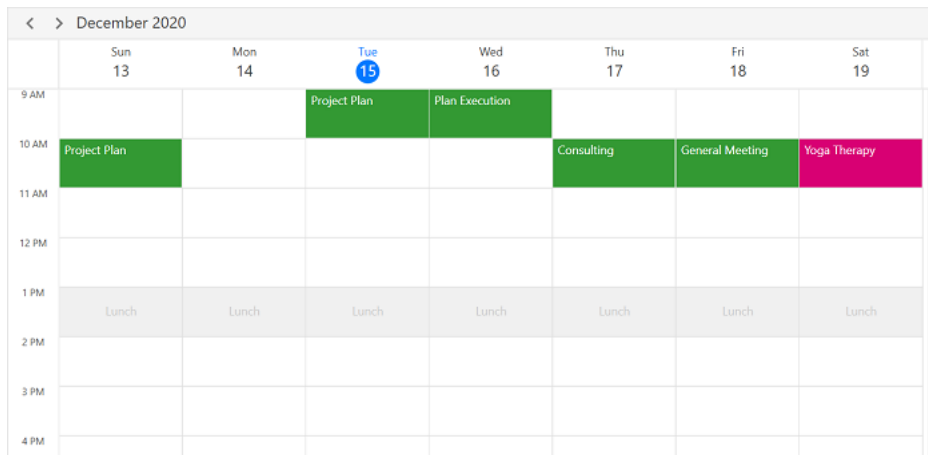
```

Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.SpecialTimeRegions.Add(new SpecialTimeRegion
{
    StartTime = new System.DateTime(2020, 12, 13, 13, 0, 0),
    EndTime = new System.DateTime(2020, 12, 13, 14, 0, 0),
    Text = "Lunch",
    CanEdit = false,
    Background = Brushes.Black,
    Foreground = Brushes.White,
    CanMergeAdjacentRegions=true,
    RecurrenceRule = "FREQ=DAILY;INTERVAL=1"
});

```



If the [CanMergeAdjacentRegions](#) of [SpecialTimeRegion](#) is set to false. The SpecialTimeRegion will be rendering on Date basis.

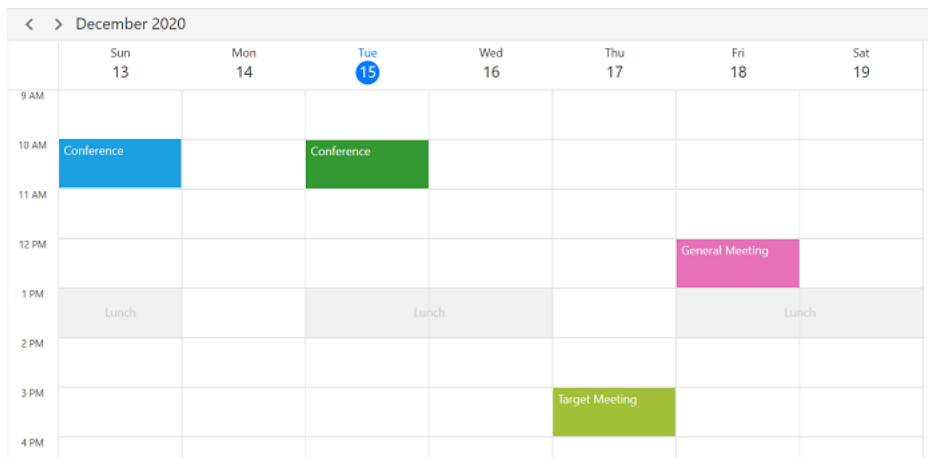


Recurrence exception dates

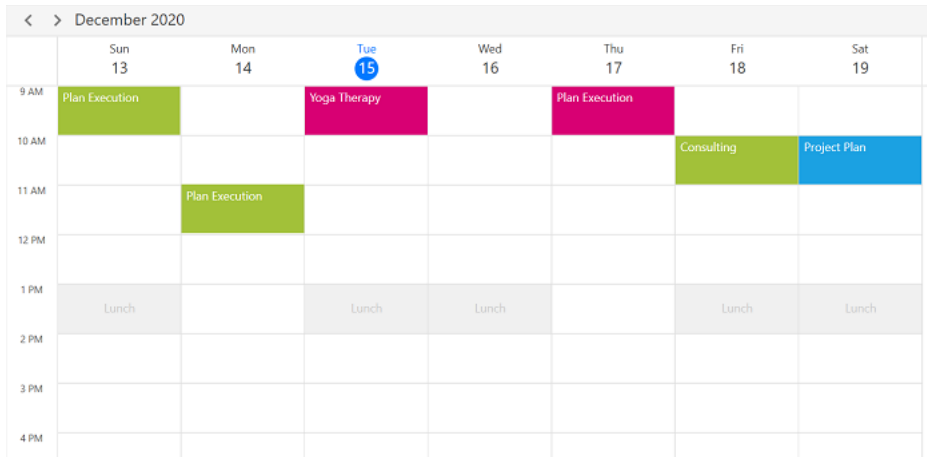
Delete any of the occurrence that is an exception from the recurrence pattern time region by using the [RecurrenceExceptionDates](#) property of [SpecialTimeRegion](#). The deleted occurrence date will be considered as a recurrence exception date.

C#

```
Schedule.ViewType = SchedulerViewType.Week;
DateTime recurrenceExceptionDates = DateTime.Now.Date.AddDays(-1);
DateTime recurrenceExceptionDates1 = DateTime.Now.Date.AddDays(2);
Schedule.DaysViewSettings.SpecialTimeRegions.Add(new SpecialTimeRegion
{
    StartTime = new System.DateTime(2020, 12, 13, 13, 0, 0),
    EndTime = new System.DateTime(2020, 12, 13, 14, 0, 0),
    Text = "Lunch",
    CanEdit = false,
    RecurrenceRule = "FREQ=DAILY; INTERVAL=1",
    CanMergeAdjacentRegions= true,
    Background = Brushes.Black,
    Foreground = Brushes.White,
    RecurrenceExceptionDates = new ObservableCollection<DateTime>()
    {
        recurrenceExceptionDates,
        recurrenceExceptionDates1,
    }
});
```



The [SpecialTimeRegion](#) in Date basis by setting the value of [CanMergeAdjacentRegions](#) is false.



Special time region customization

The `SpecialTimeRegion` background color can be customized by using the `Background` and `SpecialTimeRegionTemplate` properties of `SpecialTimeRegion` that is used to customize the text style for the image of the `SpecialTimeRegion`.

XML

```
<Window.Resources>
<DataTemplate x:Key="specialRegionTemplate">
<Grid Background="{Binding Background}"
Opacity="0.5"
HorizontalAlignment="Stretch"
VerticalAlignment="Stretch">
<Path x:Name="Fork" Data="M11,0 C11.553001,0 12,0.4469986 12,1 L12,15
C12,15.553001 11.553001,16 11,16 10.446999,16 10,15.553001 10,15 L10,7 9,7
C8.4469986,7 8,6.5530014 8,6 L8,3 C8,1.3429985 9.3429985,0 11,0 z M0,0 L1,0
1.2340002,4 1.7810001,4 2,0 3,0 3.2340002,4 3.7810001,4 4,0 5,0 5,4
C5,4.9660001 4.3140001,5.7727499 3.4029064,5.9593439 L3.4007993,5.9597201
3.9114671,14.517 C3.9594617,15.321 3.3195295,16 2.5136147,16 L2.5076156,16
C1.6937013,16 1.0517693,15.309 1.1107631,14.497 L1.7400641,5.9826035
1.6955509,5.9769421 C0.73587513,5.8301721 0,5.0005002 0,4 z" Fill="Black"
HorizontalAlignment="Center" Height="16" Stretch="Fill"
VerticalAlignment="Center" Width="12"/>
</Grid>
</DataTemplate>
</Window.Resources>
```

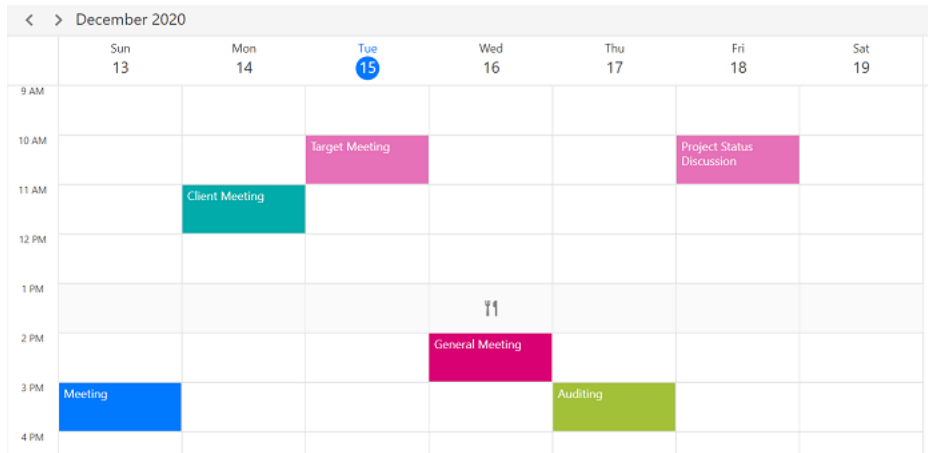
XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
<syncfusion:SfScheduler.DaysViewSettings>
<syncfusion:DaysViewSettings SpecialTimeRegionTemplate="{StaticResource
specialRegionTemplate}">
<syncfusion:DaysViewSettings.SpecialTimeRegions>
<syncfusion:SpecialTimeRegion
StartTime="2020/12/13 13:0:0"
EndTime="2020/12/13 14:0:0"
CanEdit="False"
RecurrenceRule="FREQ=DAILY;INTERVAL=1"
CanMergeAdjacentRegions="True"
```

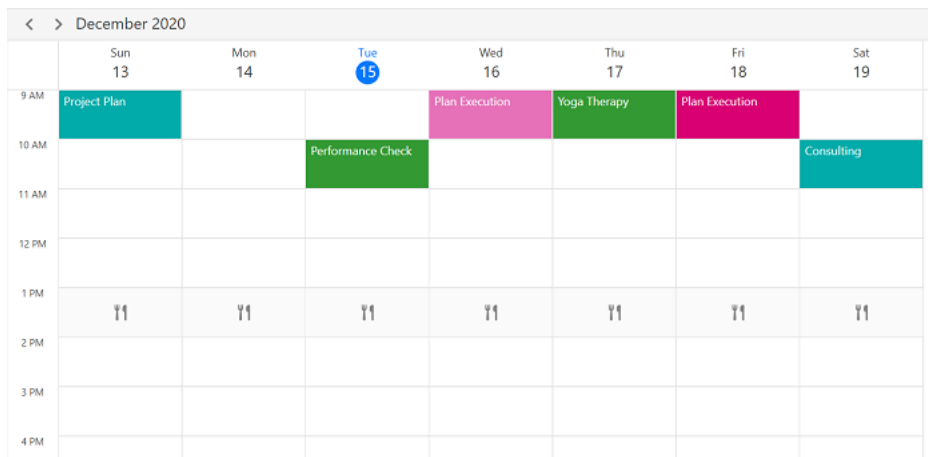
```

Foreground="Black"
Background="#FFF5F5F5"/>
</syncfusion:DaysViewSettings.SpecialTimeRegions>
</syncfusion:DaysViewSettings>
</syncfusion:SfScheduler.DaysViewSettings>

```



The [SpecialTimeRegion](#) can be customized in a Date basis by setting the value of [CanMergeAdjacentRegions](#) is false.



Full screen scheduler

Scheduler time interval height can be adjusted based on screen height by changing the value of [TimeIntervalSize](#) property to -1. It will auto-fit to the screen height and width.

XML

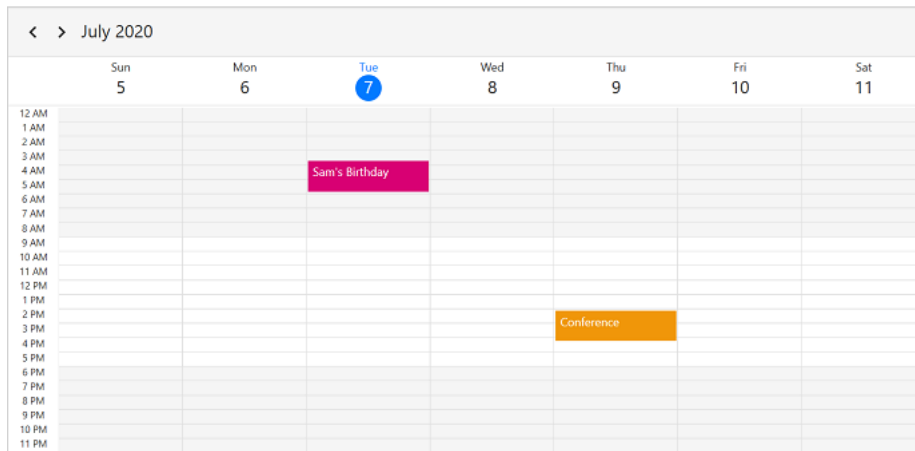
```

<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
<syncfusion:SfScheduler.DaysViewSettings>
<syncfusion:DaysViewSettings
TimeIntervalSize="-1"/>
</syncfusion:SfScheduler.DaysViewSettings>
</syncfusion:SfScheduler>

```

C#


```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.TimeIntervalSize = -1;
```



Change time ruler size

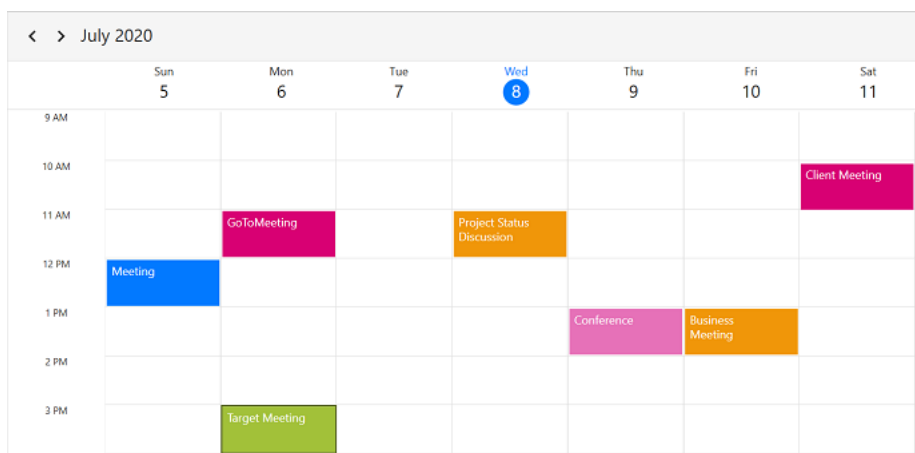
Customize the size of the time ruler view where the labels mentioning the time are placed by using the [TimeRulerSize](#) property of `DayViewSettings`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week" >
  <syncfusion:SfScheduler.DaysViewSettings>
    <syncfusion:DaysViewSettings
      TimeRulerSize="100">
    </syncfusion:DaysViewSettings>
  </syncfusion:SfScheduler.DaysViewSettings>
</syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.TimeRulerSize = 100;
```



Minimum appointment duration

The [MinimumAppointmentDuration](#) property in [DayViewSettings](#) is to set an arbitrary height to appointments when it has minimum duration in day, week, work week views, so that the subject can be readable.

C#

```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.MinimumAppointmentDuration = new
System.TimeSpan(0, 120, 0);
```

Note: NOTE

- The [MinimumAppointmentDuration](#) value will be set, when an appointment duration value lesser than [MinimumAppointmentDuration](#).
- Appointment duration value will be set, when the appointment duration value greater than [MinimumAppointmentDuration](#).
- [TimeInterval](#) value will be set, when [MinimumAppointmentDuration](#) greater than [TimeInterval](#) with lesser appointment duration.
- All day Appointment does not support [MinimumAppointmentDuration](#).

Minimum display appointments count in all day panel

You can customize the number of appointments displayed in an all-day panel using the [MinimumAllDayAppointmentsCount](#) property of [DaysViewSettings](#) in the Scheduler. By default, the appointment display count is 2, and all-day panels have more than 2 appointments, two appointments will be displayed and the remaining appointments will be displayed as appointment counts.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
<syncfusion:SfScheduler.DaysViewSettings>
<syncfusion:DaysViewSettings MinimumAllDayAppointmentsCount="3"/>
</syncfusion:SfScheduler.DaysViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Week;
this.Schedule.DaysViewSettings.MinimumAllDayAppointmentsCount = 3;
```

Customize more appointments indicator in all day panel

You can customize the default appearance of more appointments indicator in the all-day panel by using the [AllDayMoreAppointmentsIndicatorTemplate](#) property of [DaysViewSettings](#).

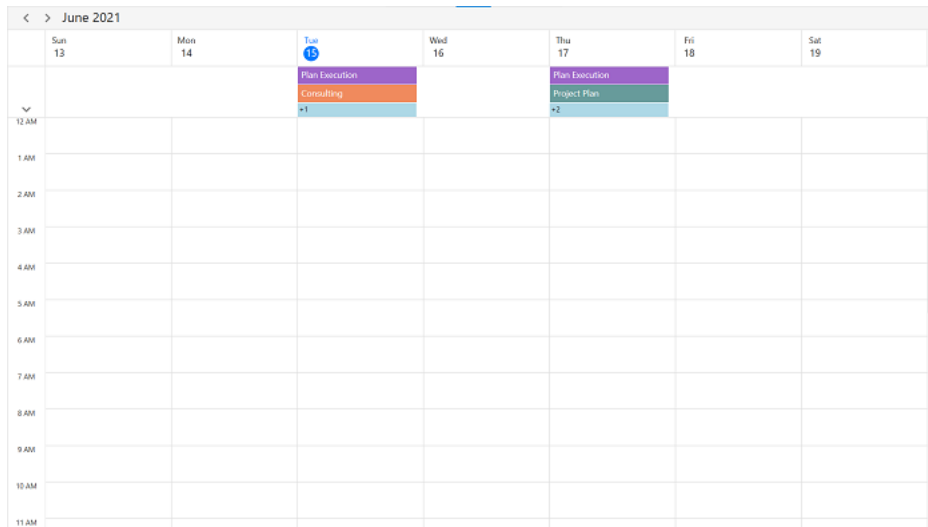
XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
<syncfusion:SfScheduler.DaysViewSettings>
<syncfusion:DaysViewSettings>
<syncfusion:DaysViewSettings.AllDayMoreAppointmentsIndicatorTemplate>
<DataTemplate >
```

```

<TextBlock Text="{Binding StringFormat=+{0}}" Background="LightBlue"
Foreground="Black"
HorizontalAlignment="Stretch" TextAlignment="Left"
VerticalAlignment="Stretch" Padding="2,3,0,0">
</TextBlock>
</DataTemplate>
</syncfusion:DaysViewSettings.AllDayMoreAppointmentsIndicatorTemplate>
</syncfusion:DaysViewSettings>
</syncfusion:SfScheduler.DaysViewSettings>
</syncfusion:SfScheduler>

```



Time text formatting

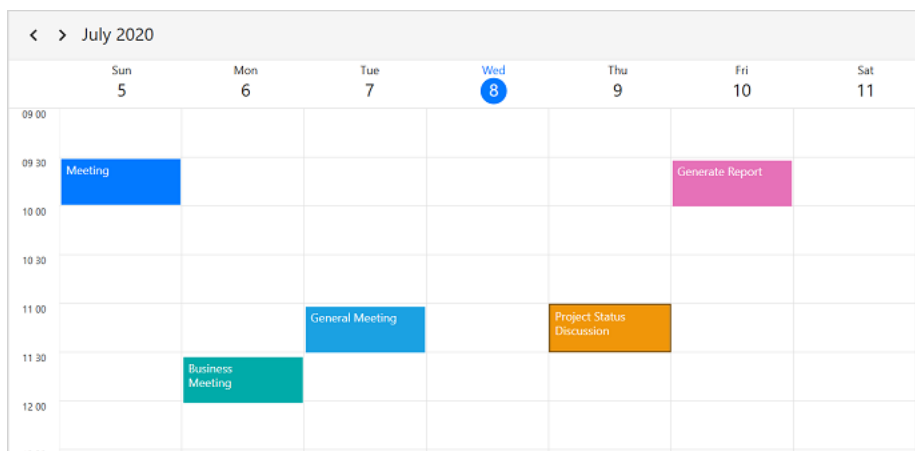
Customize the format for the labels mentioning the time, by setting the [TimeRulerFormat](#) property of [DayViewSettings](#) in [Scheduler](#).

C#

```

Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.TimeRulerFormat = "hh mm";
Schedule.DaysViewSettings.TimeInterval = new System.TimeSpan(0, 30, 0);

```



Note:

- You can customize the appointment editor time format based on scheduler time ruler format and culture.
- By default, the scheduler time ruler format is `h tt` and the appointment editor time picker format is `hh:mm tt`.

View header

Customize the default appearance of view header in day, week, work week views by setting [ViewHeaderDateFormat](#), [ViewHeaderHeight](#), [ViewHeaderDayFormat](#) and [ViewHeaderTemplate](#) of [DaysViewSettings](#).

View header text formatting

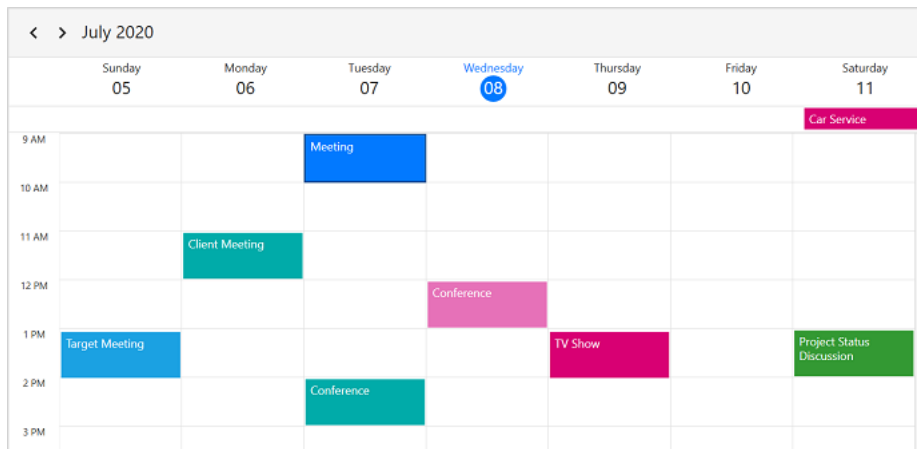
Customize the date and day format of ViewHeader by using the [ViewHeaderDateFormat](#) and [ViewHeaderDayFormat](#) properties of [DaysViewSettings](#).

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
  <syncfusion:SfScheduler.DaysViewSettings>
    <syncfusion:DaysViewSettings
      ViewHeaderDayFormat="dddd"
      ViewHeaderDateFormat="dd"/>
    </syncfusion:SfScheduler.DaysViewSettings>
  </syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.ViewHeaderDateFormat = "dd";
Schedule.DaysViewSettings.ViewHeaderDayFormat = "dddd";
```



View header height

Customize the height of the ViewHeader in a day, week, work week views by setting [ViewHeaderHeight](#) property of [DaysViewSettings](#) in [SfScheduler](#).

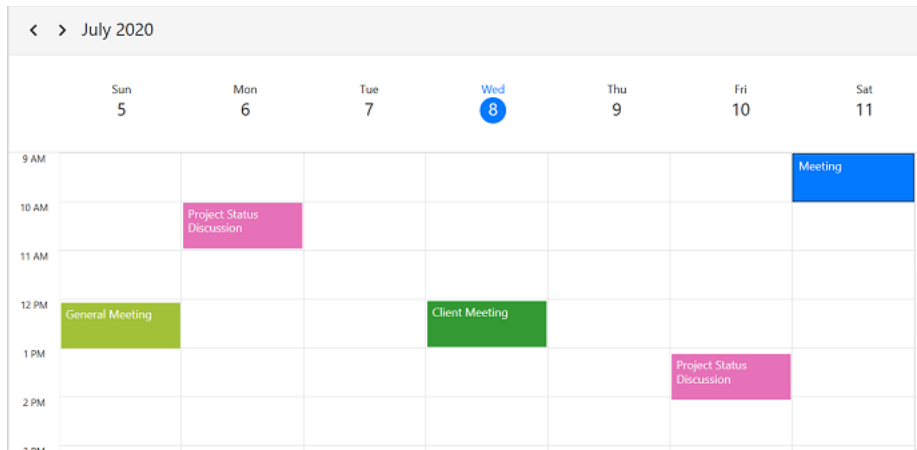
XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
  <syncfusion:SfScheduler.DaysViewSettings>
```

```
<syncfusion:DaysViewSettings
ViewHeaderHeight="100"/>
</syncfusion:SfScheduler.DaysViewSettings>
</syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DaysViewSettings.ViewHeaderHeight = 100;
```

*View header appearance customization*

Customize the default appearance of view header by setting **ViewHeaderTemplate** property of **DaysViewSettings** in **SfScheduler**.

XML

```
<Window.Resources>
<DataTemplate x:Key="viewHeaderTemplate">
<StackPanel Background="Green"
Width="2000"
VerticalAlignment="Stretch"
HorizontalAlignment="Stretch"
Orientation="Vertical">
<TextBlock
HorizontalAlignment="Left"
Margin="20,0,0,0"
Foreground="#FFFFFF"
FontFamily="Arial"
Text="{Binding DateText}"
FontSize="25"
TextTrimming="CharacterEllipsis"
TextWrapping="Wrap" />
<TextBlock
HorizontalAlignment="Left" Margin="20,0,0,0"
Foreground="#FFFFFF"
FontFamily="Arial"
Text="{Binding DayText}"
FontSize="10"
TextTrimming="CharacterEllipsis"
TextWrapping="Wrap" />
```

```

</StackPanel>
</DataTemplate>
</Window.Resources>

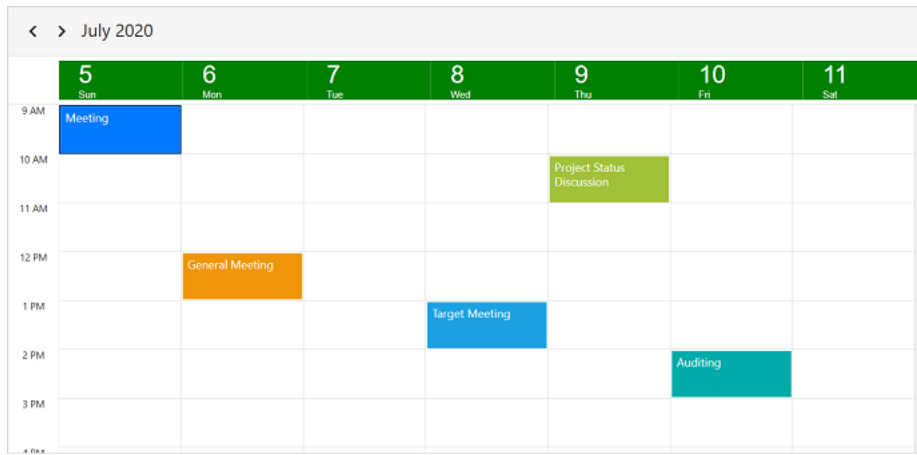
```

XML

```

<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
<syncfusion:SfScheduler.DaysViewSettings>
<syncfusion:DaysViewSettings
ViewHeaderTemplate="{StaticResource viewHeaderTemplate}" />
</syncfusion:SfScheduler.DaysViewSettings>
</syncfusion:SfScheduler>

```



Note: You can refer to our [WPF Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Timeline Views in WPF Scheduler (SfScheduler)

The **TimelineView** displays the dates in horizontal time axis with the desired day's count. Scheduler supports to display the **TimelineDay**, **TimelineWeek**, **TimelineWorkWeek**, and **TimelineMonth** views. See the past or future dates by scrolling to the right or left. Each view displays the events accurately across the time slots with an intuitive drag-and-drop feature. It provides the support to highlight the selected region of time slots and handle the interaction.

Change time interval

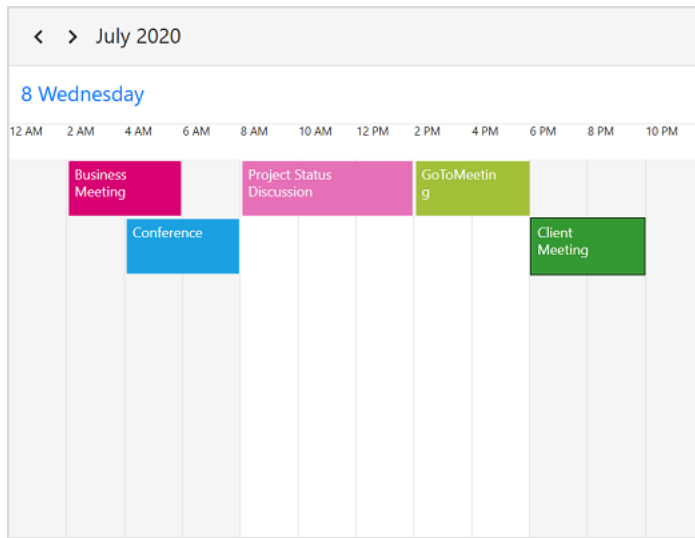
Customize the interval of timeslots in the timeline views by using the [TimeInterval](#) property of [TimelineViewSettings](#). This property will be applicable to **TimelineDay**, **TimelineWeek**, and **TimelineWorkWeek** views.

C#

```

Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.TimeInterval = new TimeSpan(0, 120, 0);

```



Note: If the `timeInterval` value (in minutes) is modified, change the time labels format by setting the `timeFormat` value to `hh:mm`.

Change time interval width

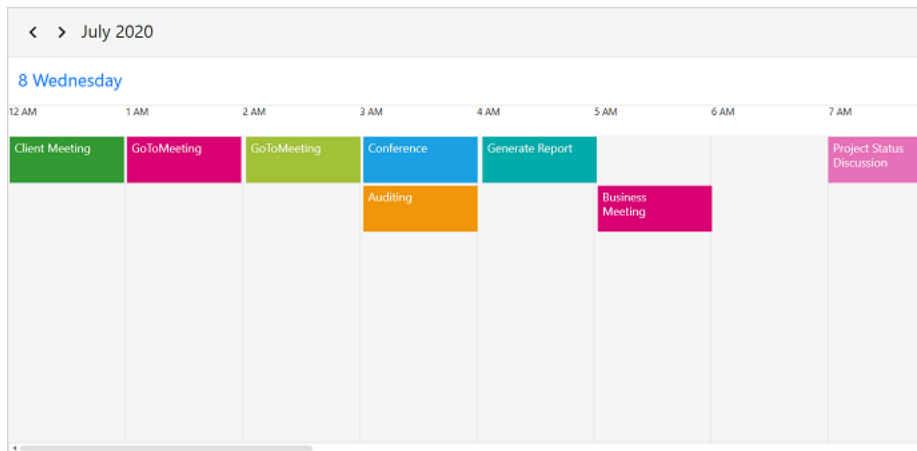
Customize the interval width of timeslots in the Timeline views by setting the [TimeIntervalSize](#) property of `TimelineViewSettings`. This property will be applicable to all timeline views. By default, its value is fifty for the `TimelineDay`, `TimelineWeek`, and `TimelineWorkWeek` views and 150 for `TimelineMonth` view.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    ViewType="TimelineWeek">
    <syncfusion:SfScheduler.TimelineViewSettings>
    <syncfusion:TimelineViewSettings
        TimeIntervalSize="120"/>
    </syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.TimeIntervalSize = 120;
```



Flexible working days and working hours

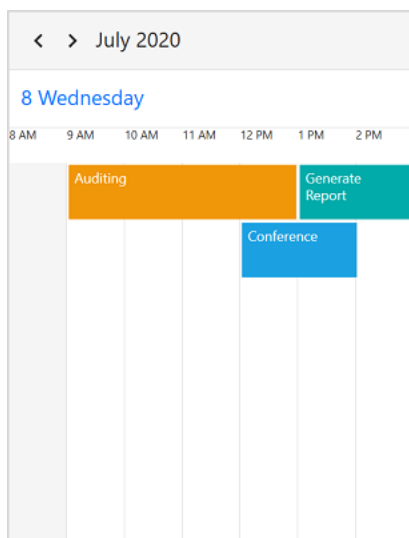
The default values for the [StartHour](#) and [EndHour](#) are 0 and 24 to show all the time slots in a timeline view. Set the [StartHour](#) and [EndHour](#) properties of [TimelineViewSettings](#) to show only the required time duration for users. Set the [StartHour](#) and [EndHour](#) in time duration to show the required time duration in minutes. The [StartHour](#) and [EndHour](#) properties are not applicable to the [TimelineMonth](#) view.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWeek">
  <syncfusion:SfScheduler.TimelineViewSettings>
    <syncfusion:TimelineViewSettings
      StartHour="8"
      EndHour="15"/>
    </syncfusion:SfScheduler.TimelineViewSettings>
  </syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.StartHour = 8;
Schedule.TimelineViewSettings.EndHour = 15;
```


**Note:**

- The [NonWorkingDays](#) property will be applicable only for `workWeek` and `TimelineWorkWeek` views only, and not applicable for the remaining views.
- Scheduler Appointments UI, which does not fall within the `StartHour` and `EndHour` will not be visible and if it falls partially, it will be clipped.
- No need to specify the decimal point values for `StartHour` and `EndHour`, if you don't want to set the minutes.
- The number of time slots will be calculated based on the total minutes of a day and time interval ($\text{total minutes of a day} ((\text{start hour} - \text{end hour}) * 60) / \text{time interval}$).
- If the custom `timeInterval` is given, then the number of time slots calculated based on the given `TimeInterval` should result in integer value ($\text{total minutes} \% \text{timeInterval} = 0$), otherwise next immediate time interval that result in the integer value divided by the total minutes of a day will be considered. For example, if `TimeInterval` = 2 Hours 15 minutes and total minutes = 1440 (24 Hours per day), then `TimeInterval` will be changed to '144' ($1440 \% 144 = 0$) by considering ($\text{total minutes} \% \text{TimeInterval} = 0$), it will return integer value for time slots rendering.
- If the custom `StartHour` and `EndHour` are given, then the number of time slots calculated based on given `StartHour` and `EndHour` should result in the integer value, otherwise next immediate `TimeInterval` will be considered until the result is the integer value. For example, if `StartHour` is 9 (09:00AM), `EndHour` is 18.25 (06:15 PM), `TimeInterval` is 30 minutes, and total minutes = 555 ($(18.25 - 9) * 60$), then the `TimeInterval` will be changed to '37 minutes' ($555 \% 37 = 0$) by considering ($\text{total minutes} \% \text{timeInterval} = 0$). It will return the integer value for time slots rendering.

Change days count

Change the day's count of timeslots in the timeline view by setting the [DaysCount](#) property of `TimelineViewSettings`. This property is only applicable for the `TimelineDay` view. By default, it's value is set to 1.

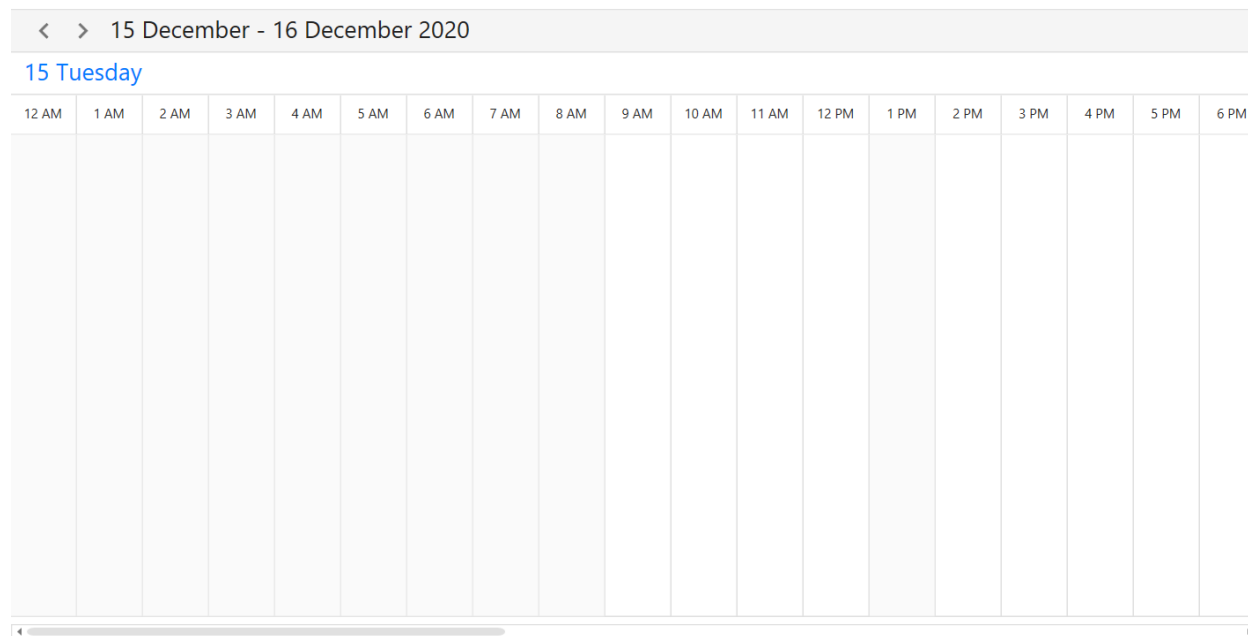
XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineDay">
  <syncfusion:SfScheduler.TimelineViewSettings>
```

```
<syncfusion:TimelineViewSettings
DaysCount="2"/>
</syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>
```

C#

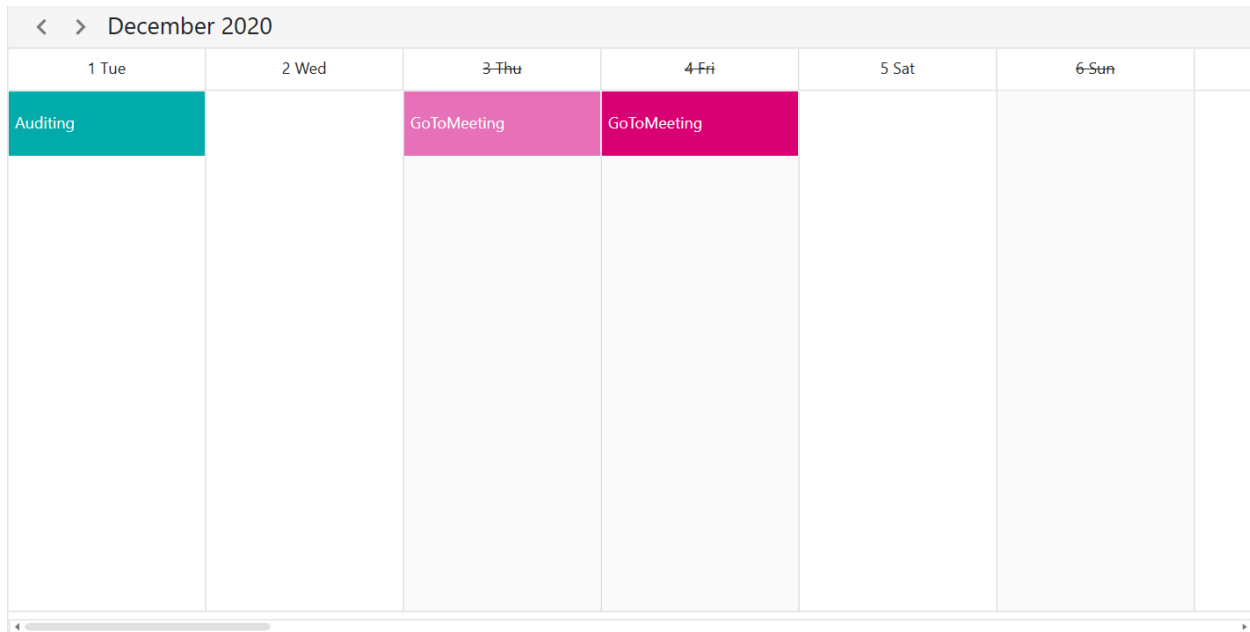
```
Schedule.ViewType = SchedulerViewType.TimelineDay;
Schedule.TimelineViewSettings.DaysCount = 2;
```

**Blackout dates**

Disable the interaction for certain dates in the scheduler **TimelineMonth** view by adding those specific dates to the [BlackoutDates](#) collection property of **SfScheduler**. Using this, allocate or restrict the specific dates for predefined events. This property is not applicable to the **TimelineDay**, **TimelineWeek**, and **TimelineWorkWeek** views.

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.BlackoutDates = GetBlackoutDates();
private ObservableCollection<DateTime> GetBlackoutDates()
{
    var blackoutDateCollection = new ObservableCollection<DateTime>()
    {
        DateTime.Now.Date.AddDays(1),
        DateTime.Now.Date.AddDays(3),
        DateTime.Now.Date.AddDays(5),
        DateTime.Now.Date.AddDays(7),
        DateTime.Now.Date.AddDays(9)
    };
    return blackoutDateCollection;
}
```



Special time regions

Restrict the user interaction such as selection and highlights specific regions of time in the timeline views by adding the [SpecialTimeRegions](#) property of [SfScheduler](#). Set the [StartTime](#) and [EndTime](#) properties of [SpecialTimeRegion](#) to create a [SpecialTimeRegion](#), use the [timeZone](#) property to set the specific timezone for the start and end time of the [SpecialTimeRegion](#). The [SpecialTimeRegion](#) will display the text or image on it that is set to the text or icon property of [SpecialTimeRegion](#). This property will be applicable to the [TimelineDay](#), [TimelineWeek](#), and [TimelineWorkWeek](#) views.

Selection restriction in timeslots

Enable or disable the touch interaction of [SpecialTimeRegion](#) using the [CanEdit](#) property of [SpecialTimeRegion](#). By default, it's value is true.

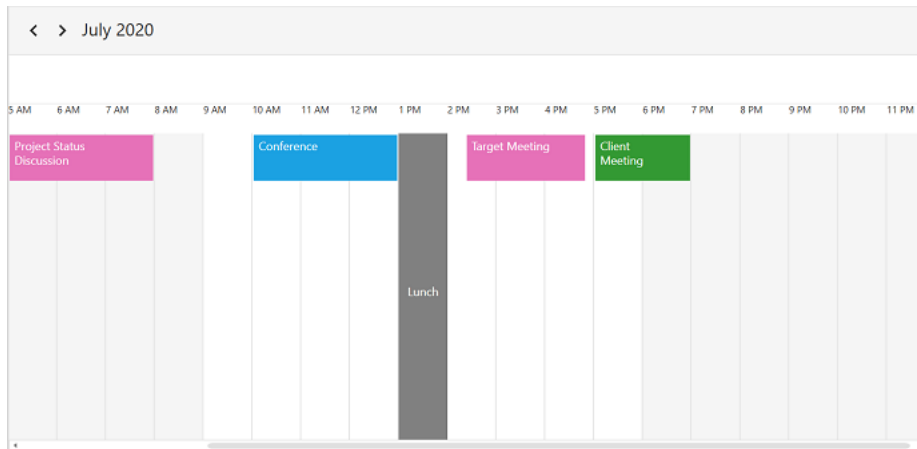
XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWeek" >
  <syncfusion:SfScheduler.TimelineViewSettings>
    <syncfusion:TimelineViewSettings>
      <syncfusion:TimelineViewSettings.SpecialTimeRegions>
        <syncfusion:SpecialTimeRegion
          StartTime="2020/07/08 13:0:0"
          EndTime="2020/07/08 14:0:0"
          CanEdit="False"
          Text="Lunch"
          Background="Black"
          Foreground="White"/>
      </syncfusion:TimelineViewSettings.SpecialTimeRegions>
    </syncfusion:TimelineViewSettings>
  </syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
```

```
Schedule.TimelineViewSettings.SpecialTimeRegions.Add(new SpecialTimeRegion
{
    StartTime = new System.DateTime(2020, 07, 08, 13, 0, 0),
    EndTime = new System.DateTime(2020, 07, 08, 14, 0, 0),
    Text = "Lunch",
    CanEdit = false,
    Background = Brushes.Black,
    Foreground = Brushes.White
});
```

**Note:**

This property only restricts the interaction on region and it does not restrict the following:

- Programmatic selection (if the user updates the selected date value dynamically).
- Does not clear the selection when the user selects the region and dynamically change the `CanEdit` property to false.
- It does not restrict appointment interaction when the appointment is placed in the region.
- It does not restrict the appointment rendering on a region, when appointments are loaded from data services or adding programmatically.

Recurring time region

The recurring time region on a daily, weekly, monthly, or yearly interval. The recurring special time regions can be created by setting the [RecurrenceRule](#) property in `SpecialTimeRegion`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWorkWeek" >
  <syncfusion:SfScheduler.TimelineViewSettings>
    <syncfusion:TimelineViewSettings>
      <syncfusion:TimelineViewSettings.SpecialTimeRegions>
        <syncfusion:SpecialTimeRegion
          StartTime="2020/07/08 13:0:0"
          EndTime="2020/07/08 14:0:0"
          CanEdit="False"
          Text="Lunch"
          Background="Black"
          Foreground="White"
          RecurrenceRule="FREQ=DAILY; INTERVAL=1" />
      </syncfusion:SpecialTimeRegions>
    </syncfusion:TimelineViewSettings>
  </syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>
```

```
</syncfusion:TimelineViewSettings.SpecialTimeRegions>
</syncfusion:TimelineViewSettings>
</syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWorkWeek;
Schedule.TimelineViewSettings.SpecialTimeRegions.Add(new SpecialTimeRegion
{
    StartTime = new System.DateTime(2020, 07, 08, 13, 0, 0),
    EndTime = new System.DateTime(2020, 07, 08, 14, 0, 0),
    Text = "Lunch",
    CanEdit = false,
    Background = Brushes.Black,
    Foreground = Brushes.White,
    RecurrenceRule= "FREQ=DAILY; INTERVAL=1"
});
```

Recurrence exception dates

Delete any of occurrence that is an exception from the recurrence pattern time region by using the [RecurrenceExceptionDates](#) property of [SpecialTimeRegion](#). The deleted occurrence date will be considered as a recurrence exception date.

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
DateTime recurrenceExceptionDates = new DateTime(2020, 07, 06, 10, 0, 0);
DateTime recurrenceExceptionDates1 = new DateTime(2020, 07, 08, 10, 0, 0);
DateTime recurrenceExceptionDates2 = new DateTime(2020, 07, 10, 10, 0, 0);
Schedule.TimelineViewSettings.SpecialTimeRegions.Add(new SpecialTimeRegion
{
    StartTime = new System.DateTime(2020, 05, 05, 12, 0, 0),
    EndTime = new System.DateTime(2020, 05, 05, 13, 0, 0),
    Text = "Lunch",
    CanEdit = false,
    Background = Brushes.Black,
    Foreground = Brushes.White,
    RecurrenceRule = "FREQ=DAILY; INTERVAL=1",
    RecurrenceExceptionDates = new ObservableCollection<DateTime>()
    {
        recurrenceExceptionDates,
        recurrenceExceptionDates1,
        recurrenceExceptionDates2,
    }
});
```

Special time region customization

The [SpecialTimeRegion](#) background color can be customized by using the [Background](#) and [SpecialTimeRegionTemplate](#) properties of [SpecialTimeRegion](#) that is used to customize the text style for the image of the [SpecialTimeRegion](#).

XML

```

<Window.Resources>
<DataTemplate x:Key="specialRegionTemplate">
<Grid Background="{Binding Background}"
HorizontalAlignment="Stretch"
VerticalAlignment="Stretch">
<Image x:Name="Image"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Source="../../Image/Fork.png"/>
</Grid>
</DataTemplate>
</Window.Resources>

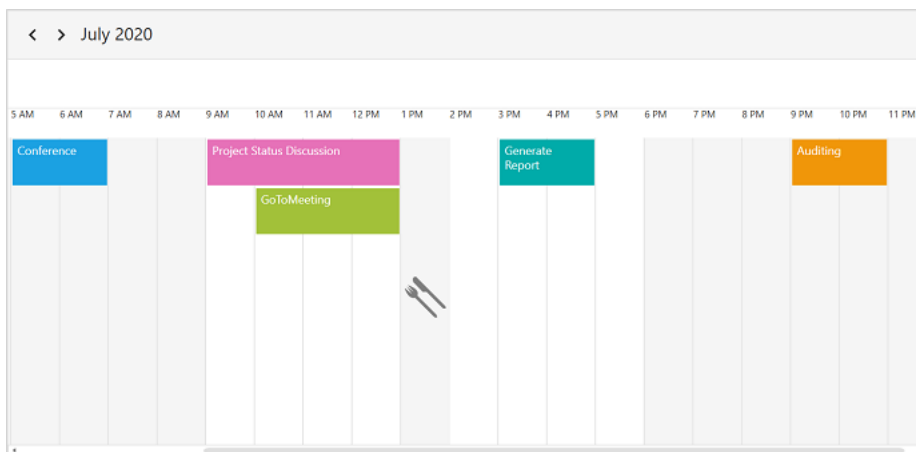
```

XML

```

<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWeek">
<syncfusion:SfScheduler.TimelineViewSettings>
<syncfusion:TimelineViewSettings
SpecialTimeRegionTemplate="{StaticResource specialRegionTemplate}">
<syncfusion:TimelineViewSettings.SpecialTimeRegions>
<syncfusion:SpecialTimeRegion
StartTime="01/01/2020 13:0:0"
EndTime="01/01/2020 14:0:0"
CanEdit="False"
Text="LUNCH"
RecurrenceRule="FREQ=DAILY;INTERVAL=1"
Foreground="Black"
Background="#FFF5F5F5"/>
</syncfusion:TimelineViewSettings.SpecialTimeRegions>
</syncfusion:TimelineViewSettings>
</syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>

```



Full screen scheduler

Scheduler time interval height can be adjusted based on screen height by changing the value of [TimeIntervalSize](#) property to -1. It will auto-fit to the screen height and width.

XML

```

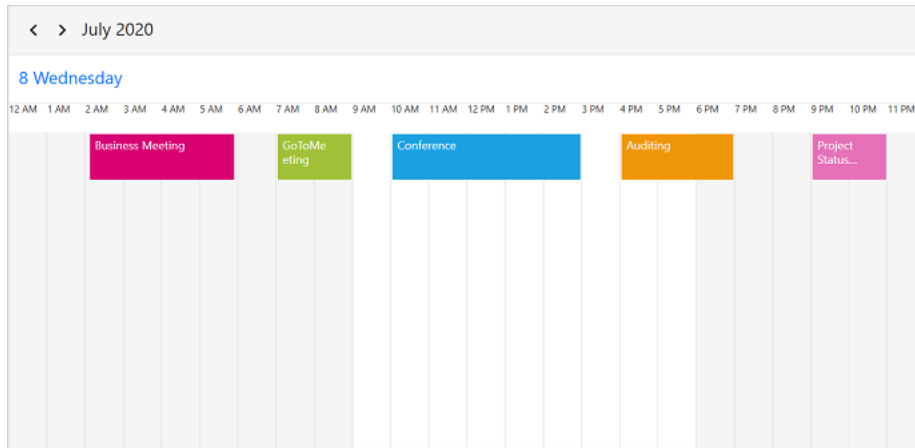
<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWeek">

```

```
<syncfusion:SfScheduler.TimelineViewSettings>
<syncfusion:TimelineViewSettings
TimeIntervalSize="-1"/>
</syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.TimeIntervalSize = -1;
```

**Change time ruler size**

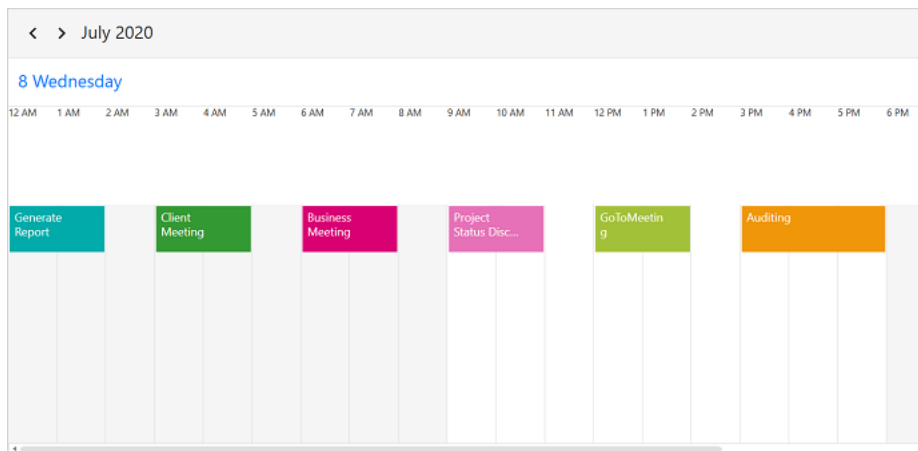
Customize the size of the time ruler view where the labels mentioning the time are placed by using the [TimeRulerSize](#) property of [TimelineViewSettings](#). This property will be applicable to [TimelineDay](#), [TimelineWeek](#), and [TimelineWorkWeek](#) views.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWeek" >
<syncfusion:SfScheduler.TimelineViewSettings>
<syncfusion:TimelineViewSettings
TimeRulerSize="100">
</syncfusion:TimelineViewSettings>
</syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.TimeRulerSize = 100;
```



Minimum appointment duration

The [MinimumAppointmentDuration](#) property in the [TimelineViewSettings](#) is to set an arbitrary height to appointments when it has a minimum duration in timeline view so that the subject can be readable. This property will not be applicable for the [TimelineMonth](#) view.

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.MinimumAppointmentDuration = new
System.TimeSpan(0, 120, 0);
```

Note:

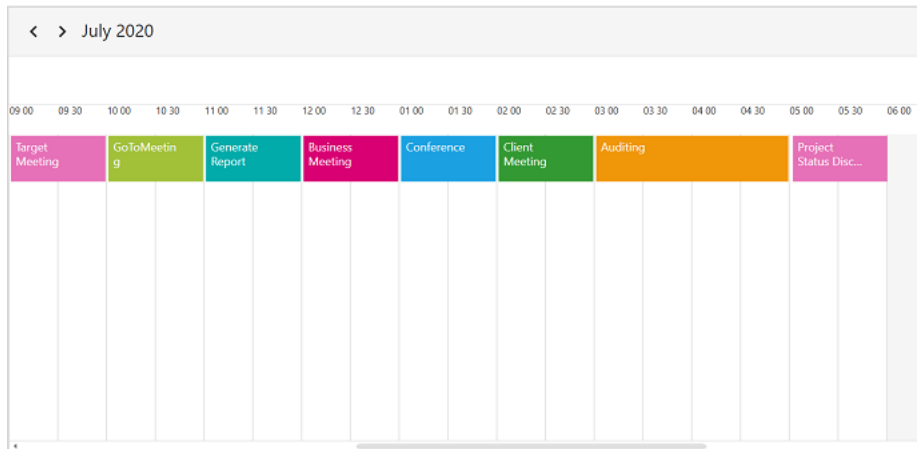
- The [MinimumAppointmentDuration](#) value will be set, when an appointment duration value is lesser than [MinimumAppointmentDuration](#).
- Appointment duration value will be set, when the appointment duration value is greater than [MinimumAppointmentDuration](#).
- [TimeInterval](#) value will be set, when [MinimumAppointmentDuration](#) is greater than [TimeInterval](#) with lesser appointment duration.
- All day Appointment does not support [MinimumAppointmentDuration](#).

Time text formatting

Customize the format for the labels mentioning the time, by setting the [TimeRulerFormat](#) property of [TimelineViewSettings](#) in the [Scheduler](#). This property will not applicable for the [TimelineMonth](#) view.

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.TimeRulerFormat = "hh mm";
Schedule.TimelineViewSettings.TimeInterval = new System.TimeSpan(0, 30, 0);
```


**Note:**

- You can customize the appointment editor time format based on the scheduler time ruler format and culture.
- By default, the scheduler time ruler format is `h tt` and the appointment editor time picker format is `hh:mm tt`.

View header

Customize the default appearance of view header in the timeline views by setting the [ViewHeaderDateFormat](#), [ViewHeaderHeight](#), [ViewHeaderDayFormat](#), and [ViewHeaderTemplate](#) of `TimelineViewSettings`.

View header text formatting

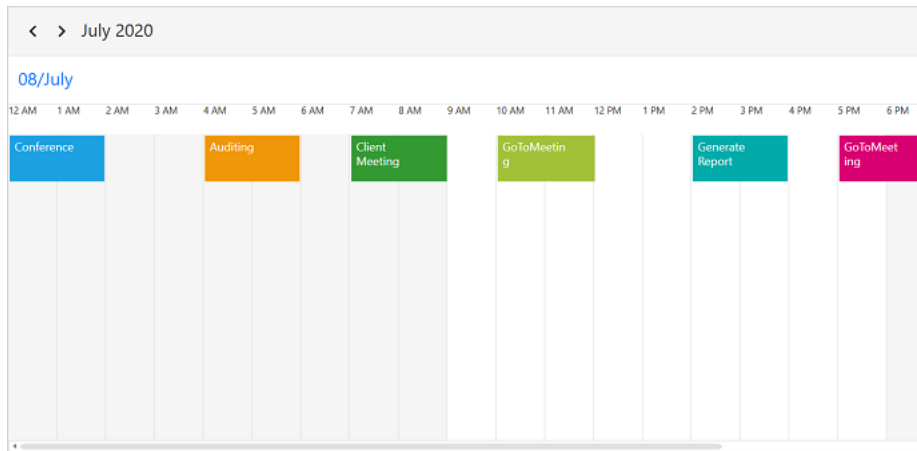
Customize the date and day format of ViewHeader by using the `ViewHeaderDateFormat` and `ViewHeaderDayFormat` properties of `TimelineViewSettings`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWeek">
  <syncfusion:SfScheduler.TimelineViewSettings>
    <syncfusion:TimelineViewSettings
      ViewHeaderDateFormat="dd/MMMM"/>
    </syncfusion:TimelineViewSettings>
  </syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.ViewHeaderDateFormat="dd/MMMM";
```



View header height

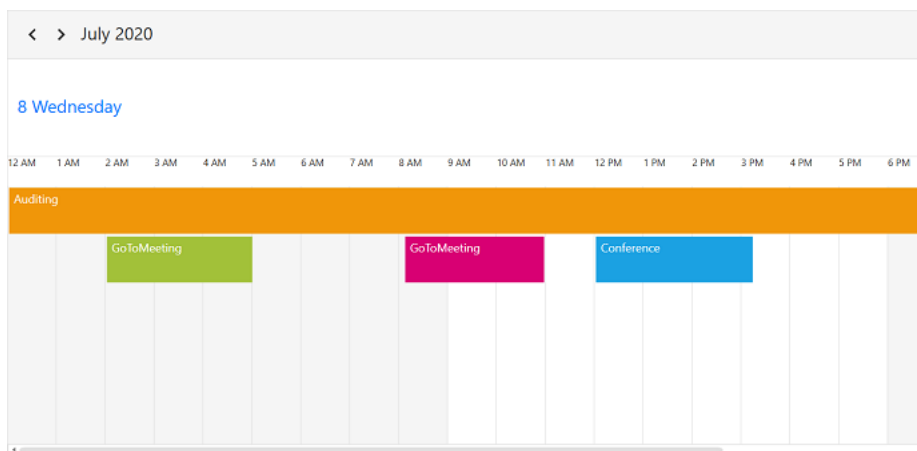
Customize the height of the ViewHeader in timeline views by setting `ViewHeaderHeight` property of `TimelineViewSettings` in `SfScheduler`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWeek">
  <syncfusion:SfScheduler.TimelineViewSettings>
    <syncfusion:TimelineViewSettings
      ViewHeaderHeight="100"/>
    </syncfusion:TimelineViewSettings>
  </syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.ViewHeaderHeight = 100;
```



View header appearance customization

Customize the default appearance of view header by setting `ViewHeaderTemplate` property of `TimelineViewSettings` in `SfScheduler`.

XML

```
<Window.Resources>
```

```

<DataTemplate x:Key="viewHeaderTemplate">
<StackPanel Background="Green"
Width="2000"
VerticalAlignment="Stretch"
HorizontalAlignment="Stretch"
Orientation="Vertical">
<TextBlock
HorizontalAlignment="Left"
Margin="20,0,0,0"
Foreground="FFFFFF"
FontFamily="Arial"
Text="{Binding DateText}"
FontSize="25"
TextTrimming="CharacterEllipsis"
TextWrapping="Wrap" />
<TextBlock
HorizontalAlignment="Left" Margin="20,0,0,0"
Foreground="FFFFFF"
FontFamily="Arial"
Text="{Binding DayText}"
FontSize="10"
TextTrimming="CharacterEllipsis"
TextWrapping="Wrap" />
</StackPanel>
</DataTemplate>
</Window.Resources>

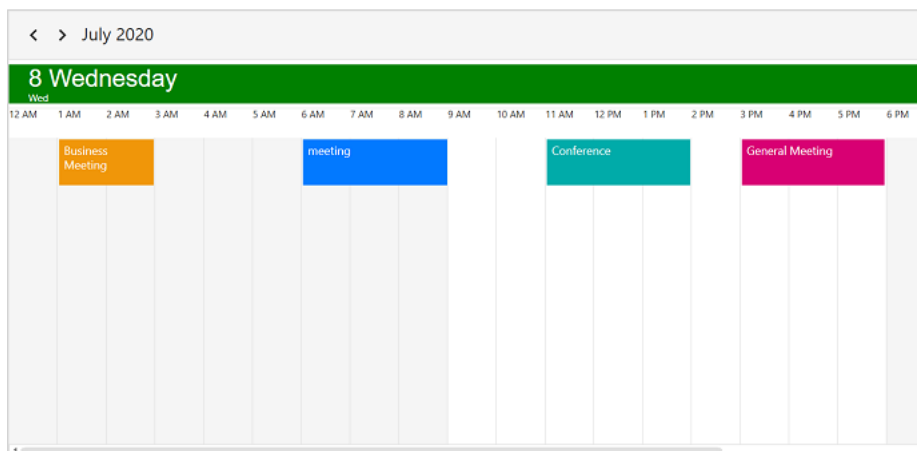
```

XML

```

<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWeek">
<syncfusion:SfScheduler.TimelineViewSettings>
<syncfusion:TimelineViewSettings
ViewHeaderTemplate="{StaticResource viewHeaderTemplate}" />
</syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>

```



Appointment height

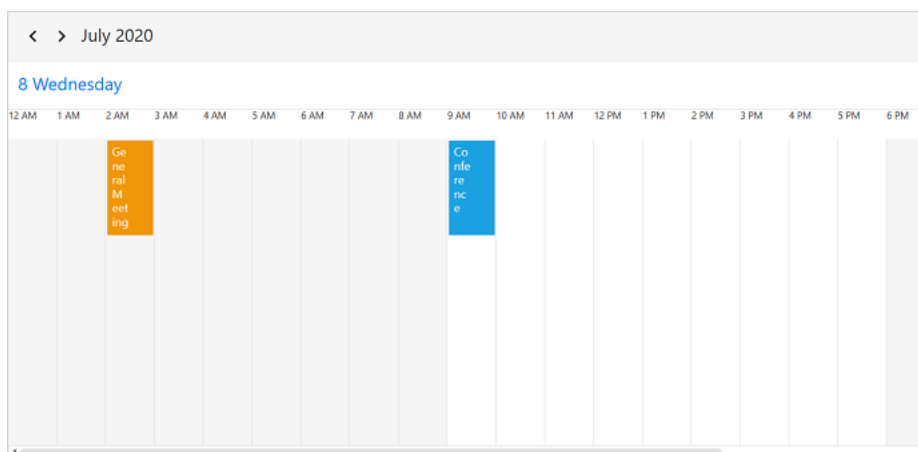
Customize the height of the appointment in `TimelineView` using the [TimelineAppointmentHeight](#) property of the `TimelineViewSettings`. By default, it's value is fifty for the `TimelineWeek`, `TimelineWorkWeek`, and `TimelineDay` views and twenty for the `TimelineMonth` view.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="TimelineWeek">
  <syncfusion:SfScheduler.TimelineViewSettings>
    <syncfusion:TimelineViewSettings
      TimelineAppointmentHeight="100">
    </syncfusion:TimelineViewSettings>
  </syncfusion:SfScheduler.TimelineViewSettings>
</syncfusion:SfScheduler>
```

C#

```
Schedule.ViewType = SchedulerViewType.TimelineWeek;
Schedule.TimelineViewSettings.TimelineAppointmentHeight = 100;
```



Note: You can refer to our [WPF Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Month View in WPF Scheduler (SfScheduler)

The month view of the scheduler displays the days of a specific month and current month, by default initially. The current date color is differentiated from other dates of the current month.

Month agenda view

The scheduler month view displays a divided agenda view that is used to show the selected date's appointments below the month. Show the agenda view by setting the [ShowAgendaView](#) property to true in the [MonthViewSettings](#).

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
  ViewType="Month" >
  <syncfusion:SfScheduler.MonthViewSettings>
    <syncfusion:MonthViewSettings ShowAgendaView="True"/>
  </syncfusion:MonthViewSettings>
</syncfusion:SfScheduler>
```

```
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.MonthViewSettings.ShowAgendaView = true;
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3 Generate Report	4
5	6	7	8 Generate Report Conference	9	10	11
12	13	14	15 Project Status Discussi...	16 GoToMeeting	17	18
19	20 Auditing	21	22	23	24	25
26	27	28	29	30	31	1 FootBall Match
2	3	4 General Meeting	5	6	7	8
Wed 08 Generate Report 09:00 AM - 10:00 AM Conference 09:00 AM - 10:00 AM						

NOTE

- An agenda view displays text as **No Selected Date** until a date is selected.
- If there is no appointment on a selected day, the agenda view displays the text as **No Events**.

Agenda view height

Customize the month agenda view height from the Scheduler by using the [AgendaViewHeight](#) property of MonthViewSettings. By default, the agenda view will occupy 30% of the Scheduler height.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month" >
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings AppointmentDisplayMode="Indicator"
ShowAgendaView="True"
AgendaViewHeight="300"/>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
```

```

this.Schedule.MonthViewSettings.AppointmentDisplayMode =
AppointmentDisplayMode.Indicator;
this.Schedule.MonthViewSettings.ShowAgendaView = true;
this.Schedule.MonthViewSettings.AgendaViewHeight = 300;

```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8
Mon 20	<div>Client Meeting 11:00 AM - 12:00 PM</div> <div>Project Status Discussion 03:00 PM - 04:00 PM</div>					

Appointment display mode

Handle the Scheduler month view appointment display by using the [AppointmentDisplayMode](#) property of MonthViewSettings. By default, the AppointmentDisplayMode is set to Appointment. By using the AppointmentDisplayMode, set the month view appointments display as follows.

- None: Appointment will not be displayed.
- Indicator: Appointment will be denoted as the circle.
- Appointment: Appointment subject will be displayed in the month cell.

XML

```

<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month" >
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings AppointmentDisplayMode="Appointment"/>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>

```

C#

```

this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.MonthViewSettings.AppointmentDisplayMode =
AppointmentDisplayMode.Appointment;

```

< > August 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
26 Wedding Anniversary	27	28 Business Meeting	29 Project Status Discussi...	30	31 Generate Report General Meeting	1 Sam's Birthday
2	3 Project Status Discussi...	4 Project Status Discussi...	5	6	7 Target Meeting	8
9	10 Target Meeting Conference Client Meeting	11	12	13	14 Generate Report	15
16	17 GoToMeeting	18	19 General Meeting Auditing	20	21 Auditing	22
23	24	25 Target Meeting Target Meeting	26	27 General Meeting	28	29
30	31	1	2	3	4	5

Appointment display count

Customize the number of appointments displayed in a month cell using the [AppointmentDisplayCount](#) property of a [MonthViewSettings](#) in the Scheduler. By default, the appointment display count is 3, and the month cell has more than 3 appointments, 3 appointments will be displayed and the remaining appointments in the month cell will be displayed as more appointments.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month" >
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings AppointmentDisplayMode="Indicator"
AppointmentDisplayCount="4"/>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.MonthViewSettings.AppointmentDisplayMode =
AppointmentDisplayMode.Indicator;
this.Schedule.MonthViewSettings.AppointmentDisplayCount = 4;
```

Note:

- By clicking more option, the scheduler navigates to the day view.
- Appointment height will be changed based on the [AppointmentDisplayCount](#) property.

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Month navigation direction

The month view of a Scheduler can be navigated in both horizontal and vertical directions. Change the direction of the navigation using the [MonthNavigationDirection](#) property of `MonthViewSettings`. By default, the month navigation direction is set to `Horizontal`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    ViewType="Month" >
    <syncfusion:SfScheduler.MonthViewSettings>
    <syncfusion:MonthViewSettings MonthNavigationDirection="Vertical"/>
    </syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.MonthViewSettings.MonthNavigationDirection =
    MonthNavigationDirection.Vertical;
```


^ v July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30 Business Meeting	1	2 Auditing	3	4 FootBall Match
5	6	7	8	9	10 Target Meeting	11
12	13	14	15	16	17	18
19	20 Client Meeting	21	22	23	24	25
26	27	28	29	30	31 Project Status Discussi...	1

Date format

Customize the date format of the scheduler month view by using the [DateFormat](#) property of `MonthViewSettings`. By default, the month date format is `d`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    ViewType="Month" >
    <syncfusion:SfScheduler.MonthViewSettings>
    <syncfusion:MonthViewSettings DateFormat="dd"/>
    </syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.MonthViewSettings.DateFormat = "dd";
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	01	02	03	04
05	06	07	08	09	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	01
02	03	04	05	06	07	08

View header

Customize the default appearance of view header in month view by setting `ViewHeaderDayFormat`, `DateFormat`, `ViewHeaderHeight` and `ViewHeaderTemplate` of `TimelineViewSettings`.

View header text formatting

Customize the day format of the Scheduler view header by using the `ViewHeaderDayFormat` property of `MonthViewSettings`. By default, the month view header day format is `ddd`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month" >
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings ViewHeaderDayFormat="dddd"/>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.MonthViewSettings.ViewHeaderDayFormat = "dddd";
```

< > July 2020						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

View header height

Customize the view header height by using the [ViewHeaderHeight](#) property of `MonthViewSettings`. By default, the `ViewHeaderHeight` is set to 50.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month" >
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings ViewHeaderHeight="100"/>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.MonthViewSettings.ViewHeaderHeight = 100;
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

View header appearance customization

Customize the default appearance of the month view header by using the [ViewHeaderTemplate](#) property of `MonthViewSettings`.

XML

```
<Window.Resources>
<Style TargetType="syncfusion:ViewHeaderControl">
<Setter Property="Background" Value="Blue"/>
</Style>
</Window.Resources>
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month" >
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings>
<syncfusion:MonthViewSettings.ViewHeaderTemplate>
<DataTemplate>
<TextBlock FontFamily="Segoe UI"
FontSize="20"
FontStyle="Italic"
Foreground="Red"
Background="AntiqueWhite"
Text="{Binding DayText}"/>
</DataTemplate>
</syncfusion:MonthViewSettings.ViewHeaderTemplate>
</syncfusion:MonthViewSettings>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Leading and Trailing days visibility

Customize the leading and trailing days visibility of the scheduler month view by using the [LeadingDaysVisibility](#) and the [TrailingDaysVisibility](#) properties of `MonthViewSettings`. By default, the `LeadingDaysVisibility` and `TrailingDaysVisibility` are set to `Visible`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    ViewType="Month" >
    <syncfusion:SfScheduler.MonthViewSettings>
    <syncfusion:MonthViewSettings
        LeadingDaysVisibility="Collapsed"
        TrailingDaysVisibility="Collapsed"/>
    </syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.MonthViewSettings.LeadingDaysVisibility =
    Visibility.Collapsed;
this.Schedule.MonthViewSettings.TrailingDaysVisibility =
    Visibility.Collapsed;
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Blackout dates

Disable the interaction for certain dates in the scheduler month view by adding those specific dates to the [BlackoutDates](#) collection property of the [SfScheduler](#). Using this, allocate or restrict specific dates for the predefined events.

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.BlackoutDates = GetBlackoutDates();
private ObservableCollection<DateTime> GetBlackoutDates()
{
    var blackoutDateCollection = new ObservableCollection<DateTime>()
    {
        DateTime.Now.Date.AddDays(1),
        DateTime.Now.Date.AddDays(2),
        DateTime.Now.Date.AddDays(3),
        DateTime.Now.Date.AddDays(4),
        DateTime.Now.Date.AddDays(5)
    };
    return blackoutDateCollection;
}
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Customize blackout dates appearance

Customize the blacked out dates (trailing, leading, normal days, and current date) by using the setting style.

XML

```
<Window.Resources>
<Style TargetType="syncfusion:MonthCell">
<Style.Triggers>
<Trigger Property="DayType" Value="NormalDay, BlockOutDay">
<Setter Property="Foreground" Value="Black"/>
<Setter Property="Background" Value="Gray"/>
</Trigger>
<Trigger Property="DayType" Value="LeadingDay, BlockoutDay">
<Setter Property="Foreground" Value="Black"/>
<Setter Property="Background" Value="Green"/>
</Trigger>
<Trigger Property="DayType" Value="TrailingDay, BlockoutDay">
<Setter Property="Foreground" Value="Red"/>
<Setter Property="Background" Value="Yellow"/>
</Trigger>
<Trigger Property="DayType" Value="Today, BlockoutDay">
<Setter Property="Foreground" Value="Red"/>
<Setter Property="Background" Value="Green"/>
</Trigger>
</Style.Triggers>
</Style>
</Window.Resources>
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Show week number

Display the week number of a year in the scheduler month view by setting the [ShowWeekNumber](#) property of `MonthViewSettings` to `true`. By default, it is set to `false`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    ViewType="Month" >
    <syncfusion:SfScheduler.MonthViewSettings>
    <syncfusion:MonthViewSettings ShowWeekNumber="True"/>
    </syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Month;
this.Schedule.MonthViewSettings.ShowWeekNumber = true;
```


< > July 2020							
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	1	2	3	4
28	5	6	7	8	9	10	11
29	12	13	14	15	16	17	18
30	19	20	21	22	23	24	25
31	26	27	28	29	30	31	1
32	2	3	4	5	6	7	8

Customize week number template

Customize the default appearance of a week number template in the month view by using the [WeekNumberTemplate](#) property of `MonthViewSettings`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month" >
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings ShowWeekNumber="True">
<syncfusion:MonthViewSettings.WeekNumberTemplate>
<DataTemplate>
<Grid >
<Label Foreground="Red"
Content="{Binding}"
VerticalAlignment="Center"
HorizontalAlignment="Center"/>
</Grid>
</DataTemplate>
</syncfusion:MonthViewSettings.WeekNumberTemplate>
</syncfusion:MonthViewSettings>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

< > July 2020							
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	1	2	3	4
28	5	6	7	8	9	10	11
29	12	13	14	15	16	17	18
30	19	20	21	22	23	24	25
31	26	27	28	29	30	31	1
32	2	3	4	5	6	7	8

Customize month cell appearance

Using the DataTemplate

Customize the default appearance of the month cell by using the [MonthCellTemplate](#) property of `MonthViewSettings`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month">
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings>
<syncfusion:MonthViewSettings.MonthCellTemplate>
<DataTemplate>
<Border Background="BlueViolet">
<TextBlock HorizontalAlignment="Center" Foreground="White" Text="{Binding
DateTime.Day}"/>
</Border>
</DataTemplate>
</syncfusion:MonthViewSettings.MonthCellTemplate>
</syncfusion:MonthViewSettings>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Using the DataTemplateSelector

Customize the default appearance of the month cell by using the [MonthCellTemplateSelector](#) property of MonthViewSettings.

The `DataTemplateSelector` can choose a `DataTemplate` at runtime based on the value of a data-bound to Scheduler month cell using the `MonthCellTemplate`. It allows to choose a different data template for each month's cell, customizing the appearance of a particular month cell based on certain conditions.

XML

```
<Window.Resources>
<local:MonthCellTemplateSelector x:Key="monthCellTemplateSelector"/>
</Window.Resources>
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month"
ItemsSource="{Binding Appointments}">
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings AppointmentDisplayMode="Indicator"
MonthCellTemplateSelector="{StaticResource monthCellTemplateSelector}">
</syncfusion:MonthViewSettings>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
public class MonthCellTemplateSelector : DataTemplateSelector
{
    public MonthCellTemplateSelector()
    {
        var mainWindow = App.Current.MainWindow as Window;
        this.MonthAppointmentTemplate =
mainWindow.Resources["monthAppointmentTemplate"] as DataTemplate;
        this.MonthCellDatesTemplate = mainWindow.Resources["monthCellTemplate"] as
DataTemplate;
    }
}
```

```

}
public DataTemplate MonthAppointmentTemplate { get; set; }
public DataTemplate MonthCellDatesTemplate { get; set; }
public override DataTemplate SelectTemplate(object item, DependencyObject container)
{
    var appointments = item as List<ScheduleAppointment>;
    var cell = container as MonthCell;
    if (cell.DateTime.Date == DateTime.Now.Date)
        cell.Foreground = Brushes.Black;
    if (appointments == null || appointments.Count == 0)
    {
        cell.DataContext = cell;
        return MonthCellDatesTemplate;
    }
    else
    {
        MonthCellViewModel monthCellViewModel = new MonthCellViewModel();
        monthCellViewModel.Foreground = cell.Foreground;
        monthCellViewModel.DateText = cell.DateText;
        monthCellViewModel.MonthCellAppointments = appointments;
        cell.DataContext = monthCellViewModel;
        return MonthAppointmentTemplate;
    }
}
}
}

```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28 ♥️ 🦄	29 👑	30	1	2 🕒	3 ♥️ 🏰	4 👑
5 🔴	6 🦄	7	8 🔊	9 🦄	10 🔊	11
12	13 🔴	14 ♥️	15 🔴	16 🔊	17 🔴	18 🦄
19	20	21 🔴	22	23 🏰 🔊	24	25 🔴
26	27 🏰 🔊	28	29	30 🔴	31	1 🔊 ♥️ 🔊
2 🔴	3 🔴	4	5	6 🦄	7	8 🔴

Customize month view appointments

Using the DataTemplate

Customize the default appearance of the month cell appointment by using the [AppointmentTemplate](#) property of `MonthViewSettings`.

XML

```

<syncfusion:SfScheduler x:Name="Schedule" ViewType="Month"
ItemsSource="{Binding Appointments}">
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings>
<syncfusion:MonthViewSettings.AppointmentTemplate>
<DataTemplate>
<TextBlock
Background="{Binding Data.BackgroundColor}"
Text="{Binding Data.EventName}"
HorizontalAlignment="Stretch"
TextTrimming="CharacterEllipsis"
Foreground="{Binding Data.ForegroundColor}"
TextWrapping="Wrap"
FontStyle="Italic" />
</DataTemplate>
</syncfusion:MonthViewSettings.AppointmentTemplate>
</syncfusion:MonthViewSettings>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>

```

< > December 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	1	2	3	4	5
Plan Execution	Yoga Therapy	Consulting	General Meeting	Consulting	Project Plan	Yoga Therapy
6	7	8	9	10	11	12
Consulting		Project Plan	Plan Execution	Project Plan	Project Plan	Yoga Therapy
13	14	15	16	17	18	19
Performance Check		Consulting	Project Plan	Plan Execution	Consulting	Consulting
20	21	22	23	24	25	26
Project Plan		Plan Execution	Plan Execution	Plan Execution	Consulting	Performance Check
27	28	29	30	31	1	2
Plan Execution		Project Plan	Project Plan	Plan Execution	Yoga Therapy	Consulting
3	4	5	6	7	8	9
Consulting	General Meeting	Consulting	Project Plan		Plan Execution	Yoga Therapy

Using the DataTemplateSelector

Customize the default appearance of the month view appointments by using the [AppointmentTemplateSelector](#) property of `MonthViewSettings`.

The `DataTemplateSelector` can choose a `DataTemplate` at runtime based on the value of a data-bound to Scheduler month appointments using the `AppointmentTemplate`. It allows to choose a different data template for each month's cell, customizing the appearance of a particular appointment based on certain conditions.

XML

```

<Window.Resources>
<local:MonthViewAppointmentTemplateSelector
x:Key="appointmentTemplateSelector">
<local:MonthViewAppointmentTemplateSelector.CurrentDayAppointmentTemplate>
<DataTemplate>
<Border>
<TextBlock Text="{Binding Subject}"
FontStyle="Italic"
Foreground="Red"

```

```
TextWrapping="Wrap"
TextTrimming="WordEllipsis"/>
</Border>
</DataTemplate>
</local:MonthViewAppointmentTemplateSelector.CurrentDayAppointmentTemplate>
<local:MonthViewAppointmentTemplateSelector.DefaultAppointmentTemplate>
<DataTemplate>
<TextBlock Text="{Binding Subject}"
TextWrapping="Wrap"
TextTrimming="WordEllipsis"/>
</DataTemplate>
</local:MonthViewAppointmentTemplateSelector.DefaultAppointmentTemplate>
</local:MonthViewAppointmentTemplateSelector>
</Window.Resources>
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month" >
<syncfusion:SfScheduler.MonthViewSettings>
<syncfusion:MonthViewSettings AppointmentTemplateSelector="{StaticResource
appointmentTemplateSelector}"/>
</syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

C#

```
public class MonthViewAppointmentTemplateSelector : DataTemplateSelector
{
    public DataTemplate CurrentDayAppointmentTemplate { get; set; }
    public DataTemplate DefaultAppointmentTemplate { get; set; }
    public override DataTemplate SelectTemplate(object item, DependencyObject
container)
    {
        var app = item as ScheduleAppointment;
        if (app.StartTime.Date == DateTime.Today.Date)
            return this.CurrentDayAppointmentTemplate;
        else
            return this.DefaultAppointmentTemplate;
    }
}
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29 Target Meeting	30	1	2 Business Meeting	3	4
5	6	7	8 Conference	9	10	11 TV Show
12	13	14	15	16	17 Auditing	18
19	20	21 Target Meeting	22	23	24	25
26	27	28 General Meeting	29	30 Target Meeting	31	1
2	3	4	5	6	7	8

Customize more appointments indicator in month cell

You can customize the default appearance of more appointments indicator in a month cell by using the [MoreAppointmentsIndicatorTemplate](#) property of the `MonthViewSettings`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Month">
  <syncfusion:SfScheduler.MonthViewSettings>
    <syncfusion:MonthViewSettings>
      <syncfusion:MonthViewSettings.MoreAppointmentsIndicatorTemplate>
        <DataTemplate>
          <TextBlock Text = "{Binding StringFormat=+{0}}" Background = "#EAEAEA"
            Foreground = "Black" Padding="0,5,0,0">
          </TextBlock>
        </DataTemplate>
      </syncfusion:MonthViewSettings.MoreAppointmentsIndicatorTemplate>
    </syncfusion:MonthViewSettings>
  </syncfusion:SfScheduler.MonthViewSettings>
</syncfusion:SfScheduler>
```

< > June 2021						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	31	1	2	3	4	5
6	7	8	9	10 General Meeting Project Plan +2	11	12
13	14	15	16	17	18	19
20	21	22	23 Plan Execution General Meeting +5	24 Plan Execution Plan Execution +2	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

Note: You can refer to our [WPF Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Resource Grouping in WPF Scheduler (SfScheduler)

The [WPF Scheduler](#) resource view will allow to group appointments based on the resources or dates, arranged by the column or row in the day, week, workweek, timeline day, timeline week, timeline workweek and timeline month views. It also allows to share the events or appointments to the multiple resources and resource appointment details can be edited by using a built-in appointment editor.

Grouping by Resources

Resources can be added to the scheduler by setting the [ResourceGroupType](#) property as [Resource](#) in [SfScheduler](#). Set the [Id](#), [Name](#), [Foreground](#) and [Background](#) properties of [SchedulerResource](#) to create a resource. Add the resource to the scheduler by using the [ResourceCollection](#) property of [SfScheduler](#) and also add or remove the scheduler resources dynamically.

Note: No resource view will be displayed, even a resource added using the [ResourceCollection](#) property when the [ResourceGroupType](#) property value is set to [None](#).

C#

```
// Adding schedule resource in the scheduler resource collection.
var ResourceCollection = new ObservableCollection<SchedulerResource>()
{
    new SchedulerResource() { Name = "Sophia", Background = new
    SolidColorBrush(Colors.Red), Id = "1000" },
    new SchedulerResource() { Name = "Zoey Addison", Background = new
    SolidColorBrush(Colors.Blue), Id = "1001" },
    new SchedulerResource() { Name = "James William", Background = new
    SolidColorBrush(Colors.Yellow), Id = "1002" },
};
// Adding the scheduler resource collection to the schedule resources of
SfScheduler.
schedule.ResourceCollection = ResourceCollection;
```

XML


```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week"
ResourceGroupType="resource" ResourceCollection="{Binding
ResourceCollection}">
```

Note: [View sample in GitHub](#)

Resource Grouping types

Group the resource order by **Date** or order by **Resource** using the [ResourceGroupType](#) property of SfScheduler.

Note: Group the resource order in the day, week, work week, timeline day, timeline week, timeline workweek and timeline month views.

Resource

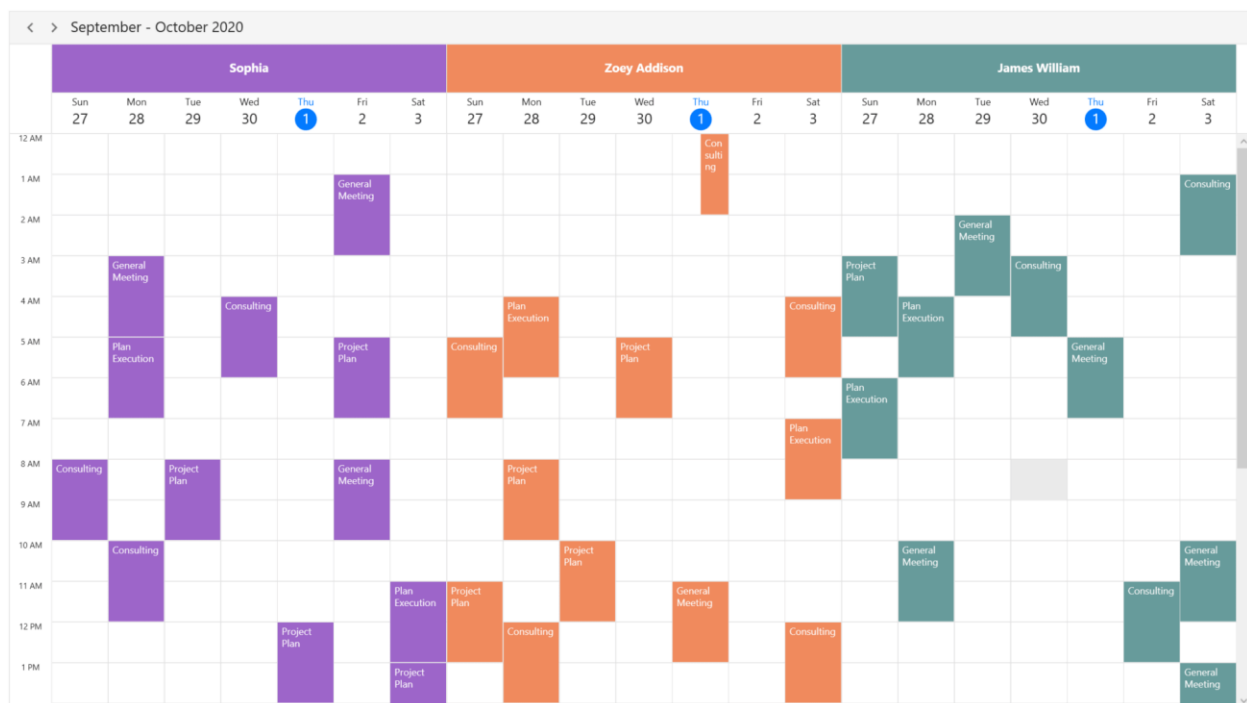
The **ResourceGroupType** is set to **Resource** to group the number of dates under each resource.

XML

```
<Schedule:SfScheduler Name="schedule" ViewType="Week"
ResourceGroupType="Resource"/>
```

C#

```
schedule.ViewType = SchedulerViewType.Week;
schedule.ResourceGroupType = ResourceGroupType.Resource;
```



Date

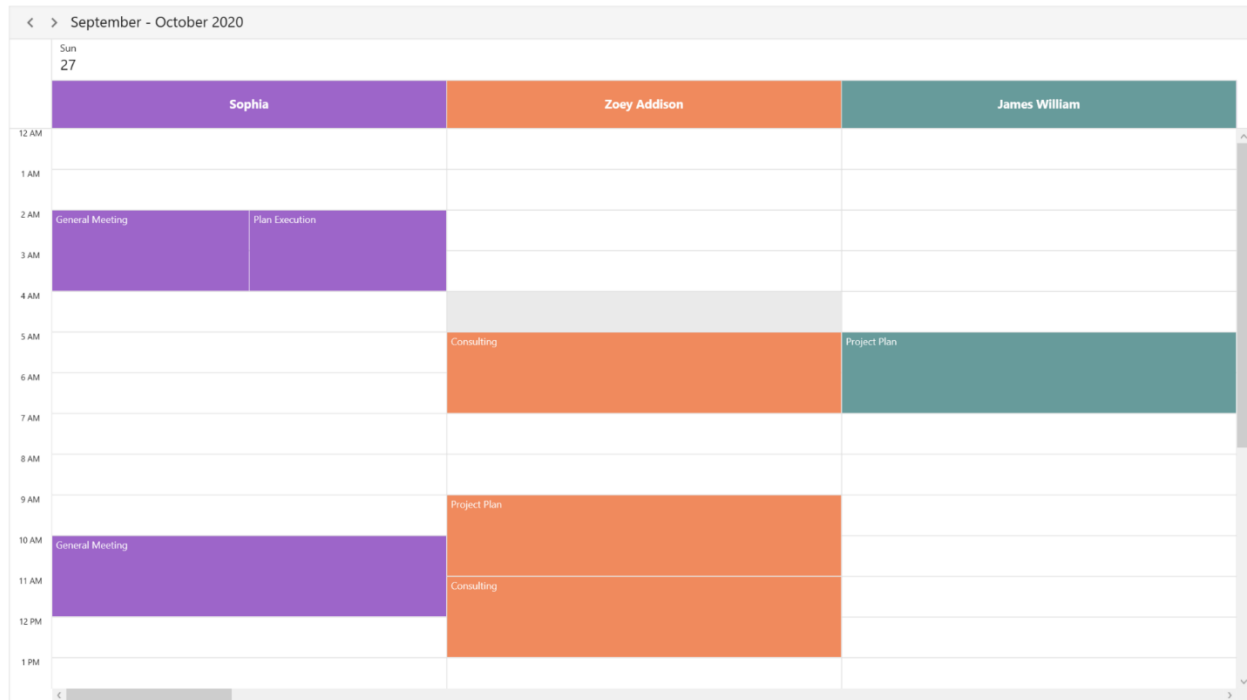
The **ResourceGroupType** is set to **Date** to group the number of resources under each date.

XML

```
<Schedule:SfScheduler Name="schedule" ViewType="Week"
ResourceGroupType="Date"/>
```

C#

```
schedule.ViewType = SchedulerViewType.Week;
schedule.ResourceGroupType = ResourceGroupType.Date;
```



Assigning resources to appointments

Appointments associated with scheduler `ResourceCollection` will be displayed when set schedule resource `Id` in the [ScheduleAppointment](#) by using the [ResourceIdCollection](#) for `ResourceGroupType` set as `Resource` or `Date`. Also assign the resources to [recurrence appointments](#).

C#

```
ScheduleAppointmentCollection scheduleAppointmentCollection = new
ScheduleAppointmentCollection();
var appointments = new ScheduleAppointment()
{
    StartTime = DateTime.Now.AddMinutes(20),
    EndTime = DateTime.Now.AddHours(2),
    Subject = "General Meeting",
    ResourceIdCollection = new ObservableCollection<object> () { "1000", "1001"
};
scheduleAppointmentCollection.Add(appointments);
this.schedule.ItemsSource = scheduleAppointmentCollection;
```

Note: • When `ResourceIdCollection` is not added to 'ScheduleAppointment' then the appointment will not be displayed in, when `ResourceGroupType` is set as `Resource` or `Date`.

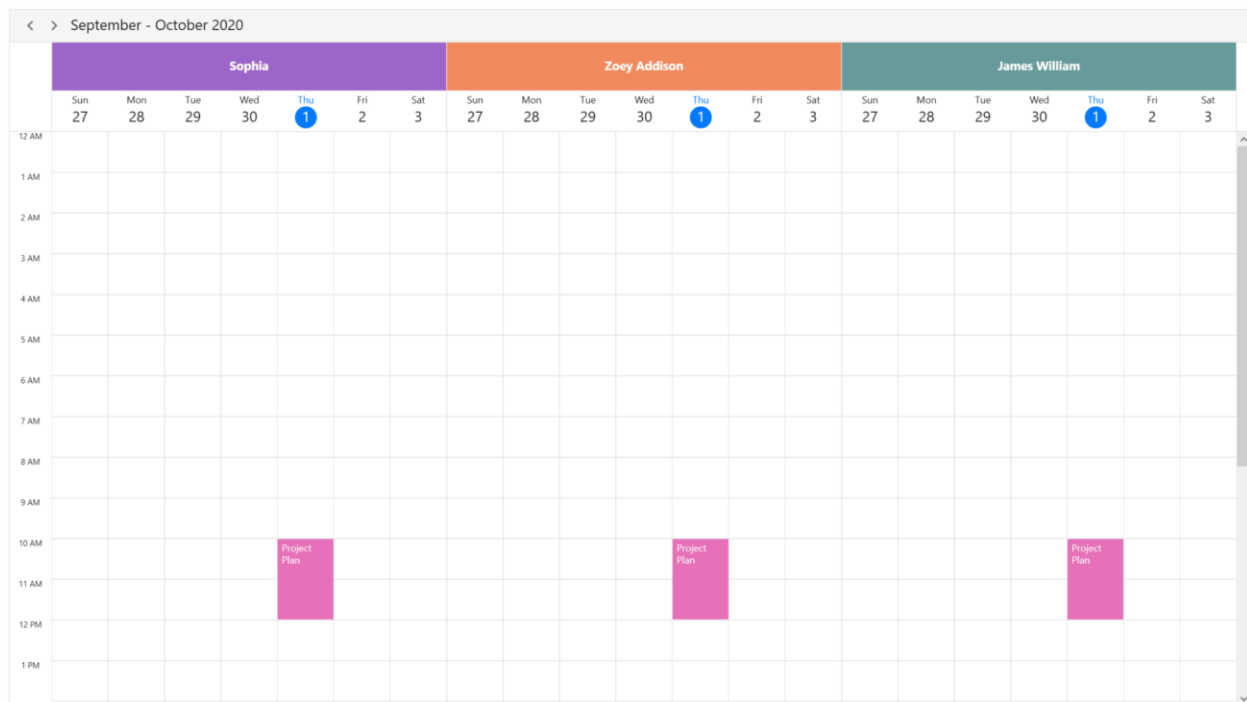
- When `ResourceGroupType` is set as `None`, resource view will be collapsed and all scheduler `DataSource` events will be displayed.
- Also add or remove the appointment resources dynamically.

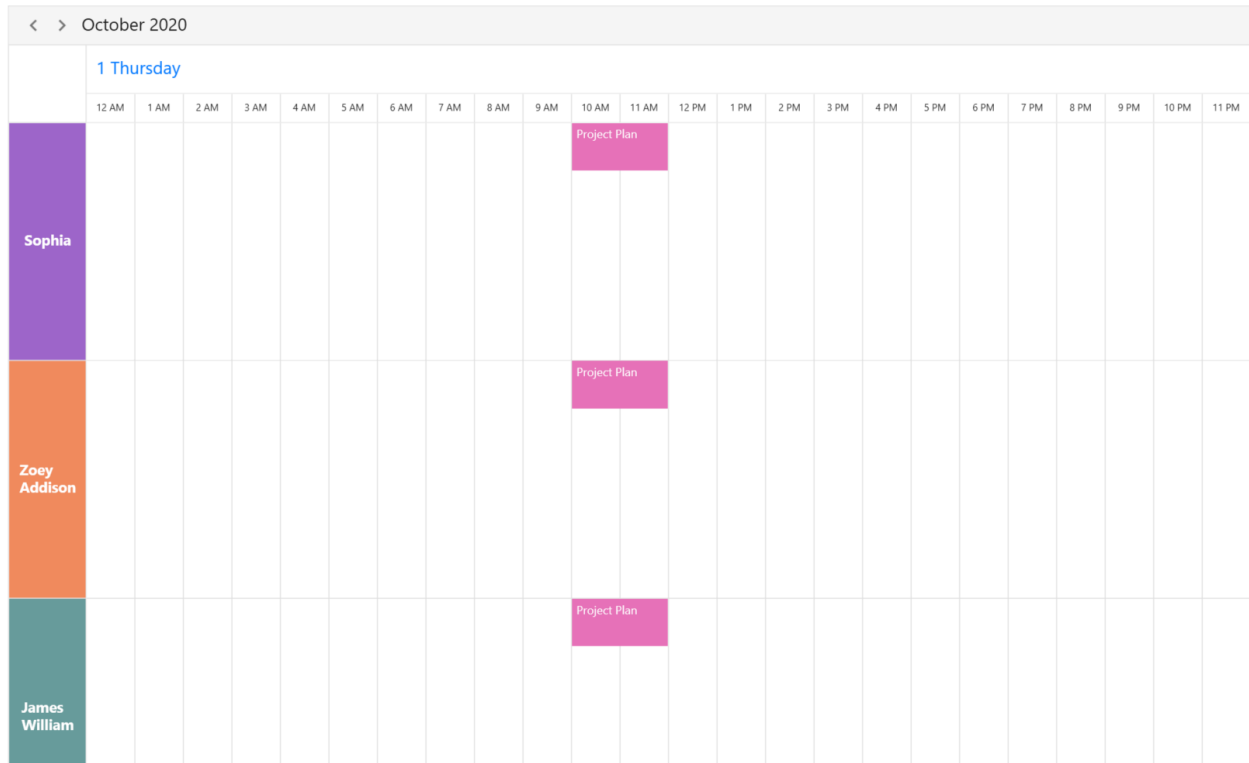
Multiple resource sharing

Multiple resources can share the same events or appointments. If the appointment details are edited or updated, then the changes will reflect on all other shared instances simultaneously.

C#

```
ScheduleAppointmentCollection scheduleAppointmentCollection = new
ScheduleAppointmentCollection();
var appointments = new ScheduleAppointment()
{
    StartTime = new DateTime(2020, 10, 01, 10, 0, 0),
    EndTime = new DateTime(2020, 10, 01, 12, 0, 0),
    Subject = "Project Plan",
    ResourceIdCollection = new ObservableCollection<object>() { "1000",
    "1001", "1002" }
    AppointmentBackground = new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFE671B8")),
};
scheduleAppointmentCollection.Add(appointments);
this.schedule.ItemsSource = scheduleAppointmentCollection;
```





Scheduler Resource Mapping

Schedule supports full data binding to [ResourceCollection](#). Specify the [ResourceMapping](#) attribute to map the properties in the underlying data source to the schedule resource.

| Property Name | Description |

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

| Name | Maps the property name of the custom class, which is equivalent to Name in the ScheduleResource. |

| Id | Maps the property name of custom class, which is equivalent to Id in ScheduleResource. |

| Background | Maps the property name of custom class, which is equivalent to Background in ScheduleResource. |

| Foreground | Maps the property name of custom class, which is equivalent to Foreground in ScheduleResource. |

Note: Custom resource class should contain a mandatory field for resource **Id**.

Create business object for Resource

Create a custom class `Employee` with mandatory fields `Name`, `Id`, `ForegroundColor` and `BackgroundColor`. Also assign the resources to [recurrence appointments](#).

C#

```
public class Employee
{
```

```

public string Name {get; set;}
public string Id {get; set;}
public Brush BackgroundColor {get; set; }
public Brush ForegroundColor {get; set; }
}

```

Note: • Inherit this class from `INotifyPropertyChanged` for dynamic changes in custom data.

- [SchedulerResource.Data](#) property is used to get the details of the custom data.

Map the properties of `Employee` class with `SfScheduler` control using Scheduler `ResourceMapping`.

XML

```

<Schedule:SfScheduler Name="schedule" ViewType="Week"
ResourceGroupType="Resource">
<Schedule:SfScheduler.ResourceMapping>
<Schedule:ResourceMapping Id="Id" Name="Name" Background="BackgroundColor"
Foreground="ForegroundColor"/>
</Schedule:SfScheduler.ResourceMapping>
</Schedule:SfScheduler>

```

C#

```

// Schedule data mapping for custom resource.
ResourceMapping resourceMapping = new ResourceMapping();
resourceMapping.Name = "Name";
resourceMapping.Id = "Id";
resourceMapping.Background = "BackgroundColor";
resourceMapping.Foreground = "ForegroundColor";
schedule.ResourceMapping = resourceMapping;

```

Assign resource object collection

Add the resources of `Employee` collection that can be assigned to the scheduler using the `ResourceCollection` property which is of `IEnumerable` type. Also add or remove scheduler resources dynamically.

C#

```

// Creating and Adding custom resource in scheduler resource collection.
var ResourceCollection = new ObservableCollection<Employee>()
{
    new Employee () {Name = "Sophia", BackgroundColor = new
SolidColorBrush(Colors.Red), Id = "1000", ForegroundColor = new
SolidColorBrush(Colors.White) },
    new Employee () {Name = "Zoey Addison", BackgroundColor = new
SolidColorBrush(Colors.Blue), Id = "1001" , ForegroundColor = new
SolidColorBrush(Colors.Red)},
    new Employee () {Name = "James William", BackgroundColor = new
SolidColorBrush(Colors.Yellow), Id = "1002" , ForegroundColor = new
SolidColorBrush(Colors.Yellow)},
};
//Adding schedule resource collection to schedule resources.
schedule.ResourceCollection = ResourceCollection;

```

Assign the resource objects to appointment business object

Associate scheduler **ResourceMapping** to the custom appointment by mapping resource **Id** in the **ResourceIdCollection** property of **AppointmentMapping**. Custom appointments associated with the scheduler resources will be displayed when **ResourceGroupType** set as **Resource** or **Date**. Also assign resources to recurrence appointments.

C#

```

/// <summary>
/// Represents the custom data properties.
/// </summary>
public class Meeting
{
    public string EventName {get; set;}
    public DateTime From {get; set;}
    public DateTime To {get; set;}
    public ObservableCollection<object> Resources {get; set;}
}

```

Note: Inherit this class from the **INotifyPropertyChanged** for dynamic changes in custom data.

Map those properties of **Meeting** class to schedule appointment by using **AppointmentMapping** properties.

{%tabs %}

XML

```

<syncfusion:SfScheduler x:Name="Schedule" ItemsSource="{Binding
Appointments}" ViewType="Week">
  <syncfusion:SfScheduler.AppointmentMapping>
  <syncfusion:AppointmentMapping
    Subject="EventName"
    StartTime="From"
    EndTime="To"
    ResourceIdCollection ="Resources"/>
  </syncfusion:SfScheduler.AppointmentMapping>
</syncfusion:SfScheduler>

```

C#

```

//Schedule data mapping for custom appointments
AppointmentMapping dataMapping = new AppointmentMapping();
dataMapping.Subject = "EventName";
dataMapping.StartTime = "From";
dataMapping.EndTime = "To";
dataMapping.AppointmentBackground = "Color";
dataMapping.ResourceIdCollection= "Resources";
Schedule.AppointmentMapping = dataMapping;

```

Schedule meetings for a Resource by setting **From**, **To** and **Resources** of Meeting class.

{%tabs %}

C#

```

Meeting meeting = new Meeting ();
meeting.From = new DateTime(2020, 07, 01, 10, 0, 0);
meeting.To = meeting.From.AddHours(1);
meeting.EventName = "Meeting";
meeting.Resources = new ObservableCollection<object> { (Resources[0] as
Employee).Id, (Resources[1] as Employee).Id };
var Meetings = new ObservableCollection<Meeting> ();
Meetings.Add(meeting);
schedule.ItemsSource = Meetings;

```

Note: [View sample in GitHub](#)

Resource header size

Customize the resource header size in the day, week, workweek, timeline day, timeline week, timeline workweek and timeline month views by using the [ResourceHeaderSize](#) property of [DaysViewSettings](#) or [TimelineViewSettings](#) in SfScheduler.

Resource header size in days view

[DaysViewSettings](#) applicable for [Day](#), [Week](#) and [WorkWeek](#) views. By default, value of this property is set to 50.

XML

```

<Schedule:SfScheduler Name="schedule" ViewType="Week"
ResourceGroupType="Resource">
<Schedule:SfScheduler.DaysViewSettings>
<Schedule:DaysViewSettings ResourceHeaderSize="100"/>
</Schedule:SfScheduler.DaysViewSettings>
</Schedule:SfScheduler>

```

C#

```

schedule.DaysViewSettings.ResourceHeaderSize = 100;

```

Resource header size in timeline view

[TimelineViewSettings](#) applicable for timeline day, timeline week, timeline workweek and timeline month views. By default, value of this property is set to 50.

XML

```

<Schedule:SfScheduler Name="schedule" ViewType="TimelineWeek"
ResourceGroupType="Resource">
<Schedule:SfScheduler.TimelineViewSettings>
<Schedule:TimelineViewSettings ResourceHeaderSize="100"/>
</Schedule:SfScheduler.TimelineViewSettings>
</Schedule:SfScheduler>

```

C#

```

schedule.TimelineViewSettings.ResourceHeaderSize = 80;

```

Resource auto height

The resource row height gets auto-adjusted based on the number of overlapping appointments occupied on the same time range by setting [RowAutoHeight](#) property as `true` in [TimelineViewSettings](#). By default, value of this property is set to `false`.

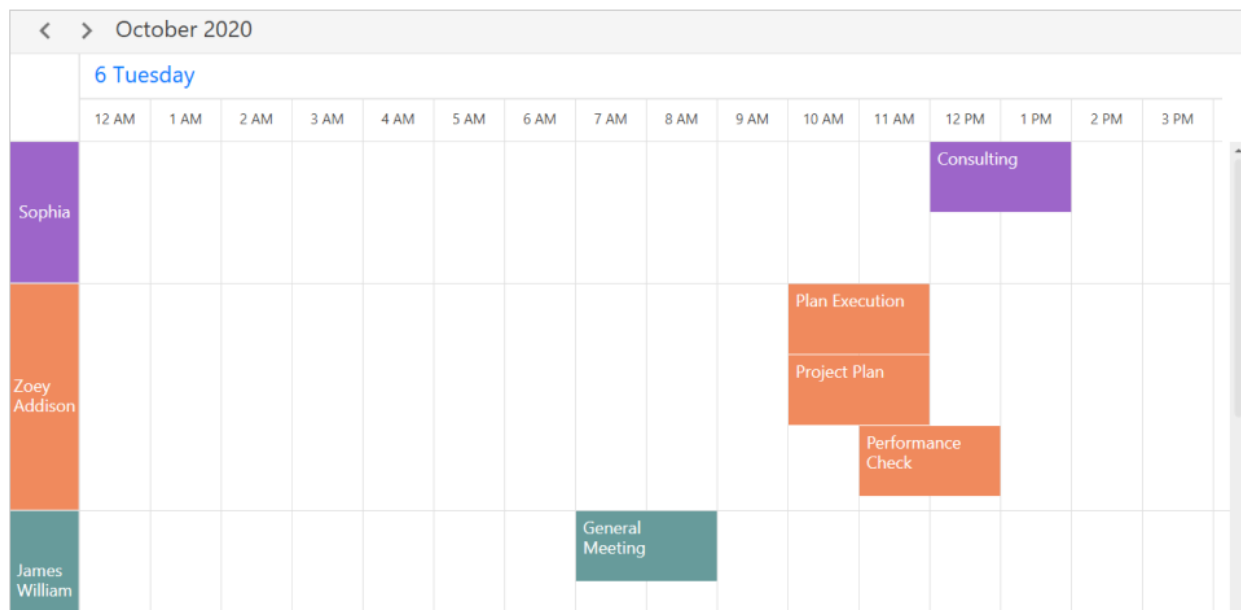
XML

```
<Schedule:SfScheduler Name="schedule"
ViewType="TimelineWeek"
ResourceGroupType="Resource">
<Schedule:SfScheduler.TimelineViewSettings>
<Schedule:TimelineViewSettings RowAutoHeight="True" />
</Schedule:SfScheduler.TimelineViewSettings>
</Schedule:SfScheduler>
```

C#

```
schedule.TimelineViewSettings.RowAutoHeight = true;
```

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



Note:

- This auto row height adjustment is applicable only on all the Timeline views such as timeline day, timeline week, timeline workweek and timeline month views.
- If auto resource row height is enabled then [VisibleResourceCount](#) will not be applicable and if resources have no appointments, then [RowMinHeight](#) will be considered as default resource row height.

Resource minimum height

You can customize the minimum row height of visible resources in timeline day, timeline week, timeline workweek and timeline month views by using the [RowMinHeight](#) property of [TimelineViewSettings](#) in [SfScheduler](#). By default, value of this property is set to 50.

XML

```
<Schedule:SfScheduler Name="schedule"
ViewType="TimelineWeek"
ResourceGroupType="Resource">
<Schedule:SfScheduler.TimelineViewSettings>
<Schedule:TimelineViewSettings RowMinHeight="100" />
</Schedule:SfScheduler.TimelineViewSettings>
</Schedule:SfScheduler>
```

C#

```
schedule.TimelineViewSettings.RowMinHeight = 100;
```

Note: The minimum resource row height adjusted based on view port size and the [VisibleResourceCount](#) will not be applicable.

Visible resource count

Customize the number of visible resources in day, week, workweek, timeline day, timeline week, timeline workweek and timeline month views by using the [VisibleResourceCount](#) property of [DaysViewSettings](#) or [TimelineViewSettings](#) in [SfScheduler](#).

Note: Visible resource count exceed count of schedule [ResourceCollection](#) count then schedule [ResourceCollection](#) count will be displayed.

Visible resource count in days view

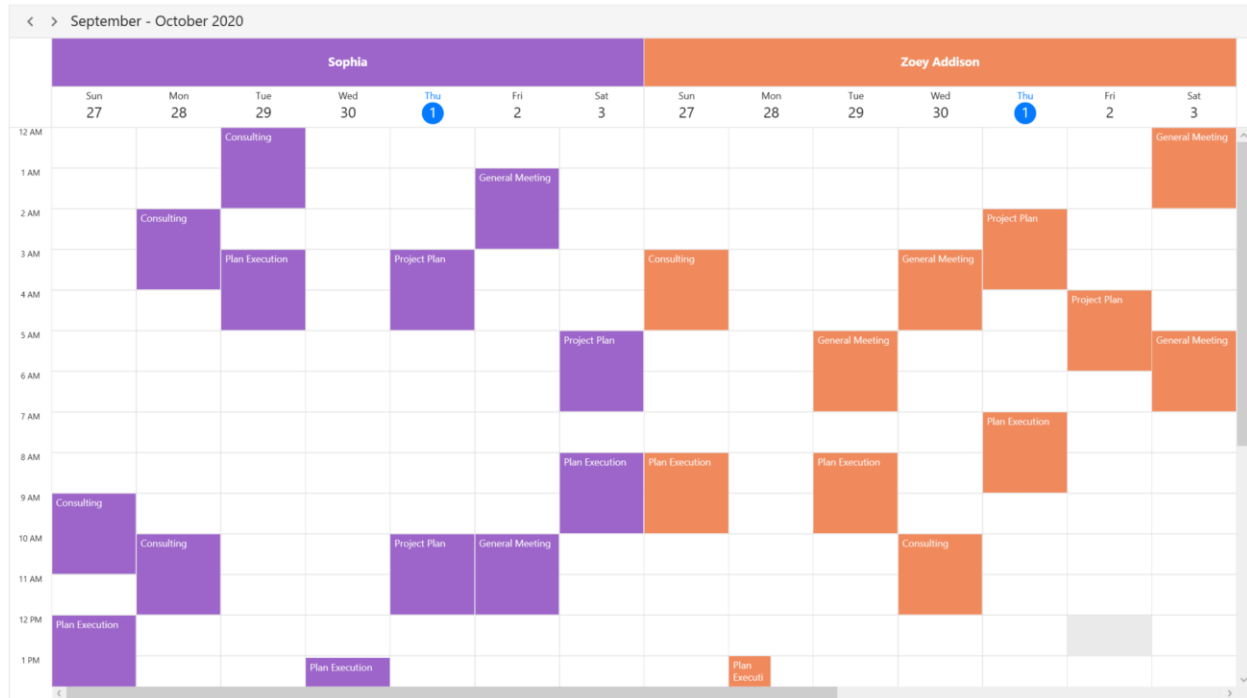
[DaysViewSettings](#) applicable for [Day](#), [Week](#) and [WorkWeek](#) views. By default, value of this property is set to 3.

XML

```
<Schedule:SfScheduler Name="schedule" ViewType="Week"
ResourceGroupType="Resource">
<Schedule:SfScheduler.DaysViewSettings>
<Schedule:DaysViewSettings VisibleResourceCount="2"/>
</Schedule:SfScheduler.DaysViewSettings>
</Schedule:SfScheduler>
```

C#

```
schedule.DaysViewSettings.VisibleResourceCount = 2;
```



Visible resource count in timeline views

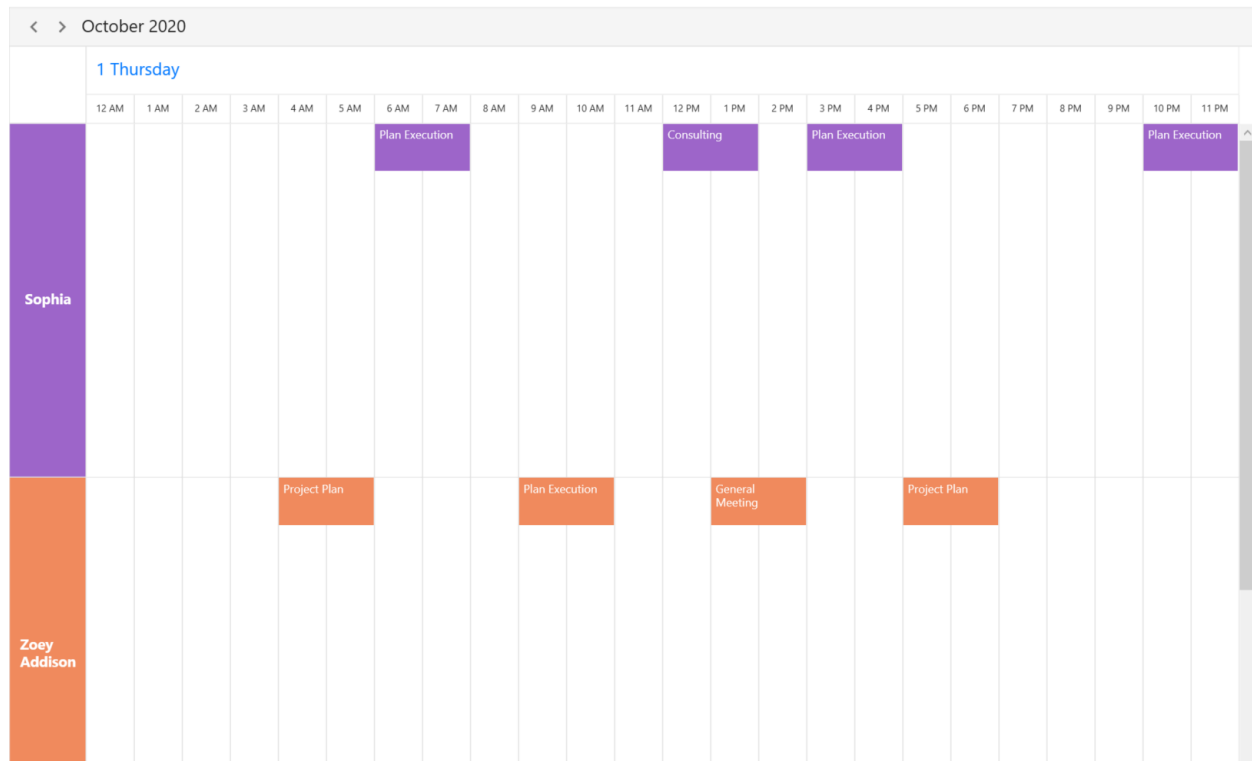
TimelineViewSettings applicable for timeline day, timeline week, timeline workweek and timeline month views. By default, value of this property is set to 3.

XML

```
<Schedule:SfScheduler Name="schedule" ViewType="TimelineDay"
ResourceGroupType="Resource">
<Schedule:SfScheduler.TimelineViewSettings>
<Schedule:TimelineViewSettings VisibleResourceCount="2"/>
</Schedule:SfScheduler.TimelineViewSettings>
</Schedule:SfScheduler>
```

C#

```
schedule.TimelineViewSettings.VisibleResourceCount = 2;
```



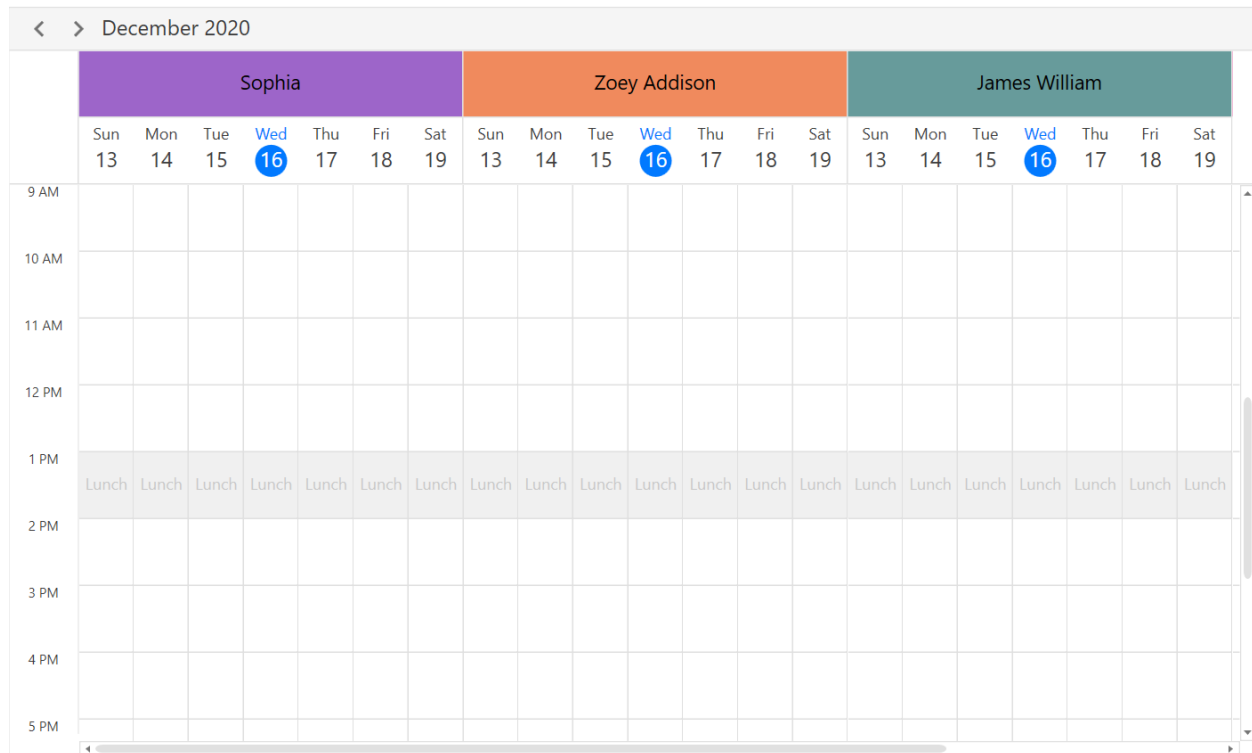
Assign resources to special time regions

Special time region can be created based on the resources in day, week, workweek, timeline day, timeline week, timeline workweek and timeline month views.

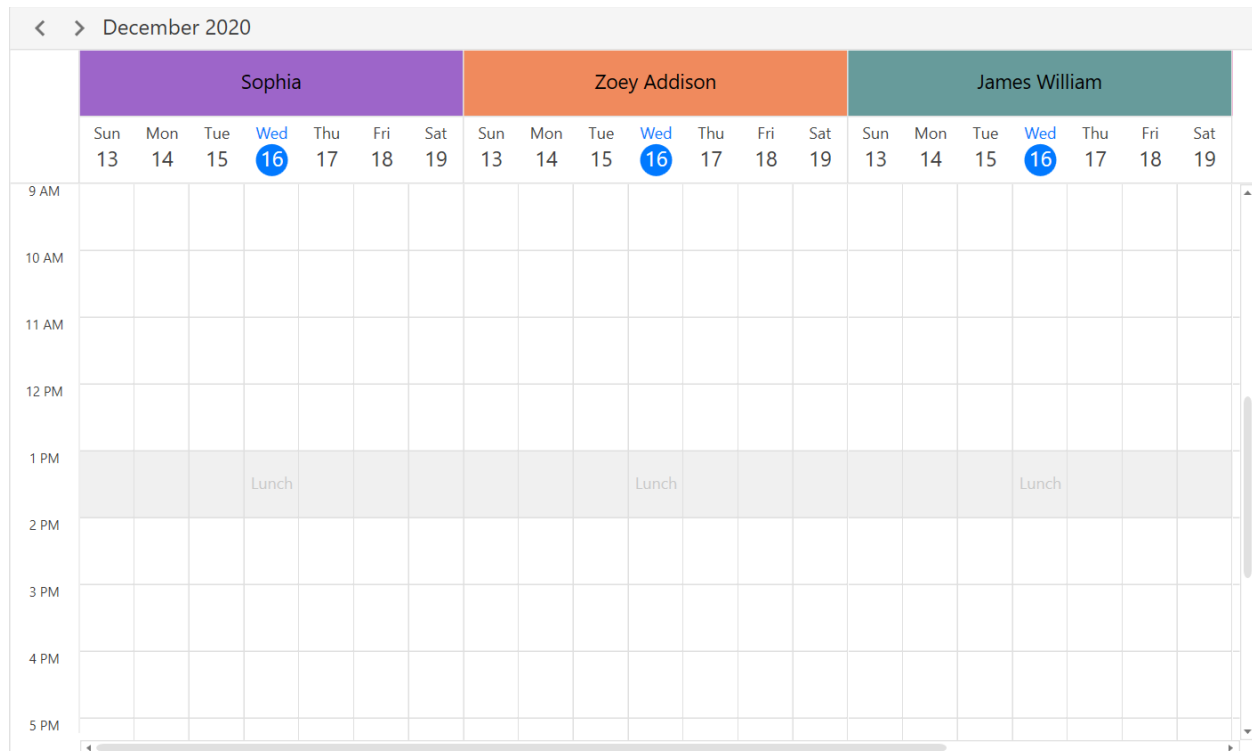
Assign resources to special time regions in days view

C#

```
Schedule.DaysViewSettings.SpecialTimeRegions.Add(new SpecialTimeRegion
{
    StartTime = new System.DateTime(2020, 12, 13, 13, 0, 0),
    EndTime = new System.DateTime(2020, 12, 13, 14, 0, 0),
    Text = "Lunch",
    CanEdit = false,
    Background = Brushes.Black,
    Foreground = Brushes.White,
    RecurrenceRule = "FREQ=DAILY;INTERVAL=1",
    CanMergeAdjacentRegions = false,
    ResourceIdCollection = new ObservableCollection<object>() { "0", "1", "2" }
});
```



The [SpecialTimeRegion](#) in a Time basis by setting the value of [CanMergeAdjacentRegions](#) to True.



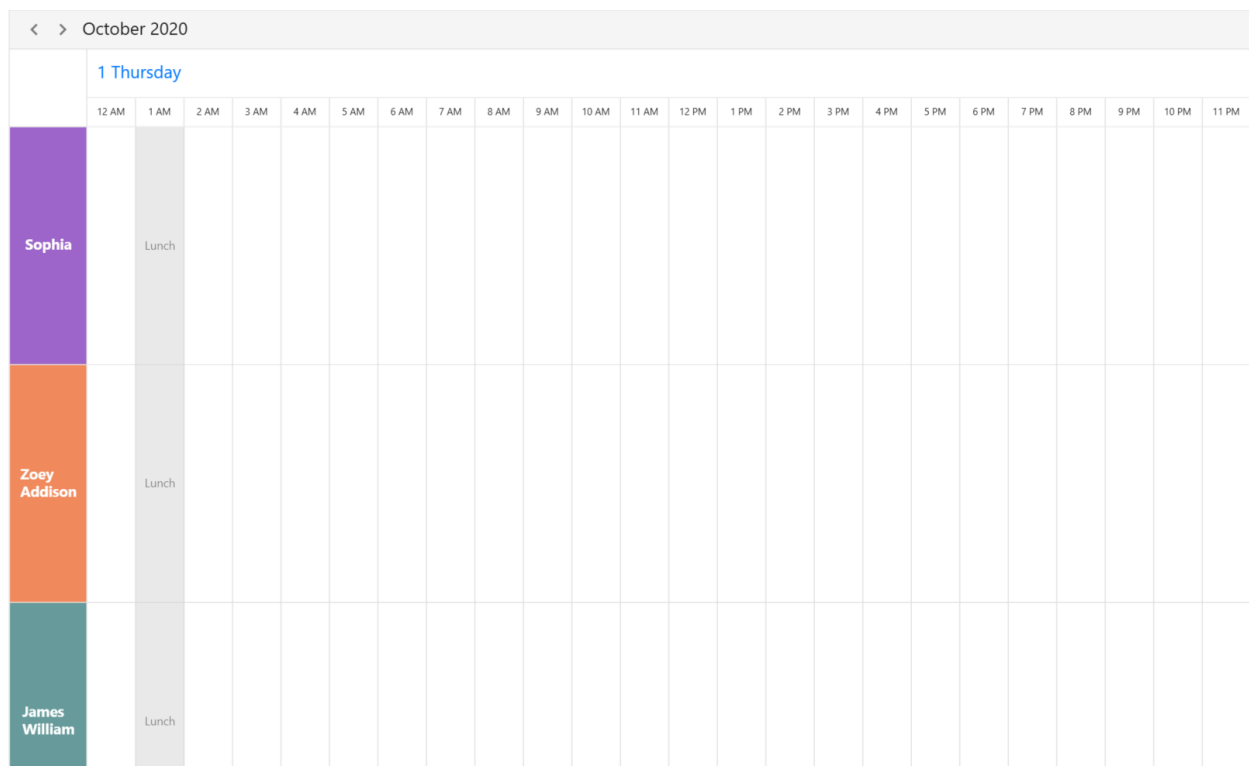
Assign resources to special time regions in timeline view

C#

```

this.schedule.TimelineViewSettings.SpecialTimeRegions.Add(new
SpecialTimeRegion
{
    StartTime = new System.DateTime(2020, 09, 27, 1, 0, 0),
    EndTime = new System.DateTime(2020, 09, 27, 2, 0, 0),
    Text = "Lunch",
    CanEdit = false,
    Background = new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#D3D3D3")),
    Foreground = new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#000000")),
    RecurrenceRule = "FREQ=DAILY;INTERVAL=1",
    ResourceIdCollection = new ObservableCollection<object>() { "1001", "1002",
"1003" }
});

```



Note: [View sample in GitHub](#)

Appearance customization

Resource UI customization using a template and template selectors support.

Customize resource appearance using ResourceHeaderTemplate

XML

```

<Window.Resources>
<DataTemplate x:Key="DayViewResourceTemplate">
<Grid Background="Transparent">
<Border BorderThickness="0.3,0.3,0,0.3" BorderBrush="Gray" >
<StackPanel VerticalAlignment="Center" Orientation="Vertical">

```

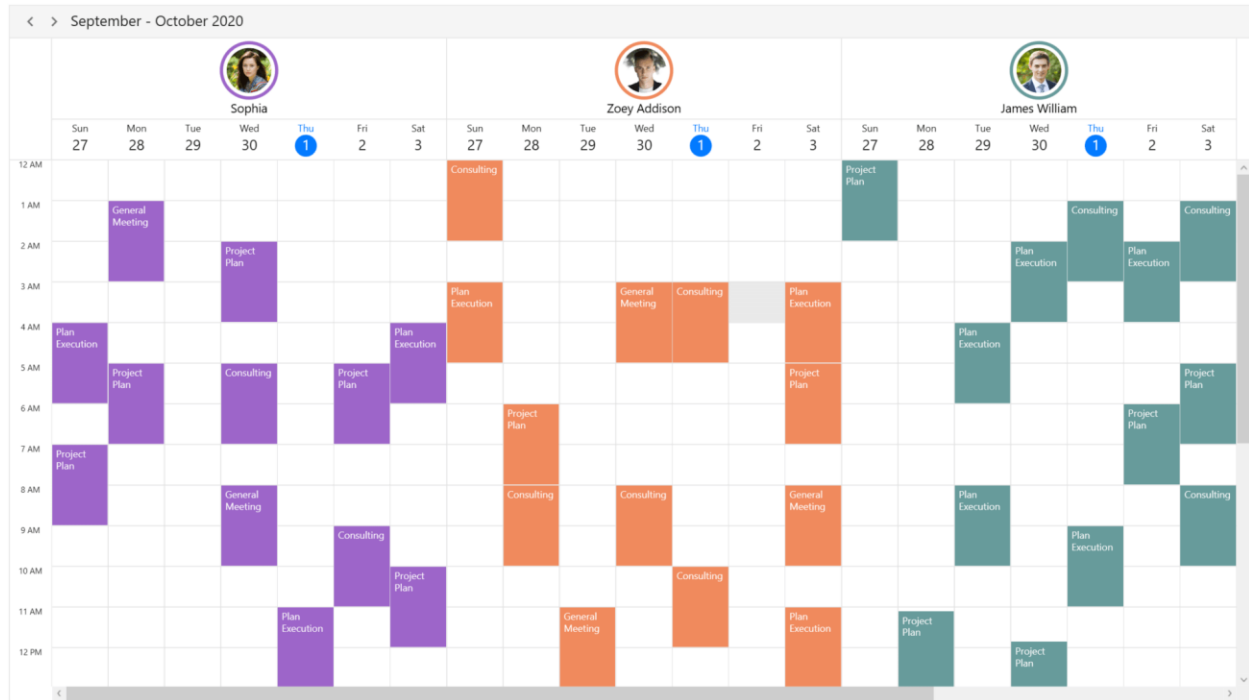
```

<Border CornerRadius="36" Height="72" Width="72" BorderThickness="4"
BorderBrush="{Binding Data.BackgroundBrush}">
<Border CornerRadius="36" Height="64" Width="64" BorderThickness="4"
BorderBrush="White">
<Image HorizontalAlignment="Center" VerticalAlignment="Center" Width="55"
Height="55" Source="{Binding Data.ImageSource}" />
</Border>
</Border>
<TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
FontSize="15"
Foreground="Black" Text="{Binding Data.Name}" />
</StackPanel>
</Border>
</Grid>
</DataTemplate>
</Window.Resources>
//used to find Image Source and Name properties
<Window.DataContext>
<local:Employee />
</Window.DataContext>
<Grid Name="grid">
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week"
ResourceGroupType="Resource" ResourceCollection="{Binding
ResourceCollection}"
ResourceHeaderTemplate="{StaticResource DayViewResourceTemplate}">
<syncfusion:SfScheduler.ResourceMapping>
<syncfusion:ResourceMapping Id="Id" Name="Name" Background="BackgroundBrush"
Foreground="ForegroundBrush"/>
</syncfusion:SfScheduler.ResourceMapping>
</syncfusion:SfScheduler>
</Grid>

```

Note

- By default, the `SchedulerResource` is set as the `DataContext` for the `ResourceHeaderTemplate` for both `SchedulerResource` and custom data object in the `ResourceCollection`.
- The custom data object can be bound in the `ResourceHeaderTemplate` by using the property of [SchedulerResource.Data](#).



Note: [View sample in GitHub](#)

Customize resource appearance using *ResourceHeaderTemplateSelector*

XML

```
<Window.Resources>
<DataTemplate x:Key="DayViewResourceTemplate">
<Grid Background="Transparent">
<Border BorderThickness="0.3,0.3,0,0.3" BorderBrush="Gray" >
<StackPanel VerticalAlignment="Center" Orientation="Vertical">
<Border CornerRadius="36" Height="72" Width="72" BorderThickness="4"
BorderBrush="{Binding Data.BackgroundBrush}">
<Border CornerRadius="36" Height="64" Width="64" BorderThickness="4"
BorderBrush="White">
<Image HorizontalAlignment="Center" VerticalAlignment="Center" Width="55"
Height="55" Source="{Binding Data.ImageSource}" />
</Border>
</Border>
<TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
FontSize="15"
Foreground="Black" Text="{Binding Data.Name}" />
</StackPanel>
</Border>
</Grid>
</DataTemplate>
<DataTemplate x:Key="TimelineViewResourceTemplate">
<Grid Background="Transparent">
<StackPanel VerticalAlignment="Center" Orientation="Vertical">
<Border CornerRadius="36" Height="72" Width="72" BorderThickness="4"
BorderBrush="{Binding Data.BackgroundBrush}">
<Border CornerRadius="36" Height="64" Width="64" BorderThickness="4"
BorderBrush="Transparent">
<Image HorizontalAlignment="Center" VerticalAlignment="Center"
```

```

Width="55"
Height="55"
Source="{Binding Data.ImageSource}" />
</Border>
</Border>
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
FontSize="15"
Text="{Binding Data.Name}" />
</StackPanel>
</Grid>
</DataTemplate>
<Window.Resources>
<Grid>
<Grid.DataContext>
<local:BindingViewModel/>
</Grid.DataContext>
<Grid.Resources>
<local:ResourceTemplateSelector x:Key="resourceTemplateSelector"
DayViewResourceTemplate="{StaticResource DaysViewResourceTemplate}"
TimelineViewResourceTemplate="{StaticResource TimelineResourceTemplate}" />
</Grid.Resources>
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ResourceGroupType="Resource}"
ResourceCollection="{Binding Resources}"
ItemsSource="{Binding ResourceAppointments}" HeaderHeight="32"
DisplayDate="{Binding DisplayDate}"
ResourceHeaderTemplateSelector="{StaticResource resourceTemplateSelector}">
</Grid>

```

Creating a ResourceHeaderTemplateSelector

C#

```

public class ResourceTemplateSelector : DataTemplateSelector
{
    /// <summary>
    /// Initializes a new instance of the <see cref="ResourceTemplateSelector" /> class.
    /// </summary>
    public ResourceTemplateSelector()
    {
    }

    public DataTemplate DayViewResourceTemplate {get; set;}
    public DataTemplate TimelineViewResourceTemplate {get; set;}
    /// <summary>
    /// Template selection method
    /// </summary>
    /// <param name="item">return the object</param>
    /// <param name="container">return the bindable object</param>
    /// <returns>return the template</returns>
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
    }
}

```



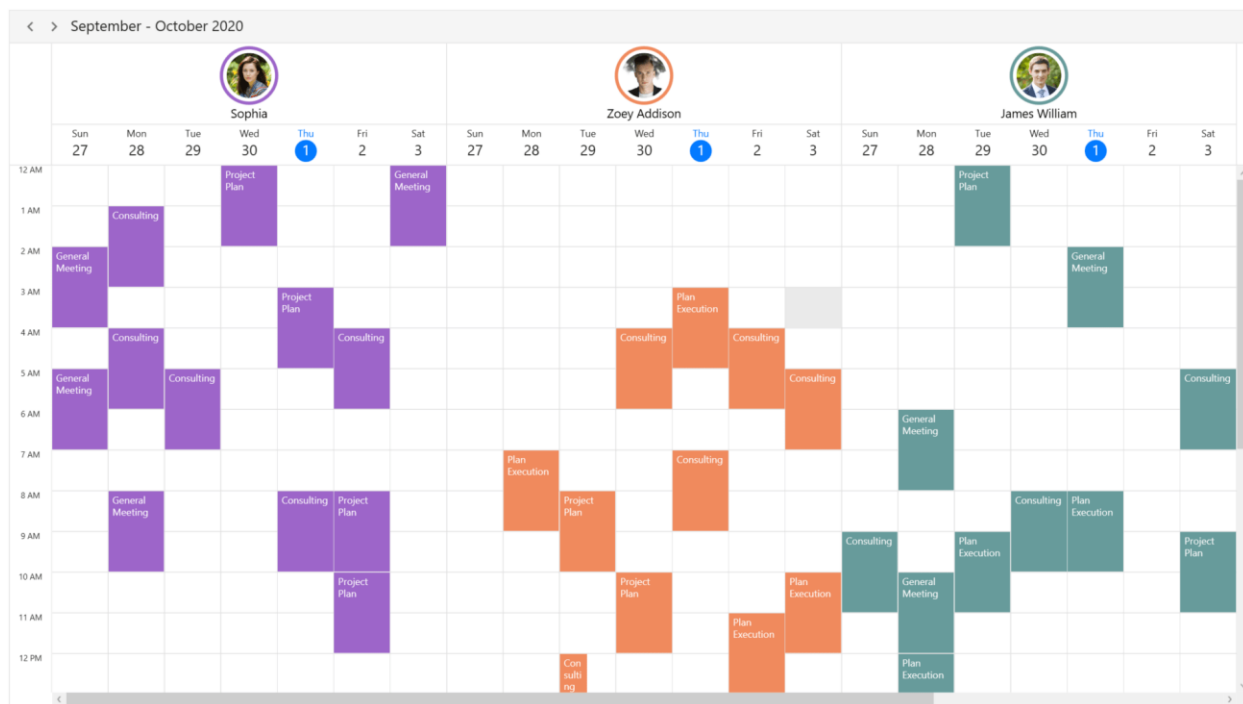
```

var schedule =
Syncfusion.Windows.Shared.VisualUtils.FindVisualParent<SfScheduler>(containe
r);
if (schedule == null)
return null;
if (schedule.ViewType == SchedulerViewType.Day || schedule.ViewType ==
SchedulerViewType.Week || schedule.ViewType == SchedulerViewType.WorkWeek)
return DayViewResourceTemplate;
else
return TimelineViewResourceTemplate;
}
}

```

Note

- By default, the **SchedulerResource** is set as the **DataContext** for the **ResourceHeaderTemplateSelector** for both **SchedulerResource** and custom data object in the **ResourceCollection**.
- The custom data object can be bound in the **ResourceHeaderTemplateSelector** by using the property of **SchedulerResource.Data**.



Note: [View sample in GitHub](#)

Appointments in WPF Scheduler (SfScheduler)

The **SfScheduler** control has a built-in capability to handle the appointment arrangement internally based on the **ScheduleAppointmentCollection**. The **WPF Scheduler** supports to render normal, all-day appointments, spanned appointment, recurring appointments, and recurrence exception dates appointments.

The [ScheduleAppointment](#) is a class that includes the specific scheduled appointment. It has some basic properties such as [StartTime](#), [EndTime](#), [Subject](#), and some additional information about the appointment can be added with [Notes](#), [Location](#), and [IsAllDay](#) properties.

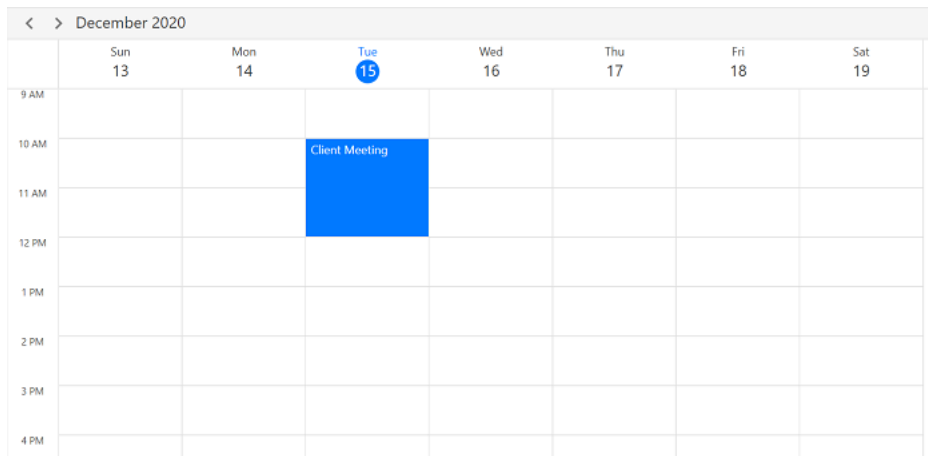
{%tabs %}

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ItemsSource="{Binding scheduleAppointmentCollection}">
</syncfusion:SfScheduler>
```

C#

```
// Creating an instance for schedule appointment collection
var scheduleAppointmentCollection = new ScheduleAppointmentCollection();
//Adding schedule appointment in the schedule appointment collection
scheduleAppointmentCollection.Add(new ScheduleAppointment()
{
    StartTime = new DateTime(2020, 12, 15, 10, 0, 0),
    EndTime = new DateTime(2020, 12, 15, 12, 0, 0),
    Subject = "Client Meeting",
    Location = "Hutchison road",
});
//Adding schedule appointment collection to the ItemSource of SfScheduler
Schedule.ItemsSource = scheduleAppointmentCollection;
```



Note:

- The Scheduler supports the functionality that arranges the appointments based on their start time and duration for the normal appointments in a day, week and workweek views.
- In an all-day panel of the day, week and workweek views, span and all day appointments are ordered and rendered based on the start date-time of an appointment that consists time duration of the appointment, followed by `IsSpanned`, `IsAllDay`, appointments respectively.
- In Timeline views, all the appointments (span, all day, and normal) are ordered and rendered based on the start date-time of an appointment that consists of the time duration of the appointment, followed by `IsSpanned`, `IsAllDay`, and normal appointments respectively.

[View sample in GitHub](#)

Scheduler item source and Mapping

The **Scheduler** supports to bind any collection that implements the **IEnumerable** interface to populate appointments. Map the properties in business object to [ScheduleAppointment](#) by configuring the [AppointmentMapping](#) property. The following table that property shows mapping property details to [ScheduleAppointment](#).

Property Name	Description
StartTime	Maps the property name of a custom class, which is equivalent to the StartTime of ScheduleAppointment .
EndTime	Maps the property name of a custom class, which is equivalent to the EndTime of ScheduleAppointment .
StartTimeZone	Maps the property name of a custom class, which is equivalent to the StartTimeZone of ScheduleAppointment .
EndTimeZone	Maps the property name of a custom class, which is equivalent to the EndTimeZone of ScheduleAppointment .
Subject	Maps the property name of a custom class, which is equivalent to the Subject of ScheduleAppointment .
Id	Maps the property name of a custom class, which is equivalent to the Id of ScheduleAppointment .
AppointmentBackground	Maps the property name of a custom class, which is equivalent to the AppointmentBackground of ScheduleAppointment .
Foreground	Maps the property name of a custom class, which is equivalent to the Foreground of ScheduleAppointment .
IsAllDay	Maps the property name of a custom class, which is equivalent to the IsAllDay of ScheduleAppointment .
RecurrenceRule	Maps the property name of a custom class, which is equivalent to the RecurrenceRule of ScheduleAppointment .
RecurrenceId	Maps the property name of a custom class, which is equivalent to the RecurrenceId of ScheduleAppointment .
Notes	Maps the property name of a custom class, which is equivalent to the Notes of ScheduleAppointment .

Location	Maps the property name of a custom class, which is equivalent to the Location of ScheduleAppointment.
RecurrenceExceptionDates	Maps the property name of a custom class, which is equivalent to the RecurrenceExceptionDates of ScheduleAppointment.
ResourceIdCollection	Maps the property name of a custom class, which is equivalent to the ResourceIdCollection of ScheduleAppointment.

Note: The CustomAppointment class should contain event start and end date time fields as mandatory

Creating business objects

Create a custom class **Meeting** with mandatory fields **From**, **To** and **EventName**.

C#

```

/// <summary>
/// Represents the custom data properties.
/// </summary>
public class Meeting
{
    public string EventName { get; set; }
    public DateTime From { get; set; }
    public DateTime To { get; set; }
    public Brush BackgroundColor { get; set; }
    public Brush ForegroundColor { get; set; }
}

```

Note: You can map those properties of the **Meeting** class to schedule appointment by using the AppointmentMapping properties.

{%tabs %}

XML

```

<syncfusion:SfScheduler x:Name="Schedule"
    ViewType="Week"
    ItemsSource="{Binding Meetings}">
    <syncfusion:SfScheduler.AppointmentMapping>
    <syncfusion:AppointmentMapping
        Subject="EventName"
        StartTime="From"
        EndTime="To"
        AppointmentBackground="BackgroundColor"
        Foreground="ForegroundColor"/>
    </syncfusion:SfScheduler.AppointmentMapping>
</syncfusion:SfScheduler>

```

C#

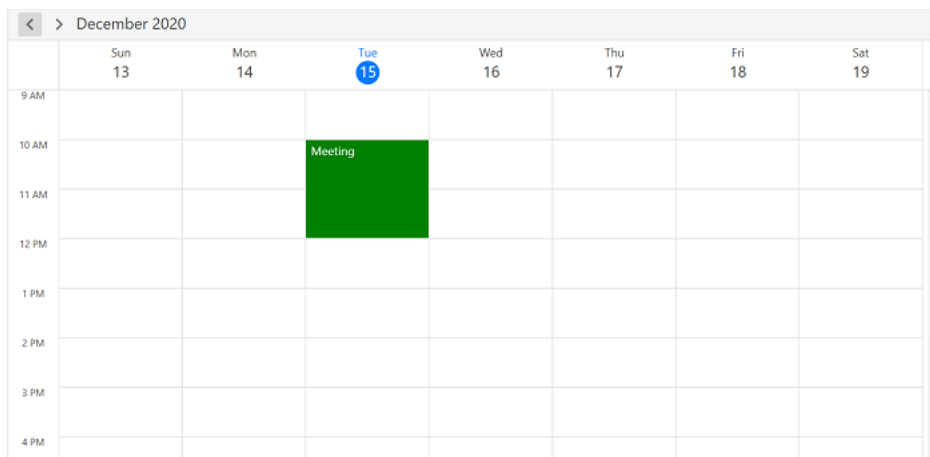
```
//Schedule data mapping for custom appointments
AppointmentMapping dataMapping = new AppointmentMapping();
dataMapping.Subject = "EventName";
dataMapping.StartTime = "From";
dataMapping.EndTime = "To";
dataMapping.AppointmentBackground = "BackgroundColor";
dataMapping.Foreground = "ForegroundColor";
Schedule.AppointmentMapping = dataMapping;
```

Schedule the meetings for a day by setting **From** and **To** of Meeting class. Create meetings of type **ObservableCollection<Meeting>** and assign those appointments collection Meetings to the **ItemsSource** property which is of **IEnumerable** type.

C#

```
//Creating an instance for the custom appointment class
Meeting meeting = new Meeting();
//Setting the start time of an event
meeting.From = new DateTime(2020, 12, 15, 10, 0, 0);
//Setting the end time of an event
meeting.To = meeting.From.AddHours(2);
//Setting the subject for an event
meeting.EventName = "Meeting";
//Setting the background color for an event
meeting.BackgroundColor = new SolidColorBrush(Colors.Green);
//Setting the foreground color for an event
meeting.ForegroundColor = new SolidColorBrush(Colors.White);
//Creating an instance for the collection of custom appointments
var Meetings = new ObservableCollection<Meeting>();
//Adding a custom appointment in the CustomAppointmentCollection
Meetings.Add(meeting);
//Adding custom appointments in the ItemsSource of SfScheduler
Schedule.ItemsSource = Meetings;
```

{% endtabs%}



Note: [View sample in GitHub](#)

Spanned appointments

Spanned Appointment is an appointment that lasts more than 24 hours. It doesn't block out the time slots in the `SfScheduler`, it will render in [AllDayAppointmentPanel](#) exclusively.

{%tabs %}

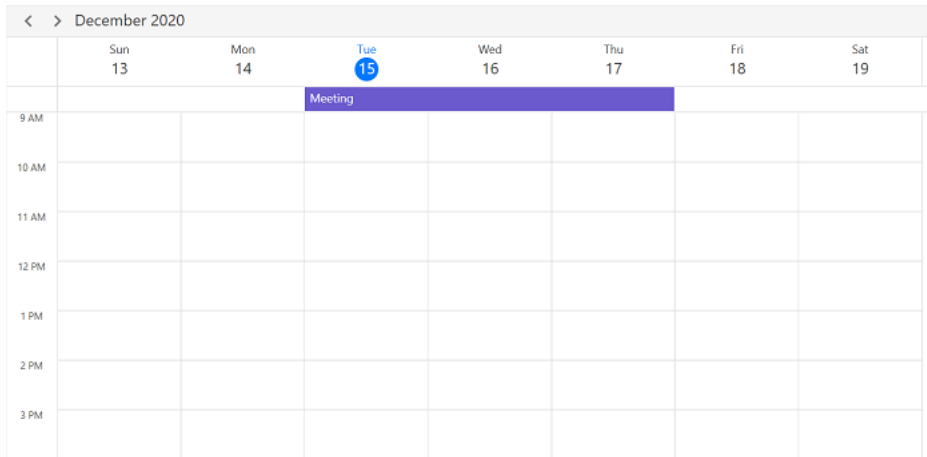
XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ItemsSource="{Binding Meetings}">
<syncfusion:SfScheduler.AppointmentMapping>
<syncfusion:AppointmentMapping
Subject="EventName"
StartTime="From"
EndTime="To"
AppointmentBackground="BackgroundColor"
Foreground="ForegroundColor">
</syncfusion:AppointmentMapping>
</syncfusion:SfScheduler.AppointmentMapping>
</syncfusion:SfScheduler>
```

C#

```
// Creating an instance for the collection of custom appointments
var Meetings = new ObservableCollection<Meeting>();
// Creating an instance for the custom appointment class
Meeting meeting = new Meeting();
// Setting the start time of an event
meeting.From = new DateTime(2020, 12, 15, 10, 0, 0);
// Setting the end time of an event
meeting.To = meeting.From.AddDays(2).AddHours(1);
// Setting the subject for an event
meeting.EventName = "Meeting";
// Setting the background color for an event
meeting.BackgroundColor = new SolidColorBrush(Colors.SlateBlue);
// Setting the foreground color for an event
meeting.ForegroundColor = new SolidColorBrush(Colors.White);
// Adding a custom appointment in the CustomAppointmentCollection
Meetings.Add(meeting);
//Adding schedule appointment collection to the ItemsSource of SfSchedule
Schedule.ItemsSource = Meetings;
```

{% endtabs %}

**Note:**

- In an all-day panel of the day, week and workweek views, span and all day appointments are ordered and rendered based on the start date-time of the appointment that consists time duration of an appointment, followed by `IsSpanned`, `IsAllDay`, appointments respectively.

[View sample in GitHub](#)

All day appointments

The all-Day appointment is an appointment that is scheduled for a whole day. It can be set by using the `IsAllDay` property in the [ScheduleAppointment](#).

{%tabs %}

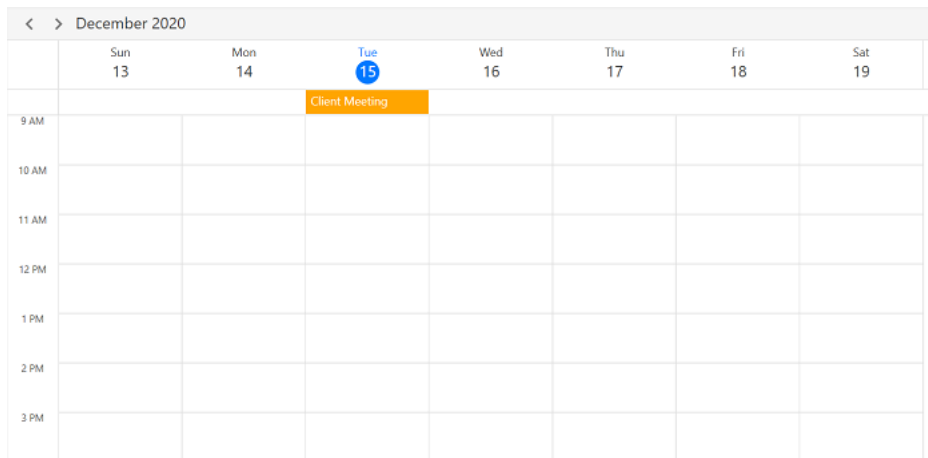
XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ItemsSource="{Binding scheduleAppointmentCollection}">
</syncfusion:SfScheduler>
```

C#

```
// Creating an instance for schedule appointment collection
var scheduleAppointmentCollection = new ScheduleAppointmentCollection();
//Adding schedule appointment in the schedule appointment collection
scheduleAppointmentCollection.Add(new ScheduleAppointment()
{
    StartTime = new DateTime(2020, 12, 15, 10, 0, 0),
    EndTime = new DateTime(2020, 12, 15, 12, 0, 0),
    Subject = "Client Meeting",
    Location = "Hutchison road",
    AppointmentBackground = Brushes.Orange,
    Foreground = Brushes.White,
    IsAllDay = true,
});
//Adding the schedule appointment collection to the ItemsSource of
SfScheduler
Schedule.ItemsSource = scheduleAppointmentCollection;
```

{% endtabs%}

**Note:**

- Appointment that lasts for an entire day (exact 24 hours) will be considered as an all-day appointment without setting the `IsAllDay` property. For example: From 06/29/2020 12:00AM to 06/30/2020 12:00AM.
- In an all-day panel of the day, week and workweek views, span and all day appointments are ordered and rendered based on the start date-time of an appointment that consists time duration of the appointment, followed by `IsSpanned`, `IsAllDay`, appointments respectively.

[View sample in GitHub](#)

Recurrence Appointment

The recurring appointment on a daily, weekly, monthly, or yearly interval. Recurring appointments can be created by setting the `RecurrenceRule` property in `ScheduleAppointment`.

Recurrence rule

The `RecurrenceRule` is a string value (RRULE) that contains the details of the recurrence appointments such as repeat type - daily or weekly or monthly or yearly, how many times it needs to be repeated, the interval duration, also the time period to render the appointment, and more. The `RecurrenceRule` has the following properties and based on this property value, the recurrence appointments are rendered in the SfScheduler with its respective time period.

PropertyName	Purpose
FREQ	Maintains the Repeat type value of the appointment. (Example: Daily, Weekly, Monthly, Yearly, Every week day) Example:FREQ=DAILY;INTERVAL=1
INTERVAL	Maintains the interval value of the appointments. For example, while creating the daily appointment at an interval of 2, the appointments are rendered on the days Monday, Wednesday and Friday. (creates the appointment on all days by leaving the interval of one day gap) Example:FREQ=DAILY;INTERVAL=1
COUNT	It holds the appointment's count value. For example, when the recurrence appointment count value is 10, it means 10 appointments are created in the recurrence series. Example:FREQ=DAILY;INTERVAL=1;COUNT=10

UNTIL	This property is used to store the recurrence end date value. For example, when the end date of appointment is set as 6/30/2020, the UNTIL property holds the end date value when the recurrence actually ends. Example:FREQ=DAILY;INTERVAL=1;UNTIL=20200725
BYDAY	It holds the "DAY" values of an appointment to render. For example, while creating the weekly appointment, select the day(s) from the day options (Monday/Tuesday/Wednesday/Thursday/Friday/Saturday/Sunday). When Monday is selected, the first two letters of the selected day "MO" is stored in the "BYDAY" property. While selecting the multiple days, the values are separated by commas. Example:FREQ=WEEKLY;INTERVAL=1;BYDAY=MO,WE;COUNT=10
BYMONTHDAY	This property is used to store the date value of the Month while creating the Month recurrence appointment. For example, while creating a Monthly recurrence appointment in the date 3, it means the BYMONTHDAY holds the value 3 and creates the appointment on 3rd day of every month. Example:FREQ=MONTHLY;BYMONTHDAY=3;INTERVAL=1;COUNT=10
BYMONTH	This property is used to store the index value of the selected Month while creating the yearly appointments. For example, while creating the yearly appointment in the Month June, it means the index value for June month is 6 and it is stored in the BYMONTH field. The appointment is created on every 6th month of a year. Example:FREQ=YEARLY;BYMONTHDAY=16;BYMONTH=6;INTERVAL=1;COUNT=10
BYSETPOS	This property is used to store the index value of the week. For example, while creating the monthly appointment in second week of the month, the index value of the second week (2) is stored in BYSETPOS. Example:FREQ=MONTHLY;BYDAY=MO;BYSETPOS=2;UNTIL=20200725

Adding recurrence appointment

The SfScheduler appointment recurrenceRule is used to populate the required recurring appointment collection in a specific pattern. The RRULE can be directly set to the [RecurrenceRule](#) property of [ScheduleAppointment](#).

{%tabs %}

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ItemsSource="{Binding scheduleAppointmentCollection}">
</syncfusion:SfScheduler>
```

C#

```
// Creating an instance for schedule appointment collection
var scheduleAppointmentCollection = new ScheduleAppointmentCollection();
//Adding schedule appointment in schedule appointment collection
var scheduleAppointment = new ScheduleAppointment()
{
    Id = 1,
    StartTime = new DateTime(2020, 12, 13, 11, 0, 0),
```

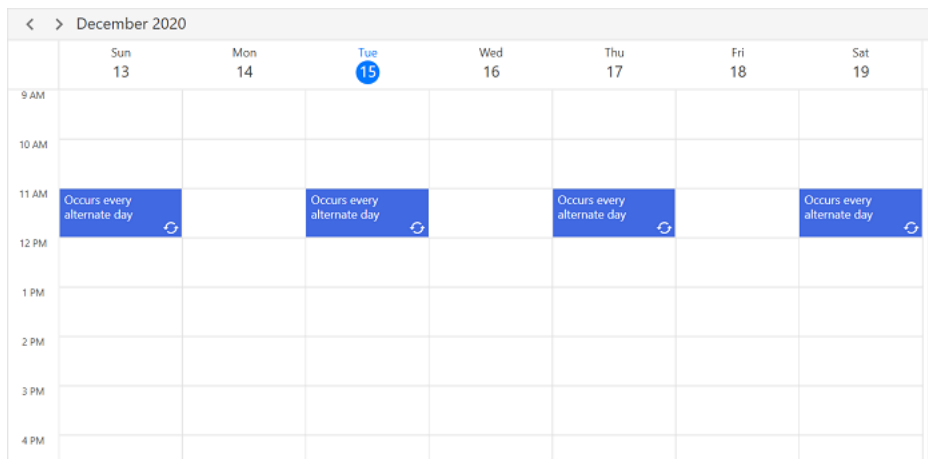
```

EndTime = new DateTime(2020, 12, 13, 12, 0, 0),
Subject = "Occurs every alternate day",
AppointmentBackground = Brushes.RoyalBlue,
Foreground = Brushes.White,
};
//Creating recurrence rule
scheduleAppointment.RecurrenceRule = "FREQ=DAILY;INTERVAL=2;COUNT=10";
//Adding schedule appointment to the schedule appointment collection
scheduleAppointmentCollection.Add(scheduleAppointment);
//Setting the schedule appointment collection to the ItemsSource of
SfScheduler
Schedule.ItemsSource = scheduleAppointmentCollection;

```

{% endtabs%}

Note: [View sample in GitHub](#)



Creating the custom recurrence appointment

For creating the custom recurrence appointment, create a custom class `Meeting` with mandatory fields `from`, `to`, and `recurrenceRule`.

C#

```

/// <summary>
/// Represents custom data properties.
/// </summary>
public class Meeting
{
    public string EventName { get; set; }
    public DateTime From { get; set; }
    public DateTime To { get; set; }
    public Brush BackgroundColor { get; set; }
    public Brush ForegroundColor { get; set; }
    public string RecurrenceRule { get; set; }
    public object Id { get; set; }
}

```

Note: Map those properties of `Meeting` class to schedule appointment by using the `AppointmentMapping` properties.

{%tabs %}

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ItemsSource="{Binding
Appointments}" ViewType="Week">
<syncfusion:SfScheduler.AppointmentMapping>
<syncfusion:AppointmentMapping
Subject="EventName"
StartTime="From"
EndTime="To"
Id="Id"
AppointmentBackground="BackgroundColor"
Foreground="ForegroundColor"
RecurrenceRule="RecurrenceRule"
IsAllDay="IsAllDay"/>
</syncfusion:SfScheduler.AppointmentMapping>
</syncfusion:SfScheduler>
```

C#

```
//Schedule data mapping for custom appointments
AppointmentMapping dataMapping = new AppointmentMapping();
dataMapping.Subject = "EventName";
dataMapping.StartTime = "From";
dataMapping.EndTime = "To";
dataMapping.Id = "Id";
dataMapping.AppointmentBackground = "BackgroundColor";
dataMapping.Foreground = "ForegroundColor";
dataMapping.RecurrenceRule = "RecurrenceRule";
Schedule.AppointmentMapping = dataMapping;
```

Schedule the recurring meetings for daily, weekly, monthly, or yearly interval by setting the [RecurrenceRule](#) of Meeting class. Create meetings of type `ObservableCollection<Meeting>` and assign those appointments collection Meetings to the [ItemsSource](#) property which is of `IEnumerable` type.

C#

```
// Creating an instance for the custom appointment class
Meeting meeting = new Meeting();
// Setting the start time of an event
meeting.From = new DateTime(2020, 12, 13, 10, 0, 0);
// Setting the end time of an event
meeting.To = meeting.From.AddHours(2);
// Setting the Id for an event
meeting.Id = 1;
// Setting the subject for an event
meeting.EventName = "Client Meeting";
// Setting the background color for an event
meeting.BackgroundColor = new SolidColorBrush(Colors.Gray);
// Setting the foreground color for an event
meeting.ForegroundColor = new SolidColorBrush(Colors.White);
// Creating recurrence rule
meeting.RecurrenceRule = "FREQ=DAILY;INTERVAL=1;COUNT=10";
// Creating an instance for collection of custom appointments
```

```
var Meetings = new ObservableCollection<Meeting>();  
// Adding a custom appointment in CustomAppointmentCollection  
Meetings.Add(meeting);  
// Adding custom appointments in ItemsSource of SfSchedule  
Schedule.ItemsSource = Meetings;
```

Note: [View sample in GitHub](#)

How to get the Recurrence editor field values from RRULE?

Get the Recurrence properties from the [RRULE](#) using the [RRuleParser](#) method of [SfScheduler](#).

C#

```
DateTime dateTime = new DateTime(2020, 6, 30, 10, 0, 0);  
RecurrenceProperties recurrenceProperties =  
RecurrenceHelper.RRuleParser("FREQ=DAILY; INTERVAL=1; COUNT=3", dateTime);
```

Recurrence properties retrieved from above method,

```
recurrenceProperties.RecurrenceType = RecurrenceType.Daily;
```

```
recurrenceProperties.Interval = 1;
```

```
recurrenceProperties.RecurrenceCount = 3;
```

```
recurrenceProperties.RecurrenceRange = RecurrenceRange.Count;
```

How to get the recurrence dates from RRULE?

Get the occurrences date-time list of recurring appointment from the RRULE using the [GetRecurrenceDateTimeCollection](#) method of [SfScheduler](#).

C#

```
DateTime dateTime = new DateTime(2020, 6, 27, 9, 0, 0);  
IEnumerable<DateTime> dateCollection =  
RecurrenceHelper.GetRecurrenceDateTimeCollection("FREQ=DAILY; INTERVAL=1; COUNT=3", dateTime);
```

The following occurrence dates can be retrieved from the given RRULE:

```
var date0 = 6/27/2020;
```

```
var date1 = 6/28/2020;
```

```
var date2 = 6/29/2020;
```

How to get pattern appointment for the specified occurrence?

Gets the pattern appointment for the specified occurrence.

To get the pattern appointment by using the following event and passing a parameter as [Scheduler](#) and Specified [Appointment](#).

C#

```
Schedule.AppointmentTapped += Schedule_AppointmentTapped;  
private void Schedule_AppointmentTapped(object sender, AppointmentTappedArgs e)  
{
```

```
if (e.Appointment != null)
{
    var patternAppointment =
    RecurrenceHelper.GetPatternAppointment(this.Schedule, e.Appointment);
}
```

Note:

- For custom appointment, pass e.Appointment.Data as a param and get the custom appointment details from the Data property of ScheduleAppointment.
- If a specified occurrence is changed, the GetPatternAppointment method returns the pattern appointment of exception appointment.

How to get occurrence appointment at the specified date?

Gets an occurrence at the specified date within a series of recurring appointments.

To get a specific appointment by using the following event and passing a parameter as Scheduler, Specified Appointment and specified DateTime.

C#

```
Schedule.AppointmentTapped += Schedule_AppointmentTapped;
private void Schedule_AppointmentTapped(object sender, AppointmentTappedArgs e)
{
    if (e.Appointment != null)
    {
        var occurrenceAppointment =
        RecurrenceHelper.GetOccurrenceAppointment(this.Schedule, e.Appointment, new
        DateTime(2020, 12, 20));
    }
}
```

Note: If an occurrence at the specified date is deleted or not present, then the GetOccurrenceAppointment method returns null.

Recurrence pattern exceptions

Delete or change any recurrence pattern appointment by handling exception dates and exception appointments to that recurring appointment.

Recurrence exception dates

Delete any occurrence appointment, which is exception from the recurrence pattern appointment by adding exception dates to the recurring appointment.

Recurrence exception appointment

Change any occurrence appointment which is exception from recurrence pattern appointment by adding the recurrence exception appointment in the SfScheduler [ItemsSource](#).

Creating the recurrence exceptions for schedule appointment

Add the recurrence exception appointments and recurrence exception dates to ScheduleAppointment or remove them from the [ScheduleAppointment](#) by using its [RecurrenceExceptionDates](#) property.

Delete occurrence from recurrence pattern appointment or adding exception dates to recurrence pattern appointment

Delete any of occurrence which is exception from recurrence pattern appointment by using the `RecurrenceExceptionDates` property of `ScheduleAppointment`. The deleted occurrence date will be considered as recurrence exception date.

{%tabs %}

XML

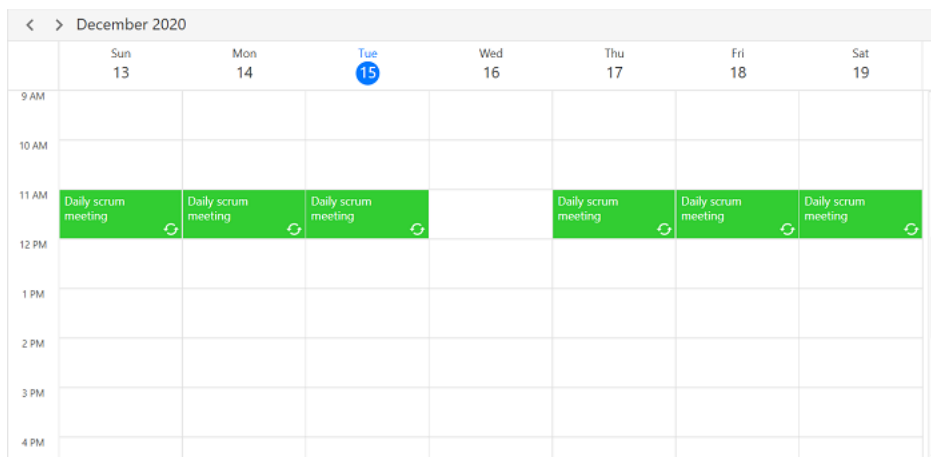
```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ItemsSource="{Binding scheduleAppointmentCollection}">
</syncfusion:SfScheduler>
```

C#

```
// Creating an instance for schedule appointment collection
var scheduleAppointmentCollection = new ScheduleAppointmentCollection();
// Recurrence and exception appointment
var scheduleAppointment = new ScheduleAppointment
{
    Id = 1,
    Subject = "Daily scrum meeting",
    StartTime = new DateTime(2020, 12, 13, 11, 0, 0),
    EndTime = new DateTime(2020, 12, 13, 12, 0, 0),
    AppointmentBackground = new SolidColorBrush(Colors.LimeGreen),
    Foreground = new SolidColorBrush(Colors.White),
    RecurrenceRule = "FREQ=DAILY; INTERVAL=1; COUNT=10"
};
//Adding the recurring or pattern appointment to the Schedule
AppointmentCollection.
scheduleAppointmentCollection.Add(scheduleAppointment);
//Add the ExceptionDates to avoid occurrence on specific dates.
DateTime exceptionDate = scheduleAppointment.StartTime.AddDays(3).Date;
scheduleAppointment.RecurrenceExceptionDates = new
ObservableCollection<DateTime>()
{
    exceptionDate,
};
//Setting AppointmentCollection as ItemSource of SfScheduler.
Schedule.ItemsSource = scheduleAppointmentCollection;
```

Note: [View sample in GitHub](#)

Note: Exception dates should be Universal Time Coordinates (UTC) time zone.



Add exception appointment to the recurrence pattern

Add an exception appointment which is changed or modified occurrence of the recurrence pattern appointment to the [ItemsSource](#) of Scheduler. To add a changed occurrence, ensure to set the [RecurrenceId](#) of that occurrence and add the date of that occurrence to the [RecurrenceExceptionDates](#) of recurrence pattern appointment. The [RecurrenceId](#) of changed occurrence should hold the exact recurrence pattern appointment [Id](#).

{%tabs %}

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ItemsSource="{Binding scheduleAppointmentCollection}">
</syncfusion:SfScheduler>
```

C#

```
// Creating an instance for schedule appointment collection
var appointmentCollection = new ScheduleAppointmentCollection();
// Recurrence and exception appointment
var scheduleAppointment = new ScheduleAppointment
{
    Id = 1,
    Subject = "Daily scrum meeting",
    StartTime = new DateTime(2020, 12, 13, 11, 0, 0),
    EndTime = new DateTime(2020, 12, 13, 12, 0, 0),
    AppointmentBackground = new
        SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF00BFFF")),
    Foreground = new
        SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFFFFF")),
    RecurrenceRule = "FREQ=DAILY;INTERVAL=1;COUNT=10"
};
//Adding the recurring or pattern appointment to the AppointmentCollection.
appointmentCollection.Add(scheduleAppointment);
//Add ExceptionDates to avoid occurrence on specific dates.
DateTime changedExceptionDate =
    scheduleAppointment.StartTime.AddDays(3).Date;
scheduleAppointment.RecurrenceExceptionDates = new
    ObservableCollection<DateTime>()
```

```

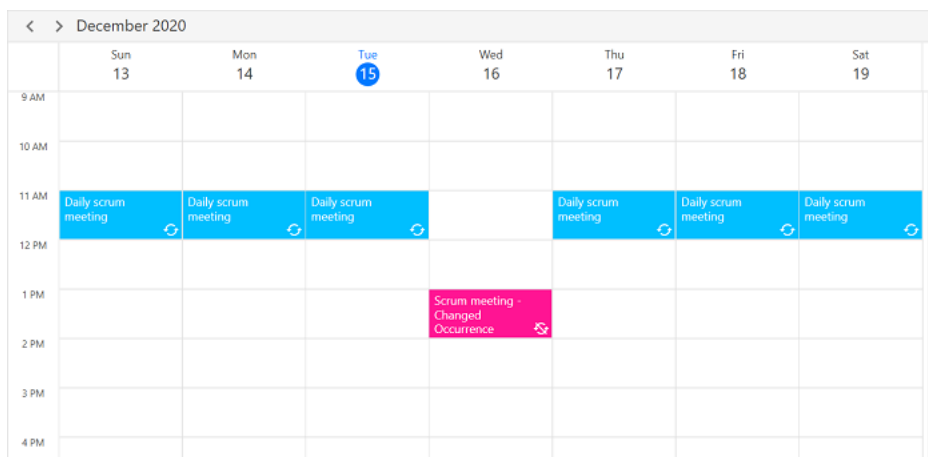
{
    changedExceptionDate,
};
//Creating an exception occurrence appointment by changing the start time or
end time.
// RecurrenceId is set to 1, so it will be the changed occurrence for the
above-created pattern appointment.
var exceptionAppointment = new ScheduleAppointment()
{
    Id = 2,
    Subject = "Scrum meeting - Changed Occurrence",
    StartTime = new DateTime(changedExceptionDate.Year,
        changedExceptionDate.Month, changedExceptionDate.Day, 13, 0, 0),
    EndTime = new DateTime(changedExceptionDate.Year,
        changedExceptionDate.Month, changedExceptionDate.Day, 14, 0, 0),
    AppointmentBackground = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFFF1493")),
    Foreground = new
    SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFFFFF")),
    RecurrenceId = 1
};
// Adding an exception occurrence appointment to the AppointmentCollection.
appointmentCollection.Add(exceptionAppointment);
//Setting the AppointmentCollection as a ItemSource of SfScheduler.
Schedule.ItemsSource = appointmentCollection;

```

Note: • The `RecurrenceId` of an exception appointment and the `Id` of its pattern appointment should have the same value.

- The Exception recurrence appointment does not have the `RecurrenceRule`, so for an exception appointment, it will be reset to empty.
- The exception appointment should have different `Id` with original pattern appointment `Id`.
- The exception appointment should be a normal appointment and should not be created as recurring appointment, since its occurrence is from recurrence pattern.
- The recurrenceExceptionDates should be in a Universal Time Coordinates (UTC) time zone.

Note: [View sample in GitHub](#)



Create recurrence exceptions for custom appointment

Add the recurrence exception appointments and recurrence exception dates to the CustomAppointment or remove them from CustomAppointment, create a custom class `Meeting` with mandatory fields [RecurrenceExceptionDates](#) and [RecurrenceId](#).

Delete occurrence from the recurrence pattern appointment or adding exception dates to recurrence pattern appointment

Delete any occurrence which is an exception from the recurrence pattern appointment by using the [RecurrenceExceptionDates](#) property of [AppointmentMapping](#) class which is used to map the exception dates to the schedule recurrence appointment. The deleted occurrence date will be considered as recurrence exception date.

To add the exception dates in the recurrence series of custom appointment, add the `RecurrenceExceptionDates`, `EventName`, `From`, `To`, `Color`, `RecurrenceRule` properties to the custom class `Meeting`.

C#

```
public class Meeting
{
    public ObservableCollection<DateTime> RecurrenceExceptions { get; set; } =
    new ObservableCollection<DateTime>();
    public string EventName { get; set; }
    public DateTime From { get; set; }
    public DateTime To { get; set; }
    public object Id { get; set; }
    public Brush BackgroundColor { get; set; }
    public Brush ForegroundColor { get; set; }
    public string RecurrenceRule { get; set; }
    public object RecurrenceId { get; set; }
}
```

Map this custom property `RecurrenceExceptionDates` of custom class with the `RecurrenceExceptionDates` property of `AppointmentMapping` class to map the exception dates to the scheduled appointment.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ItemsSource="{Binding customAppointmentCollection}>
<syncfusion:SfScheduler.AppointmentMapping>
<syncfusion:AppointmentMapping
Subject="EventName"
StartTime="From"
EndTime="To"
Id="Id"
AppointmentBackground="BackgroundColor"
Foreground="ForegroundColor"
IsAllDay="IsAllDay"
StartTimeZone="StartTimeZone"
RecurrenceRule="RecurrenceRule"
RecurrenceExceptionDates="RecurrenceExceptions"
EndTimeZone="EndTimeZone"
RecurrenceId="RecurrenceId"/>
```

```
</syncfusion:SfScheduler.AppointmentMapping>
</syncfusion:SfScheduler>
```

C#

```
// Creating an instance for custom appointment collection
ObservableCollection<Meeting> customAppointmentCollection = new
ObservableCollection<Meeting>();
// data mapping for custom appointments.
AppointmentMapping dataMapping = new AppointmentMapping();
dataMapping.RecurrenceExceptionDates = "RecurrenceExceptionDates";
// Create the new exception date.
var exceptionDate = new DateTime(2020, 07, 09);
//Adding custom appointment in the custom appointment collection
var recurrenceAppointment = new Meeting()
{
    From = new DateTime(2020, 07, 05, 10, 0, 0),
    To = new DateTime(2020, 07, 05, 11, 0, 0),
    EventName = "Occurs Daily",
    BackgroundColor = new SolidColorBrush(Colors.LightSeaGreen),
    ForegroundColor = new SolidColorBrush(Colors.White),
    Id = 1
};
// Creating recurrence rule
recurrenceAppointment.RecurrenceRule = "FREQ=DAILY;COUNT=20";
// Add RecurrenceExceptionDates to appointment.
recurrenceAppointment.RecurrenceExceptions = new
ObservableCollection<DateTime>()
{
    exceptionDate
};
//Adding custom appointment in the custom appointment collection
customAppointmentCollection.Add(recurrenceAppointment);
//Adding custom appointment collection to the ItemsSource of SfScheduler
Schedule.ItemsSource = customAppointmentCollection;
```

Note: Exception dates should be Universal Time Coordinates (UTC) time zone.

Note: [View sample in GitHub](#)

< > December 2020	Sun 13	Mon 14	Tue 15	Wed 16	Thu 17	Fri 18	Sat 19
9 AM							
10 AM							
11 AM	Occurs Daily	Occurs Daily	Occurs Daily		Occurs Daily	Occurs Daily	Occurs Daily
12 PM							
1 PM							
2 PM							
3 PM							
4 PM							

Add an exception appointment to the recurrence pattern

Add an exception appointment which is changed or modified occurrence of the recurrence pattern appointment to the [ItemsSource](#) of Scheduler. To add the changed occurrence, ensure to set the [RecurrenceId](#) of that occurrence and add the date of that occurrence to [RecurrenceExceptionDates](#) of recurrence pattern appointment. The [RecurrenceId](#) of changed occurrence should hold the exact recurrence pattern appointment [Id](#)

Map the equivalent properties of [Id](#), [RecurrenceId](#) and [RecurrenceExceptionDates](#) properties from the business object to the [Id](#) and [RecurrenceExceptionDates](#) properties of [AppointmentMapping](#).

Add the created exception recurrence appointment to the SfScheduler `ItemsSource`.

C#

```
//// Creating an instance for schedule appointment collection
public ObservableCollection<Meeting> RecursiveAppointmentCollection
{
    get;
    set;
}
```

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ItemsSource="{Binding RecursiveAppointmentCollection}">
<syncfusion:SfScheduler.AppointmentMapping>
<syncfusion:AppointmentMapping
Subject="EventName"
StartTime="From"
EndTime="To"
Id="Id"
AppointmentBackgroundColor="BackgroundColor"
ForegroundColor="ForegroundColor"
RecurrenceRule="RecurrenceRule"
RecurrenceId="RecurrenceId"/>
</syncfusion:SfScheduler.AppointmentMapping>
</syncfusion:SfScheduler>
```

C#

```
this.RecursiveAppointmentCollection = new ObservableCollection<Meeting>();
//Adding custom appointment in the custom appointment collection
Meeting dailyEvent = new Meeting
{
    EventName = "Daily scrum meeting",
    From = new DateTime(2020, 12, 13, 11, 0, 0),
    To = new DateTime(2020, 12, 13, 12, 0, 0),
    BackgroundColor = new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF00BFFF")),
    ForegroundColor = new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFFFFF")),
    RecurrenceRule = "FREQ=DAILY;INTERVAL=1;COUNT=10",
    Id = 1
};
```

```

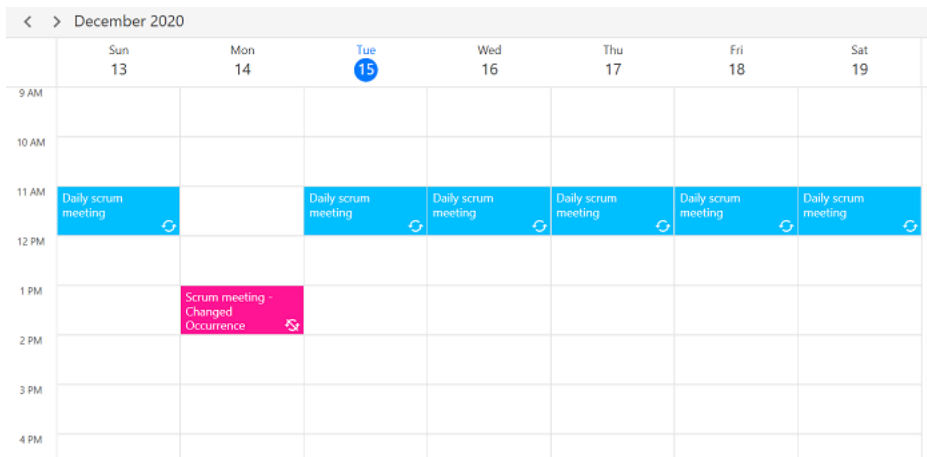
//Adding custom appointment in the custom appointment collection
RecursiveAppointmentCollection.Add(dailyEvent);
//Add ExceptionDates to avoid occurrence on specific dates.
DateTime changedExceptionDate = DateTime.Now.AddDays(-1).Date;
dailyEvent.RecurrenceExceptions = new ObservableCollection<DateTime>()
{
    changedExceptionDate
};
//Change start time or end time of an occurrence.
Meeting changedEvent = new Meeting
{
    EventName = "Scrum meeting - Changed Occurrence",
    From = new DateTime(changedExceptionDate.Year, changedExceptionDate.Month,
        changedExceptionDate.Day, 13, 0, 0),
    To = new DateTime(changedExceptionDate.Year, changedExceptionDate.Month,
        changedExceptionDate.Day, 14, 0, 0),
    BackgroundColor = new
        SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFFF1493")),
    ForegroundColor = new
        SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFFFFF")),
    RecurrenceRule = "FREQ=DAILY; INTERVAL=1; COUNT=10",
    Id = 2,
    RecurrenceId = 1
};
RecursiveAppointmentCollection.Add(changedEvent);
//Adding custom appointment collection to the ItemsSource of SfScheduler
Schedule.ItemsSource = RecursiveAppointmentCollection;

```

Note: The **RecurrenceId** of exception appointment and the **Id** of its pattern appointment should have same value.

- The Exception recurrence appointment does not have the **RecurrenceRule**, so for an exception appointment, it will be reset to empty.
 - The exception appointment should have different **Id** with original pattern appointment **Id**.
 - The exception appointment should be a normal appointment and should not be created as recurring appointment, since its occurrence from the recurrence pattern.
 - The recurrenceExceptions should be in a Universal Time Coordinates (UTC) time zone.
-

Note: [View sample in GitHub](#)



Appearance customization

The default appearance of schedule appointment can be customized in all the views by using the [AppointmentTemplate](#) and [AppointmentTemplateSelector](#) properties of [ViewSettingsBase](#). Use the [AllDayAppointmentTemplate](#) property of [DaysViewSettings](#) to customize the appearance of all day appointments in day, week and work week views.

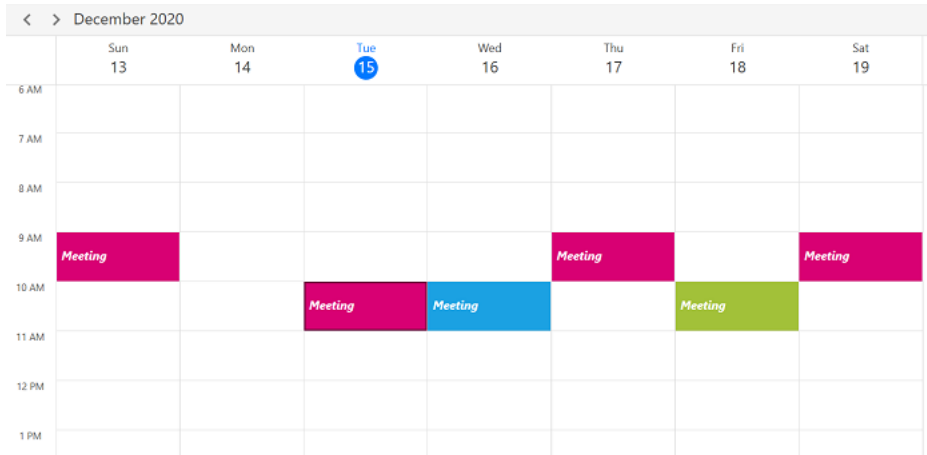
XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week"
ItemsSource="{Binding Appointments}">
<syncfusion:SfScheduler.DaysViewSettings>
<syncfusion:DaysViewSettings>
<syncfusion:DaysViewSettings.AppointmentTemplate>
<DataTemplate>
<StackPanel Background="{Binding Data.BackgroundColor}"
VerticalAlignment="Stretch"
HorizontalAlignment="Stretch"
Orientation="Horizontal">
<TextBlock Margin="5"
VerticalAlignment="Center"
Text="Meeting"
TextTrimming="CharacterEllipsis"
Foreground="{Binding Data.ForegroundColor}" TextWrapping="Wrap"
FontStyle="Italic"
TextAlignment="Left"
FontWeight="Bold"/>
</StackPanel>
</DataTemplate>
</syncfusion:DaysViewSettings.AppointmentTemplate>
</syncfusion:DaysViewSettings>
</syncfusion:SfScheduler.DaysViewSettings>
</syncfusion:SfScheduler>
```

Note

- By default, the `ScheduleAppointment` is set as the `DataContext` for the `AppointmentTemplate` and `AppointmentTemplateSelector` for both `ScheduleAppointment` and custom data object in the `ItemsSource` of `SfScheduler`.

- The custom data object can be bound in the `AppointmentTemplate` and `AppointmentTemplateSelector` by using the property of [ScheduleAppointment.Data](#).



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

Appointment selection border brush

You can customize the appointment selection border brush by using the [SelectionBorderBrush](#) property in the [AppointmentControl](#). If the `AppointmentControl` has default style, the appointment selection border color will be updated based on the selected appointment background color.

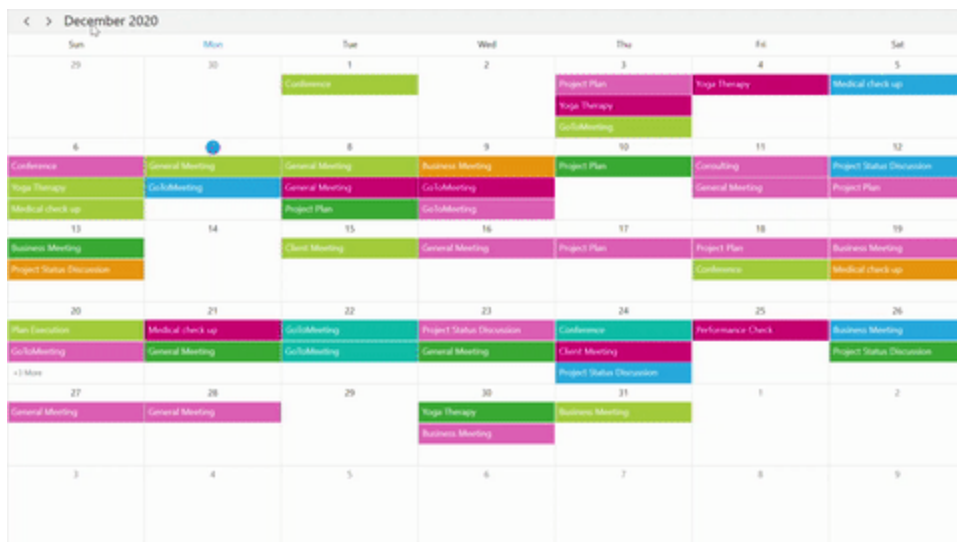
XML

```
<Window.Resources>
<Style TargetType="syncfusion:AppointmentControl">
<Setter Property="BorderBrush" Value="Blue" />
<Setter Property="SelectionBorderBrush" Value="Red" />
<Setter Property="BorderThickness" Value="2" />
</Style>
</Window.Resources>
```

Note: [View sample in GitHub](#)

Load On Demand in WPF Scheduler (SfScheduler)

The [WPF Scheduler](#) supports to load appointment on demand with loading indicator and it improves the loading performance when there are appointments range for multiple years.



QueryAppointments event

The [QueryAppointments](#) event is used to load appointments in on-demand for the visible date range. Start and stop the loading indicator animation before and after the appointments loaded using the [ShowBusyIndicator](#).

The [QueryAppointmentsEventArgs](#) has the following members which provides information for the [QueryAppointments](#) event.

[VisibleDateRange](#) - Gets the current visible date range of scheduler that is used to load the appointments.

XML

```
<syncfusion:SfScheduler x:Name="Scheduler"
    ViewType="Month" QueryAppointments="Scheduler_QueryAppointments">
</syncfusion:SfScheduler>
```

C#

```
Scheduler.QueryAppointments += Scheduler_QueryAppointments;
private void Scheduler_QueryAppointments(object sender,
    QueryAppointmentsEventArgs e)
{
    Scheduler.ShowBusyIndicator = true;
    Scheduler.ItemsSource =
    this.GenerateSchedulerAppointments(e.VisibleDateRange);
    Scheduler.ShowBusyIndicator = false;
}
private IEnumerable GenerateSchedulerAppointments(DateRange dateRange)
{
    Random ran = new Random();
    int daysCount = (dateRange.ActualEndDate - dateRange.ActualStartDate).Days;
    var appointments = new ObservableCollection<SchedulerModel>();
    for (int i = 0; i < 50; i++)
    {
        var startTime = dateRange.ActualStartDate.AddDays(ran.Next(0, daysCount +
            1)).AddHours(ran.Next(0, 24));
        appointments.Add(new SchedulerModel
```

```

{
    From = startTime,
    To = startTime.AddHours(1),
    EventName = subjectCollection[ran.Next(0, subjectCollection.Count)],
    Color = brush[ran.Next(0, brush.Count)],
});
}
return appointments;
}

```

The [QueryAppointments](#) will be raised once any one of the following action is taken.

- Once the [ViewChanged](#) event is raised, the [QueryAppointments](#) will be raised.
- If the appointment has been added, removed, or changed (Resize, drag, and drop) in the current time visible date range, then the [QueryAppointments](#) event will not be triggered. Since the appointments for that visible date range have been loaded already.
- The [QueryAppointments](#) event is triggered when the Schedule [ResourceCollection](#) is updated to load appointments based on the changed resource collection.
- The [QueryAppointments](#) event will be triggered when the [ResourceGroupType](#) is changed.

Load On Demand command

The Scheduler notifies by [LoadOnDemandCommand](#) when the user changes the visible date range. Get a visible date range from the [QueryAppointmentsEventArgs](#). The default value for this [ICommand](#) is null. The [QueryAppointmentsEventArgs](#) passed as a command parameter.

Define a ViewModel class that implements command and handle it by the CanExecute and Execute methods to check and execute on-demand loading. In the execute method, perform the following operations,

- Start and stop the loading indicator animation before and after the appointments loaded using the [ShowBusyIndicator](#).
- Once the appointment collection is got, load into the scheduler items source.

XML

```

<syncfusion:SfScheduler x:Name="Scheduler"
    ViewType="Month"
    ShowBusyIndicator="{Binding ShowBusyIndicator}"
    LoadOnDemandCommand="{Binding LoadOnDemandCommand}"
    ItemsSource="{Binding Events}">
</syncfusion:SfScheduler>

```

C#

```

public class LoadOnDemandViewModel : NotificationObject
{
    public ICommand LoadOnDemandCommand { get; set; }
    private IEnumerable events;
    public IEnumerable Events
    {
        get { return events; }
    }
}

```



```

set
{
events = value;
this.RaisePropertyChanged("Events");
}
}
private bool showBusyIndicator;
public bool ShowBusyIndicator
{
get { return showBusyIndicator; }
set
{
showBusyIndicator = value;
this.RaisePropertyChanged("ShowBusyIndicator");
}
}
public SchedulerViewModel()
{
this.LoadOnDemandCommand = new DelegateCommand(ExecuteOnDemandLoading,
CanExecuteOnDemandLoading);
}
public event PropertyChangedEventHandler PropertyChanged;
public async void ExecuteOnDemandLoading(object parameter)
{
this.ShowBusyIndicator = true;
await Task.Delay(1000);
await
Application.Current.MainWindow.Dispatcher.BeginInvoke(DispatcherPriority.App
licationIdle, new Action(() =>
{
this.Events = this.GenerateSchedulerAppointments((parameter as
QueryAppointmentsEventArgs).VisibleDateRange);
}));
this.ShowBusyIndicator = false;
}
private bool CanExecuteOnDemandLoading(object sender)
{
return true;
}
private IEnumerable GenerateSchedulerAppointments(DateRange dateRange)
{
Random ran = new Random();
int daysCount = (dateRange.ActualEndDate - dateRange.ActualStartDate).Days;
var appointments = new ObservableCollection<SchedulerModel>();
for (int i = 0; i < 50; i++)
{
var startTime = dateRange.ActualStartDate.AddDays(ran.Next(0, daysCount +
1)).AddHours(ran.Next(0, 24)); appointments.Add(new SchedulerModel
{
From = startTime,
To = startTime.AddHours(1),
EventName = subjectCollection[ran.Next(0, subjectCollection.Count)],
Color = brush[ran.Next(0, brush.Count)],
});
}
return appointments;
}

```

```
}

```

The [LoadOnDemandCommand](#) will be invoked once any one of the following actions is taken.

- Once the [ViewChanged](#) event is raised, the [LoadOnDemandCommand](#) will also be raised.
- If the appointment has been added, removed, or changed (Resize, drag, and drop) in the current time visible date range, then the [LoadOnDemandCommand](#) will not be triggered. Since the appointments for that visible date range have been loaded already.
- The [LoadOnDemandCommand](#) is triggered when the Scheduler [ResourceCollection](#) is updated to load the appointments based on the changed resource collection.
- The [LoadOnDemandCommand](#) will be triggered when the [ResourceGroupType](#) is changed.

Load On Demand for recurring appointment

The scheduler will add the occurrences of recurrence series based on the visible date range, use the [RecurrenceHelper.GetRecurrenceDateTimeCollection](#) to compare and load the recurrence appointment on demand in the ItemsSource.

- The recurrence appointment should be added to the Scheduler [ItemsSource](#) until the date of recurrence ends.
- If [RecurrenceRule](#) added with count or end date, use the [RecurrenceHelper.GetRecurrenceDateTimeCollection](#) method to get the recurrence date collection and compare recursive dates in the current visible date range. Then add the recurrence appointment in the scheduler [ItemsSource](#) If the recursive dates are in the current visible date range.
- If the [RecurrenceRule](#) is added without an end date, then the recurrence appointment should be added in the scheduler [ItemsSource](#) when all the visible dates changed from the recurrence start date.

Note: [View sample in GitHub](#)

Appointment Editing in WPF Scheduler (SfScheduler)

This section explains how to handle appointment editing in [WPF scheduler](#) and also explains about the appointment resizing.

Adding appointments

Scheduler supports to add a new appointment by using the [Appointment Editor](#) UI window. Open this window by double-clicking on a time cell or month cell or view header.

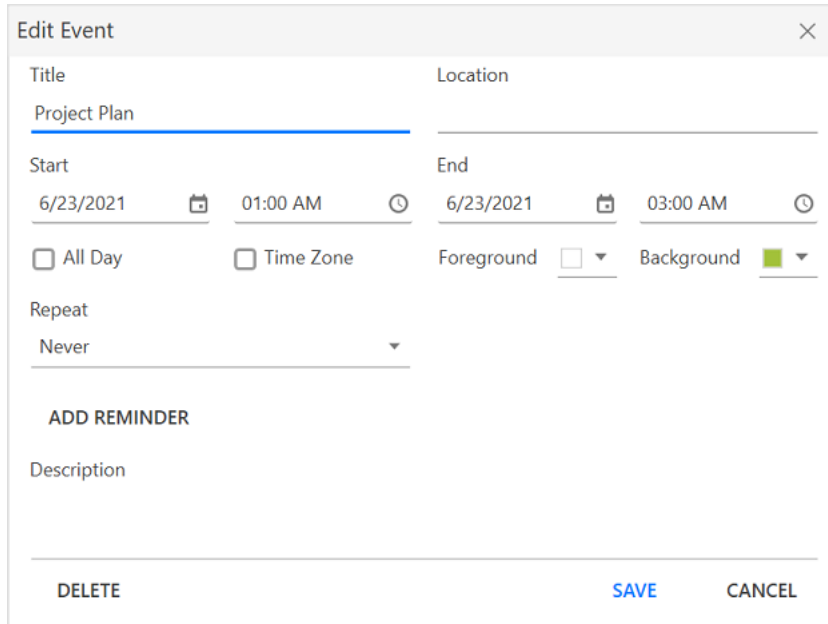
Note:

- If [AllowViewNavigation](#) is true, the current view should be navigated to the respective day or timeline day views by single-clicking on the date in the view header. Other than the date by double-clicking on the view header cell, the appointment editor window will be opened, and by default, the [AllDay](#) checkbox will be checked in the appointment editor window.
- All-day appointments can be created by double-clicking on the view header and not applicable for the month view header.

Editing appointment

Scheduler supports to edit the appointment by using **Appointment Editor** UI window. Open this window by double clicking on the appointment.

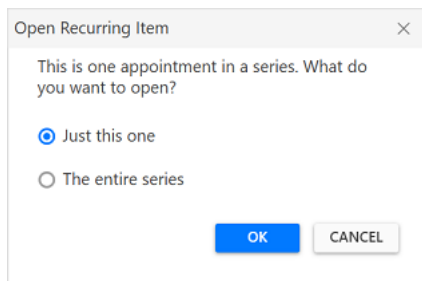
Appointment editor window



Edit the appointments in appointment editor window. This changes will be saved back in appointment and mapped data object when using data binding.

Edit recurring appointment

Scheduler supports to edit the recurrence appointment. The following window will appear when the recurrence appointment is edited to select, whether to edit only the particular occurrence or appointment series.



Handle the opening of recurrence popup window using [RecurringAppointmentEditMode](#) property in [AppointmentEditorOpeningEventArgs](#) by handling [AppointmentEditorOpening](#) event.

AppointmentEditorOpening event

When the appointment editor UI window is opened to add or update appointment, then Scheduler notifies by [AppointmentEditorOpening](#) event.

[AppointmentEditorOpeningEventArgs](#) has following members which provides the information for [AppointmentEditorOpening](#) event.

[Appointment](#) - Gets the selected appointment details which is being updated. It will be null when adding new appointment through appointment editor.

[DateTime](#) - Get the DateTime of time slot or month cell where user double clicked.

[Cancel](#) - To avoid the default appointment editor showing by enabling this property.

[RecurrenceEditMode](#) - Get or Sets the edit mode to perform the edit option to edit the occurrence or series for recurrence appointment. The default value of [RecurrenceEditMode](#) is [User](#).

- [User](#) - Default window dialog will appear when editing a recurrence appointment to select the edit option from the end-user itself.
- [Occurrence](#) - Edit the particular occurrence alone in recurrence appointment. Default window dialog will not appear.
- [Series](#) - Edit the entire series in recurrence appointment. Default window dialog will not appear.

For example, To use custom appointment editor window instead of default appointment editor window, handle [AppointmentEditorOpening](#) event.

C#

```
this.Schedule.AppointmentEditorOpening += Schedule_AppointmentEditorOpening;
private void Schedule_AppointmentEditorOpening(object sender,
AppointmentEditorOpeningEventArgs e)
{
    //To handle the default appointment editor window by setting the e.Cancel
    value as true.
    e.Cancel = true;
    if (e.Appointment != null)
    {
        //Display the custom appointment editor window to edit the appointment
    }
    else
    {
        //Display the custom appointment editor window to add new appointment
    }
}
```

- [Resource](#) - gets the resource of an appointment under which the appointment is located.

Visible/Collapse the built-in editors in appointment editor window

Programmatically visible or collapse the editors by setting the [AppointmentEditorOptions](#) property in [SchedulerAppointmentEditorWindow](#). By default, the value of [AppointmentEditorOptions](#) is set to [AppointmentEditorOptions.All](#) in the [SchedulerAppointmentEditorWindow](#) that displays all the appointment editors. The following code shows how to collapse the [Reminder](#) and [Resource](#) editors by handling the [AppointmentEditorOpening](#) event.

C#

```
this.Schedule.AppointmentEditorOpening += Schedule_AppointmentEditorOpening;
private void Schedule_AppointmentEditorOpening(object sender,
AppointmentEditorOpeningEventArgs e)
{

```

```
e.AppointmentEditorOptions = AppointmentEditorOptions.All |
(~AppointmentEditorOptions.Background & ~AppointmentEditorOptions.Foreground
& ~AppointmentEditorOptions.Reminder & ~AppointmentEditorOptions.Resource);
}
```

Note:

- The basic editors such that **Subject**, **Location**, **Start Hour** and **End Hour** of the scheduler appointment editor will not be collapsed.

AppointmentEditorClosing event

When the appointment editor window is closed after adding or editing the schedule appointment, Scheduler notifies by [AppointmentEditorClosing](#) event.

[AppointmentEditorClosingEventArgs](#) has the following members which provides the information for [AppointmentEditorClosing](#) event.

Handled - Gets or sets a value that indicates whether the scheduler can update the underlying appointments collection or appointment based on the action performed in the appointment editor. If the value is true, scheduler does not perform the action so the code has to be written in the handler and perform the action. The default value of Handled is **false**.

Appointment - Gets the details of updated or newly added appointment.

Cancel - To avoid the default appointment editor closing by enabling this property.

Action - Gets the action of appointment which is Add, Edit, Delete or Cancel.

- Add - Specifies that the appointment is newly added through appointment editor.
- Edit - Specifies that the appointment is edited through appointment editor.
- Delete - Specifies that the appointment is deleted through appointment editor.
- Cancel - Specifies that the appointment editing is canceled through appointment editor.

For example, to handle the appointment adding for today's date, user can handle the `AppointmentEditorClosing` event.

C#

```
this.Schedule.AppointmentEditorClosing += Schedule_AppointmentEditorClosing;
private void Schedule_AppointmentEditorClosing(object sender,
AppointmentEditorClosingEventArgs e)
{
    var appointment = e.Appointment as ScheduleAppointment;
    if (appointment != null)
    {
        if (appointment.StartTime.Day == DateTime.Now.Day)
        e.Handled = true;
    }
}
```

- `Resource` - gets the resource collection of edited appointment.

Disable appointment editing

To disable appointment editing functionality, Set `AppointmentEditFlag` property to `None`. In this case, add, edit, resize and drag & drop the appointments cannot be able performed.

XML

```
<syncfusion:SfScheduler
x:Name="Schedule"
AppointmentEditFlag="None">
</syncfusion:SfScheduler>
```

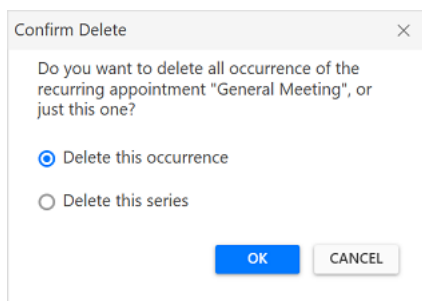
Delete appointments

Scheduler supports two ways to remove the selected appointment.

1. Pressing delete key.
2. Using appointment editor window.

Delete recurring appointment

Scheduler supports to delete the recurrence appointment. The following window will appear when the user deletes the recurrence appointment. Select the delete option to make the changes for occurrence or appointment series.



AppointmentDeleting event

Scheduler notifies by [AppointmentDeleting](#) event, when user deletes the appointment.

[AppointmentDeletingEventArgs](#) has following members which provides information for [AppointmentDeleting](#) event.

[Appointment](#) - Gets the selected appointment to delete.

[Cancel](#) - To avoid appointment deleting by enabling this property.

[RecurrenceEditMode](#) - Gets or sets whether to delete series or occurrence, when delete a recurrence appointment. The default value of RecurrenceEditMode is [User](#).

- User - Default window dialog will appear when deleting a recurrence appointment to select the edit option from the end-user itself.
- Occurrence - Delete the particular occurrence alone in recurrence appointment. Default window dialog will not appear.
- Series - Delete the entire series in recurrence appointment. Default window dialog will not appear.

C#

```
Schedule.AppointmentDeleting += Schedule_AppointmentDeleting;  
private void Schedule_AppointmentDeleting(object sender,  
AppointmentDeletingEventArgs e)  
{  
    //To notify when restrict appointment delete  
    e.Cancel = true;  
}
```

Appointment Resizing

Scheduler has support to resize the selected appointment. This support is available for all views except 'Month' view.

Disable appointment resize

Scheduler supports to disable the appointment resizing by setting [AppointmentEditFlag](#) property except [Resize](#). In this case, appointment resizing cannot be performed.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"  
AppointmentEditFlag="Add, DragDrop, Edit">  
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.AppointmentEditFlag = AppointmentEditFlag.Add |  
AppointmentEditFlag.DragDrop | AppointmentEditFlag.Edit;
```

AppointmentResizing event

Scheduler notifies by [AppointmentResizing](#) event when user resize an appointment.

[AppointmentResizingEventArgs](#) has following members which provides information for [AppointmentResizing](#) event.

[Appointment](#) - Gets the appointment being resized.

[Action](#) - Gets the current action being performed while resizing an appointment.

- **Starting** - Denotes the event occurred when the user mouse over the appointment to resize an appointment (before showing resize cursor).
- **Progressing** - Denotes the event occurred when the user resizing an appointment.
- **Committing** - Denotes the event occurred when the user ends the resizing by releasing pointer to commit the changed to underlying appointment.
- **Canceling** - Denotes the event occurred before canceling the resize operation, when the user press `Esc` key when resizing operation in progress.

[StartTime](#) - Gets the updated start time of the appointment in resizing operation.

[EndTime](#) - Gets the updated end time of the appointment in resizing operation.

[CanContinueResize](#) - Gets or sets a value indicating whether resizing the operation should be continued or canceled. Set this property when Action is Starting, Progressing, Canceling. This property won't have any effect for when Action is Committing.

[CanCommit](#) - Gets or sets a value indicating whether to update underlying appointment when resizing operation is completed. Set this property when Action is Canceling and Committing. This property won't have any effect for when Action is Starting and Progressing.

C#

```
this.Schedule.AppointmentResizing += Schedule_AppointmentResizing;
private void Schedule_AppointmentResizing(object sender,
AppointmentResizingEventArgs e)
{
    //To notify when resizing the appointment.
}
```

- **Resource** - gets the resource of an appointment under which the appointment is located.

Note: You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Reminder in WPF Scheduler (SfScheduler)

[WPF Scheduler](#) alerts you for a particular appointment with a reminder window when enabling the [EnableReminder](#) property. Reminder window supports to **Dismiss** or **DismissAll** or set the **SnoozeTime** for the reminder appointments.

Enable reminder

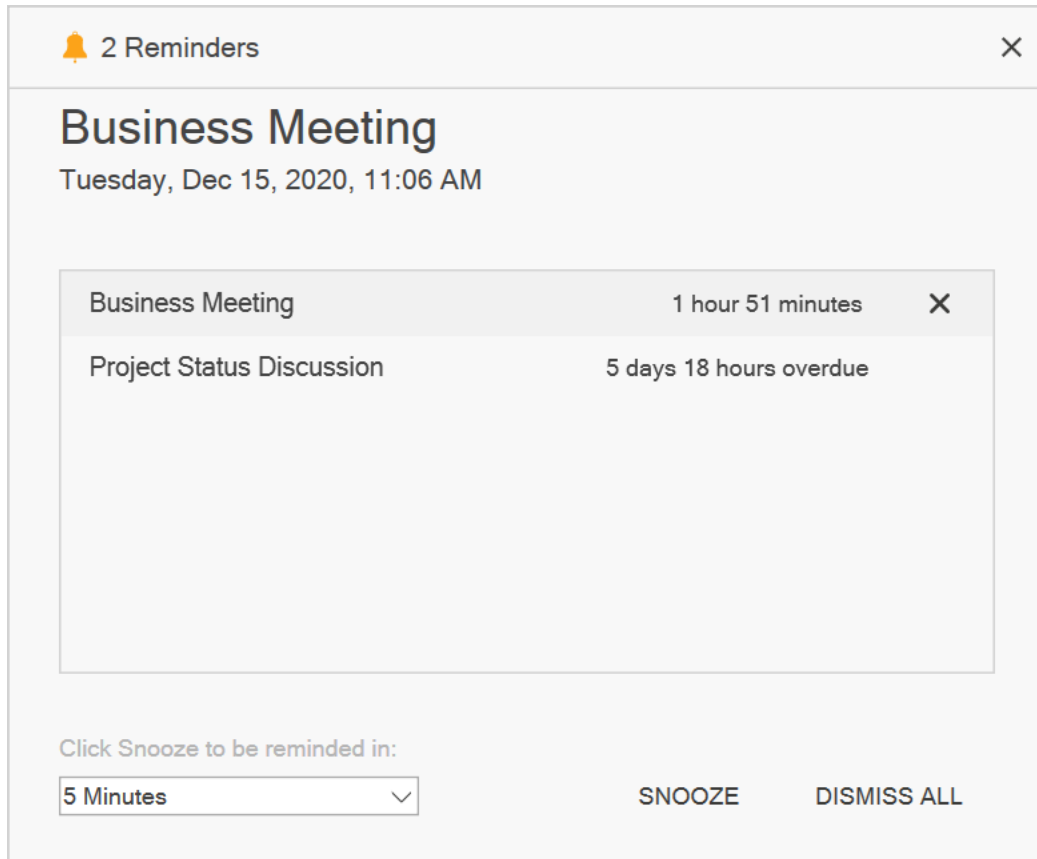
Reminder can be set by setting the [EnableReminder](#) property to **true**. The reminder time can be set using the [Reminders](#) property of [ScheduleAppointment](#).

XML

```
<syncfusion:SfScheduler x:Name="Scheduler">
```



```
ViewType="Week"
EnableReminder="True" >
</syncfusion:SfScheduler>
```



Adding reminders

Configure the appointment reminders with [SchedulerReminder](#). The `SchedulerReminder` has the following properties for reminder alert,

Properties	Description
ReminderTimeInterval	Gets or sets the time interval that decides to open the reminder alert window before the appointment's start time.
ReminderAlertTime	Gets the reminder time that decides when to show a reminder alert of the appointment.
Appointment	Gets the appointment details for which the reminder is created.
Data	Gets the reminder data object associated with the <code>SchedulerReminder</code> .
IsDismissed	Gets or sets whether the reminder is dismissed.

XML

```
<Grid.DataContext>
<local:ReminderViewModel/>
</Grid.DataContext>
<syncfusion:SfScheduler x:Name="Schedule"
ItemsSource="{Binding Events}"
EnableReminder="True">
</syncfusion:SfScheduler>
```

C#

```
public class ReminderViewModel
{
    ...
    public ScheduleAppointmentCollection Events { get; set; } = new
ScheduleAppointmentCollection();
    this.Events.Add(new ScheduleAppointment()
    {
        StartTime = DateTime.Now,
        EndTime = DateTime.Now.AddHours(1),
        AppointmentBackground = new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF339933")),
        Subject = "Conference",
        Reminders = new ObservableCollection<SchedulerReminder>
        {
            new SchedulerReminder { ReminderTimeInterval = new TimeSpan(0)},
        }
    });
}
```

Note: [View sample in GitHub](#)

Creating business object for reminder

Reminders supports to map the custom object with the [ScheduleAppointment.Reminders](#).

C#

```
/// <summary>
/// Represents custom data properties.
/// </summary>
public class Event
{
    public Event()
    {
    }

    public DateTime From { get; set; }
    public DateTime To { get; set; }
    public bool IsAllDay { get; set; }
    public string EventName { get; set; }
    public string Notes { get; set; }
    public string StartTimeZone { get; set; }
    public string EndTimeZone { get; set; }
    public Brush Color { get; set; }
    public object RecurrenceId { get; set; }
    public object Id { get; set; }
```

```

public string RecurrenceRule { get; set; }
public ObservableCollection<DateTime> RecurrenceExceptions { get; set; }
public ObservableCollection<Reminder> Reminders { get; set; }
}

```

The [ReminderMapping](#) provides the mapping information about the [SchedulerReminder](#) properties to the [Data](#) object. ReminderMapping has the following properties for reminder alert,

- [ReminderTimeInterval](#) - Maps the property name of custom class, which is equivalent for the SchedulerReminder.ReminderTimeInterval.
- [IsDismissed](#) - Maps the property name of custom class, which is equivalent for the SchedulerReminder.IsDismissed.

C#

```

/// <summary>
/// Represents custom data properties.
/// </summary>
public class Reminder
{
    /// <summary>
    /// Gets or sets the value indicating whether the reminder is dismissed or not.
    /// </summary>
    public bool Dismissed { get; set; }
    /// <summary>
    /// Gets or sets the value to display reminder alert before appointment start time.
    /// </summary>
    public TimeSpan TimeInterval { get; set; }
}

```

Map those properties of the [Meeting](#) class with the [SfScheduler](#) control by using the [AppointmentMapping](#) and map [CustomReminder](#) properties with [SchedulerReminder](#) by using the [ReminderMapping](#).

XML

```

<syncfusion:SfScheduler x:Name="Schedule"
ItemsSource="{Binding Events}"
EnableReminder="True">
<syncfusion:SfScheduler.AppointmentMapping>
<syncfusion:AppointmentMapping
Subject="EventName"
StartTime="From"
EndTime="To"
AppointmentBackground="Color"
IsAllDay="IsAllDay"
StartTimeZone="StartTimeZone"
EndTimeZone="EndTimeZone"
RecurrenceExceptionDates="RecurrenceExceptions"
RecurrenceRule="RecurrenceRule"
RecurrenceId="RecurrenceId"

```

```

Reminders="Reminders">
<syncfusion:AppointmentMapping.ReminderMapping>
<syncfusion:ReminderMapping IsDismissed="Dismissed"
ReminderTimeInterval="TimeInterval"/>
</syncfusion:AppointmentMapping.ReminderMapping>
</syncfusion:AppointmentMapping>
</syncfusion:SfScheduler.AppointmentMapping>
</syncfusion:SfScheduler>

```

C#

```

public class ReminderViewModel
{
    ...
    public ObservableCollection<Event> Events { get; set; } = new
    ObservableCollection<Event>();
    this.Events.Add(new Event()
    {
        From = DateTime.Now,
        To = DateTime.Now.AddHours(1),
        Color = new
        SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF339933")),
        EventName = "Conference",
        Reminders = new ObservableCollection<Reminder>
        {
            new Reminder { TimeInterval = new TimeSpan(0)},
        }
    });
}

```

Note: [View sample in GitHub](#)

ReminderAlertOpening event

Scheduler notifies by the [ReminderAlertOpening](#) event when appearing in the reminder window. The [ReminderAlertOpeningEventArgs](#) has following properties,

- [Reminders](#) - Gets a list of reminders that are used to display the appointment reminders in the reminder alert window.
- [Cancel](#) - To avoid the reminder window opening by enabling this property.

C#

```

scheduler.ReminderAlertOpening += Scheduler_ReminderAlertOpening;
private void Scheduler_ReminderAlertOpening(object sender,
ReminderAlertOpeningEventArgs e)
{
    var reminders = e.Reminders;
    var appointment = e.Reminders[0].Appointment;
}

```

ReminderAlertActionChanged events

The Scheduler notifies by the [ReminderAlertActionChangedEvent](#) when appointment reminders' dismiss, dismiss all, or snooze action is changed in the Scheduler reminder alert window.

The [ReminderAlertActionChangedEventArgs](#) has following properties which provides information for the `ReminderAlertActionChanged` event.

[ReminderAction](#) - Gets the appointment reminder actions such as dismiss, dismiss all, and snooze performed in the reminder alert window and specifies the appointment reminder that was dismissed in the reminder alert window, the reminders for all appointments were dismissed in the reminder alert window and the appointment reminder is snoozed in the reminder alert window.

[Reminders](#) - Gets the reminders collection where dismiss, dismiss all, or snooze action is performed in the reminder alert window.

[SnoozeTime](#) - Gets the snooze time value of an appointment in the reminder alert window.

C#

```
this.Schedule.ReminderAlertActionChanged +=
OnScheduleReminderAlertActionChanged;
private void OnScheduleReminderAlertActionChanged(object sender,
Syncfusion.UI.Xaml.Scheduler.ReminderAlertActionChangedEventArgs e)
{
    if (e.ReminderAction == ReminderAction.Dismiss)
    {
        var reminder = e.Reminders[0];
    }
    else if (e.ReminderAction == ReminderAction.DismissAll)
    {
        var reminders = e.Reminders;
    }
    else if (e.ReminderAction == ReminderAction.Snooze)
    {
        var reminder = e.Reminders[0];
        var snoozeTime = e.SnoozeTime;
    }
}
```

Note: You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Appointment drag and drop in WPF Scheduler (SfScheduler)

The [WPF Scheduler](#) supports to reschedule the appointment by performing the drag and drop operation.

Disable drag and drop

The Scheduler supports to disable the appointment drag and drop by setting [AppointmentEditFlag](#) property except `DragDrop`. In this case, appointment drag & drop will not be able to perform.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
AppointmentEditFlag="Add,Edit,Resize">
</syncfusion:SfScheduler>
```

C#

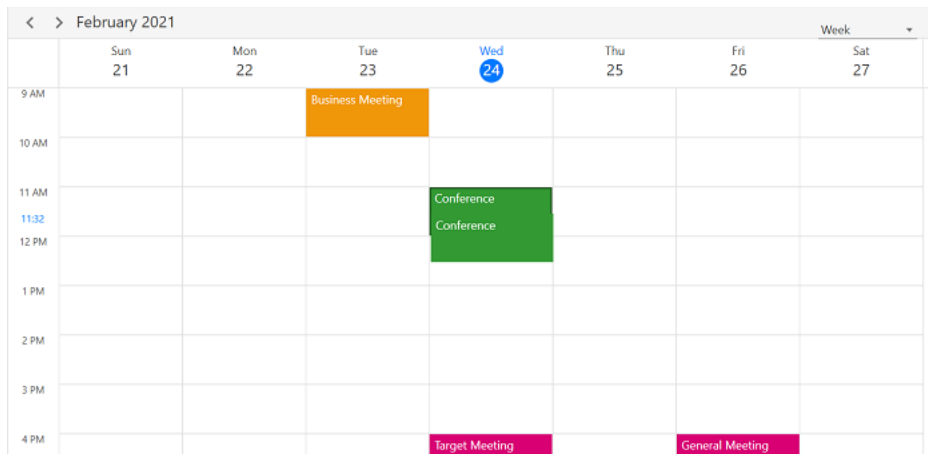
```
this.Schedule.AppointmentEditFlag = AppointmentEditFlag.Add |
AppointmentEditFlag.Edit | AppointmentEditFlag.Resize;
```

Show/Hide the time indicator on appointment dragging

Show or hide the time indicator at a specific time when to drag the appointment by using the [ShowTimeIndicator](#) property of [DragDropSettings](#) is set to true.

C#

```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DragDropSettings.ShowTimeIndicator = true;
```



Note:

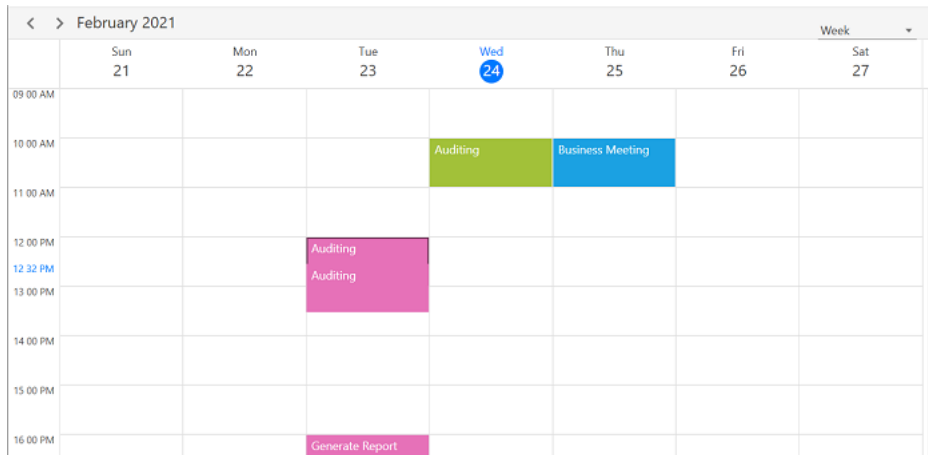
- Not applicable for Month and Timeline Month views.
- If the [TimeRulerSize](#) property value is zero to collapse the time ruler labels, then drag time indicator will not be shown.

Appointment dragging time indicator text formatting

Customize the format for the appointment dragging time indicator format by setting the [TimeIndicatorFormat](#) property of [DragDropSettings](#) in Scheduler.

C#

```
Schedule.ViewType = SchedulerViewType.Week;
Schedule.DragDropSettings.TimeIndicatorFormat = "HH mm tt";
```

**Note:**

- When dragging the appointment, the time indicator overlaps with the time ruler label position in the day, week and workweek views.

AppointmentDragOver event

Scheduler notifies by [AppointmentDragOver](#) while dragging the appointment.

[AppointmentDragOverEventArgs](#) has following members which provides information for AppointmentDragOver event.

[Appointment](#) - Gets the Appointment that is dragging.

[DraggingPoint](#) - Gets the dragging point of schedule appointment UI.

[DraggingTime](#) - Gets the dragging time of the dragging appointment object.

[SourceResource](#) - Gets the [SchedulerResource](#) where the appointment was located before starting the dragging.

[TargetResource](#) - Gets the [SchedulerResource](#) where the appointment is currently being dragged over.

C#

```
this.Schedule.AppointmentDragOver += Schedule_AppointmentDragOver;
private void Schedule_AppointmentDragOver(object sender,
AppointmentDragEventArgs e)
{
    //To notify when dragging the appointment.
}
```

AppointmentDragStarting event

Scheduler notifies by [AppointmentDragStarting](#) when start to drag the appointment.

[AppointmentDragStartingEventArgs](#) has following members which provides information for AppointmentDragStarting event.

[Appointment](#) - Get the selected appointment.

[Cancel](#) - To avoid appointment dragging by enabling this property.

[Resource](#) - Gets the resource of an appointment under which the appointment is located.

C#

```

this.Schedule.AppointmentDragStarting += Schedule_AppointmentDragStarting;
private void Schedule_AppointmentDragStarting(object sender,
AppointmentDragStartingEventArgs e)
{
    //To notify when start to drag the appointment.
}

```

AppointmentDropping event

Scheduler is notified by [AppointmentDropping](#) when the appointment is dropped.

[AppointmentDroppingEventArgs](#) has following members which provides information for [AppointmentDropping](#) event.

[Appointment](#) - Gets the selected appointment that is dragged and dropped.

[DropTime](#) - Gets the dropped time of the dragged appointment.

[Cancel](#) - To avoid appointment dropping by enabling this property.

[SourceResource](#) - Gets the [SchedulerResource](#) where the appointment was located before starting the dragging.

[TargetResource](#) - Gets the [SchedulerResource](#) where the appointment is currently being dragged over.

C#

```

this.Schedule.AppointmentDropping += Schedule_AppointmentDropping;
private void Schedule_AppointmentDropping(object sender,
AppointmentDroppingEventArgs e)
{
    //To notify when the appointment is dropping.
}

```

Note: You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Time Zone in WPF Scheduler (SfScheduler)

[WPF Scheduler](#) allows to create appointments in various time zones and display them in the respective time zone or any other time zone. Use the time zone in the following four different ways:

- Create appointments in different time zones.
- Display appointments based on the client's time zone.
- Display appointments based on scheduler time zone.
- Display appointments at the same time everywhere regardless of client's time zone.

The following Time Zone's respective countries are added to cover all the time zone regions. Use any of the time zones from the following list for scheduler time zone.

Time Zone	Region	UTC Offset
Samoa Standard Time	Pacific/Apia	UTC - 13:00

Dateline Standard Time	Etc/GMT+12	UTC - 12:00
UTC-11	Pacific/Midway	UTC - 11:00
Hawaiian Standard Time	Pacific/Honolulu	UTC - 10:00
Alaskan Standard Time	America/Anchorage	UTC - 09:00
Pacific Standard Time	America/Los_Angeles	UTC - 08:00
Pacific Standard Time (Mexico)	America/Santa_Isabel	UTC - 08:00
Mountain Standard Time	America/Denver	UTC - 07:00
Mountain Standard Time (Mexico)	America/Chihuahua	UTC - 07:00
US Mountain Standard Time	America/Phoenix	UTC - 07:00
Canada Central Standard Time	America/Regina	UTC - 06:00
Central America Standard Time	America/Guatemala	UTC - 06:00
Central Standard Time	America/Chicago	UTC - 06:00
Eastern Standard Time	America/New_York	UTC - 05:00
SA Pacific Standard Time	America/Bogota	UTC - 05:00
US Eastern Standard Time	America/Indianapolis	UTC - 05:00
Venezuela Standard Time	America/Caracas	UTC - 04:30
Atlantic Standard Time	America/Halifax	UTC - 04:00
Central Brazilian Standard Time	America/Cuiaba	UTC - 04:00
Pacific SA Standard Time	America/Santiago	UTC - 04:00
Paraguay Standard Time	America/Asuncion	UTC - 04:00
SA Western Standard Time	America/La_Paz	UTC - 04:00
Newfoundland Standard Time	America/St_Johns	UTC - 03:30
Bahia Standard Time	America/Bahia	UTC - 03:00
Argentina Standard Time	America/Buenos_Aires	UTC - 03:00
E. South America Standard Time	America/Sao_Paulo	UTC - 03:00
Greenland Standard Time	America/Godthab	UTC - 03:00
Montevideo Standard Time	America/Montevideo	UTC - 03:00
SA Eastern Standard Time	America/Cayenne	UTC - 03:00
UTC-02	America/Noronha	UTC - 02:00

Azores Standard Time	Atlantic/Azores	UTC - 01:00
Cape Verde Standard Time	Atlantic/Cape_Verde	UTC - 01:00
GMT Standard Time	Europe/London	UTC
Greenwich Standard Time	Atlantic/Reykjavik	UTC
Morocco Standard Time	Africa/Casablanca	UTC
UTC	America/Danmarkshavn	UTC
Central Europe Standard Time	Europe/Budapest	UTC + 01:00
Central European Standard Time	Europe/Warsaw	UTC + 01:00
Namibia Standard Time	Africa/Windhoek	UTC + 01:00
Romance Standard Time	Europe/Paris	UTC + 01:00
W. Central Africa Standard Time	Africa/Lagos	UTC + 01:00
W. Europe Standard Time	Europe/Berlin	UTC + 01:00
Egypt Standard Time	Africa/Cairo	UTC + 02:00
FLE Standard Time	Europe/Kiev	UTC + 02:00
GTB Standard Time	Europe/Bucharest	UTC + 02:00
Israel Standard Time	Asia/Jerusalem	UTC + 02:00
Libya Standard Time	Africa/Tripoli	UTC + 02:00
Middle East Standard Time	Asia/Beirut	UTC + 02:00
South Africa Standard Time	Africa/Johannesburg	UTC + 02:00
Syria Standard Time	Asia/Damascus	UTC + 02:00
Turkey Standard Time	Europe/Istanbul	UTC + 02:00
Arab Standard Time	Asia/Riyadh	UTC + 03:00
Arabic Standard Time	Asia/Baghdad	UTC + 03:00
Belarus Standard Time	Europe/Minsk	UTC + 03:00
E. Africa Standard Time	Africa/Nairobi	UTC + 03:00
Jordan Standard Time	Asia/Amman	UTC + 03:00
Kaliningrad Standard Time	Europe/Kaliningrad	UTC + 03:00
Iran Standard Time	Asia/Tehran	UTC + 03:30
Arabian Standard Time	Etc/GMT-4	UTC + 04:00

Azerbaijan Standard Time	Asia/Baku	UTC + 04:00
Caucasus Standard Time	Asia/Yerevan	UTC + 04:00
Georgian Standard Time	Asia/Tbilisi	UTC + 04:00
Mauritius Standard Time	Indian/Mauritius	UTC + 04:00
Russia Time Zone 3	Europe/Samara	UTC + 04:00
Russian Standard Time	Europe/Moscow	UTC + 04:00
Afghanistan Standard Time	Asia/Kabul	UTC + 04:30
Pakistan Standard Time	Asia/Karachi	UTC + 05:00
West Asia Standard Time	Asia/Tashkent	UTC + 05:00
India Standard Time	Asia/Calcutta	UTC + 05:30
Sri Lanka Standard Time	Asia/Colombo	UTC + 05:30
Nepal Standard Time	Asia/Kathmandu	UTC + 05:45
Bangladesh Standard Time	Asia/Dhaka	UTC + 06:00
Central Asia Standard Time	Asia/Almaty	UTC + 06:00
Ekaterinburg Standard Time	Asia/Yekaterinburg	UTC + 06:00
Myanmar Standard Time	Asia/Rangoon	UTC + 06:30
SE Asia Standard Time	Asia/Bangkok	UTC + 07:00
N. Central Asia Standard Time	Asia/Novosibirsk	UTC + 07:00
China Standard Time	Asia/Shanghai	UTC + 08:00
North Asia Standard Time	Asia/Krasnoyarsk	UTC + 08:00
Singapore Standard Time	Asia/Singapore	UTC + 08:00
Taipei Standard Time	Asia/Taipei	UTC + 08:00
Ulaanbaatar Standard Time	Asia/Ulaanbaatar	UTC + 08:00
W. Australia Standard Time	Australia/Perth	UTC + 08:00
Korea Standard Time	Asia/Seoul	UTC + 09:00
North Asia East Standard Time	Asia/Irkutsk	UTC + 09:00
Tokyo Standard Time	Asia/Tokyo	UTC + 09:00
AUS Central Standard Time	Australia/Darwin	UTC + 09:30
Cen. Australia Standard Time	Australia/Adelaide	UTC + 09:30

AUS Eastern Standard Time	Australia/Sydney	UTC + 10:00
E. Australia Standard Time	Australia/Brisbane	UTC + 10:00
Tasmania Standard Time	Australia/Hobart	UTC + 10:00
West Pacific Standard Time	Pacific/Port Moresby	UTC + 10:00
Yakutsk Standard Time	Asia/Yakutsk	UTC + 10:00
Central Pacific Standard Time	Pacific/Guadalcanal	UTC + 11:00
Russia Time Zone 10	Asia/Srednekolymsk	UTC + 11:00
Vladivostok Standard Time	Asia/Vladivostok	UTC + 11:00
Fiji Standard Time	Pacific/Fiji	UTC + 12:00
Magadan Standard Time	Asia/Magadan	UTC + 12:00
New Zealand Standard Time	Pacific/Auckland	UTC + 12:00
Russia Time Zone 11	Asia/Kamchatka	UTC + 12:00
UTC+12	Pacific/Tarawa	UTC + 12:00
Tonga Standard Time	Pacific/Tongatapu	UTC + 13:00
Line Islands Standard Time	Pacific/Kiritimati	UTC + 14:00

Create appointments in different time zones

Create appointments at different time zones using the [StartTimeZone](#) and [EndTimeZone](#) properties of [ScheduleAppointment](#). An appointment's start time and end time are calculated based on the given time zone information for the start time and end time. Set different time zones to the StartTimeZone and EndTimeZone properties.

Use the [StartTime](#) and [EndTime](#) properties of [ScheduleAppointment](#) to get the exact start time and end time of an appointment. By using the [ActualStartTime](#) and [ActualEndTime](#) properties, get the exact appointment rendering time.

C#

```
var appointments = new ScheduleAppointmentCollection();
appointments.Add(new ScheduleAppointment()
{
    Subject = "Meeting",
    StartTime = DateTime.Now,
    EndTime = DateTime.Now.AddHours(1),
    StartTimeZone = "India Standard Time",
    EndTimeZone = "India Standard Time"
});
```

NOTE

- If the recurring appointment is converted to another time zone, then the whole sequence will be recalculated according to the new time zone information.
- If an all-day appointment is created, its start time and end time will be set to 12 A.M. and 12 A.M. by default, so time zone is not applicable for all-day appointments.
- Scheduler supports daylight saving time.
- The time zone support is applicable for custom appointments too, so map the corresponding property.
- Use [TimeZone](#) for custom appointments by mapping the [StartTimeZone](#) and [EndTimeZone](#) custom properties of [AppointmentMapping](#).

Display Appointments based on client's time zone

Display the appointments based on the client's local time zone in scheduler. For example, consider a scenario that you are in North Carolina and you want to set up a meeting at 10 A.M. on North Carolina time. You have colleagues in London and Chennai, and they also need to participate. The time for this meeting will be 3 P.M. (15:00) in London and 5.30 A.M. in Chennai. When each view your Scheduler, you need to see the appointment displayed relative to your local time zones 5.30 A.M., 10 A.M., and 3 P.M., respectively. It can be achieved by setting schedule time zone to default (it will consider your device's local time zone as schedule time zone) and appointment's time zone to **Eastern Standard Time (North Carolina)** [as you are in North Carolina and its time zone is Eastern Standard Time].

Display appointments based on Scheduler time zone

Set the specific time zone to schedule using the [TimeZone](#) property of scheduler. On this scenario, the appointments will be displayed in UTC time when the [StartTimeZone](#) and [EndTimeZone](#) properties of [ScheduleAppointment](#) are set to null. The appointments will be displayed in UTC time based on the given scheduler time zone.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    TimeZone="Central America Standard Time"/>
```

C#

```
this.Schedule.TimeZone = "Central America Standard Time";
```

Display appointments at same time everywhere regardless of client's time zone

Display the appointments at the same time everywhere without considering the time zone while setting the [TimeZone](#) property of the scheduler, the [StartTimeZone](#) and [EndTimeZone](#) properties of [ScheduleAppointment](#) to null. The appointments will be displayed based on the given [StartTime](#) and [EndTime](#) of appointment everywhere without considering the time zone.

Updating StartTime and EndTime after drag and drop appointment based on time zone.

After rescheduling an appointment using drag and drop, appointment's start and end time value will be updated based on scheduler time zone and appointment's time zone.

For an example, consider the local time zone is **India Standard Time**, if you drag an appointment from 9 AM and drop this on 1 PM and the scheduler's [TimeZone](#) is not set and the appointment's [StartTimeZone](#) and [EndTimeZone](#) has set as **AUS Central Standard Time (Darwin)** then

appointment's start time and end time value will be converted from Local time zone to appointment time zone and the appointment's start time will be saved at 9 AM.

If you set scheduler's `TimeZone` as `AUS Central Standard Time (Darwin)` and the appointment's `StartTimeZone` and `EndTimeZone` as `Central Standard Time (Mexico)` then the appointment's start time and end time value has converted from scheduler's time zone to appointment time zone and the appointment's start time will be saved at 3.30 AM of next day.

If you set scheduler's `TimeZone` as `AUS Central Standard Time (Darwin)` and appointment's time zone was not set then the appointment's start time and end time value converted from scheduler time zone to `UTC` time zone and the appointment's start time will be saved at 10.30 PM.

Note: You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Date Navigations in WPF Scheduler (SfScheduler)

Range for visible dates

Visible dates can be restricted between certain range of dates, using [MaximumDate](#) and [MinimumDate](#) properties in [SfScheduler](#). It is applicable in all the schedule views.

Minimum display date

`MinimumDate` will restrict date navigations features of backward, and also it doesn't allow to swipe the control using touch gesture beyond the minimum date range. The dates before the minimum date will be disabled in the schedule.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    MinimumDate="2020-05-05 10:0:0">
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.MinimumDate = new DateTime(2020, 05, 05, 10, 0, 0);
```

Maximum display date

`MaximumDate` will restrict date navigations features of forward, and also it doesn't allow to swipe the control using touch gesture beyond the maximum date range. The dates beyond the maximum date will be disabled in the schedule.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    MaximumDate="2020-10-05 10:0:0">
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.MaximumDate = new DateTime(2020, 10, 05, 10, 0, 0);
```

Programmatic date navigation

Programmatically navigate the dates in scheduler by using the [DisplayDate](#) property of `SfScheduler`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    DisplayDate="2020-07-05 10:0:0">
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.DisplayDate = new DateTime(2020, 07, 05, 10, 0, 0);
```

Note: Date navigation before the minimum date will be reset to the scheduler minimum date and date navigation beyond the maximum date will be reset to the scheduler maximum date.

Programmatic date selection

Programmatically select the dates in scheduler by using the [SelectedDate](#) property of SfScheduler.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    SelectedDate="2020-07-10 10:0:0">
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.SelectedDate = new DateTime(2020, 07, 10, 10, 0, 0);
```

Note: Selection before minimum dates and beyond maximum dates using the [SelectedDate](#) is not possible.

Programmatically change to adjacent dates

By default, the date can be navigated to next and previous views using touch gesture, by swiping the control from right to left and left to right direction. The view can be also changed programmatically using the [Forward](#) and [Backward](#) methods available in SfScheduler.

Forward

Use the [Forward](#) method of SfScheduler for viewing the next immediate visible dates in the scheduler. It will move to the next month if the scheduler view is month, similarly it will move to the next week for week view and the next day for day view.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
    MaximumDate="2020-10-05 10:0:0">
</syncfusion:SfScheduler>
<Button x:Name="Forward"
    Content="fwd"
    Click="Forward_Click">
</Button>
```

C#

```
private void Forward_Click(object sender, RoutedEventArgs e)
{
```

```
Schedule.Forward();  
}
```

Backward

Use the **Backward** method of SfScheduler for viewing the previous immediate visible dates in the scheduler. It will move to the previous month if the scheduler view is month, similarly it will move to the previous week for week view and previous day for day view.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"  
MinimumDate="2020-05-05 10:0:0">  
</syncfusion:SfScheduler>  
<Button x:Name="Backward"  
Content="bwd"  
Click="Backward_Click">  
</Button>
```

C#

```
private void Backward_Click(object sender, RoutedEventArgs e)  
{  
    Schedule.Backward();  
}
```

Allow view navigation

You can quickly navigate to the respective day or timeline day view by single-clicking on the date in month cell or view header of the following scheduler views such as week, work week, month, timeline week, timeline work week, and timeline month views by using the [AllowViewNavigation](#) property of the scheduler.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"  
ViewType="Week"  
AllowViewNavigation="True">  
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Week;  
this.Schedule.AllowViewNavigation = true;
```

Note:

- The **AllowViewNavigation** is not applicable for day and timeline day views.
- If the [ShowAgendaView](#) is true in a month view, the month view should navigate to the day view by single-clicking on the agenda date view header, otherwise, the month view should navigate to the day view by single-clicking on the date in a month cell.

Show date picker

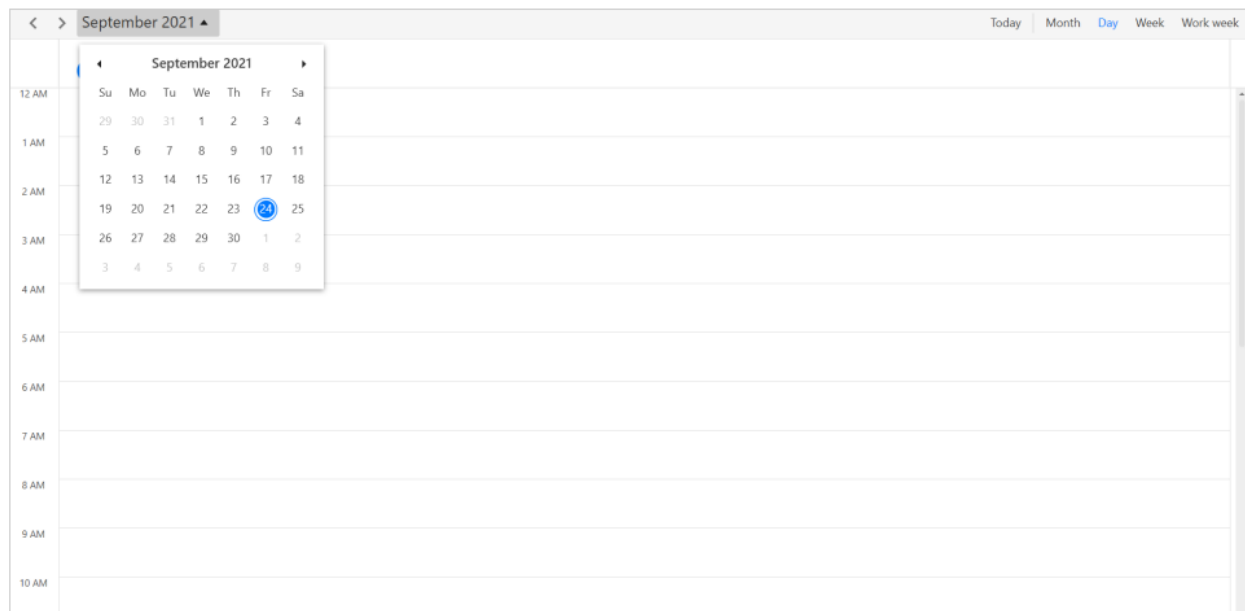
You can enable the date picker for the scheduler by using the [ShowDatePickerButton](#) property in the [SfScheduler](#), which displays the date picker and Today button in the header view. It allows you to quickly navigate to today and different scheduler views.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
ShowDatePickerButton="True">
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Week;
this.Schedule.ShowDatePickerButton = true;
```



Note: The header DatePicker will not be shown if [CalendarIdentifier](#) is specified other than [GregorianCalendar](#).

Allowed views

You can quickly navigate to the different scheduler views by using the [AllowedViewTypes](#) property in the [SfScheduler](#). The views set to this property will display as a view button in the scheduler header view. This UI will be responsive as showing more icons and will be updated based on the window size change.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Week"
AllowedViewTypes="Month,Week,WorkWeek,Day,TimelineDay,TimelineWeek,TimelineW
orkWeek,TimelineMonth" >
```

C#

```
this.Schedule.ViewType = SchedulerViewType.Week;
this.Schedule.AllowedViewTypes = AllowedSchedulerViewTypes.Week |
AllowedSchedulerViewTypes.WorkWeek | AllowedSchedulerViewTypes.Day |
AllowedSchedulerViewTypes.Month | AllowedSchedulerViewTypes.TimelineDay |
AllowedSchedulerViewTypes.TimelineMonth |
AllowedSchedulerViewTypes.TimelineWeek |
AllowedSchedulerViewTypes.TimelineWorkWeek;
```

Note: [View sample in GitHub](#)

Header in WPF Scheduler (SfScheduler)

Change the header height, date format and appearance of [SfScheduler](#).

Header height

Change the scheduler header height by using [HeaderHeight](#) property of SfScheduler. By default, the header height is 50.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
HeaderHeight="100" >
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.HeaderHeight = 100;
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Header date format

Change the Scheduler header date format of scheduler by using the [HeaderDateFormat](#) property of SfScheduler. By default, the header date format is `MMMM yyyy`.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
HeaderDateFormat="MMM/yyyy">
</syncfusion:SfScheduler>
```

C#

```
this.Schedule.HeaderDateFormat = "MMM/yyyy";
```

< > Jul/2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Appearance customization

Customize header appearance using Style

Style the Scheduler header appearance using [SchedulerHeaderControl](#) in scheduler. Change the background color, textStyle, and borderBrush etc. by setting style property for SchedulerHeaderControl.

XML

```
<Style TargetType="syncfusion:SchedulerHeaderControl">
<Setter Property="Background" Value="LightCyan"/>
<Setter Property="Foreground" Value="Red"/>
<Setter Property="FontStyle" Value="Italic"/>
<Setter Property="BorderBrush" Value="LightCoral"/>
<Setter Property="BorderThickness" Value="2"/>
</Style>
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Customize header appearance using DataTemplate

Customize header appearance of scheduler by using [HeaderTemplate](#) property of SfScheduler.

XML

```
<syncfusion:SfScheduler x:Name="Schedule">
  <syncfusion:SfScheduler.HeaderTemplate>
    <DataTemplate >
      <TextBlock FontStyle="Italic"
        Foreground="Blue"
        FontSize="25"
        Text="{Binding}"/>
    </DataTemplate>
  </syncfusion:SfScheduler.HeaderTemplate>
</syncfusion:SfScheduler>
```

< > July 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Note: You can refer to our [WPF Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Events in WPF Scheduler (SfScheduler)

CellTapped

The [CellTapped](#) event occurs when the user clicks or touches the cell in Scheduler.

This event receives two arguments namely `this` that handles `SfScheduler` and [CellTappedEventArgs](#) as objects.

The [CellTappedEventArgs](#) object contains the following properties:

- [Appointment](#) - returns Tapped appointment values.
 1. If [ItemsSource](#) added with custom business object then tapped custom Appointment details can get by using Appointment [Data](#) property in Cell tapped arguments.
 2. The tapped appointment is a Recurrence appointment it will return the parent recurrence appointment values.
 3. The appointment details get for month view if [AppointmentDisplaymode](#) as [Appointment](#), or else it will be null for month view.
- [Appointments](#)- returns Tapped Month cell appointments values if AppointmentDisplayMode as indicator. Tapped Month Cell has a Recurrence appointment it will return the parent recurrence appointment values. It will be null for Day or Week or WorkWeek views.
- [IsMoreAppointments](#)- specifies whether more appointments are tapped or not in month view. It will be applicable only for Month view which has AppointmentDisplaymode as Appointment.
- [CancelNavigation](#)- specifies whether day view navigation should be disabled when clicking more appointments in month view. It will be applicable for month view which has AppointmentDisplaymode as Appointment and click the More appointments in month cell.
- [DateTime](#)- gets the date-time of the tapped cell.
- [TimeInterval](#)- gets the date-time interval of the tapped cell. It is not applicable for month view.

- [Resource](#) - gets the resource associated with the timeslot cell where user tapped.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month"
CellTapped="Scheduler_CellTapped" >
</syncfusion:SfScheduler>
```

C#

```
private void Scheduler_CellTapped(object sender, CellTappedEventArgs e)
{
    var dateTime = e.DateTime.ToString();
}
```

CellDoubleTapped

The [CellDoubleTapped](#) event occurs when the user double clicks the cell in Scheduler. This event receives two arguments namely `this` that handles `SfScheduler` and [CellDoubleTappedEventArgs](#) as objects. The base class of the [CellDoubleTappedEventArgs](#) is [CellTappedEventArgs](#)

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month"
CellDoubleTapped="Scheduler_CellDoubleTapped">
</syncfusion:SfScheduler>
```

C#

```
private void Scheduler_CellDoubleTapped(object sender,
CellDoubleTappedEventArgs e)
{
    var dateTime = e.DateTime.ToString();
}
```

CellLongPressed

The [CellLongPressed](#) event occurs when the user long presses the cell in Scheduler. This event receives two arguments namely `this` that handles `SfScheduler` and [CellLongPressedEventArgs](#) as objects. The base class of the [CellLongPressedEventArgs](#) is [CellTappedEventArgs](#)

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month"
CellLongPressed="Schedule_CellLongPressed">
</syncfusion:SfScheduler>
```

C#

```
private void Schedule_CellLongPressed(object sender,
CellLongPressedEventArgs e)
{
}
```

```
var dateTime = e.DateTime.ToString();  
}
```

SelectionChanged

The [SelectionChanged](#) event occurs after the selection is changed in Scheduler. This event receives two arguments namely `this` that handles `SfScheduler` and [SelectionChangedEventArgs](#) as objects.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"  
ViewType="Month"  
SelectionChanged="Schedule_SelectionChanged">  
</syncfusion:SfScheduler>
```

C#

```
private void Schedule_SelectionChanged(object sender,  
SelectionChangedEventArgs e)  
{  
var newdate = e.NewValue.ToString();  
var olddate = e.OldValue.ToString();  
}
```

The [SelectionChangedEventArgs](#) object contains the following properties:

- [OldValue](#) - gets an old selected date.
- [NewValue](#)- gets a new selected date.

SelectionChanging

The [SelectionChanging](#) event occurs when the selection gets changing in Scheduler. This event receives two arguments namely `this` that handles `SfScheduler` and [SelectionChangingEventArgs](#) as objects.

The [SelectionChanging](#) object contains the following properties:

- [OldValue](#) - gets an old selected date.
- [NewValue](#)- gets a new selected date.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"  
ViewType="Month"  
SelectionChanging="Schedule_SelectionChanging">  
</syncfusion:SfScheduler>
```

C#

```
private void Schedule_SelectionChanging(object sender,  
SelectionChangingEventArgs e)  
{  
var newdate = e.NewValue.ToString();  
var olddate = e.OldValue.ToString();  
}
```

ViewHeaderCellTapped

The [ViewHeaderCellTapped](#) event occurs when the user clicks or touches the view header in Scheduler. This event receives two arguments namely `this` that handles `SfScheduler` and [ViewHeaderCellTappedEventArgs](#) as objects.

The [ViewHeaderCellTappedEventArgs](#) object contains the following properties:

- [DateTime](#) - gets the corresponding date time.
- [Resource](#) - gets the resource when tapped on view header in day, week, work week and timeline views.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month"
ViewHeaderCellTapped="Schedule_ViewHeaderCellTapped">
</syncfusion:SfScheduler>
```

C#

```
private void Schedule_ViewHeaderCellTapped(object sender,
ViewHeaderCellTappedEventArgs e)
{
    var dateTime = e.DateTime.ToString();
}
```

HeaderTapped

The [HeaderTapped](#) event occurs when the user clicks or touches the Scheduler header. This event receives two arguments namely `this` that handles `SfScheduler` and [HeaderTappedEventArgs](#) as objects.

The [HeaderTappedEventArgs](#) object contains the following properties:

- [DateTime](#) - gets the corresponding date time.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month"
HeaderTapped="Schedule_HeaderTapped">
</syncfusion:SfScheduler>
```

C#

```
private void Schedule_HeaderTapped(object sender, HeaderTappedEventArgs e)
{
    var dateTime = e.DateTime.ToString();
}
```


WeekNumberTapped

The [WeekNumberTapped](#) event occurs when the user clicks or touches the week number in month view. This event receives two arguments namely [this](#) that handles [SfScheduler](#) and [WeekNumberTappedEventArgs](#) as objects.

The [WeekNumberTappedEventArgs](#) object contains the following properties:

- [Month](#)- gets the corresponding month.
- [WeekNumber](#)- gets the corresponding week number.
- [Year](#)- gets the corresponding year.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month"
WeekNumberTapped="Schedule_WeekNumberTapped">
</syncfusion:SfScheduler>
```

C#

```
private void Schedule_WeekNumberTapped(object sender,
WeekNumberTappedEventArgs e)
{
var weeknumber = e.WeekNumber.ToString();
}
```

AppointmentTapped

The [AppointmentTapped](#) event occurs when schedule appointments get tapped in all views. This event receives two arguments namely [this](#) that handles [SfScheduler](#) and [AppointmentTappedArgs](#) as objects.

The [AppointmentTappedArgs](#) object contains the following properties:

- [Appointment](#)- gets the custom appointment details
- [SelectedDate](#)- gets the SelectedDate details
- [Resource](#) - gets the resource details under which the appointment is located.

XML

```
<syncfusion:SfScheduler x:Name="Schedule"
ViewType="Month"
AppointmentTapped="Schedule_AppointmentTapped">
</syncfusion:SfScheduler>
```

C#

```
private void Schedule_AppointmentTapped(object sender, AppointmentTappedArgs
e)
{
var appointment = e.Appointment.ToString();
}
```

Note: You can refer to our [WPF Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

ContextMenu and Commands in WPF Scheduler (SfScheduler)

The [WPF Scheduler](#) has support to define a context menu for appointments, time slots, and month cells are right-clicked. It will also have the built-in `RoutedUICommands` support for handling the context menu to add, edit, and delete appointments. There are two types of `ContextMenu`.

- [CellContextMenu](#)
- [AppointmentContextMenu](#)

Note: [View sample in GitHub](#)

Cell context menu

Set the context menu for time slot and month cells by using the [SfScheduler.CellContextMenu](#) property. The `CellContextMenu` will appear only when the time slot or month cells are right-clicked.

Note:

- The menu items which bind the [SchedulerCommands.Edit](#) and [SchedulerCommands.Delete](#) built-in commands will be disabled in the `CellContextMenu`.
- While binding the menu item using the `CommandBinding`, get the command parameter as

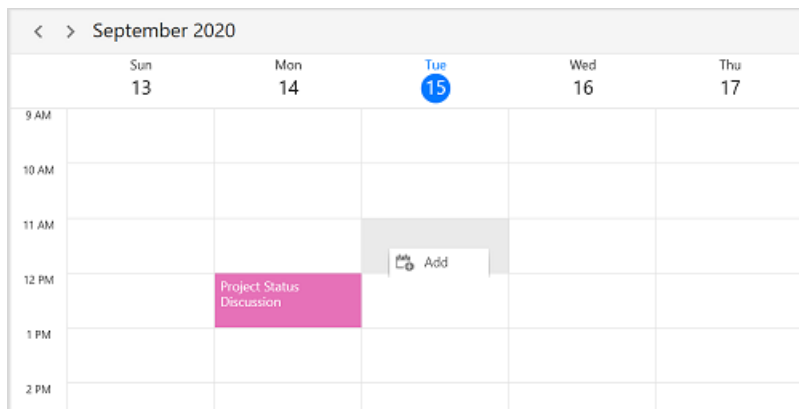
[SchedulerContextMenuInfo](#)

that contains the Appointment or DateTime of the corresponding cell.

- By default, the cell context menu will be opened when holding on any timeslot or month cell. The appointment context menu will be opened by holding, only if the appointment's drag and drop is disabled using the [AppointmentEditFlag](#) property.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
  <syncfusion:SfScheduler.CellContextMenu>
    <ContextMenu>
      <MenuItem Command="{Binding Source={x:Static
Member=syncfusion:SchedulerCommands.Add}}" CommandParameter="{Binding}"
CommandTarget="{Binding ElementName=Schedule}" Header="Add">
    </MenuItem>
  </ContextMenu>
</syncfusion:SfScheduler.CellContextMenu>
</syncfusion:SfScheduler>
```



Appointment context menu

Set the context menu for schedule appointments by using the [SfScheduler.AppointmentContextMenu](#) property. The [AppointmentContextMenu](#) will be displayed only on appointments that are right-clicked.

NOTE

- The menu item which binds the [SchedulerCommands.Add](#) command, will be disabled in the [SfScheduler.AppointmentContextMenu](#).
- While binding the menu item using the CommandBinding, get the command parameter as

[SchedulerContextMenuInfo](#)

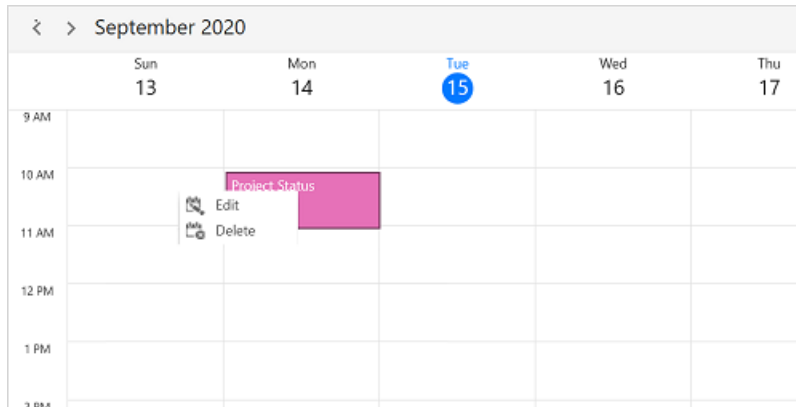
that contains the Appointment or DateTime of the corresponding cell.

- In the month view, the [AppointmentContextMenu](#) opens when the [MonthViewSettings.AppointmentDisplayMode](#) is set to [AppointmentDisplayMode.Appointment](#).
- To enable the touch context menu for appointments in the scheduler, by disabling the appointment drag and drop by setting the [AppointmentEditFlag](#) property except for DragDrop. In this case, the appointment drag & drop cannot be performed. The [AppointmentContextMenu](#) will be displayed only on appointments and the appointment selection, should be performed.

XML

```
<syncfusion:SfScheduler x:Name="Schedule" ViewType="Week">
  <syncfusion:SfScheduler.AppointmentContextMenu>
    <ContextMenu>
      <MenuItem Command="{Binding Source={x:Static
Member=syncfusion:SchedulerCommands.Edit}}"
CommandParameter="{Binding}" CommandTarget="{Binding ElementName=Schedule}"
Header="Edit">
    </MenuItem>
      <MenuItem Command="{Binding Source={x:Static
Member=syncfusion:SchedulerCommands.Delete}}"
CommandParameter="{Binding}"
Header="Delete">
    </MenuItem>
    </ContextMenu>
  </AppointmentContextMenu>
</SfScheduler>
```

```
</syncfusion:SfScheduler.AppointmentContextMenu>
</syncfusion:SfScheduler>
```



SchedulerContextMenuOpening event

The [SchedulerContextMenuInfo](#) event occurs while opening the [AppointmentContextMenu](#) or [CellContextMenu](#) in the SfScheduler.

[SchedulerContextMenuOpeningEventArgs](#) has the following members which provides the information about the [SchedulerContextMenuOpening](#) event.

- **MenuInfo** – Returns the [SchedulerContextMenuInfo](#) which contains the information about date time, appointment of the element opens the context menu. The [AppointmentContextMenu](#) and [CellContextMenu](#) received this information as a [DataContext](#).
- **MenuType** – Gets the element type for which the context menu opens.
- **ContextMenu** – It represents a shortcut context menu that is being opened.

Note: You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Calendar Types in WPF Scheduler (SfScheduler)

This section describes how to change the calendar types of scheduler control using the [CalendarIdentifier](#).

Types of Calendar

The scheduler control supports the different types of calendars such as Gregorian, Korean, Hebrew, and more. You can change the calendar types by using the [CalendarIdentifier](#) property in [Scheduler](#). The default value of the [CalendarIdentifier](#) property is [GregorianCalendar](#).

Supported Calendars	Unsupported Calendars
<i>GregorianCalendar</i> <i>HebrewCalendar</i> <i>HijriCalendar</i> <i>KoreanCalendar</i> <i>TaiwanCalendar</i> <i>ThaiCalendar</i>	<i>JulianCalendar</i> <i>JapaneseCalendar</i> * Lunar type calendars

UmAlQuraCalendar PersianCalendar	
-------------------------------------	--

Note:

- When the [CalendarIdentifier](#) and [FlowDirection](#) properties are set, the [FlowDirection](#) property is given higher precedence. If you want to override this behavior set [FlowDirection](#) after [CalendarIdentifier](#).
- The scheduler uses the [CalendarIdentifier](#) property to determine which calendar to use to localize and format the header date, view header day, and date, time ruler, and DatePicker, and TimePicker in the appointment editor.
- By default, the scheduler uses the [GregorianCalendar](#) for the app's preferred language.
- The Scheduler Time mode (12 hour or 24 hour) does not change depending on the calendar type; however, the time format can be changed depending on the calendar type by using [Day view time text formatting](#) and [Timeline view time text formatting](#).

XML

```
<scheduler:SfScheduler x:Name="Schedule"
CalendarIdentifier="HijriCalendar" />
```

C#

```
this.Schedule.CalendarIdentifier = "HijriCalendar";
```

Today								1443 صفر < >
الأحد	الاثنين	الثلاثاء	الأربعاء	الخميس	الجمعة	السبت		
19	20	21	22	23	24	25		
ص 12								
ص 1								
ص 2								
ص 3								
ص 4								
ص 5								
ص 6								
ص 7								
ص 8								
ص 9								
ص 10								

DateTime values in Calendar types

All the DateTime values can be given such as [DisplayDate](#), [SelectedDate](#), [BlackoutDates](#), Appointment [StartTime](#), and [EndTime](#), SpecialTimeRegion [StartTime](#) and [EndTime](#) values in two ways when calendar identifier is specified other than [GregorianCalendar](#).

- Create an appointment with a start and end time value by declaring the calendar type and relevant calendar type date.

C#

```
// Creating an instance for the schedule appointment collection.
var appointments = new ScheduleAppointmentCollection();
// Adding schedule appointment in the schedule appointment collection.
appointments.Add(new ScheduleAppointment()
{
    Subject = "Meeting",
    // StartTime and EndTime value specified with calendar type and respective
    // calendar date.
    StartTime = new DateTime(1443, 02, 22, 10, 0, 0, new HijriCalendar()),
    EndTime = new DateTime(1443, 02, 22, 11, 0, 0, new HijriCalendar()),
});
// Adding the schedule appointment collection to the ItemsSource.
this.scheduler.ItemsSource = appointments;
```

- Create an appointment with a start and end time by declaring the local system date; in that case, the system date will be converted to the relevant calendar type date.

C#

```
// Creating an instance for the schedule appointment collection.
var appointments = new ScheduleAppointmentCollection();
// Adding schedule appointment in the schedule appointment collection.
appointments.Add(new ScheduleAppointment()
{
    Subject = "Meeting",
    // StartTime and EndTime values specified with local system date will be
    // converted to the Hijiri calendar mentioned.
    StartTime = new DateTime(2021, 09, 29, 10, 0, 0, 0),
    EndTime = new DateTime(2021, 09, 29, 11, 0, 0, 0),
});
// Adding the schedule appointment collection to the ItemsSource.
this.scheduler.ItemsSource = appointments;
```

Note: [View sample in GitHub](#)

Localization in WPF Scheduler (SfScheduler)

Localization is the process of customizing the user interface, based on a culture specific to a particular country or region in order to display the regional data. The culture is represented by a unique string, for example, —en-US || for U.S. English and — fr-FR || for French (common).

Localization is the key feature that provides solutions to global customers with the help of localized resource files provided by the control. The Scheduler supports localization, and creates a resource file for any culture to be applied in the scheduler.

Set Current UI Culture to the Application

Application culture can be changed by setting [CurrentUICulture](#).

C#

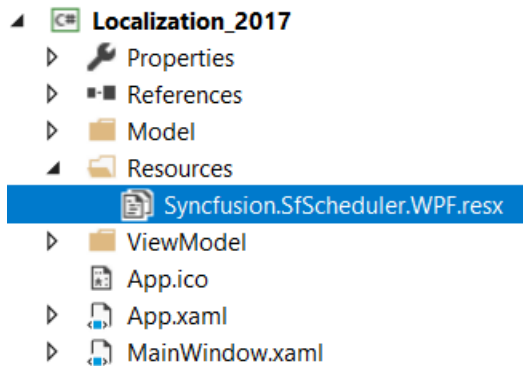
```
public MainWindow()  
{  
    this.InitializeComponent();  
    System.Threading.Thread.CurrentThread.CurrentUICulture = new  
    System.Globalization.CultureInfo("fr-FR");  
}
```

Localization using Resource file

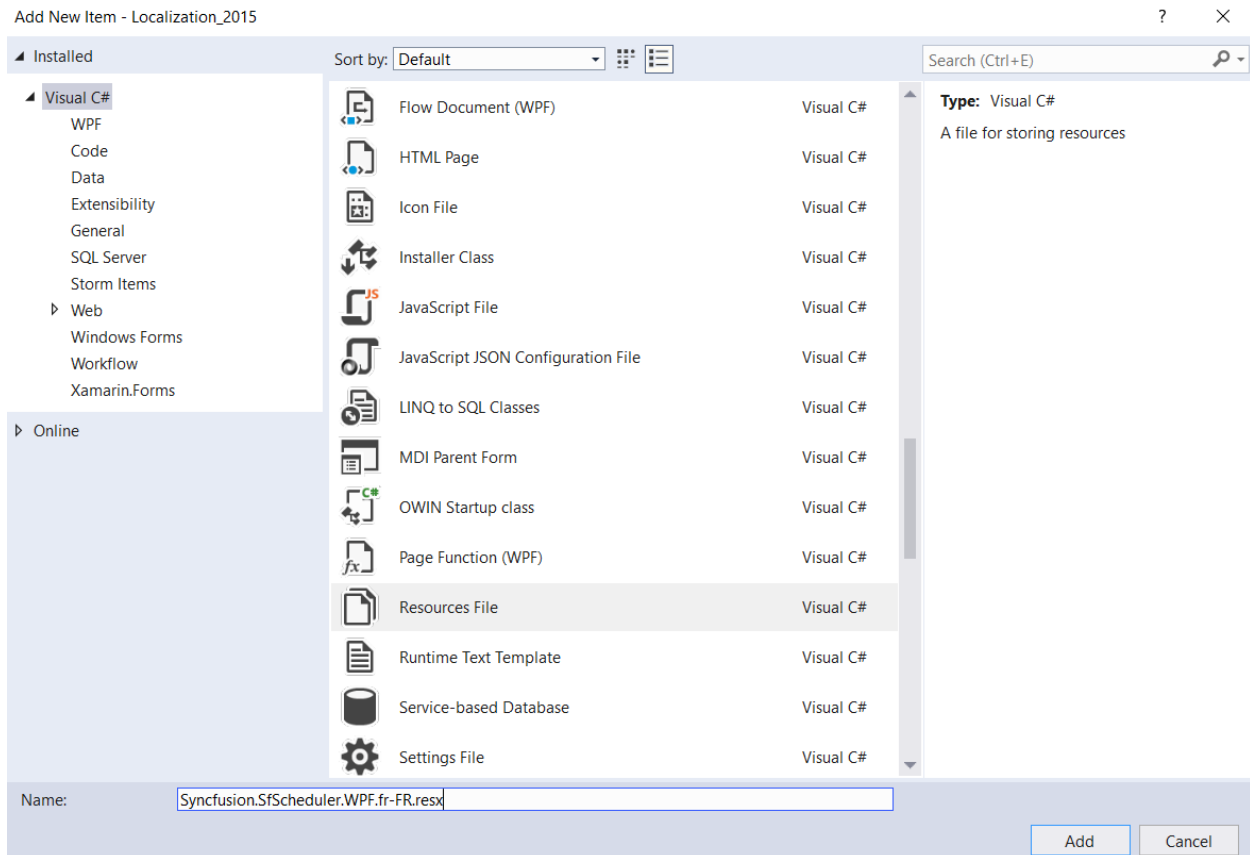
To localize the Scheduler based on `CurrentUICulture` using resource files, follow the below steps.

1. Create new folder, named as **Resources** in the application.
2. Add the default resource file of Scheduler into **Resources** folder.

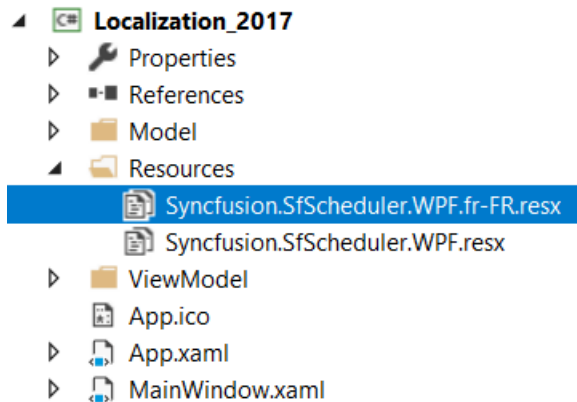
Note: [View Syncfusion.SfScheduler.WPF.resx in GitHub](#)



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.SfScheduler.WPF.<culture name>.resx**. For example, give the name as **Syncfusion.SfScheduler.WPF.de.resx** for German culture.
5. The culture name that indicates the name of language and country.



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



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

Syncfusion.SfScheduler.WPF.fr-FR.resx

Strings Add Resource Remove Resource Access Modifier: No code gen

Name	Value	Comment
AllDay	Toute la journée	Text for All Day
April	avril	Text for April
August	août	Text for August
Cancel	Annuler	Content for Cancel button
Count	Compter	Text for Count
Daily	du quotidien	Text for Daily
Day	journée	Text for Day
Days	Journées)	Text for Day(s)
December	décembre	Text for December
Delete	Supprimer	Content for Delete button
DeleteOccurrence	Supprimer cette occurrence	Content for single radio button that is displayed
DeleteRecurringDialogContent	Voulez-vous supprimer toutes les occurrences du rendez-vous récurrent [], ou	'[]' replaced by selected appointment subject
DeleteRecurringDialogHeader	Confirmation de la suppression	Title content of dialog that is displayed when
DeleteSeries	Supprimer cette série	Content for series radio button that is displayed
Description	La description	Text for Description
EditEventTitle	Modifier l'événement	Text for Edit Event
EditRecurringDialogContent	Il s'agit d'un rendez-vous d'une série. Que voulez-vous ouvrir?	Edit confirmation dialog content that is displayed
EditRecurringDialogHeader	Ouvrir un élément récurrent	Title content of dialog that is displayed when
End	Fin	Text for End
EndTimezone	Fin du fuseau horaire	Text for End Timezone
February	février	Text for February
First	Première	Text for First

< > juillet 2020

dim.	lun.	mar.	mer.	jeu.	ven.	sam.
28	29	Nouvel évènement			3	4
5	6	Titre Meeting Début 14/07/2020 <input checked="" type="checkbox"/> Toute la journée Répéter Jamais La description sauver Annuler			10	11
12	13				17	18
19	20				24	25
26	27				31	1
2	3	4	5	6	7	8

Note: [View sample in GitHub](#). You can refer to our [WPF Scheduler](#) feature tour page for its groundbreaking feature representations.

Accessibility Support in WPF Scheduler (SfScheduler)

Screen reader support

The `WPF Scheduler` can easily be accessed by the screen readers. Please find the following table to get the spoken feedback about the inner element contents of the screen.

Month view

View	Accessibility Format	Example
Month cell	dddd, MMMM d, yyyy	Sunday, May 30, 2021
ViewHeader	dddd	Sunday
Appointment (all-day)	Subject All-day	Meeting All day
Appointment	Subject h:mm t - h:mm tt dddd, MMMM d, yyyy	General Meeting 10:00 AM-11:00 AM Tuesday, June 8, 2021
Spanning Appointment	Subject dddd, MMMM d, yyyy h:mm:ss tt - dddd, MMMM d, yyyy h:mm:ss tt	Plan Execution Friday, May 28, 2021 10:00:00 AM-Sunday, May 30, 2021 11:00:00 AM
Week number	Week number	Week Number 22
Month agenda view with No date selected	No Date Selected	No Date Selected
Month Agenda view with No events	dddd, MMMM d, yyyy No Events	Wednesday, June 2, 2021 No Events
Month Agenda view list appointment	dddd, dddd, MMMM d, yyyy	Thursday, June 24, 2021
BlackoutDates	Blackout Day dddd, MMMM d, yyyy	Blackout Day-Saturday, June 26, 2021

Day, week and workweek views

View	Accessibility Format	Example
Timeslot cell	dddd, MMMM d, yyyy h:mm:ss tt	Sunday, June 20, 2021 12:00:00 AM
ViewHeader	dddd, MMMM d, yyyy	Sunday, June 20, 2021
Appointment (all-day)	Subject All-day	Meeting All day
Appointment	Subject h:mm t - h:mm tt dddd, MMMM d, yyyy	General Meeting 10:00 AM-11:00 AM Tuesday, June 8, 2021

Spanning Appointment	Subject dddd, MMMM d, yyyy h:mm:ss tt - dddd, MMMM d, yyyy h:mm:ss tt	Plan Execution Friday, May 28, 2021 10:00:00 AM-Sunday, May 30, 2021 11:00:00 AM
Time ruler	Time ruler	12 AM

Timeline views

View	Accessibility Format	Example
Timeslot cell	dddd, MMMM d, yyyy h:mm:ss tt	Sunday, June 20, 2021 12:00:00 AM
ViewHeader	dddd, MMMM d, yyyy	Sunday, June 20, 2021
Appointment (all-day)	Subject All-day	Meeting All day
Appointment	Subject h:mm t - h:mm tt dddd, MMMM d, yyyy	General Meeting 10:00 AM-11:00 AM Tuesday, June 8, 2021
Spanning Appointment	Subject dddd, MMMM d, yyyy h:mm:ss tt - dddd, MMMM d, yyyy h:mm:ss tt	Plan Execution Friday, May 28, 2021 10:00:00 AM-Sunday, May 30, 2021 11:00:00 AM
Time ruler	Time ruler	12 AM

Scheduler header

View	Accessibility Format	Example
Day, Week, Work week, Month, Timeline day and Timeline month	MMMM yyyy	June 2021
Timeline week and Timeline work week	MMMM yyyy - MMMM yyyy	28 June - 2 July 2021
Previous navigation button	Previous navigation button	Previous navigation button
Next navigation button	Next navigation button	Next navigation button

Scheduler resource header

View	Accessibility Format	Example
Resource header	Name Id	Sophia 1000

Appointment editor window

View	Accessibility Format	Example
Add appointment	Title	New event
Edit appointment	Title	Edit event
Title textbox	Title Edit Subject Text Enter line	Title Edit Meeting Enter line
Location textbox	Title Edit Location Text Enter line	Location Edit Chennai Enter line
Notes textbox	Title Edit NotesText Enter line	Notes Edit (description) Enter line
Save button	Save button	Save button
Cancel button	Cancel button	Cancel button
Delete button	Delete button	Delete button
Timezone/All-day check box	All day checkbox	All day checkbox checked/Unchecked
Date/time picker	Date/time value edit	12/1/2021 edit
Combo box	Combobox content	Custom

Recurrence editing window

View	Accessibility Format	Example
Editing or Deleting an recurrence appointment	Title	Open Recurring Item / Confirm Delete
Radio button	Content ControlType	Just this one RadioButton / Delete this occurrence
Button	Content ControlType	OK button / Cancel button

Reminder window

View	Accessibility Format	Example
Reminder window	Title	4 reminders
Dismiss all button	Dismiss all button	Dismiss all button

Snooze button	Snooze button	Snooze button
Dismiss button	Dismiss button	Dismiss button
Reminder appointments selected	subject Selected appointment index out of over all index.	Conference 1 of 4

Keyboard navigation

The **SfScheduler** supports selection using keyboard interactions.

Day, Week and WorkWeek views

Navigation Shortcut Keys	Descriptions
Right arrow	Moves selection to the same time slot on the next day.
Left arrow	Moves selection to the same time slot on the previous day.
Down arrow	Moves selection to the next time slot directly below the currently selected time slot.
Up arrow	Moves selection to the previous time slot directly above the currently selected time slot.

Timeline views

Navigation Shortcut Keys	Descriptions
Right arrow	Moves selection to the next time slot of the currently selected time slot.
Left arrow	Moves selection to the previous time slot of the currently selected time slot.

Month view

Navigation Shortcut Keys	Descriptions
Right arrow	Moves selection to the next date of the currently selected date.
Left arrow	Moves selection to the previous date of the currently selected date.
Down arrow	Moves selection to the date directly below the currently selected date on the next row.
Up arrow	Moves selection to the date directly above the currently selected date on the previous row.

Appointments

Navigation Shortcut Keys	Descriptions
Tab	Moves selection to the next appointment of the currently selected appointment.
Shift + Tab	Moves selection to the previous appointment of the currently selected appointment.
Delete	Deletes the selected appointment from appointments collection.

View navigations

Ctrl + Alt + 1 => DayView	Moves the view to day view.
Ctrl + Alt + 2 => WeekView	Moves the view to week view.
Ctrl + Alt + 3 => WorkWeekView	Moves the view to work week view.
Ctrl + Alt + 4 => MonthView	Moves the view to work week view.
Ctrl + Alt + 5 => TimelineDayView	Moves the view to Timeline day view.
Ctrl + Alt + 6 => TimelineWeekView	Moves the view to Timeline week view.
Ctrl + Alt + 7 => TimelineWorkWeekView	Moves the view to Timeline work week view.
Ctrl + Alt + 8 => TimelineMonthView	Moves the view to Timeline month view.

Note: You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

Migrating from SfSchedule to SfScheduler in WPF Scheduler

SfScheduler is a new Scheduler control introduced in 18.2.0.45 Version. The SfScheduler control is used to schedule and manage the appointments through an intuitive user interface, similar to the Outlook calendar. This section helps to identify the equivalent SfSchedule features or APIs in the SfScheduler.

Adding Reference

SfSchedule Assembly Name: Syncfusion.SfSchedule.WPF

SfSchedule Namespace Name: Syncfusion.UI.Xaml.Schedule

SfScheduler Assembly Name: Syncfusion.SfScheduler.WPF

SfScheduler Namespace Name: Syncfusion.UI.Xaml.Scheduler

The following code example shows xmlns namespace for SfScheduler control. Include the Syncfusion schema in WPF and both the charts are available in the WPF schema.

SfSchedule

XML

```
xmlns:schedule="http://schemas.syncfusion.com/wpf"
```

SfScheduler

XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
(or)
xmlns:syncfusion="clr-
namespace:Syncfusion.UI.Xaml.Scheduler;assembly=Syncfusion.SfScheduler.WPF"
```

Initialization

Both [SfSchedule](#) and [SfScheduler](#) almost have the same set of features. But the SfScheduler control offers a rich set of features over SfSchedule.

Major improvements of SfScheduler

- Improved the scheduler control rendering performance.
- Improved the performance on switching the views such as day, week, work week, month, and timeline views.
- Performance has been improved while loading more events or appointments.
- Schedule view or date swiping and scrolling interaction improved.
- Schedule elements easily customized using the Style and Template according to the WPF standard such as events or appointments, month cell.
- Support Recurrence pattern exception dates and exception appointments.
- Support special time region for Timeslot views.

Note: In the future, new features & enhancements will be added only in **SfScheduler**. It is recommended to use the **SfScheduler**.

The following table shows the API comparison between SfSchedule and SfScheduler.

SfSchedule	SfScheduler	Description
ScheduleType	ViewType	Gets or sets a value which determines Scheduler ViewType.
FirstDayOfWeek	FirstDayOfWeek	Gets or sets the day of the week in Schedule.
NonWorkingDays	NonWorkingDays	Gets or sets the non working days.
Appointments , ItemsSource	ItemsSource	Gets or sets a items source to the scheduler.
ScheduleAppointment	ScheduleAppointment	Gets or sets the appointments data in schedule control.

AppointmentMapping	AppointmentMapping	Gets or sets the mapping to the appointment.
Resource	ResourceCollection	Gets or sets the Resource grouping for schedule.

The following table compares the [ScheduleAppointment](#) APIs,

SfSchedule(ScheduleAppointment)	SfScheduler(ScheduleAppointment)	Description
Subject	Subject	Gets or sets the subject for the appointment.
Notes	Notes	Gets or sets the notes for an appointment.
Location	Location	Gets or sets the location for an appointment.
AppointmentBackground	AppointmentBackground	Gets or sets the appointment color.
StartTime	StartTime	Gets or sets the start date and time of the appointment.
EndTime	EndTime	Gets or sets the end date and time of the appointment.
-	ActualStartTime	Gets the internal start time which is converted based on start time zone applied.
-	ActualEndTime	Gets the internal end time which is converted based on start time zone applied.
StartTimeZone	StartTimeZone	Gets or sets time zone for the start time of the appointment.
EndTimeZone	EndTimeZone	Gets or sets time zone for the end time of the appointment.
IsRecursive	IsRecursive	Gets a value indicating whether the appointment is recurrence appointment or not.
AllDay	IsAllDay	Gets or sets a value indicating whether the appointment's duration is equal to one day or not.

RecurrenceRule	RecurrenceRule	Gets or sets a value indicating whether the appointment should be recursive.
RecursiveExceptionDates	RecurrenceExceptionDates	Gets or sets the properties for maintaining recurrence rule exception Dates.
-	RecurrenceId	Gets or sets an unique ID for referring recurrence appointment.
-	Data	Gets the data object associated with appointment.
-	Type	Gets the type of appointment.

The following table compares the [AppointmentMapping](#) APIs,

SfSchedule(AppointmentMapping)	SfScheduler(AppointmentMapping)	Description
SubjectMapping	Subject	Gets or sets the Subject property for mapping to the schedule appointment.
NotesMapping	Notes	Gets or sets the Notes property for mapping to the schedule appointment.
LocationMapping	Location	Gets or sets the Location property for mapping to the schedule appointment.
AppointmentBackgroundMapping	AppointmentBackground	Gets or sets the AppointmentBackground property for mapping to the schedule appointment.
StartTimeMapping	StartTime	Gets or sets the StartTime property for mapping to the schedule appointment.
EndTimeMapping	EndTime	Gets or sets the EndTime property for mapping to the schedule appointment.
StartTimeZoneMapping	StartTimeZone	Gets or sets the StartTimeZone property for mapping to the schedule appointment.

EndTimeZoneMapping	EndTimeZone	Gets or sets the EndTimeZone property for mapping to the schedule appointment.
AllDayMapping	IsAllDay	Gets or sets the IsAllDay property for mapping to the schedule appointment.
RecurrenceRuleMapping	RecurrenceRule	Gets or sets the RecurrenceRule property for mapping to the schedule appointment.
RecursiveExceptionDatesMapping	RecurrenceExceptionDates	Gets or sets the RecurrenceExceptionDates property for mapping to the schedule appointment.
-	RecurrenceId	Gets or sets the RecurrenceId property for mapping to the schedule appointment.

See the list of the rich set of features in [SfScheduler](#) over [SfSchedule](#) as follows:

Rich set of features in [SfScheduler](#) over [SfSchedule](#).

Feature	Description
Events	Appointments contain information on events scheduled at specific times. In addition to default appointments, the users can use their own collections to connect a business entity to an appointment by mapping their fields, such as start time, end time, subject, notes, and recurrence.
Data Binding	The scheduler control supports to bind any collection that implements the IEnumerable interface to populate appointments.
Recurring Events	Easily configure recurring events on a daily, weekly, monthly, or yearly basis. Also skip or change the occurrence of a recurring appointment.

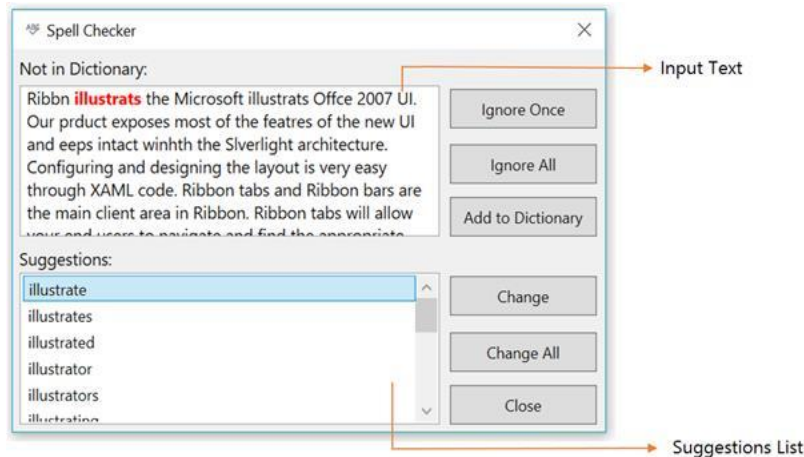
Note: You can refer to our [WPF Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Scheduler example](#) to know how to schedule and manage appointments through an intuitive user interface, similar to the Outlook calendar.

SfSpellChecker

WPF SpellChecker (SfSpellChecker) Overview

[SfSpellChecker](#) control provides a simple and intuitive interface to check for spelling errors in text editor controls. You can perform spell checking on text editor control and it will also provide suggestions for the misspelled words through dialog and context menu. You can use spell check for any language(culture) input text and custom dictionary supports.

Control structure



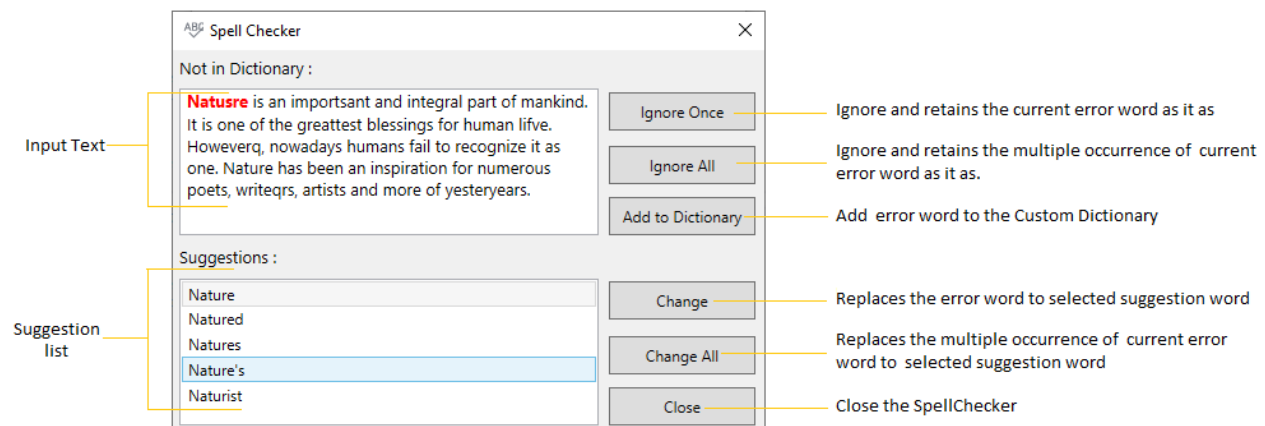
Features

- Supports Context Menu suggestion.
- Supports spell check for any language(culture)
- Supports Custom Dictionary to provide suggestions.
- Provide built-in options to Ignore, Ignore All, Replace, Replace All for error words in spell checker dialog.
- Support to Ignore Email, URL, Numbers, Mixed and Upper case words from spell check.
- Highlights the error words.

Getting Started with WPF SpellChecker (SfSpellChecker)

This section explains how to create a [WPF SpellChecker](#) (SfSpellChecker) and spell check the text.

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 SfSpellChecker to an application

Spell checking operation can be done on text editor controls through **SfSpellChecker** in WPF application.

You can add the **SfSpellChecker** to an application by the following steps,

1. Create a WPF project in Visual Studio and include following assembly.
 - o Syncfusion.SfSpellChecker.WPF
2. Add **TextBox** control and set [SfSpellChecker.SpellChecker](#) attached property to perform spell check.

XML

```
<Grid>
<StackPanel>
<TextBox
Text="Natusre is an importsant and integral part of mankind. It is one of
the greatest blessings for human lifve. Howeverq, nowadays humans fail to
recognize it as one. Nature has been an inspiration for numerous poets,
writeqrs, artists and more of yesteryears."
Name="textbox"
TextWrapping="Wrap">
<!--Adding Spellchecker to the TextBox-->
<syncfusion:SfSpellChecker.SpellChecker>
<syncfusion:SfSpellChecker
x:Name="spellChecker"
EnableSpellCheck="True"/>
</syncfusion:SfSpellChecker.SpellChecker>
</TextBox>
<Button
Content="Spell Check"
Click="SpellCheck_ButtonClick"
HorizontalAlignment="Center"></Button>
</StackPanel>
</Grid>
```

C#

```
//Creating a spell checker instance
SfSpellChecker spellChecker = new SfSpellChecker();
//Enabling the spell check
spellChecker.EnableSpellCheck = true;
//Assigning a spellchecker to the TextBox
SfSpellChecker.SetSpellChecker(textbox, spellChecker);
```

3. If you want to open the **SfSpellChecker** while clicking on the **Spell Check** button, call the [PerformSpellCheckUsingDialog](#) method inside the **SpellCheck ButtonClick** method.

C#

```
//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
```

```
spellChecker.PerformSpellCheckUsingDialog();  
}
```

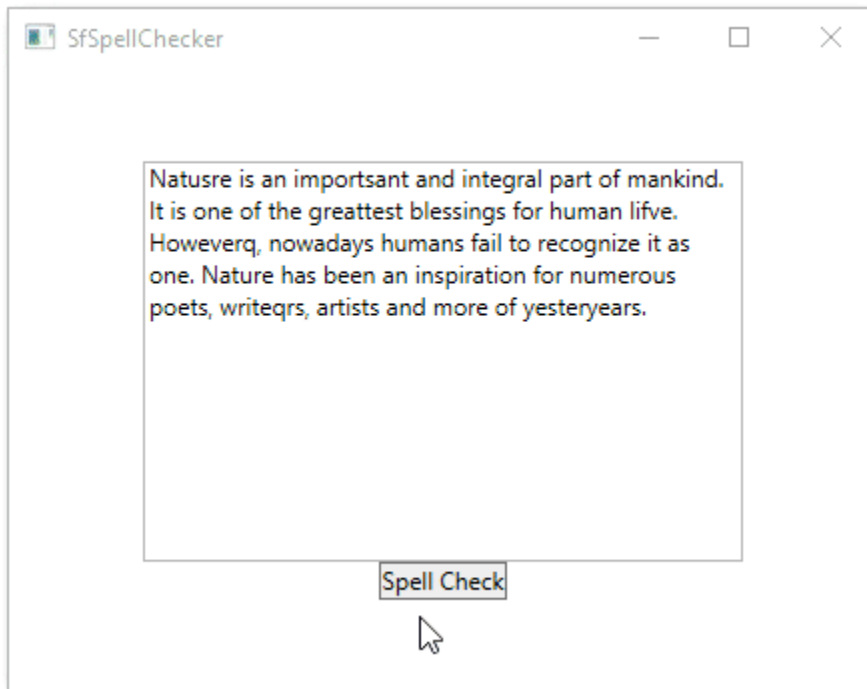
Natusre is an importantsant and integral part of mankind.
It is one of the greatestest blessings for human lifve.
Howeverq, nowadays humans fail to recognize it as
one. Nature has been an inspiration for numerous
poets, writeqrs, artists and more of yesteryears.

Spell Check

Note: View [Sample](#) in GitHub

Fix spelling mistakes using spell check dialog

1. You can open a SfSpellChecker by clicking Spell Check button and then SpellChecker opened as pop-up with TextSpellEditor.
2. The error words are highlight with Red foreground.
3. You can replace error words with correct words by double click the suitable word listed in the suggestion listbox or select the suggestion word from the listbox and then press the Change button.



Note: View [Sample](#) in GitHub

Fix spelling mistakes using context menu

You can simply correct the spell error words by choosing the correct option from listed suggestions from the ContextMenu. You can get the suggestion words by right click on the error word. The Error words are differentiated by red underlining. You can disable the context menu suggestion by using the `EnableContextMenu` property value as `false`. The default value of `EnableContextMenu` property is `true`.

XML

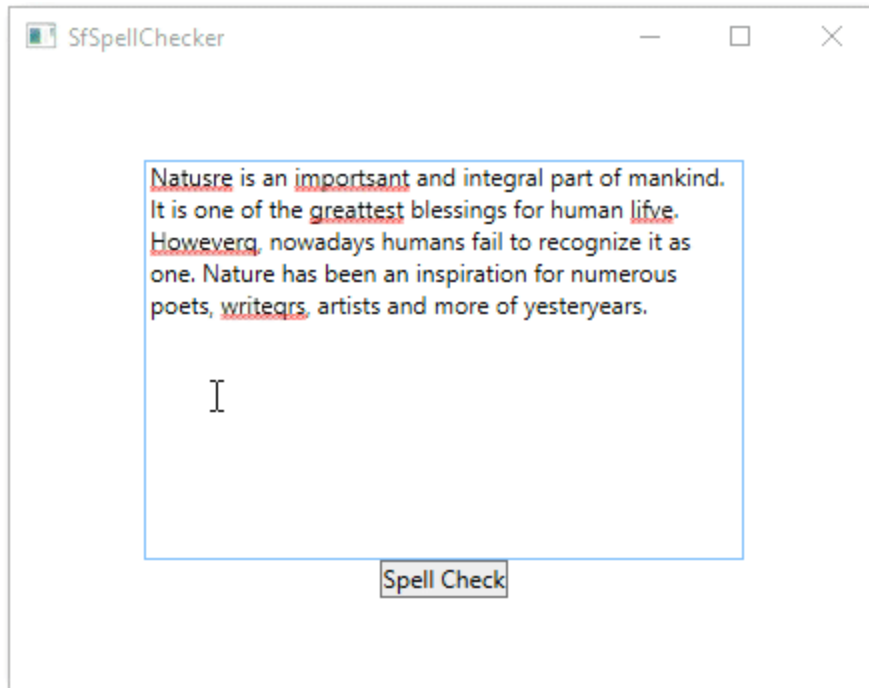
```
<Grid>
<StackPanel>
<TextBox
Text="Natusre is an importsant and integral part of mankind. It is one of
the greatest blessings for human lifve. Howeverq, nowadays humans fail to
recognize it as one. Nature has been an inspiration for numerous poets,
writeqrs, artists and more of yesteryears."
Name="textbox"
TextWrapping="Wrap"
VerticalContentAlignment="Top">
<!--Adding Spellchecker to the TextBox-->
<syncfusion:SfSpellChecker.SpellChecker>
<syncfusion:SfSpellChecker
x:Name="spellChecker"
<!--Enable Contextmenu to spellcheck-->
EnableContextMenu="True"
EnableSpellCheck="True"/>
</syncfusion:SfSpellChecker.SpellChecker>
</TextBox>
<Button
Content="Spell Check"
Click="SpellCheck_ButtonClick"
HorizontalAlignment="Center"></Button>
</StackPanel>
</Grid>
```

C#

```
//Enable Contextmenu to spellcheck
spellChecker.EnableContextMenu = true;
spellChecker.EnableSpellCheck = true;
```

C#

```
//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
spellChecker.PerformSpellCheckUsingDialog();
}
```



Note: View [Sample](#) in GitHub

Disable spell checking

If you want to disable the spell check operation, use the `EnableSpellCheck` property value as `false`. If the `EnableSpellCheck` property value is `false`, you will not be able to use both the context menu and SpellCheck dialogue to perform spell checking operations. The default value of `EnableSpellCheck` property is `true`.

XML

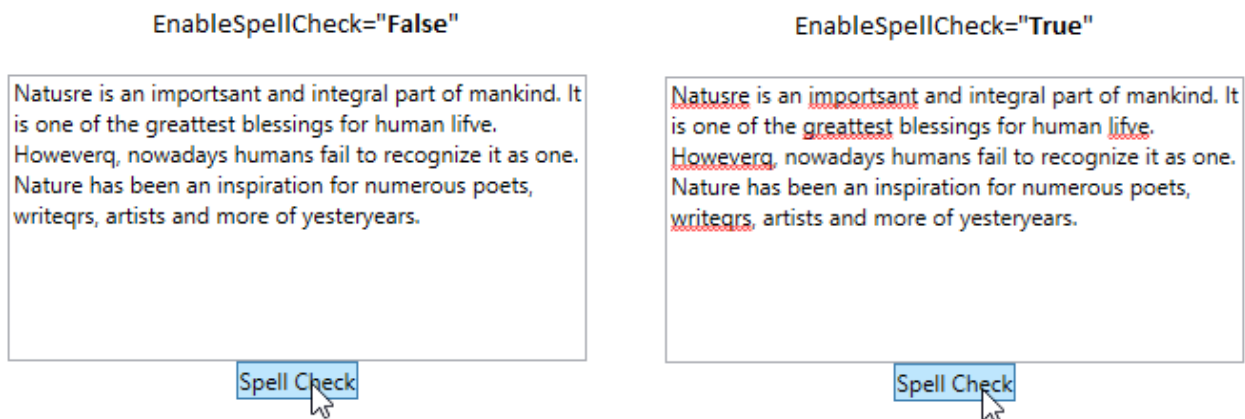
```
<Grid>
  <StackPanel>
    <TextBox
      Text="Natusre is an importants and integral part of mankind. It is one of
      the greatttest blessings for human lifve. Howeverq, nowadays humans fail to
      recognize it as one. Nature has been an inspiration for numerous poets,
      writeqrs, artists and more of yesteryears."
      Name="textbox"
      TextWrapping="Wrap">
      <!--Adding Spellchecker to the TextBox-->
      <syncfusion:SfSpellChecker.SpellChecker>
        <syncfusion:SfSpellChecker
          x:Name="spellChecker"
          <!--Restrict the spell check operation-->
          EnableSpellCheck="False"/>
        </syncfusion:SfSpellChecker.SpellChecker>
      </TextBox>
      <Button
        Content="Spell Check"
        Click="SpellCheck_ButtonClick"
        HorizontalAlignment="Center"></Button>
      </StackPanel>
    </Grid>
```

C#

```
//Restrict the spell check operation
spellChecker.EnableSpellCheck = false;
```

C#

```
//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
    spellChecker.PerformSpellCheckUsingDialog();
}
```



Note: View [Sample](#) in GitHub

Get suggestions for misspelled word

You can get the suggestion list by passing the error word in the below methods.

- [GetSuggestions](#) - To get a list of suggestion words for an error word
- [GetPhoneticWords](#) - To get a list of phonetic words for an error word
- [GetAnagrams](#) - To get a list of anagram words for an error word

XML

```
<Grid>
<StackPanel>
<TextBox
    Text="Natusre is an importsant and integral part of mankind. It is one of
    the greatest blessings for human lifve. Howeverq, nowadays humans fail to
    recognize it as one. Nature has been an inspiration for numerous poets,
    writeqrs, artists and more of yesteryears."
    Name="textbox"
    TextWrapping="Wrap">
<!--Adding Spellchecker to the TextBox-->
<syncfusion:SfSpellChecker.SpellChecker>
<syncfusion:SfSpellChecker
    x:Name="spellChecker"
    <!--Enable Contextmenu to spellcheck-->
    EnableContextMenu="True"
```



```

EnableSpellCheck="True"/>
</syncfusion:SfSpellChecker.SpellChecker>
</TextBox>
<Button
Content="Spell Check"
Click="SpellCheck_ButtonClick"
HorizontalAlignment="Center"></Button>
</StackPanel>
</Grid>

```

C#

```

//Enable Contextmenu to spellcheck
spellChecker.EnableContextMenu = true;

```

C#

```

//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
    spellChecker.PerformSpellCheckUsingDialog();
}

```

Ignore SpellCheck for particular types of text

If you want to ignore the error words such as a format like email id's and link addresses, HTML tags, combination of words and numbers, combination of upper and lower case words, use the respective property value as **true** from the following table,

Property | Description | Example

IgnoreEmailAddress | Specifies whether or not to ignore email address during Spell Check. The Default value is False. | Ex: john@abc.com

IgnoreHtmlTags | Specifies whether or not to ignore HTML tags during Spell Check. The Default value is False. | Ex: < html></ html>

IgnoreUrl | Specifies whether or not to ignore Internet address during Spell Check. The Default value is False. | Ex: https://help.syncfusion.com

IgnoreMixedCaseWords | Specifies whether or not to ignore mixed case words during Spell Check. The Default value is False. | Ex: AbCDeFH

IgnoreUpperCaseWords | Specifies whether or not to ignore uppercase words during Spell Check. The Default value is False. | Ex: ABCDE >

IgnoreAlphaNumericWords | Specifies whether or not to Spell Check numbers or words with numbers during Spell Check. The Default value is False. | Ex: A*	ACe&981

XML

```

<Grid>
<StackPanel>
<TextBox
Text="Natusre is an importants and integral part of mankind. It is one of
the greatest blessings for human lifve. Howeverq, nowadays humans fail to

```

```

recognize it as one. Nature has been an inspiration for numerous poets,
writegrs, artists and more of yesteryears."
Name="textbox"
TextWrapping="Wrap">
<!--Adding Spellchecker to the TextBox-->
<syncfusion:SfSpellChecker.SpellChecker>
<syncfusion:SfSpellChecker
x:Name="spellChecker"
EnableContextMenu="True"
EnableSpellCheck="True"
IgnoreUrl="True"
IgnoreUpperCaseWords="True"
IgnoreAlphaNumericWords="True"
IgnoreEmailAddress="True"
IgnoreMixedCaseWords="True"
IgnoreHtmlTags="True"/>
</syncfusion:SfSpellChecker.SpellChecker>
</TextBox>
<Button
Content="Spell Check"
Click="SpellCheck_ButtonClick"
HorizontalAlignment="Center"></Button>
</StackPanel>
</Grid>

```

C#

```

spellChecker.IgnoreUrl = true;
spellChecker.IgnoreUpperCaseWords = true;
spellChecker.IgnoreAlphaNumericWords = true;
spellChecker.IgnoreEmailAddress = true;
spellChecker.IgnoreMixedCaseWords = true;
spellChecker.IgnoreHtmlTags = true;

```

C#

```

//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
spellChecker.PerformSpellCheckUsingDialog();
}

```

SpellCheck for any language(culture)

You can spell check any language(culture) by adding the respective culture to the [SfSpellChecker.Culture](#) property and add the dictionaries which contains the basic word file and grammar file to the [SfSpellChecker.Dictionaries](#) collection.

The following dictionary types are used for spell-checking,

- Hunspell
- Ispell
- OpenOffice

Note: Refer the [Load your own dictionaries for any language](#) page to know more about how to add and use the Dictionary for any culture to an application.

Add custom words to dictionary

If you want to add words that is not available in existing dictionary, you can add it using **CustomDictionary**. This dictionary does not has a grammar file, it accepts only dictionary file that contains a list of words. Users can also add words to this custom dictionary by clicking **Add to Dictionary** button available in dialog or context menu.

Note: Refer the [Adding Custom Dictionary](#) page to know more about how to add and use the custom dictionary to an application.

Event to notify when spell check is completed

By default, when the spell check is completed, it will be notified by using the message box that showing the **Spell check is completed** message. If you want to restrict that message box, you can handle the [SpellCheckCompleted](#) event and set the [SpellCheckCompletedEventArgs.ShowMessageBox](#) to **false**.

XML

```
<Grid>
  <StackPanel>
    <TextBox
      Text="Natures is an importants and integral part of mankind. It is one of
      the greatest blessings for human lifve. Howeverq, nowadays humans fail to
      recognize it as one. Nature has been an inspiration for numerous poets,
      writeqrs, artists and more of yesteryears."
      Name="textbox"
      TextWrapping="Wrap">
      <!--Adding Spellchecker to the TextBox-->
      <syncfusion:SfSpellChecker.SpellChecker>
      <syncfusion:SfSpellChecker
        x:Name="spellChecker"
        SpellCheckCompleted="SpellChecker_SpellCheckCompleted"
        EnableContextMenu="True"
        EnableSpellCheck="True"/>
      </syncfusion:SfSpellChecker.SpellChecker>
    </TextBox>
    <Button
      Content="Spell Check"
      Click="SpellCheck_ButtonClick"
      HorizontalAlignment="Center"></Button>
  </StackPanel>
</Grid>
```

C#

```
spellChecker.SpellCheckCompleted += SpellChecker_SpellCheckCompleted;
```

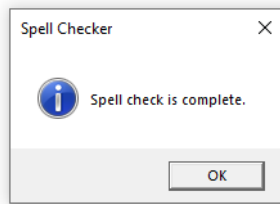
C#

```
//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
    spellChecker.PerformSpellCheckUsingDialog();
}
```

```
private void SpellChecker_SpellCheckCompleted(object sender, EventArgs e) {  
    //Restrict the message box showing  
    (e as SpellCheckCompletedEventArgs).ShowMessageBox = false;  
}
```

SpellCheckCompletedEventArgs.ShowMessageBox = **true**

nature, in the broadest sense, is the natural, physical, or material world or universe. "Nature" can refer to the phenomena of the physical world, and also to life in general. The study of nature is a large, if not the only, part of science. Although humans are part of nature, human activity is often understood as a separate category from other natural phenomena.

SpellCheckCompletedEventArgs.ShowMessageBox = **false**

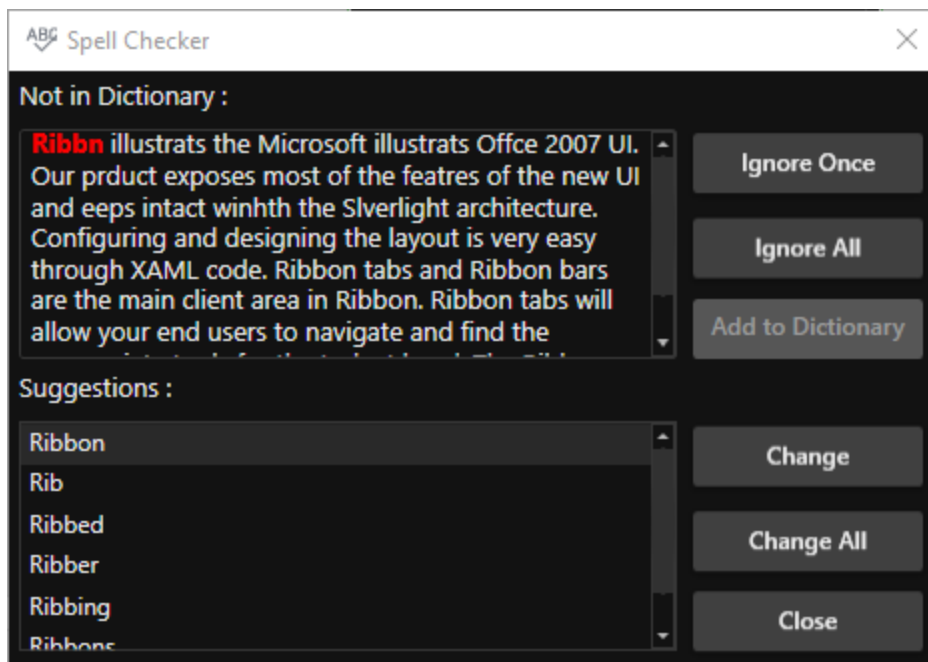
nature, in the broadest sense, is the natural, physical, or material world or universe. "Nature" can refer to the phenomena of the physical world, and also to life in general. The study of nature is a large, if not the only, part of science. Although humans are part of nature, human activity is often understood as a separate category from other natural phenomena.

Note: View [Sample](#) in GitHub

Theme

WPF SpellChecker (SfSpellChecker) supports various built-in themes. Refer to the below links to apply themes for the SfSpellChecker,

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



Custom Dictionary in WPF SpellChecker (SfSpellChecker)

You can use a default and custom dictionaries to spell check the document based on your need. You can also spell check for any culture and languages using various dictionaries.

Default SpellCheck Dictionary

[SfSpellChecker](#) provides built-in dictionary for **English** language and it provides suitable suggestion of the error words.

Load your own dictionaries for any language

You can add your own dictionary to [SfSpellChecker.Dictionaries](#) collection. **SfSpellChecker** support 3 standard dictionary file format:

- 1.Hunspell
- 2.IsPELL
- 3.OpenOffice

Note: Built-in dictionary will be disabled once custom dictionary is added to SfSpellChecker

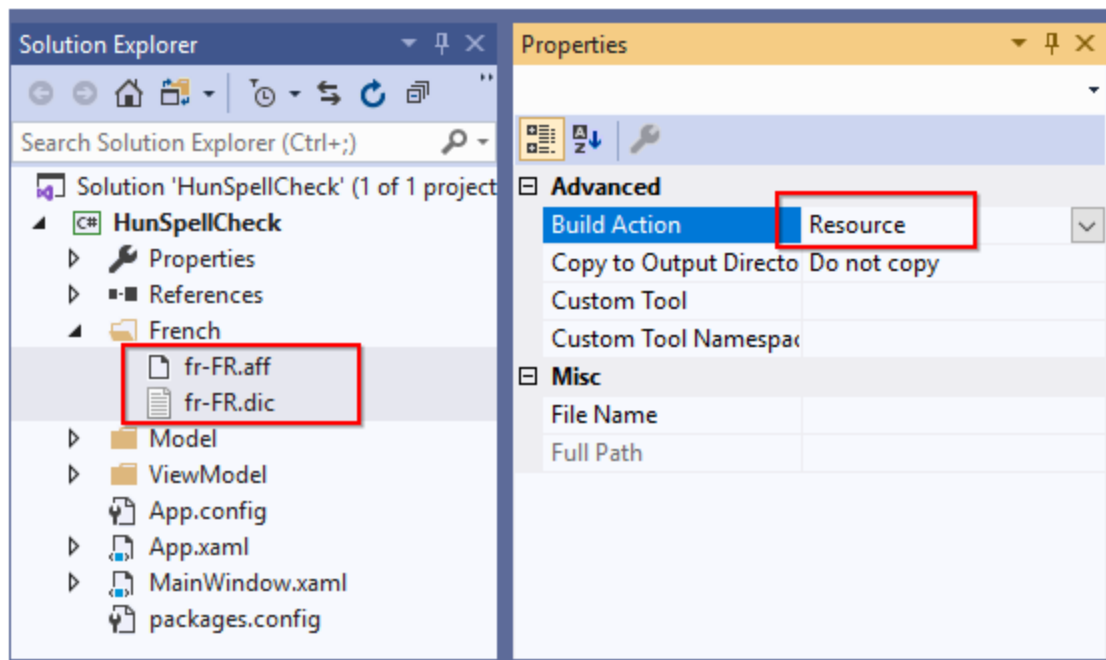
SpellCheck using Hunspell dictionary

You can check spelling mistakes using **Hunspell** dictionary format. This format contains files as follows,

- Affix file with grammar rules- ***.aff**,
- Basic Words file - ***.dic** file.

Adding Hunspell Dictionary

1. Add your [HunspellDictionary](#)'s required culture **.aff and .dic** files and add them as **Resource** into the application.



2. Create a `HunspellDictionary` instance and add the basic word & grammar file path to the `HunspellDictionary.DictionaryUri` & `HunspellDictionary.GrammarUri` properties and add the culture to the `HunspellDictionary.Culture` property.
3. Add the `HunspellDictionary` into the `SfSpellChecker.Dictionaries` collection
4. Setting the required culture to the `SfSpellChecker.Culture` property.

Note: The following code snippets shows how to add Hunspell dictionary to the SpellChecker. Please refer [Adding SfSpellChecker to an application](#) to know how to configure SfSpellChecker.

XML

```
<Grid>
  <StackPanel>
    <TextBox
      Text="Nous sommes heureux de vous avochir ici"
      Name="textbox"
      TextWrapping="Wrap">
      <!--Adding Spellchecker to the TextBox-->
      <syncfusion:SfSpellChecker.SpellChecker>
        <syncfusion:SfSpellChecker
          Culture="fr-FR"
          x:Name="spellChecker"
          EnableSpellCheck="True">
          <syncfusion:SfSpellChecker.Dictionaries>
            <!--Adding French cultured Hunspell dictionary-->
            <syncfusion:HunspellDictionary
              DictionaryUri="/HunSpellCheck;component/French/fr-FR.dic"
              GrammarUri="/HunSpellCheck;component/French/fr-FR.aff"
              Culture="fr-FR"/>
            </syncfusion:SfSpellChecker.Dictionaries>
          </syncfusion:SfSpellChecker>
        </syncfusion:SfSpellChecker.SpellChecker>
      </syncfusion:SfSpellChecker.SpellChecker>
    </StackPanel>
  </Grid>
```

```
</TextBox>
<Button
Content="Spell Check"
Click="SpellCheck_ButtonClick"
HorizontalAlignment="Center"></Button>
</StackPanel>
</Grid>
```

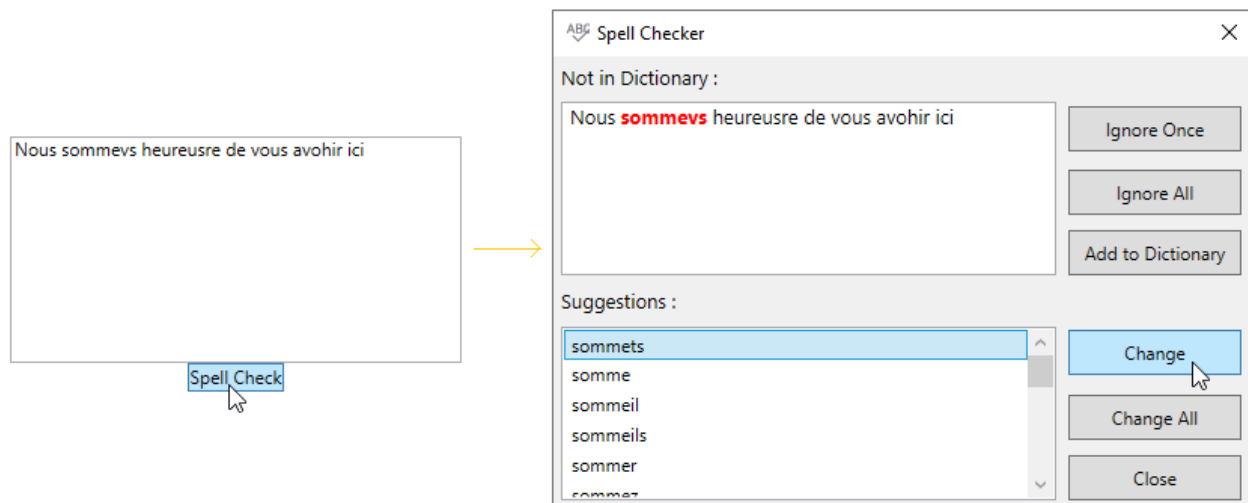
C#

```
//Creating a culture instance
CultureInfo culture = new CultureInfo("fr-FR");
SfSpellChecker spellChecker = new SfSpellChecker();
// Adding Hunspell dictionaries in Dictionaries collection
spellChecker.Dictionaries = new DictionaryCollection();
//Add French culture Hunspell dictionary
spellChecker.Dictionaries.Add(
new HunspellDictionary()
{
Culture = culture,
GrammarUri = new Uri("/HunSpellCheck;component/French/fr-FR.aff",
UriKind.Relative),
DictionaryUri = new Uri("/HunSpellCheck;component/French/fr-FR.dic",
UriKind.Relative)
}
);
//Setting a French culture for SpellChecker
spellChecker.Culture = culture;
//Assigning a spellchecker to the TextBox
SfSpellChecker.SetSpellChecker(textbox, spellChecker);
```

C#

```
//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
spellChecker.PerformSpellCheckUsingDialog();
}
```

Note: You can add multiple `HunspellDictionary` with various culture files into the `SfSpellChecker.Dictionaries` collection. Based on the `SfSpellChecker.Culture` respective `HunspellDictionary` is used for spell check.



SpellCheck using Ispell dictionary

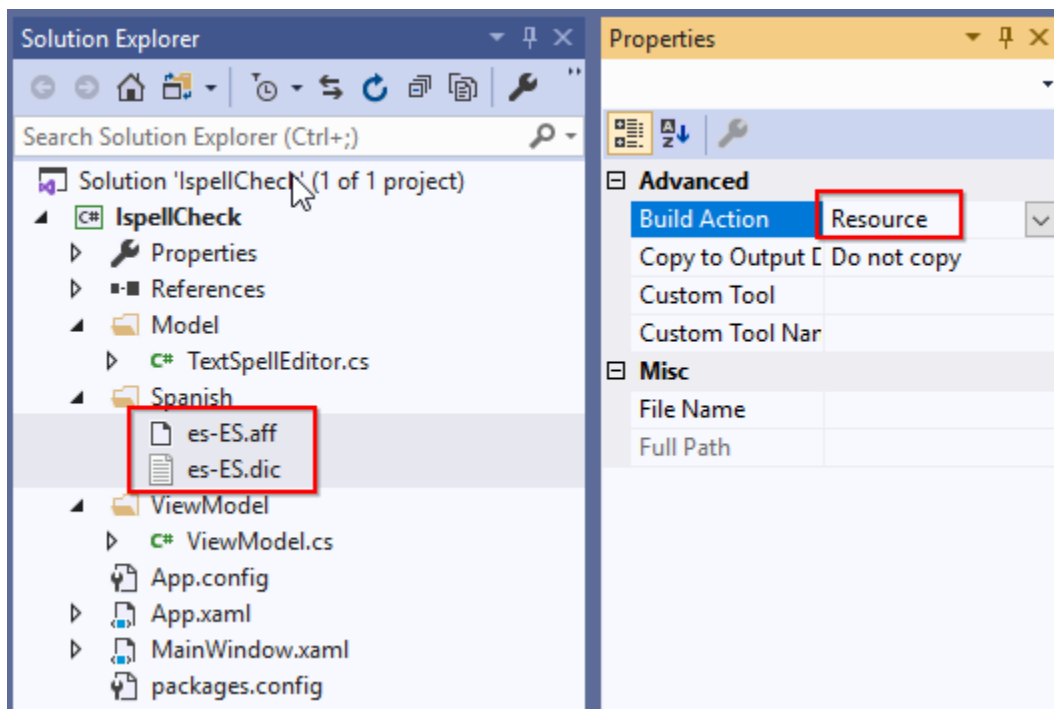
You can check spelling mistakes using **Ispell** dictionary format. This format contains files as follows,

Ispell dictionary contains two files as follows,

- Affix file with grammar rules- *.aff,
- Basic Words file - .xlg or .dic file.

Adding Ispell Dictionary

1. Add your **IspellDictionary**'s required culture **.aff and .dic** files and add them as **Resource** into the application.



2. Create a `IspellDictionary` instance and add the basic word & grammar file path to the `IspellDictionary.DictionaryUri` & `IspellDictionary.GrammarUri` properties and add the culture to the `IspellDictionary.Culture` property.
3. Add the `IspellDictionary` into the `SfSpellChecker.Dictionaries` collection
4. Setting the required culture to the `SfSpellChecker.Culture` property.

XML

```
<Grid>
<StackPanel>
<TextBox
Text="gracsq por venizr por favor ven de nuevoq"
Name="textbox"
TextWrapping="Wrap">
<!--Adding Spellchecker to the TextBox-->
<syncfusion:SfSpellChecker.SpellChecker>
<syncfusion:SfSpellChecker
Culture="es-ES"
x:Name="spellChecker"
EnableSpellCheck="True">
<syncfusion:SfSpellChecker.Dictionaries>
<!--Adding Spanish cultured Ispell dictionary-->
<syncfusion:IspellDictionary
DictionaryUri="/IspellCheck;component/Spanish/es-ES.dic"
GrammarUri="/IspellCheck;component/Spanish/es-ES.aff"
Culture="es-ES"/>
</syncfusion:SfSpellChecker.Dictionaries>
</syncfusion:SfSpellChecker>
</syncfusion:SfSpellChecker.SpellChecker>
</TextBox>
<Button
Content="Spell Check"
Click="SpellCheck_ButtonClick"
HorizontalAlignment="Center"></Button>
</StackPanel>
</Grid>
```

C#

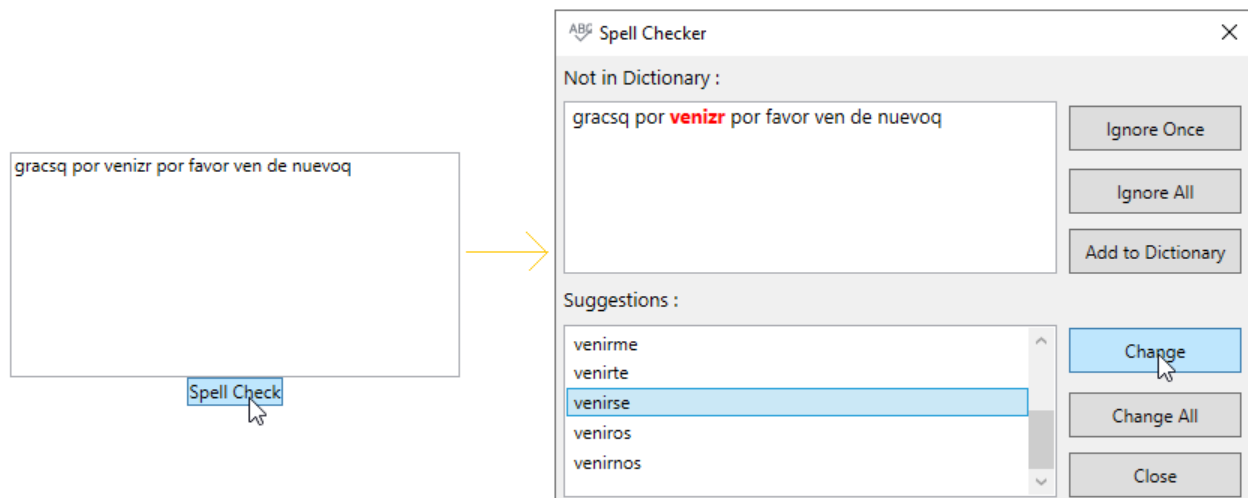
```
//Creating a culture instance
CultureInfo culture = new CultureInfo("es-ES");
SfSpellChecker spellChecker = new SfSpellChecker();
// Adding Ispell dictionaries in Dictionaries collection
spellChecker.Dictionaries = new DictionaryCollection();
//Add Spanish culture Ispell dictionary
spellChecker.Dictionaries.Add(
new IspellDictionary()
{
Culture = culture,
GrammarUri = new Uri("/IspellCheck;component/Spanish/es-ES.aff",
UriKind.Relative),
DictionaryUri = new Uri("/IspellCheck;component/Spanish/es-ES.dic",
UriKind.Relative)
}
);
```

```
//Setting a Spanish culture for SpellChecker
spellChecker.Culture = culture;
//Assigning a spellchecker to the TextBox
SfSpellChecker.SetSpellChecker(textbox, spellChecker);
```

C#

```
//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
    spellChecker.PerformSpellCheckUsingDialog();
}
```

Note: You can add multiple `IspellDictionary` with various culture files into the `SfSpellChecker.Dictionaries` collection. Based on the `SfSpellChecker.Culture` respective `IspellDictionary` is used for spell check.



SpellCheck using OpenOffice dictionary

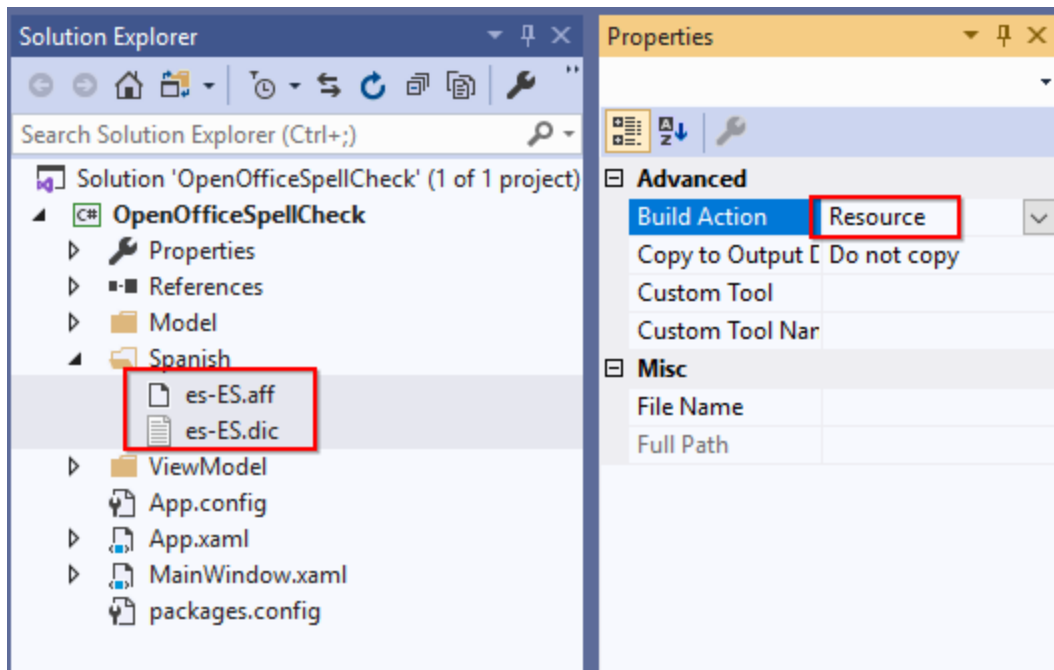
You can check spelling mistakes using `OpenOffice` dictionary format. This format contains files as follows,

`OpenOffice` dictionary contains two files as follows,

- Affix file with grammar rules- `*.aff`,
- Basic Words file - `*.dic` file.

Adding OpenOffice Dictionary

1. Add your `OpenOfficeDictionary`'s required culture `.aff and .dic` files and add them as `Resource` into the application.



2. Create a `OpenOfficeDictionary` instance and add the basic word & grammar file path to the `OpenOfficeDictionary.DictionaryUri` & `OpenOfficeDictionary.GrammarUri` properties and add the culture to the `OpenOfficeDictionary.Culture` property.
3. Add the `OpenOfficeDictionary` into the `SfSpellChecker.Dictionaries` collection
4. Setting the required culture to the `SfSpellChecker.Culture` property.

XML

```
<Grid>
<StackPanel>
<TextBox
Text="gracsq por venizr por favor ven de nuevoq"
Name="textbox"
TextWrapping="Wrap">
<!--Adding Spellchecker to the TextBox-->
<syncfusion:SfSpellChecker.SpellChecker>
<syncfusion:SfSpellChecker
Culture="es-ES"
x:Name="spellChecker"
EnableSpellCheck="True">
<syncfusion:SfSpellChecker.Dictionaries>
<!--Adding Spanish cultured OpenOffice dictionary-->
<syncfusion:OpenOfficeDictionary
DictionaryUri="/OpenOfficeSpellCheck;component/Spanish/es-ES.dic"
GrammarUri="/OpenOfficeSpellCheck;component/Spanish/es-ES.aff"
Culture="es-ES"/>
</syncfusion:SfSpellChecker.Dictionaries>
</syncfusion:SfSpellChecker>
</syncfusion:SfSpellChecker.SpellChecker>
</TextBox>
<Button
Content="Spell Check"
```

```
Click="SpellCheck_ButtonClick"
HorizontalAlignment="Center"></Button>
</StackPanel>
</Grid>
```

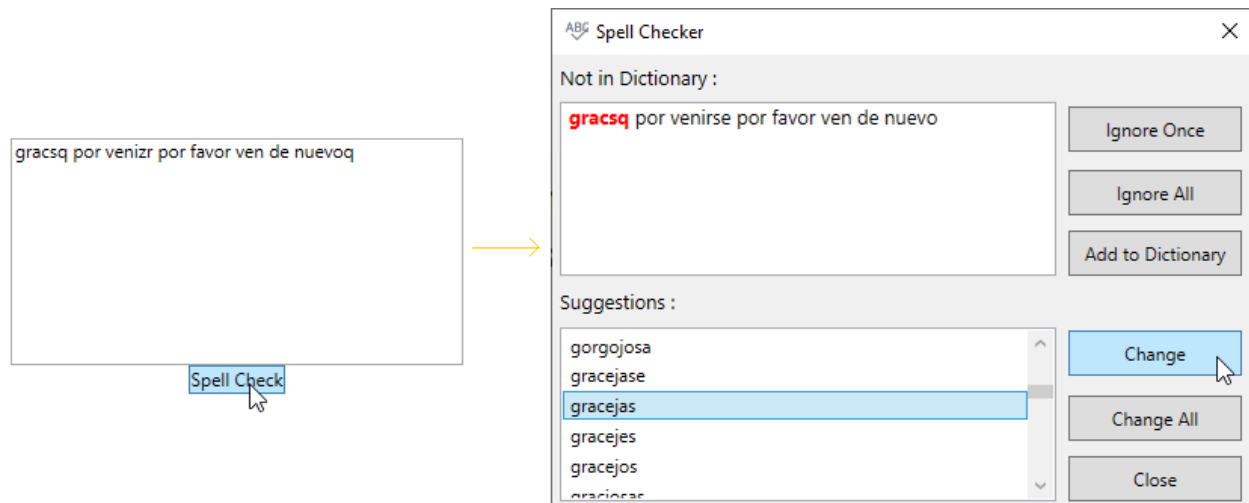
C#

```
//Creating a Spanish culture instance
CultureInfo culture = new CultureInfo("es-ES");
SfSpellChecker spellChecker = new SfSpellChecker();
// Adding OpenOffice dictionaries in Dictionaries collection
spellChecker.Dictionaries = new DictionaryCollection();
//Add Spanish culture OpenOffice dictionary
spellChecker.Dictionaries.Add(
new OpenOfficeDictionary()
{
    Culture = culture,
    GrammarUri = new Uri("/OpenOfficeSpellCheck;component/Spanish/es-ES.aff",
    UriKind.Relative),
    DictionaryUri = new Uri("/OpenOfficeSpellCheck;component/Spanish/es-ES.dic",
    UriKind.Relative)
}
);
//Setting a Spanish culture for SpellChecker
spellChecker.Culture = culture;
//Assigning a spellchecker to the TextBox
SfSpellChecker.SetSpellChecker(textbox, spellChecker);
```

C#

```
//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
    spellChecker.PerformSpellCheckUsingDialog();
}
```

Note: You can add multiple `OpenOfficeDictionary` with various culture files into the `SfSpellChecker.Dictionaries` collection. Based on the `SfSpellChecker.Culture` respective `OpenOfficeDictionary` is used for spell check.



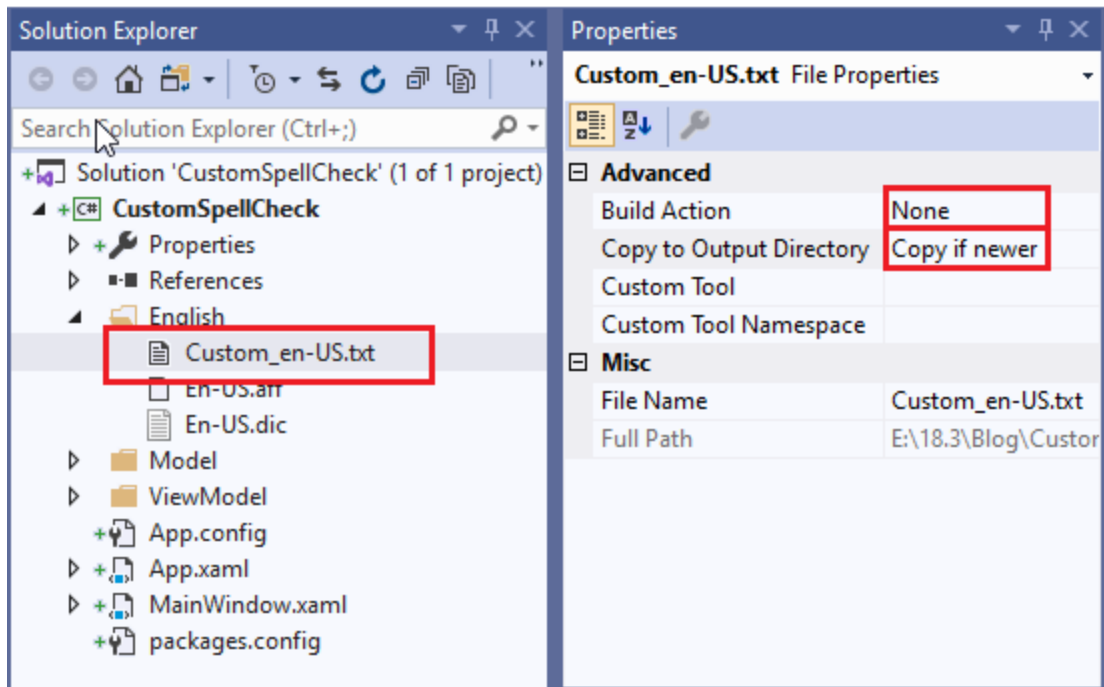
Add custom words to dictionary

If you want to add words that is not available in existing dictionary, you can add it using [CustomDictionary](#). This dictionary does not has a grammar file, it accepts only dictionary file that contains a list of words. Users can also add words to this custom dictionary by clicking **Add to Dictionary** button available in dialog or context menu.

You can add multiple **CustomDictionary** for each culture using **SfSpellChecker.Dictionaries** collection. If you load the **SfSpellChecker** with **en-US** culture, then you can add your custom words only on **en-US** cultured **CustomDictionary**.

Adding Custom Dictionary

1. Create a custom dictionary text file, set build action as **None** and set **Copy to Output Directory** to **Copy if newer**.



2. Create a `CustomDictionary` instance and add the custom word file path to the `CustomDictionary.DictionaryUri` property and add the culture to the `CustomDictionary.Culture` property.
3. Add the `CustomDictionary` into the `SfSpellChecker.Dictionaries` collection
4. Setting the required culture to the `SfSpellChecker.Culture` property.

XML

```
<Grid>
<StackPanel>
<TextBox
Text="Ribbn illustrats the Microsoft illustrats Office 2007 UI. Our prduct
exposes most of the featres of the new UI and eeps intact winhth the
Slverlight architecture. Configuring and designing the layout is very easy
through XAML code. Ribbon tabs and Ribbon bars are the main client area in
Ribbon. Ribbon tabs will allow your end users to navigate and find the
appropriate tools for the task at hand. The Ribbon bars will contain the
Ribbon tools."
Name="textbox"
TextWrapping="Wrap">
<!--Adding Spellchecker to the TextBox-->
<syncfusion:SfSpellChecker.SpellChecker>
<syncfusion:SfSpellChecker
Culture="en-US"
x:Name="spellChecker"
EnableSpellCheck="True">
<syncfusion:SfSpellChecker.Dictionaries>
<!--Adding english cultured custom dictionary-->
<syncfusion:CustomDictionary
DictionaryUri="E:/SpellcheckerDemo/bin/Debug/English/Custom_en-US.txt"
Culture="en-US"/>

```

```
<!--Adding english cultured OpenOffice dictionary-->
<syncfusion:OpenOfficeDictionary
DictionaryUri="/CustomSpellCheck;component/US/en-US.dic"
GrammarUri="/CustomSpellCheck;component/US/en-US.aff"
Culture="en-US"/>
</syncfusion:SfSpellChecker.Dictionaries>
</syncfusion:SfSpellChecker>
</syncfusion:SfSpellChecker.SpellChecker>
</TextBox>
<Button
Content="Spell Check"
Click="SpellCheck_ButtonClick"
HorizontalAlignment="Center"></Button>
</StackPanel>
</Grid>
```

C#

```
//Creating a culture instance
CultureInfo culture = new CultureInfo("en-US");
SfSpellChecker spellChecker = new SfSpellChecker();
// Get the current PROJECT directory
Uri CustomDict_uri= new Uri(Directory.GetCurrentDirectory()+
@"\English\Custom_en-US.txt", UriKind.Absolute);
//Add Custom dictionary for US culture
spellChecker.Dictionaries.Add(
new CustomDictionary()
{
Culture = culture,
DictionaryUri = CustomDict_uri
}
);
//Add US culture OpenOffice dictionary
spellChecker.Dictionaries.Add(
new OpenOfficeDictionary()
{
Culture = culture,
GrammarUri = new Uri("/CustomSpellCheck;component/US/en-US.aff",
UriKind.Relative),
DictionaryUri = new Uri("/CustomSpellCheck;component/US/en-US.dic",
UriKind.Relative)
}
//Setting a US culture for SpellChecker
spellChecker.Culture = culture;
//Assigning a spellchecker to the TextBox
SfSpellChecker.SetSpellChecker(textbox, spellChecker);
```

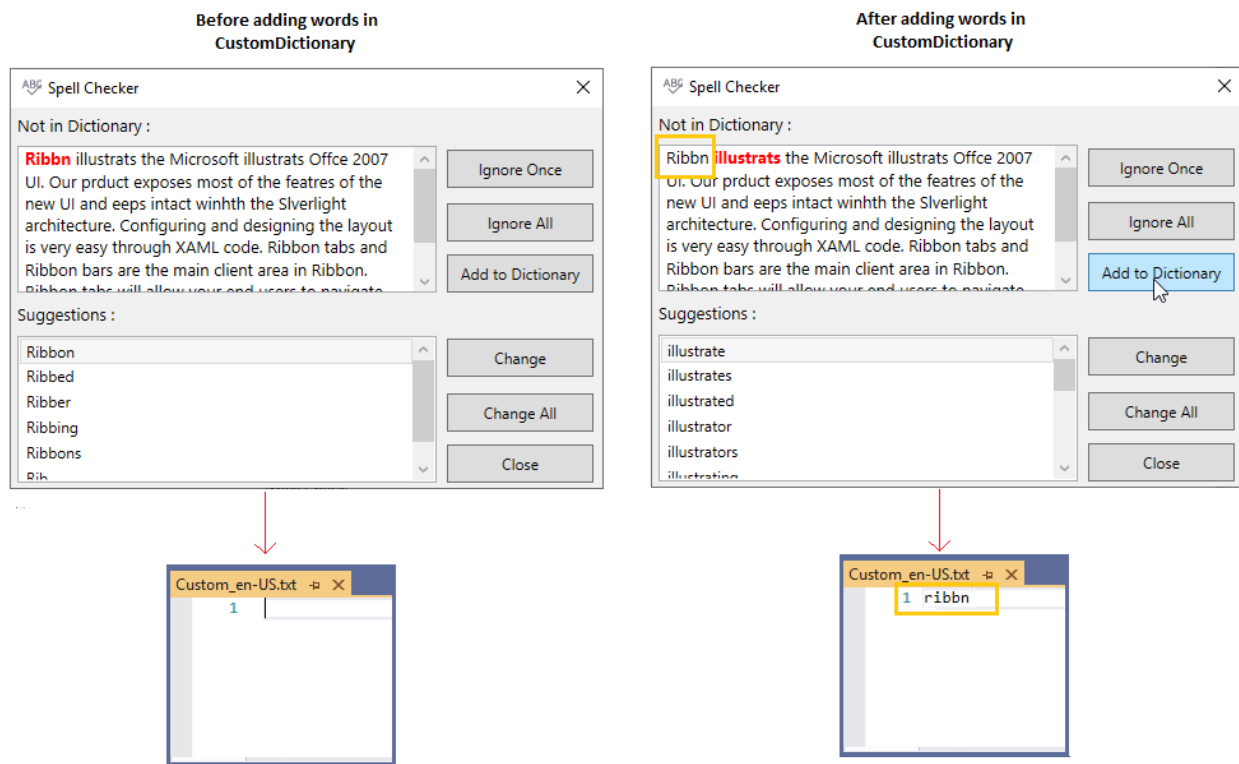
C#

```
//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
spellChecker.PerformSpellCheckUsingDialog();
}
```

Note: As custom dictionary hold extra words that is not available in standard dictionary, it is always used in conjunction with standard dictionary.

Note: If you use the custom dictionary without standard dictionary, all words that are not included in the dictionary will be shown as error words.

Note: You can add multiple CustomDictionary with various culture word files into the SfSpellChecker.Dictionaries collection. Based on the SfSpellChecker.Culture respective CustomDictionary is used for spell check.



Note: [View Sample in GitHub](#)

Switch language(Culture) at runtime

You can add Hunspell, Ispell, or OpenOffice dictionaries one or more times with various culture into the SfSpellChecker.Dictionaries collection. You can change the spell check culture at runtime by changing the SfSpellChecker.Culture property. Based on the current SfSpellChecker.Culture respective dictionary is used to spell check.

XML

```
<Grid>
<StackPanel>
<TextBox
Text="Nous sommevs heureuxre de vous avochir ici"
Name="textbox"
TextWrapping="Wrap">
<!--Adding Spellchecker to the TextBox-->
<syncfusion:SfSpellChecker.SpellChecker>
<syncfusion:SfSpellChecker
Culture="fr-FR"
```



```

x:Name="spellChecker"
EnableSpellCheck="True">
<syncfusion:SfSpellChecker.Dictionaries>
<!--Adding French cultured Hunspell dictionary-->
<syncfusion:HunspellDictionary
DictionaryUri="/HunSpellCheck;component/French/fr-FR.dic"
GrammarUri="/HunSpellCheck;component/French/fr-FR.aff"
Culture="fr-FR"/>
<!--Adding Spanish cultured Hunspell dictionary-->
<syncfusion:HunspellDictionary
DictionaryUri="/HunSpellCheck;component/Spanish/es-ES.dic"
GrammarUri="/HunSpellCheck;component/Spanish/es-ES.aff"
Culture="es-ES"/>
!--Adding english cultured Hunspell dictionary-->
<syncfusion:HunspellDictionary
DictionaryUri="/HunSpellCheck;component/US/en-US.dic"
GrammarUri="/HunSpellCheck;component/US/en-US.aff"
Culture="en-US"/>
</syncfusion:SfSpellChecker.Dictionaries>
</syncfusion:SfSpellChecker>
</syncfusion:SfSpellChecker.SpellChecker>
</TextBox>
<Button
Content="Spell Check"
Click="SpellCheck_ButtonClick"
HorizontalAlignment="Center"></Button>
</StackPanel>
</Grid>

```

C#

```

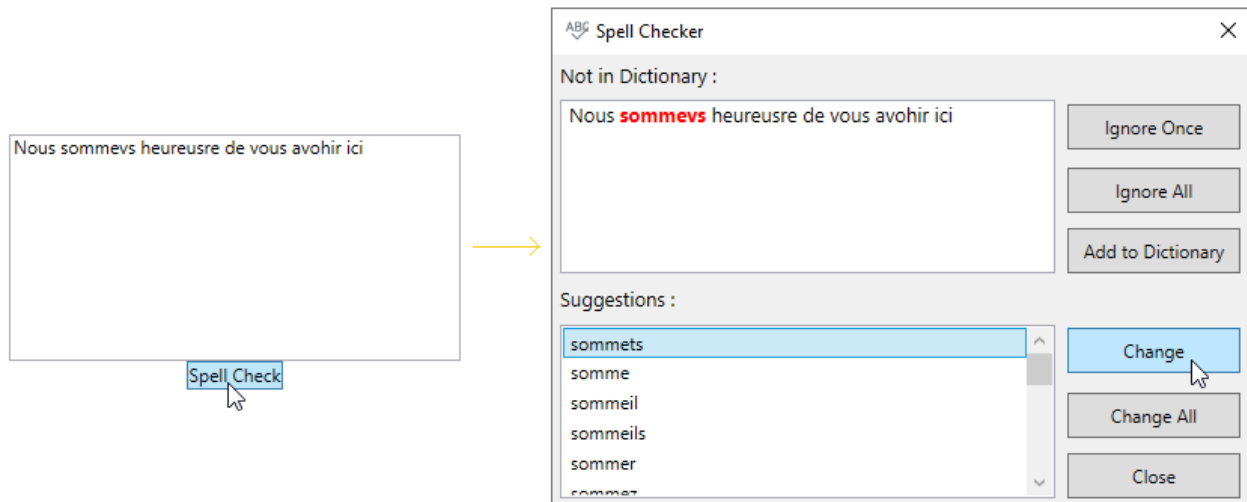
SfSpellChecker spellChecker = new SfSpellChecker();
// Adding Hunspell dictionaries in Dictionaries collection
spellChecker.Dictionaries = new DictionaryCollection();
//Add French culture Hunspell dictionary
spellChecker.Dictionaries.Add(
new HunspellDictionary()
{
Culture = new CultureInfo("fr-FR"),
GrammarUri = new Uri("/HunSpellCheck;component//French/fr-FR.aff",
UriKind.Relative),
DictionaryUri = new Uri("/HunSpellCheck;component//French/fr-FR.dic",
UriKind.Relative)
}
);
//Add Spanish culture Hunspell dictionary
spellChecker.Dictionaries.Add(
new HunspellDictionary()
{
Culture = new CultureInfo("es-ES"),
GrammarUri = new Uri("/HunSpellCheck;component//Spanish/es-ES.aff",
UriKind.Relative),
DictionaryUri = new Uri("/HunSpellCheck;component//Spanish/es-ES.dic",
UriKind.Relative)
}
);

```

```
//Add US culture Hunspell dictionary
spellChecker.Dictionaries.Add(
new HunspellDictionary()
{
Culture = new CultureInfo("en-US"),
GrammarUri = new Uri("/HunSpellCheck;component//US/en-US.aff",
UriKind.Relative),
DictionaryUri = new Uri("/HunSpellCheck;component//US/en-US.dic",
UriKind.Relative)
}
);
//Setting a required dictionary's french culture for SpellChecker
spellChecker.Culture = spellChecker.Dictionaries[0].Culture;;
//Assigning a spellchecker to the TextBox
SfSpellChecker.SetSpellChecker(textbox, spellChecker);
```

C#

```
//Call SpellCheck method to open SpellCheck on button click
private void SpellCheck_ButtonClick(object sender, RoutedEventArgs e) {
spellChecker.PerformSpellCheckUsingDialog();
}
```



Here, `SpellChecker.Culture` is `fr-FR` culture. So, `fr-FR` cultured `Hunspell` dictionary is used as speck check dictionary.

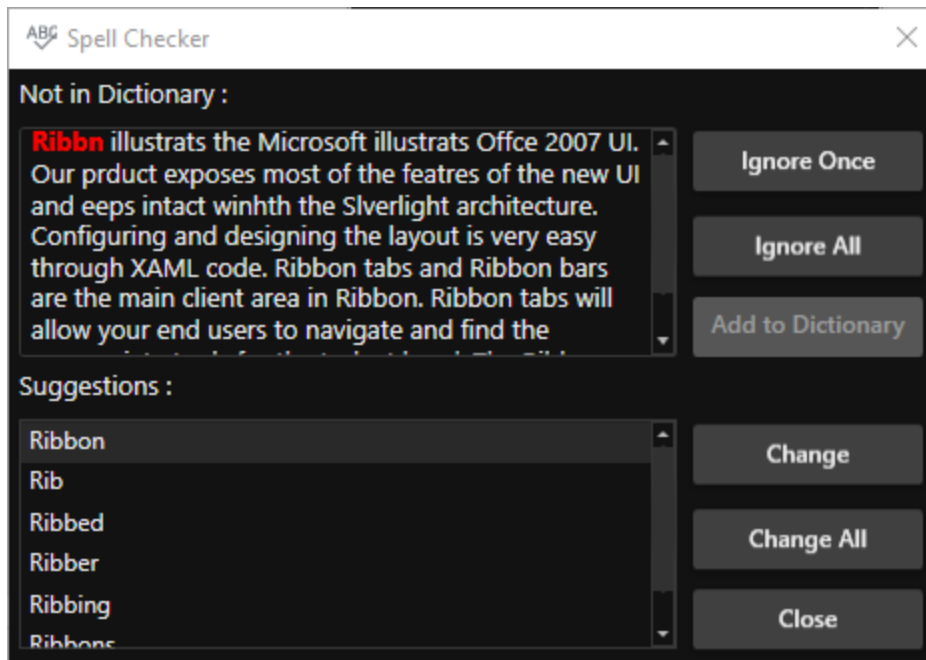
Appearance in WPF SpellChecker (SfSpellChecker)

This section explains different theming options available in [SfSpellChecker](#) control.

Theme

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

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



SplitButton

WPF Split Button Overview

The split button is a combination of a button and a menu control. The button itself provides a default selection or when the arrow is clicked, displays a dropdown list for other possible selections.

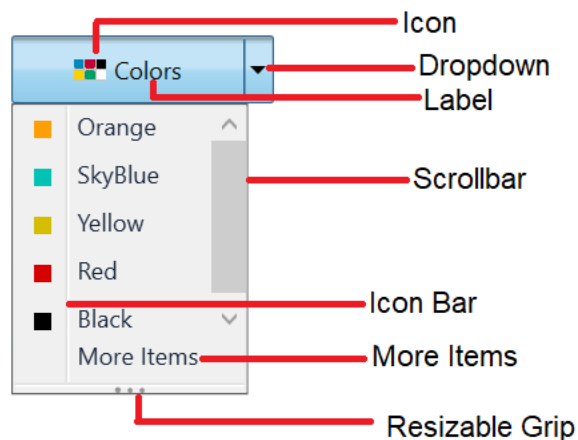
Key features

- **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 Split Button

This section provides an overview of how to work with WPF Split Button control. It describes the control structure, the control initialization and the image setting for the control and add items to the control.

Control structure



Assembly deployment

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

Creating simple application with SplitButton

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

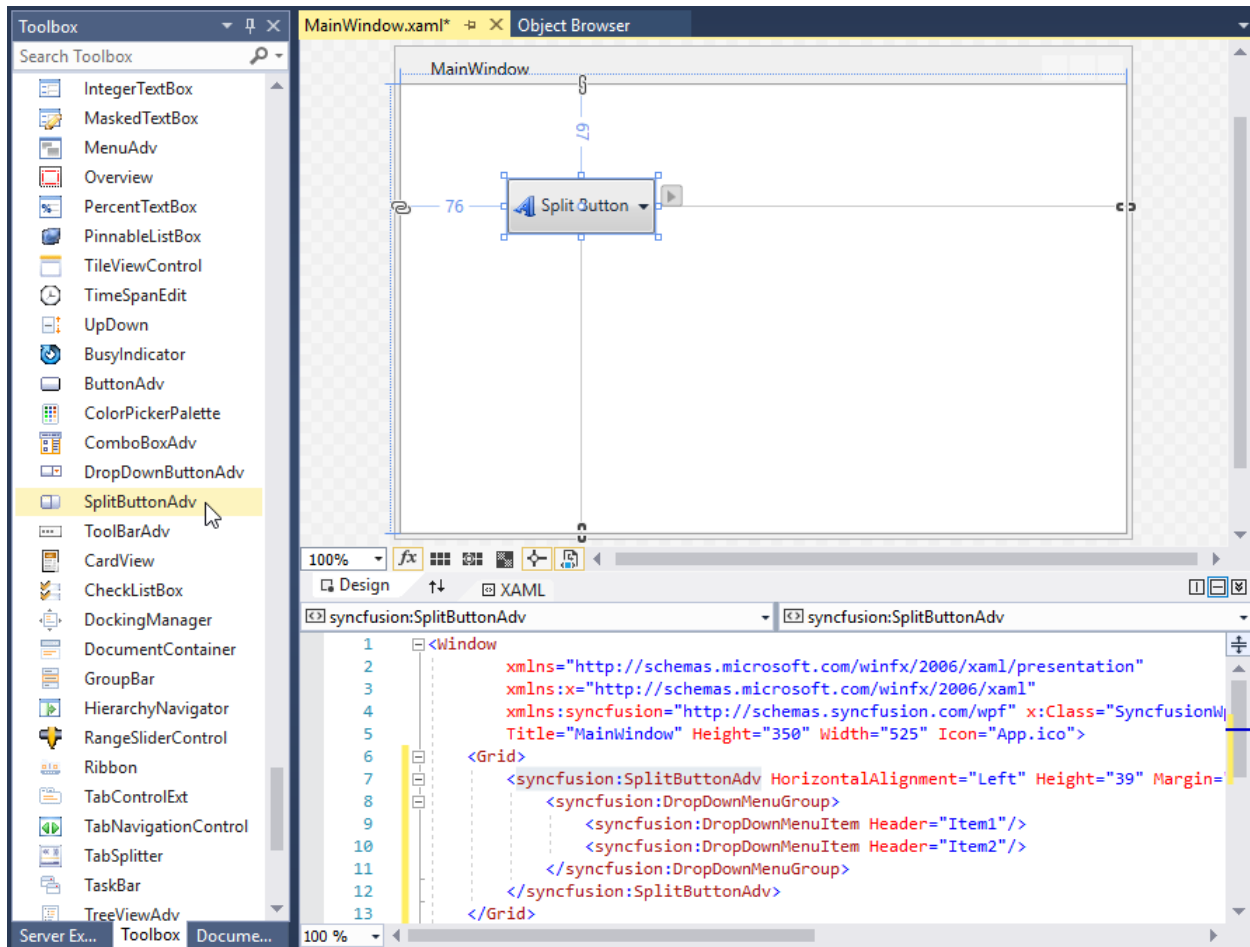
1. [Adding control via designer](#)
2. [Adding control manually in XAML](#)
3. [Adding control manually in C#](#)

Adding control via designer

WPF Split Button control can be added to the application by dragging **SplitButtonAdv** 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:SplitButtonAdv x:Name="splitButtonAdv" Label="Split 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 SplitButtonAdv control in XAML page.

XAML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:SplitButtonadv_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:SplitButtonAdv Height="44" VerticalAlignment="Center"
HorizontalAlignment="Center" Width="162"/>
</Grid>
</Window>
```

```
</Grid>
</Window>
```

Adding control manually in C#

In order to add control manually in C#, do the below steps.

1. Add the below required assembly references to the project.
 - Syncfusion.Shared.WPF
2. Import the `Syncfusion.Windows.Tools.Controls` namespace.
3. Create `SplitButtonAdv` 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:SplitButtonadv_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 ButtonSample
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SplitButtonAdv splitButtonAdv = new SplitButtonAdv();
            splitButtonAdv.Height=44;
            splitButtonAdv.Width=31;
            Root.Children.Add(splitButtonAdv);
        }
    }
}
```

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.

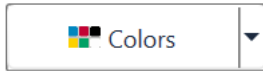
XML

```
<syncfusion:SplitButtonAdv Label="Colors" SmallIcon="Images\color.png"/>
```

C#

```
SplitButtonAdv button = new SplitButtonAdv();
```

```
button.Label = "Colors";  
button.SmallIcon = new BitmapImage(new Uri("Images\colors.png",  
UriKind.RelativeOrAbsolute));
```



Setting size mode

Size mode is used to render Split 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:SplitButtonAdv SizeMode="Small" SmallIcon="Images\color.png"  
Label="Colors"/>
```

C#

```
SplitButtonAdv button = new SplitButtonAdv();  
button.Label = "Colors";  
button.SizeMode = SizeMode.Small;  
button.SmallIcon = new BitmapImage(new Uri("Images\colors.png",  
UriKind.RelativeOrAbsolute));
```



Normal mode

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

XML

```
<syncfusion:SplitButtonAdv SizeMode="Normal" SmallIcon="Images\color.png"  
Label="Colors"/>
```

C#

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



Large mode

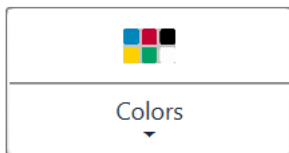
In a large size button, a large image along with the text at the bottom will be displayed.

XML

```
<syncfusion:SplitButtonAdv SizeMode="Large" LargeIcon="Images\color.png"
Label="Colors"/>
```

C#

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

XML

```
<Window x:Class="SplitButtonAdv_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:SplitButtonAdv_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="M11.403995,24.319994C12.395995,24.319994 13.199994,25.127996
13.199994,26.124 13.199994,27.121004 12.395995,27.928007 11.403995,27.928007
10.412996,27.928007 9.6089972,27.121004 9.6089972,26.124 9.6089972,25.127996
10.412996,24.319994 11.403995,24.319994z
M17.906996,23.344005C18.898996,23.344005 19.702995,24.152004
19.702995,25.148004 19.702995,26.144005 18.898996,26.952003
17.906996,26.952003 16.915996,26.952003 16.111997,26.144005
16.111997,25.148004 16.111997,24.152004 16.915996,23.344005
17.906996,23.344005z M6.6579994,19.755001C7.6499986,19.755001
8.4539975,20.563002 8.4539975,21.559 8.4539975,22.555 7.6499986,23.362999
6.6579994,23.362999 5.6670007,23.362999 4.8630012,22.555 4.8630012,21.559
```



```

4.8630012,20.563002 5.6670007,19.755001 6.6579994,19.755001z
M22.225996,18.761007C23.217995,18.761007 24.021994,19.568006
24.021994,20.565006 24.021994,21.561005 23.217995,22.369005
22.225996,22.369005 21.234997,22.369005 20.430998,21.561005
20.430998,20.565006 20.430998,19.568006 21.234997,18.761007
22.225996,18.761007z M22.123995,12.392005C23.115993,12.392005
23.919992,13.199004 23.919992,14.196004 23.919992,15.192003
23.115993,16.000004 22.123995,16.000004 21.132995,16.000004
20.328997,15.192003 20.328997,14.196004 20.328997,13.199004
21.132995,12.392005 22.123995,12.392005z
M14.34922,4.7026652C15.503031,4.733209 16.70268,5.8161396
17.219744,7.4063476 17.829737,9.2813538 17.257744,11.15236
15.943761,11.583361 14.628778,12.015363 13.068799,10.844358
12.458807,8.9693526 11.849814,7.0933465 12.420807,5.2233409
13.735789,4.7913393 13.900037,4.7374643 14.068128,4.7086047
14.237728,4.7030128 14.274828,4.7017897 14.312,4.7016796 14.34922,4.7026652z
M11.384993,2.000007C10.416991,2.0000073,9.6289967,2.7939981,9.6289967,3.7699
956L9.6289967,10.694005C9.6289967,13.375005 7.4559931,15.556995
4.7850031,15.556995 3.2369992,15.556995 1.9999997,16.800999
1.9999994,18.330997 1.9999997,24.765001 7.2070001,30
13.608001,30L14.014998,30C20.736997,30,26.205,24.502993,26.205,17.746006L26.
205,16.897007C26.205,8.6830043,19.557005,2.0000073,11.384993,2.000007z
M11.384993,0C20.660002,0,28.205,7.5800074,28.205,16.897007L28.205,17.746006C
28.205,25.606005,21.839994,32,14.014998,32L13.608001,32C6.1049951,32 -
1.6119111E-07,25.867998 0,18.330997 -1.6119111E-07,15.698002
2.1340023,13.556996 4.7559961,13.556996 6.3529962,13.556996
7.6289972,12.272008
7.6289972,10.694005L7.6289972,3.7699956C7.6289972,1.6910015,9.3139943,0,11.3
84993,0z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="normalIconTemplate">
<Grid Width="16" Height="16">
<Path
Data="M11.403995,24.319994C12.395995,24.319994 13.199994,25.127996
13.199994,26.124 13.199994,27.121004 12.395995,27.928007 11.403995,27.928007
10.412996,27.928007 9.6089972,27.121004 9.6089972,26.124 9.6089972,25.127996
10.412996,24.319994 11.403995,24.319994z
M17.906996,23.344005C18.898996,23.344005 19.702995,24.152004
19.702995,25.148004 19.702995,26.144005 18.898996,26.952003
17.906996,26.952003 16.915996,26.952003 16.111997,26.144005
16.111997,25.148004 16.111997,24.152004 16.915996,23.344005
17.906996,23.344005z M6.6579994,19.755001C7.6499986,19.755001
8.4539975,20.563002 8.4539975,21.559 8.4539975,22.555 7.6499986,23.362999
6.6579994,23.362999 5.6670007,23.362999 4.8630012,22.555 4.8630012,21.559
4.8630012,20.563002 5.6670007,19.755001 6.6579994,19.755001z
M22.225996,18.761007C23.217995,18.761007 24.021994,19.568006
24.021994,20.565006 24.021994,21.561005 23.217995,22.369005
22.225996,22.369005 21.234997,22.369005 20.430998,21.561005
20.430998,20.565006 20.430998,19.568006 21.234997,18.761007
22.225996,18.761007z M22.123995,12.392005C23.115993,12.392005
23.919992,13.199004 23.919992,14.196004 23.919992,15.192003
23.115993,16.000004 22.123995,16.000004 21.132995,16.000004
20.328997,15.192003 20.328997,14.196004 20.328997,13.199004
21.132995,12.392005 22.123995,12.392005z

```

```

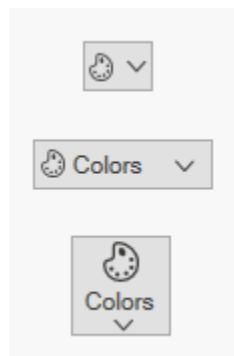
M14.34922,4.7026652C15.503031,4.733209 16.70268,5.8161396
17.219744,7.4063476 17.829737,9.2813538 17.257744,11.15236
15.943761,11.583361 14.628778,12.015363 13.068799,10.844358
12.458807,8.9693526 11.849814,7.0933465 12.420807,5.2233409
13.735789,4.7913393 13.900037,4.7374643 14.068128,4.7086047
14.237728,4.7030128 14.274828,4.7017897 14.312,4.7016796 14.34922,4.7026652z
M11.384993,2.000007C10.416991,2.0000073,9.6289967,2.7939981,9.6289967,3.7699
956L9.6289967,10.694005C9.6289967,13.375005 7.4559931,15.556995
4.7850031,15.556995 3.2369992,15.556995 1.9999997,16.800999
1.9999994,18.330997 1.9999997,24.765001 7.2070001,30
13.608001,30L14.014998,30C20.736997,30,26.205,24.502993,26.205,17.746006L26.
205,16.897007C26.205,8.6830043,19.557005,2.0000073,11.384993,2.000007z
M11.384993,0C20.660002,0,28.205,7.5800074,28.205,16.897007L28.205,17.746006C
28.205,25.606005,21.839994,32,14.014998,32L13.608001,32C6.1049951,32 -
1.6119111E-07,25.867998 0,18.330997 -1.6119111E-07,15.698002
2.1340023,13.556996 4.7559961,13.556996 6.3529962,13.556996
7.6289972,12.272008
7.6289972,10.694005L7.6289972,3.7699956C7.6289972,1.6910015,9.3139943,0,11.3
84993,0z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="largeIconTemplate">
<Grid Width="16" Height="16">
<Path
Width="16"
Height="16"
Data="M11.403995,24.319994C12.395995,24.319994 13.199994,25.127996
13.199994,26.124 13.199994,27.121004 12.395995,27.928007 11.403995,27.928007
10.412996,27.928007 9.6089972,27.121004 9.6089972,26.124 9.6089972,25.127996
10.412996,24.319994 11.403995,24.319994z
M17.906996,23.344005C18.898996,23.344005 19.702995,24.152004
19.702995,25.148004 19.702995,26.144005 18.898996,26.952003
17.906996,26.952003 16.915996,26.952003 16.111997,26.144005
16.111997,25.148004 16.111997,24.152004 16.915996,23.344005
17.906996,23.344005z M6.6579994,19.755001C7.6499986,19.755001
8.4539975,20.563002 8.4539975,21.559 8.4539975,22.555 7.6499986,23.362999
6.6579994,23.362999 5.6670007,23.362999 4.8630012,22.555 4.8630012,21.559
4.8630012,20.563002 5.6670007,19.755001 6.6579994,19.755001z
M22.225996,18.761007C23.217995,18.761007 24.021994,19.568006
24.021994,20.565006 24.021994,21.561005 23.217995,22.369005
22.225996,22.369005 21.234997,22.369005 20.430998,21.561005
20.430998,20.565006 20.430998,19.568006 21.234997,18.761007
22.225996,18.761007z M22.123995,12.392005C23.115993,12.392005
23.919992,13.199004 23.919992,14.196004 23.919992,15.192003
23.115993,16.000004 22.123995,16.000004 21.132995,16.000004
20.328997,15.192003 20.328997,14.196004 20.328997,13.199004
21.132995,12.392005 22.123995,12.392005z
M14.34922,4.7026652C15.503031,4.733209 16.70268,5.8161396
17.219744,7.4063476 17.829737,9.2813538 17.257744,11.15236
15.943761,11.583361 14.628778,12.015363 13.068799,10.844358
12.458807,8.9693526 11.849814,7.0933465 12.420807,5.2233409
13.735789,4.7913393 13.900037,4.7374643 14.068128,4.7086047
14.237728,4.7030128 14.274828,4.7017897 14.312,4.7016796 14.34922,4.7026652z
M11.384993,2.000007C10.416991,2.0000073,9.6289967,2.7939981,9.6289967,3.7699
956L9.6289967,10.694005C9.6289967,13.375005 7.4559931,15.556995

```

```

4.7850031,15.556995 3.2369992,15.556995 1.9999997,16.800999
1.9999994,18.330997 1.9999997,24.765001 7.2070001,30
13.608001,30L14.014998,30C20.736997,30,26.205,24.502993,26.205,17.746006L26.
205,16.897007C26.205,8.6830043,19.557005,2.0000073,11.384993,2.000007z
M11.384993,0C20.660002,0,28.205,7.5800074,28.205,16.897007L28.205,17.746006C
28.205,25.606005,21.839994,32,14.014998,32L13.608001,32C6.1049951,32 -
1.6119111E-07,25.867998 0,18.330997 -1.6119111E-07,15.698002
2.1340023,13.556996 4.7559961,13.556996 6.3529962,13.556996
7.6289972,12.272008
7.6289972,10.694005L7.6289972,3.7699956C7.6289972,1.6910015,9.3139943,0,11.3
84993,0z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
</Window.Resources>
<StackPanel>
<syncfusion:SplitButtonAdv x:Name="smallIcon" SizeMode="Small" Label="Login"
HorizontalAlignment="Center" VerticalAlignment="Center" Margin="10"
IconTemplate="{StaticResource smallIconTemplate}">
</syncfusion:SplitButtonAdv>
<syncfusion:SplitButtonAdv x:Name="normalIcon" SizeMode="Normal"
Label="Login" HorizontalAlignment="Center" VerticalAlignment="Center"
Margin="10" IconTemplate="{StaticResource normalIconTemplate}">
</syncfusion:SplitButtonAdv>
<syncfusion:SplitButtonAdv x:Name="largeIcon" SizeMode="Large" Label="Login"
HorizontalAlignment="Center" VerticalAlignment="Center" Margin="10"
IconTemplate="{StaticResource largeIconTemplate}">
</syncfusion:SplitButtonAdv>
</StackPanel>
</Window>

```



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

- [IconTemplate](#)
- [LargeIcon](#)
- [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_SplitButtonAdv.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_SplitButtonAdv"
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:SplitButtonAdv 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 [SplitButtonAdv](#) loads the icon in the following priority order.

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

Setting image

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

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

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

XML

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

C#

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



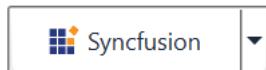
The **SmallIcon** property can be set even when the sizeMode is **Normal**.

XML

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

C#

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



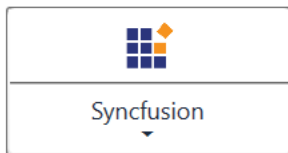
The **LargeIcon** property can be set as follows:

XML

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

C#

```
SplitButtonAdv button = new SplitButtonAdv();
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.

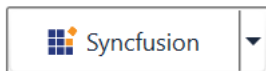
XML

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

C#

```
SplitButtonAdv splitbutton = new SplitButtonAdv();
```

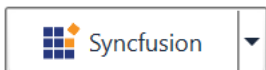
```
splitbutton.Label = "Syncfusion";
splitbutton.IconWidth=20;
splitbutton.IconHeight=20;
splitbutton.SmallIcon = new BitmapImage(new Uri("Images\\syncfusion.png",
UriKind.RelativeOrAbsolute));
```

**XML**

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

C#

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



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

IsDefault mode

The [IsDefault](#) property indicates whether the SplitButtonAdv is a Default button and is used to activate the SplitButtonAdv by pressing using Enter key. When setting the IsDefault property to true, the user can invoke the button by pressing the **Enter** key.

XML

```
<syncfusion:SplitButtonAdv x:Name="defaultButton" Label="Default"
Grid.Column="1" Grid.Row="1" VerticalAlignment="Top"
HorizontalAlignment="Center" Click="SplitButtonAdv_Click" IsDefault="True"
/>
```

Adding items to Split Button

The DropDownMenuGroup acts as a container for the Split 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 Split Button please refer to the topics [Data Binding](#) and [Command Binding](#).

XML

```
<Window x:Class="Split_Button_Menuitem_Binding.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

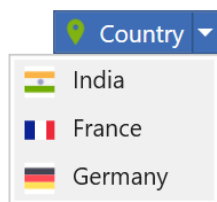
```
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:DropDown_Button_Menuitem_Binding"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:CountryViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:SplitButtonAdv Label="Country" SmallIcon="Images/flagsmall.png"
>
<syncfusion:DropDownMenuGroup ItemsSource="{Binding DropDownItems}">
<syncfusion:DropDownMenuGroup.ItemTemplate>
<DataTemplate>
<syncfusion:DropDownMenuItem Header="{Binding Name}"
HorizontalAlignment="Left">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="{Binding Flag}"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
</DataTemplate>
</syncfusion:DropDownMenuGroup.ItemTemplate>
</syncfusion:DropDownMenuGroup>
</syncfusion:SplitButtonAdv>
</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))  
        });  
    }  
}
```

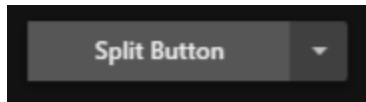


Note: View [sample](#) in GitHub.

Theme

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

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



Data Binding in WPF Split Button

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:SplitButtonAdv 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:SplitButtonAdv>

```

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="Split_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:Split_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:SplitButtonAdv x:Name="splitButton" 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
splitButton}}"
CommandParameter="{Binding .}">
                            <syncfusion:DropDownMenuItem.Icon>
                                <Image Source="{Binding Flag}" />
                            </syncfusion:DropDownMenuItem.Icon>
                        </syncfusion:DropDownMenuItem>
                    </DataTemplate>
                </syncfusion:DropDownMenuGroup.ItemTemplate>
            </syncfusion:DropDownMenuGroup>
        </syncfusion:SplitButtonAdv>
    </Grid>
</Window>
```

C#

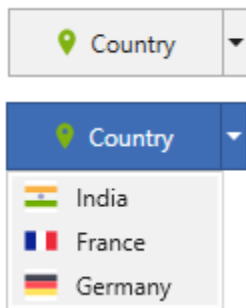
```
public class DelegateCommand<T> : ICommand
{
    private Predicate<T> _canExecute;
    private Action<T> _method;
    bool _canExecuteCache = true;
    /// <summary>
    /// Initializes a new instance of the <see cref="DelegateCommand"/> class.
    /// </summary>
    /// <param name="method">The 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 Split Button

The command and command parameter properties allow to execute any action on clicking either the button or 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="Split_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:SplitViewModel/>
</Window.DataContext>
<Grid VerticalAlignment="Center" HorizontalAlignment="Left">
<Grid.RowDefinitions>
<RowDefinition Height="30"/>
<RowDefinition Height="30"/>
<RowDefinition Height="*/>
</Grid.RowDefinitions>
<CheckBox IsChecked="{Binding CanPerformAction}" Grid.Row="0" Content="Can
perform action in split button"/>
<CheckBox IsChecked="{Binding CanPerformActionItem}" Grid.Row="1"
Content="Can perform action in drop down items"/>
<syncfusion:SplitButtonAdv Label="Country" SizeMode="Large"
LargeIcon="Images\flaglarge.png" Command="{Binding ClickCommand}"
CommandParameter="Action completed" Grid.Row="2" Height="72" Width="122">
<syncfusion:DropDownMenuGroup>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="India"
Command="{Binding DropDownCommand}" CommandParameter="India">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/india.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="France"
Command="{Binding DropDownCommand}" CommandParameter="France" >
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/france.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Germany"
Command="{Binding DropDownCommand}" CommandParameter="Germany" >
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/germany.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Canada"
Command="{Binding DropDownCommand}" CommandParameter="Canada" >
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/Canada.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="China"
Command="{Binding DropDownCommand}" CommandParameter="China" >
<syncfusion:DropDownMenuItem.Icon>
<Image Source="Images/china.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
</syncfusion:DropDownMenuGroup >
</syncfusion:SplitButtonAdv>
</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 SplitButtonAdv control.

Dropdown Menu Items in WPF Split Button

Setting icon for dropdown menu items

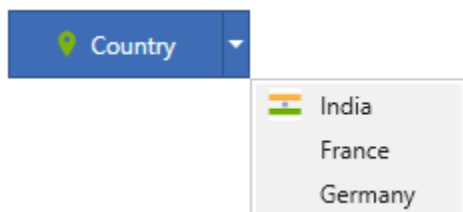
The icon option helps to provide pictorial representation of the dropdown menu item. One can apply the icon by setting the [Icon](#) property value to an image source.

XML

```
<syncfusion:SplitButtonAdv Label="Country" x:Name="splitbutton"
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:SplitButtonAdv>
```

C#

```
SplitButtonAdv splitbutton = new SplitButtonAdv();
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",
HorizontalAlignment="Left" };
DropDownMenuItem Item3 = new DropDownMenuItem() { Header ="Germany",
HorizontalAlignment="Left" };
menu.Items.Add(Item1);
menu.Items.Add(Item2);
menu.Items.Add(Item3);
splitbutton.Content = menu;
splitbutton.Label = "Country";
splitbutton.DropDirection = DropDirection.BottomRight;
splitbutton.SizeMode = SizeMode.Normal;
splitbutton.SmallIcon = new BitmapImage(new Uri("Images\country.png"));
```



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**.

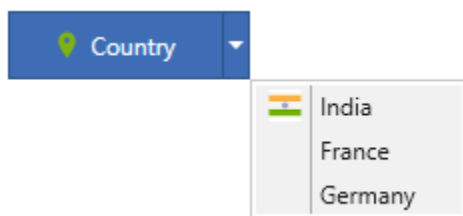
Note: The default value of [IconBarEnabled](#) is **false**.

XML

```
<syncfusion:SplitButtonAdv Label="Country" DropDirection="BottomRight"
x:Name="splitbutton" 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:SplitButtonAdv>
```

C#

```
SplitButtonAdv splitbutton = new SplitButtonAdv();
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",
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;
splitbutton.Content = menu;
splitbutton.Label = "Country";
splitbutton.SizeMode = SizeMode.Normal;
splitbutton.DropDirection = DropDirection.BottomRight;
splitbutton.SmallIcon = new BitmapImage(new Uri("Images\country.png"));
```



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:SplitButtonAdv Label="Country" DropDirection="BottomRight"
x:Name="splitbutton" 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="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: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:SplitButtonAdv>

```

C#

```

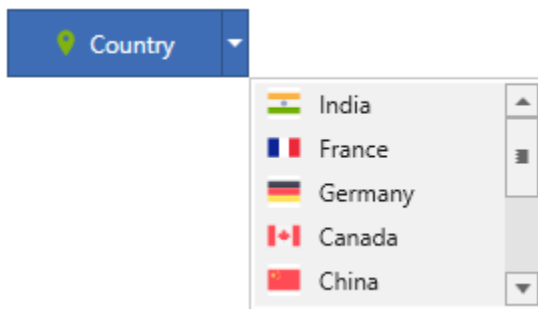
SplitButtonAdv splitbutton = new SplitButtonAdv();
DropDownMenuGroup menu = new DropDownMenuGroup();
DropDownMenuItem Item1 = new DropDownMenuItem() { Header="India",Icon=new
BitmapImage(new Uri("Images\india.png")), HorizontalAlignment="Left"};
DropDownMenuItem Item2 = new DropDownMenuItem() { Header ="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"};
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;
splitbutton.Content = menu;
splitbutton.Label = "Country";
splitbutton.SizeMode = SizeMode.Normal;
splitbutton.DropDirection = DropDirection.BottomRight;
splitbutton.SmallIcon = new BitmapImage(new Uri("Images\country.png"));

```



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:SplitButtonAdv Label="Country" DropDirection="BottomRight"
x:Name="splitbutton" 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:SplitButtonAdv>

```

C#

```

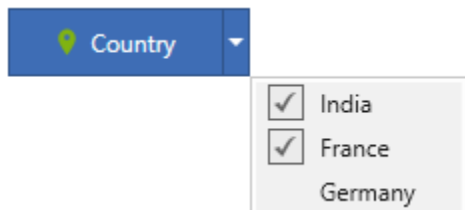
SplitButtonAdv splitbutton = new SplitButtonAdv();
DropDownMenuGroup menu = new DropDownMenuGroup();
DropDownMenuItem Item1 = new DropDownMenuItem() { Header="India",
IsChecked=true, IsCheckable=true, HorizontalAlignment="Left"};
DropDownMenuItem Item2 = new DropDownMenuItem() { Header ="France",
IsChecked=true, IsCheckable=true, HorizontalAlignment="Left"};
DropDownMenuItem Item3 = new DropDownMenuItem() { Header ="Germany",
HorizontalAlignment="Left"};
menu.Items.Add(Item1);

```

```

menu.Items.Add(Item2);
menu.Items.Add(Item3);
splitbutton.Content = menu;
splitbutton.DropDirection = DropDirection.BottomRight;
splitbutton.Label = "Country";
splitbutton.SizeMode = SizeMode.Normal;
splitbutton.SmallIcon = new BitmapImage(new Uri("Images\country.png"));

```



Resizing dropdown menu

The dropdown menu group popup 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:SplitButtonAdv Label="Country" x:Name="splitbutton"
    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:SplitButtonAdv>

```

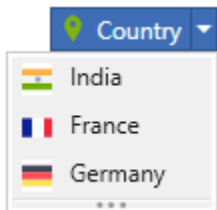
C#

```

SplitButtonAdv splitbutton = new SplitButtonAdv();
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"};
menu.Items.Add(Item1);

```

```
menu.Items.Add(Item2);
menu.Items.Add(Item3);
menu.IsResizable = true;
splitbutton.Content = menu;
splitbutton.Label = "Colors";
splitbutton.SmallIcon = new BitmapImage(new Uri("images\country.png"));
```



Note: View [sample](#) in GitHub. This sample showcases how to set the drop-down item icon, icon bar visibility, scrollbar visibility, and checkable support.

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

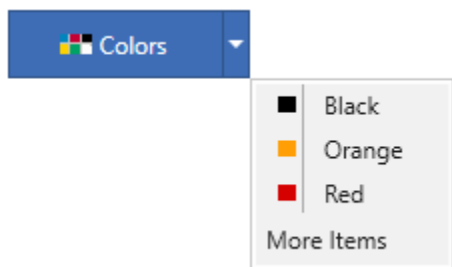
```
<Window x:Class="SplitButton_Custom_Items.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:SplitButton_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:SplitButtonAdv Label="Colors" SizeMode="Normal"
SmallIcon="Images\colors.png">
<syncfusion:DropDownMenuGroup IconBarEnabled="True" MoreItems="{Binding
Items}" 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:SplitButtonAdv>
</Grid>
</Window>
```

C#

```
using Syncfusion.Windows.Shared;
using System;
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Controls;
namespace SplitButton_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="SplitButton_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:SplitButton_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:SplitButtonAdv Label="Colors" x:Name="splitbutton"
SizeMode="Normal" SmallIcon="Images\colors.png">
<syncfusion:DropDownMenuGroup IconBarEnabled="True" MoreItems="{Binding
Colors}" IsMoreItemsIconTrayEnabled="True">
<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:SplitButtonAdv>
</Grid>
</Window>

```

C#

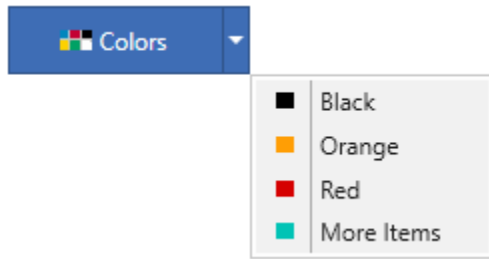
```

using Syncfusion.Windows.Shared;
using Syncfusion.Windows.Tools.Controls;
using System;
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Media.Imaging;
namespace SplitButton_Custom_Items
{
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 split button control.

Dropdown Direction in WPF Split Button

Dropdown direction is used to change the position of the popup being loaded while pressing 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**.

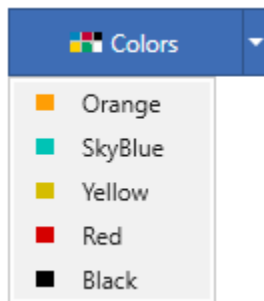
XML

```
<syncfusion:SplitButtonAdv DropDirection="BottomLeft"
    SmallIcon="images\colors.png" Label="Colors">
    <syncfusion:DropDownMenuGroup>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Orange">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="images\orange.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="SkyBlue">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="images\skyblue.png"/>
    </syncfusion:DropDownMenuItem.Icon>
    </syncfusion:DropDownMenuItem>
    <syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Yellow">
    <syncfusion:DropDownMenuItem.Icon>
    <Image Source="images\yellow.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:SplitButtonAdv>
```

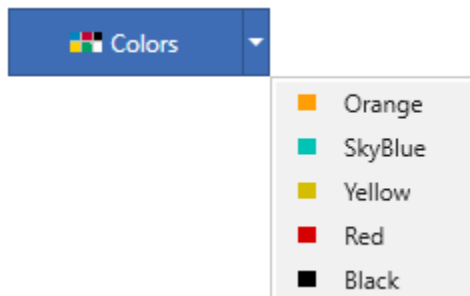
```
<syncfusion:DropDownMenuItem HorizontalAlignment="Left" Header="Black">
<syncfusion:DropDownMenuItem.Icon>
<Image Source="images\black.png"/>
</syncfusion:DropDownMenuItem.Icon>
</syncfusion:DropDownMenuItem>
</syncfusion:DropDownMenuGroup>
</syncfusion:SplitButtonAdv>
```

C#

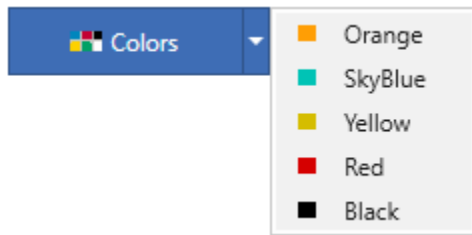
```
SplitButtonAdv splitbutton = new SplitButtonAdv();
splitbutton.Label = "Colors";
splitbutton.DropDirection = DropDirection.BottomLeft;
splitbutton.SmallIcon = new BitmapImage(new Uri("images\colors.png"));
DropDownMenuGroup menu = new DropDownMenuGroup();
DropDownMenuItem Item1 = new DropDownMenuItem() { Header="Orange", Icon=new
BitmapImage(new Uri("images\orange.png")), HorizontalAlignment="Left"};
DropDownMenuItem Item2 = new DropDownMenuItem() { Header = "Skyblue",
Icon=new BitmapImage(new Uri("images\skyblue.png")),
HorizontalAlignment="Left"};
DropDownMenuItem Item3 = new DropDownMenuItem() { Header = "Yellow", Icon=new
BitmapImage(new Uri("images\yellow.png")), HorizontalAlignment="Left"};
DropDownMenuItem Item4 = new DropDownMenuItem() { Header = "Red", Icon=new
BitmapImage(new Uri("images\red.png")), HorizontalAlignment="Left"};
DropDownMenuItem Item5 = new DropDownMenuItem() { Header = "Black", Icon=new
BitmapImage(new Uri("images\black.png")), HorizontalAlignment="Left"};
menu.Items.Add(Item1);
menu.Items.Add(Item2);
menu.Items.Add(Item3);
menu.Items.Add(Item4);
menu.Items.Add(Item5);
splitbutton.Content=menu;
```



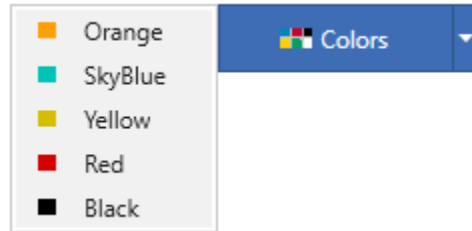
Drop Direction - BottomLeft



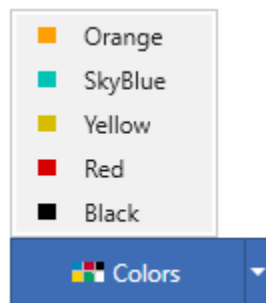
Drop Direction - BottomRight



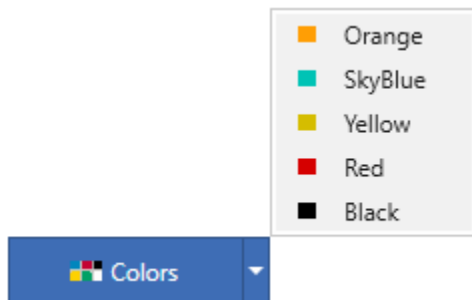
Drop Direction - Right



Drop Direction - Left



Drop Direction - TopLeft



Drop Direction - TopRight

Multiline Text in WPF Split Button (SplitButtonAdv)

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

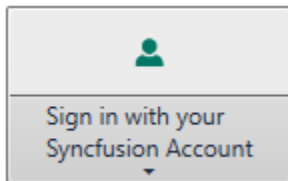
Note: This property is only applicable for large size mode of the Split Button.

XML

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

C#

```
SplitButtonAdv splitbutton = new SplitButtonAdv();
splitbutton.Label = "Sign in with your Syncfusion Account";
splitbutton.IsMultiLine = true;
splitbutton.SizeMode = SizeMode.Large;
splitbutton.SmallIcon = new BitmapImage(new Uri("image\\userlarge.png"));
```

**Events in WPF Split Button**

The Split Button 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:SplitButtonAdv x:Name="splitbutton"
    DropDownOpening="splitbutton_DropDownOpening"/>
```

C#

```
SplitButtonAdv splitbutton = new SplitButtonAdv();
splitbutton.DropDownOpening += new
CancelEventHandler(splitbutton_DropDownOpening);
private void splitbutton_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:SplitButtonAdv x:Name="splitbutton"
    DropDownOpened="splitbutton_DropDownOpened"/>
```

C#

```
SplitButtonAdv splitbutton = new SplitButtonAdv();
splitbutton.DropDownOpened += new
RoutedEventHandler(splitbutton_DropDownOpened);
private void splitbutton_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:SplitButtonAdv x:Name="splitbutton"  
    DropDownClosing="splitbutton_DropDownClosing"/>
```

C#

```
SplitButtonAdv splitbutton = new SplitButtonAdv();  
splitbutton.DropDownClosing += new  
    CancelEventHandler(splitbutton_DropDownClosing);  
private void splitbutton_DropDownClosing(object sender,  
    System.ComponentModel.CancelEventArgs e)  
{  
}
```

DropDownClosed

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

XML

```
<syncfusion:SplitButtonAdv x:Name="splitbutton"  
    DropDownClosed="splitbutton_DropDownClosed"/>
```

C#

```
SplitButtonAdv splitbutton = new SplitButtonAdv();  
splitbutton.DropDownClosed += new  
    RoutedEventHandler(splitbutton_DropDownClosed);  
private void splitbutton_DropDownClosed(object sender, RoutedEventArgs e)  
{  
}
```

Click

The events occurs when the Split Button control is clicked and any action can be handled in the respective event handler.

XML

```
<syncfusion:SplitButtonAdv x:Name="splitbutton" Click="splitbutton_Click"/>
```

C#

```
SplitButtonAdv splitbutton = new SplitButtonAdv();  
splitbuttonbutton.Click += new RoutedEventHandler(splitbuttonbutton_Click);  
private void splitbuttonbutton_Click(object sender, RoutedEventArgs e)
```

```
{
}
```

Events for dropdown menu items

Click

The events 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 events occur 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 Split Button

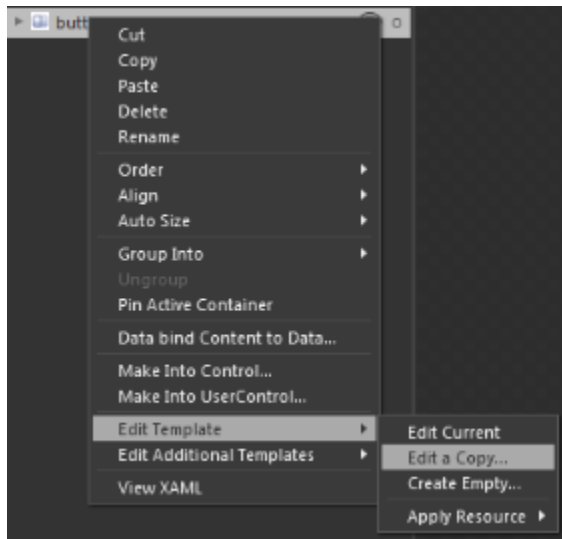
WPF styles and templates is a suite of features that allow developers and designers to create visual compelling effects and consistent appearance of the products.

This document provides information to change the visual appearance of the Split Button control. In addition, one can edit the structure of the Split Button control by using Blend and Visual Studio that helps to customize their appearances.

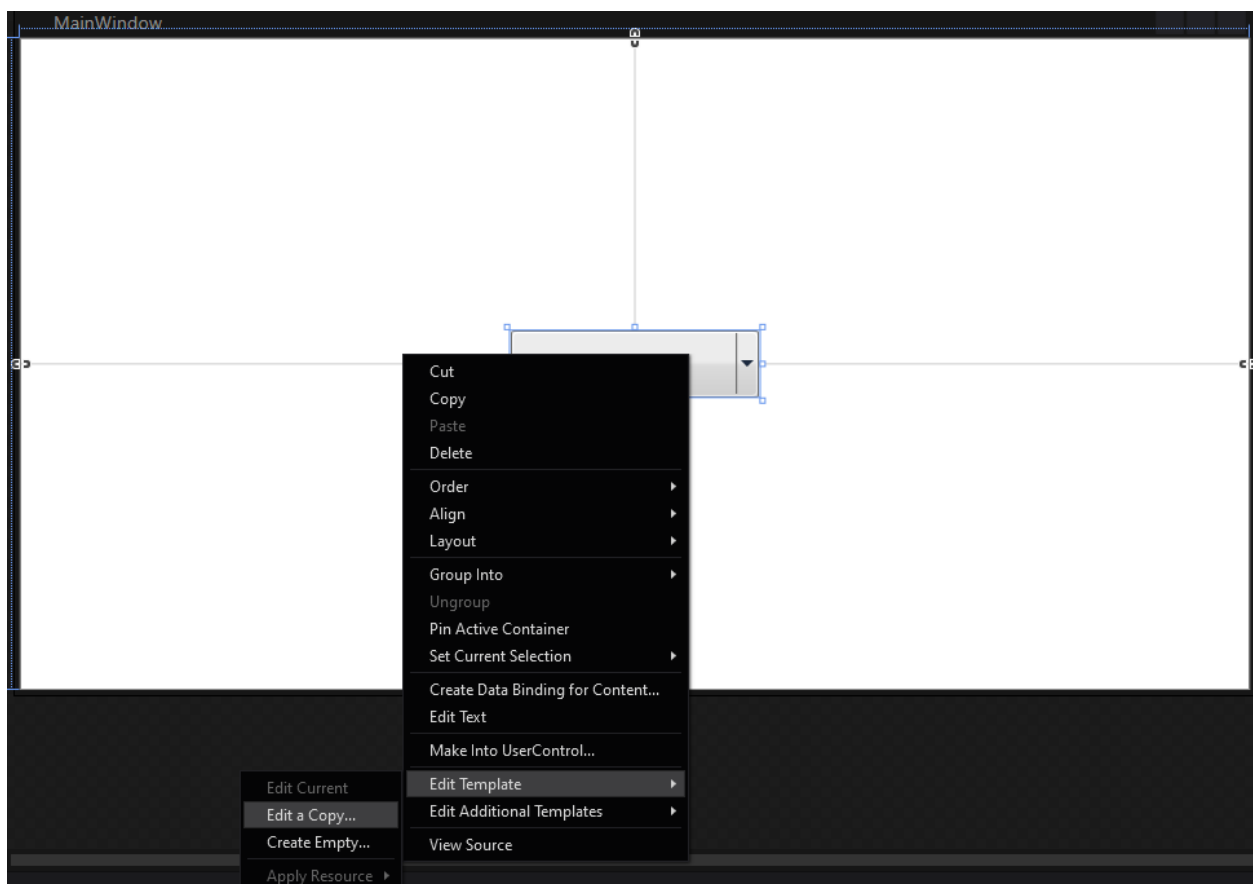
Edit appearance in Expression Blend

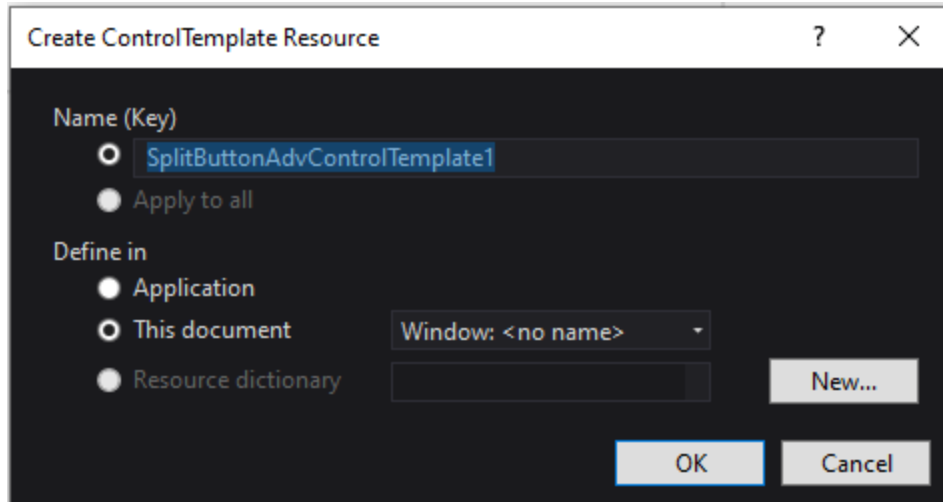
- Open the application in Expression Blend.

- Select the Split Button control from the window.



- Right click on the Split 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.

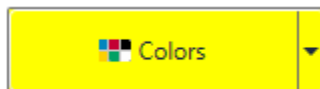




The **Create ControlTemplate Resource** dialog allows to enter the name or change the control template name, as well as choose the location for the template. When **OK** is pressed, the Split Button control template is generated by the Expression Blend in the **Resource** section. The generated XAML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

- **Create Empty...** - Creates an empty Split Button 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.

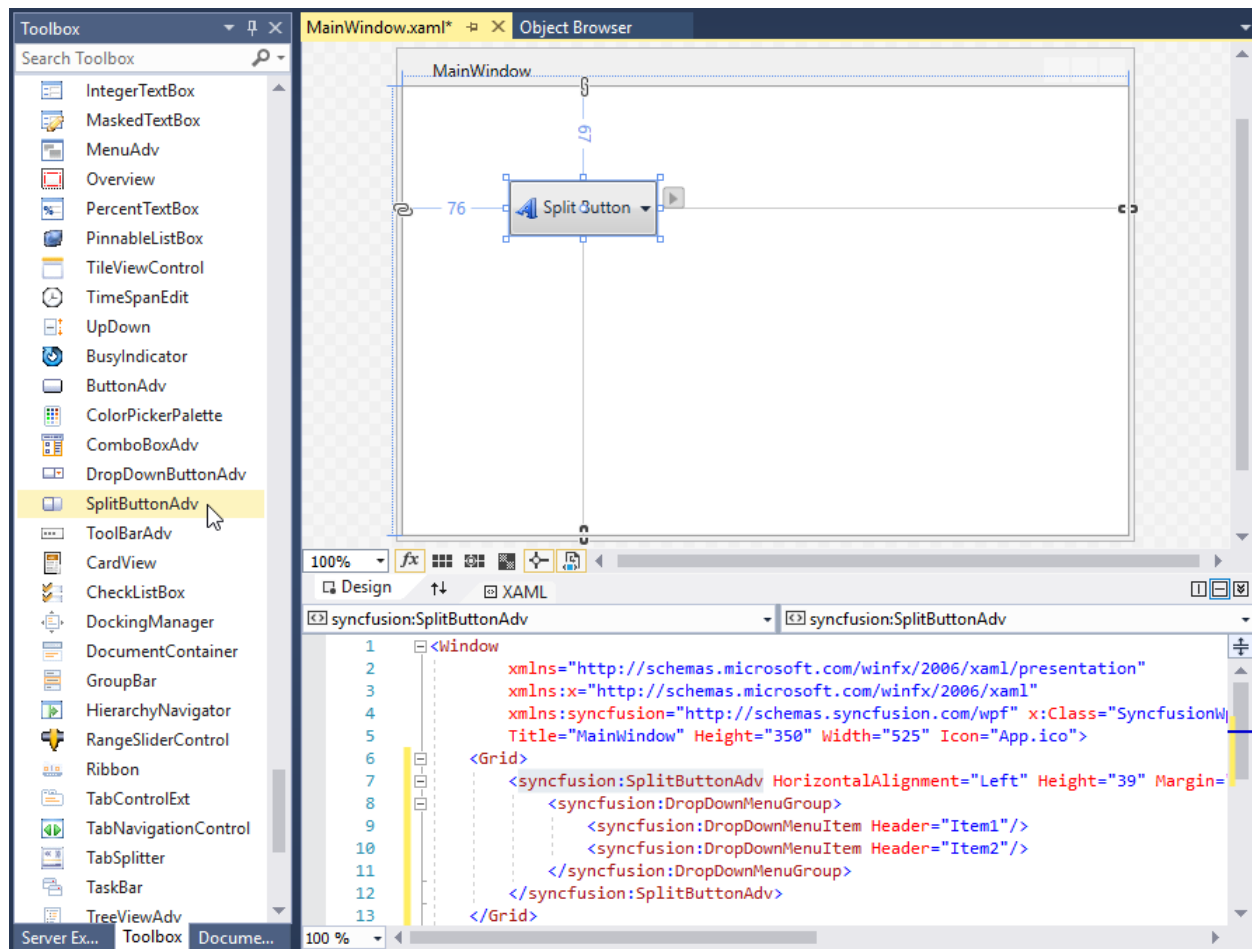


```
<Window.Resources>
  <ControlTemplate x:Key="SplitButtonAdvControlTemplate1" TargetType="{x:Type Syncfusion:SplitButtonAdv}"...>
</Window.Resources>
```

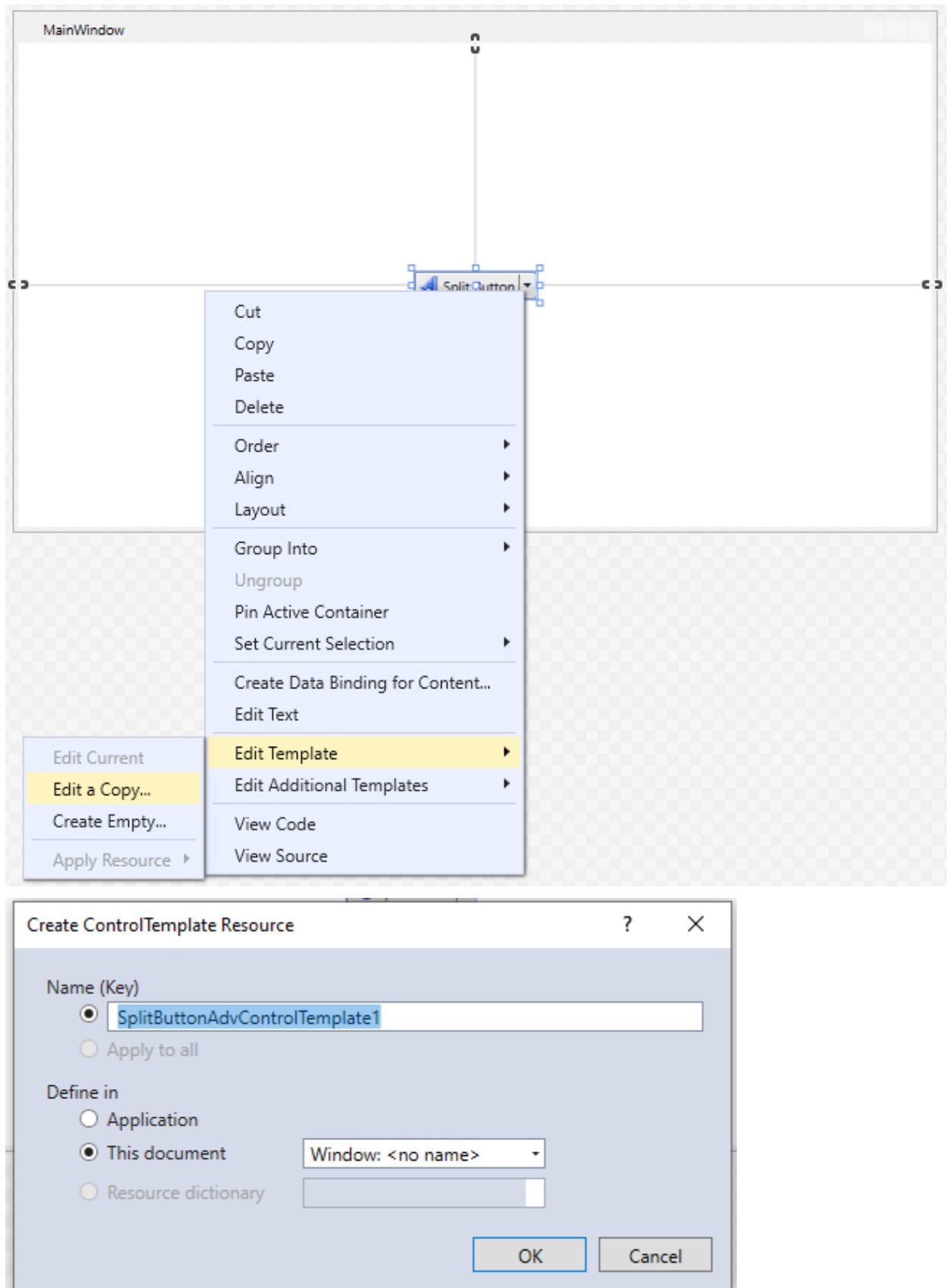
Split Button control edited in Expression Blend

[Edit appearance in Visual Studio](#)

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



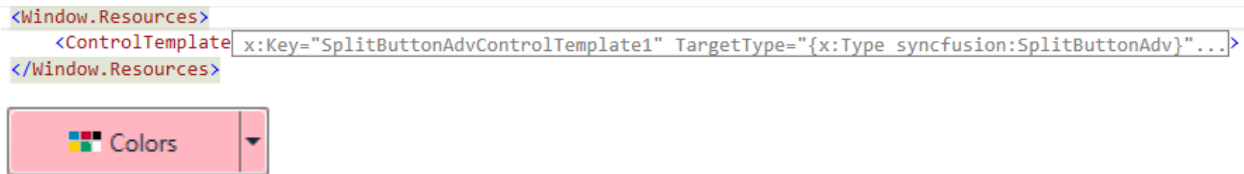
- On choosing menu option **Edit Template**, it further comprise of following two options.
- **Edit a Copy...** – Edit a copy of the default style. When selecting this option, a new dialog opens as follows.



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 Split Button control template is generated in the **Resource** section. The generated XAML can be edited in XAML view.

- **Create Empty...** - Creates an empty Split 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.

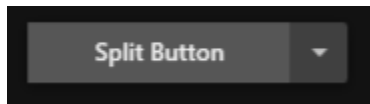


Split Button control edited in Visual Studio

Themes in WPF Split Button

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

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



SfSpreadsheet

WPF Spreadsheet (SfSpreadsheet) Overview

The SfSpreadsheet is excel inspired control that allows you to create, edit, view and format the Microsoft Excel files without excel installed. It provides absolute ease of use UI experience with integrated ribbon to cover any possible business scenario. SfSpreadsheet comes with built-in calculation engine with support for 400+ most widely used formulas.

Key Features

- **Ribbon** – Ribbon integrated with organically enhanced UI experience.
- **Editing and Selection**- Interactive support for editing and cell selection in workbook.
- **Formulas** - Provides support for 400+ most widely used formulas which any business user needs and allows you to add, remove and edit the formulas like in excel.
- **Name Manager** – Supports the name ranges in the formulas. By using the name ranges, you can specify the name of the cell range, and then you can use it in the formula more easily without hassling of remembering cell locations.
- **Data validation** – Provides support to ensure the data integrity by enforcing end users to enter valid data into the cells and if entered data does not meet the specified criteria, and error message is displayed.

- **Floating Cells**- Provides support for floating cell mode that is when the text exceeds the length of the cell, it will float the text to the adjacent cell.
- **Merge Cells** - Merge two or more adjacent cells into a single cell and display the contents of one cell in the merged cell.
- **Conditional Formatting**- Provides support for excel compatible conditional formatting and allows you to apply formats to a cell or range of cells depending on the value of cells or formula that meet specific criteria. Also provides support to define and import the conditional formatting. rules such as Data Bars, Icon Sets and Color Scales options which are used to visualize the data.
- **Cell Comments**- Supports comments that provide additional information about a cell such as what the value represents. And it would be useful if you want the end users to understand the data in the cells more deeply.
- **Hyperlinks and Bookmarks**- Provides support to import, create, and edit hyperlinks and bookmarks. The hyperlink is a convenient way to navigate or browse data within a worksheet or other worksheets in a workbook.
- **Undo/Redo** - Provides support to undo or redo the changes that you have made in the workbook.
- **Clipboard Operations** – Provides support for Cut/Copy/Paste Operations in Spreadsheet.
- **Fill Series** – Provides support to automatically fill cells with data that follows or completes a pattern.
- **Freeze panes** – Provides support to freeze rows/columns.
- **Resizing and Hiding** – Provides interactive support to resize or hide/unhide the rows and columns.
- **Charts, Pictures and Textboxes** - Provides support to import Charts, Pictures and Textboxes.
- **Sparklines** – Provides support to import Sparklines.
- **Outlines** - Provides support to group or ungroup rows and columns.
- **Workbook and Worksheet Protection**- Provides support to protect the worksheet and also supports to lock-down the structure and window of workbook, which enables you to prevent workbook from any structural change or from any change in size.
- **Conversion** – Provides support to export the workbook to PDF, HTML, Image and CSV.
- **Zooming** – Provides support to zoom in and zoom out of the worksheet view.
- **Localization** - Provides support to localize all the static text in a Ribbon and all dialogs to any desired language.
- **Supported file types** - Ability to import the different types of excel which are XLS, XLSX, XLSM, XLT, XLTX, CSV(Comma delimited) respectively.

Choose between SfSpreadsheet and Spreadsheet control

WPF suite contains **SfSpreadsheet** and **Spreadsheet** control for viewing and editing the excel files. SfSpreadsheet is an alternate for Spreadsheet control which is marked as classic control. Hence it is recommended to use SfSpreadsheet which provides better performance and rich set features over Spreadsheet control.

Below are the features that SfSpreadsheet have over Spreadsheet control,

Feature	SfSpreadsheet
Scrolling performance	Supports fast and fluid scrolling even if the excel has a huge set of data .Thus its performance is high compared to Spreadsheet control.

Copy Paste	Supports various paste options similar to excel options like Paste, Formulas, Values, Formula and Source Formatting, Values and Source Formatting and Formatting alone. It also provides a good performance compared to Spreadsheet control.
Undo/Redo	Supports undo/redo functionalities similar to those achieved with Microsoft Office-type applications. This operation records the changes in the whole workbook while Spreadsheet Control records the changes in sheet level only.
Formula calculation	Provides support for 400+ most widely used formulas and uses Multi-threading concept So, the calculation speed is also high compared to Spreadsheet control.
Floating Cells	Provides support to float cell both in display and edit mode.
Hyperlinks	Provides support for Hyperlink feature which you can create hyperlink for existing files or web page and email addresses too.
Conditional Formatting	Provides support to define and import the conditional formatting rules such as Data Bars, Icon Sets and Color Scales options which are used to visualize the data.
Data validation	Provides support for validation for cross sheet references and list validation with formula/cell references compared to Spreadsheet control.

Properties table

Below are the properties difference between SfSpreadsheet and Spreadsheet control,

SfSpreadsheet	Spreadsheet	Description
ActiveGrid	GridProperties->ActiveSpreadsheetGrid	Gets the active SpreadsheetGrid
IsCustomTabItemContextMenuEnabled	TabStyleManager -> IsCustomTabItemContextMenuEnabled	Gets or sets whether Custom ContextMenu is to be Enabled
ShowTabItemContextMenu	TabStyleManager -> ShowTabItemContextMenu	Gets or sets whether TabItemContextMenu is to be displayed
TabItemContextMenu	TabStyleManager ->TabItemContextMenu	Gets or sets the ContextMenu Items for TabItem
ActiveSheet	ExcelProperties->Workbook->ActiveSheet	Gets the Current ActiveSheet

Workbook	ExcelProperties->Workbook	
CurrentCellStyle	GridProperties->CurrentCellStyle	Gets the Style of the Current Cell
HistoryManager	GridProperties.ActiveSpreadsheetGrid.Model.CommandStack	Gets the command stack of the SfSpreadsheet. By default it has been enabled
SheetName	GridProperties->CurrentSheetName	Gets the tab sheet name

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Getting Started with WPF Spreadsheet (SfSpreadsheet)

This section helps you to get started with the SfSpreadsheet

Assemblies Deployment

Below table describes, list of assemblies required to be added in project when the SfSpreadsheet control is used in your application.

Assembly	Description
Syncfusion.SfCellGrid.WPF.dll	Contains the base and fundamental classes which holds the underlying architecture for displaying cells with virtualized behavior and selection/interaction of cells.
Syncfusion.SfGridCommon.WPF.dll	Contains the classes which holds the properties and functions of scroll viewer and disposable elements
Syncfusion.SfSpreadsheet.WPF.dll	Contains the classes that handles all the UI Operations of SfSpreadsheet such as importing of sheets, applying formulas/styles etc.
Syncfusion.Shared.WPF.dll	Contains the classes which holds the controls like Color pickers, Chromeless window, ComboBoxAdv, DateTimeEdit etc.
Syncfusion.Tools.WPF.dll	Contains the classes which holds the controls like TabControlExt, TabItemExt, Gallery, GroupBar, TabSplitter etc which are used in SfSpreadsheet
Syncfusion.XlsIO.Base.dll	Contains the base classes which is responsible for read and write in Excel files, Worksheet Manipulations, Formula calculations etc.

Below are the assemblies list that can be added when you want to enable certain features in SfSpreadsheet control.

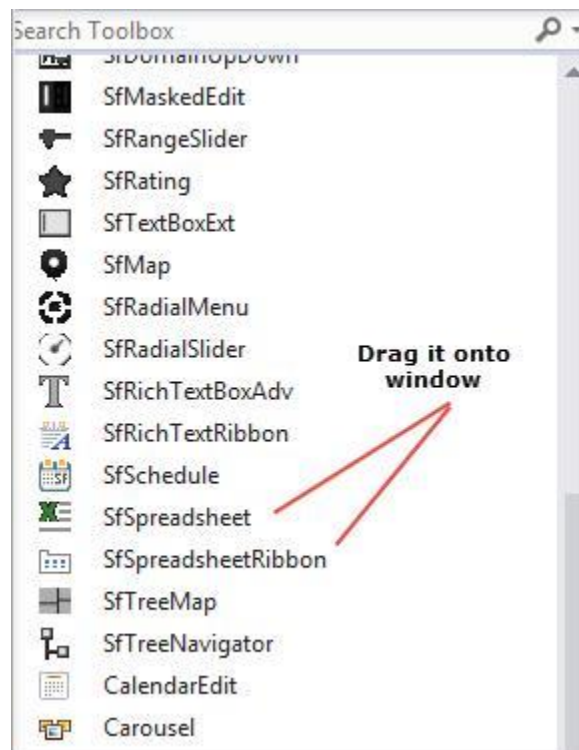
Optional Assemblies	Description
Syncfusion.SfSpreadsheetHelper.WPF.dll	Contains the classes which is responsible for importing charts and sparklines into SfSpreadsheet.
Syncfusion.ExcelChartToImageConverter.WPF.dll	Contains the classes which is responsible for converting charts as image.
Syncfusion.SfChart.WPF.dll	Contains the classes which is responsible for importing charts like Line charts, Pie charts, Sparklines etc.
Syncfusion.ExcelToPDFConverter.Base.dll	Contains the base and fundamental classes which is responsible for converting excel to PDF.
Syncfusion.Pdf.Base.dll	Contains the base and fundamental classes for creating PDF.

Create a Simple Application with Spreadsheet

SfSpreadsheet control can be added into the application either via designer or via coding.

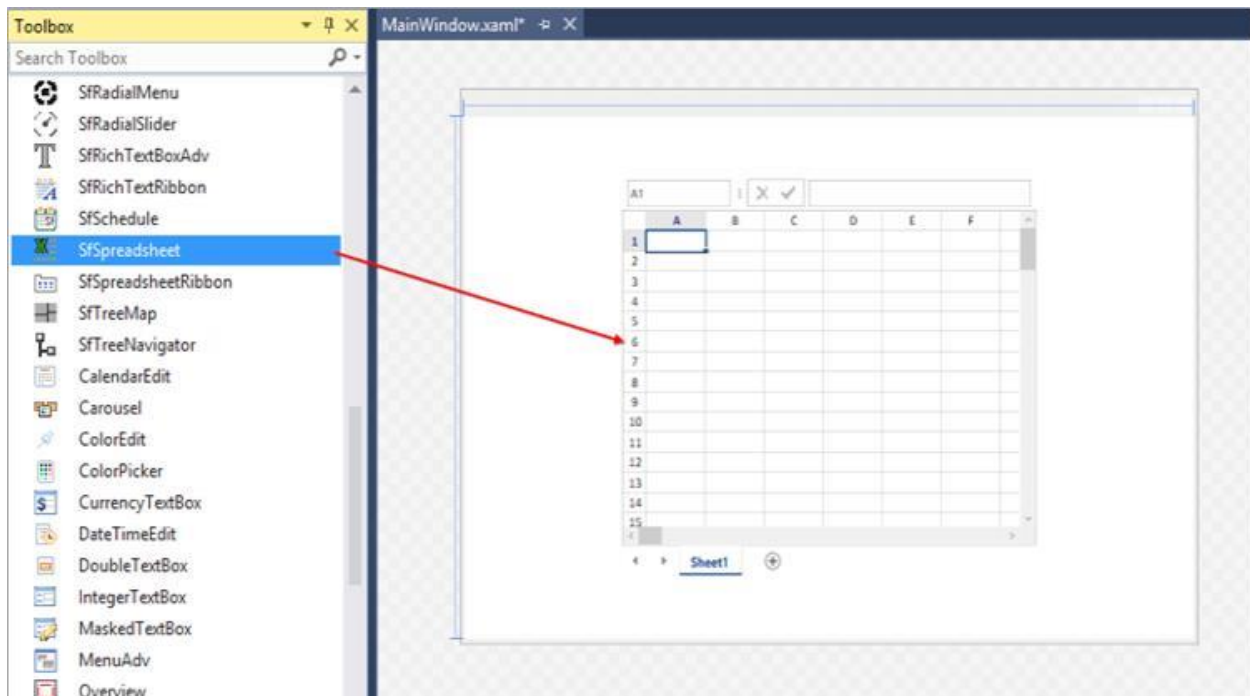
Adding a Control via Designer

1. Create new WPF application in Visual Studio. 2. Open the Visual Studio **Tool box**. Navigate to “Syncfusion Controls” tab, and find the SfSpreadsheet/SfSpreadsheetRibbon toolbox items



Syncfusion Control tab

3. Drag SfSpreadsheet and drop in the Designer area from the Toolbox



For Spreadsheet

XML

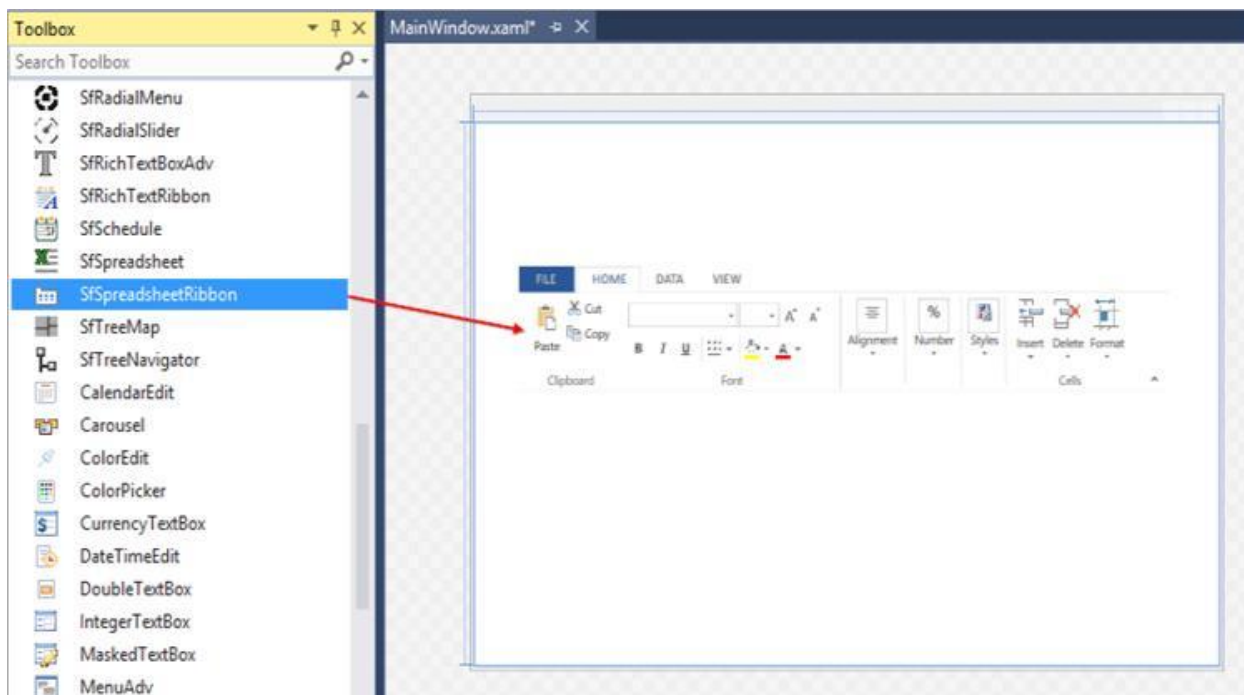
```
<syncfusion:SfSpreadsheet x:Name = spreadsheet />
```

4. Ribbon can be added to the application by dragging SfSpreadsheetRibbon to the Designer area. 5. To make an interaction between Ribbon items and SfSpreadsheet, bind the SfSpreadsheet as DataContext to the SfSpreadsheetRibbon.

For Ribbon

XML

```
<syncfusion:SfSpreadsheetRibbon DataContext= "{Binding  
ElementName=spreadsheet}" />
```



Adding Control via Coding

Spreadsheet is available in the following namespace *"Syncfusion.UI.Xaml.Spreadsheet"* and it can be created programmatically either by using XAML or C# code.

For Spreadsheet

XML

```
<Window x:Class="SpreadsheetDemo.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"
syncfusion:SkinStorage.VisualStudio="Office2013"
mc:Ignorable="d">
<syncfusion:SfSpreadsheet x:Name="spreadsheet"
FormulaBarVisibility="Visible"/>
</Window>
```

C#

```
SfSpreadsheet spreadsheet = new SfSpreadsheet();
this.grid.Children.Add(spreadsheet);
```

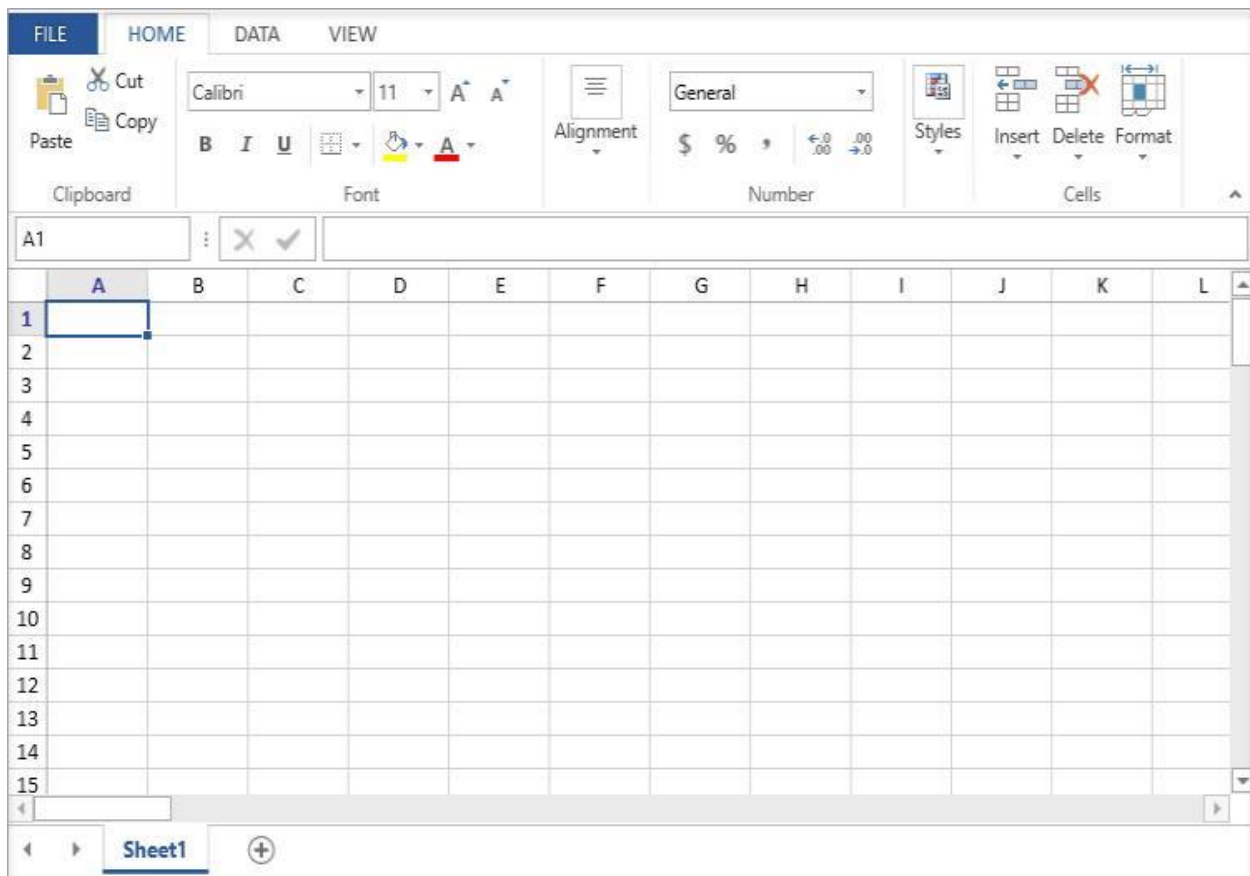
Note: To add the **SfSpreadsheetRibbon** in your application, use the **RibbonWindow** since the backstage of Ribbon will be opened only when the ribbon is loaded under the **RibbonWindow**

XML

```
<syncfusion:RibbonWindow x:Class="SpreadsheetDemo.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"  
syncfusion:SkinStorage.VisualStudio="Office2013"  
mc:Ignorable="d">  
</syncfusion:RibbonWindow>
```

You can get the following output when execute the application.



Creating a new Excel Workbook

A new workbook can be created by using a [Create](#) method with specified number of worksheets. By default, a workbook will be created with single worksheet.

C#

```
spreadsheet.Create(2);
```

Opening an existing Excel Workbook

The Excel Workbook can be opened in SfSpreadsheet using the [Open](#) method in various ways,

C#

```
//Using Stream,  
spreadsheet.Open (Stream file);  
//Using String,
```

```
spreadsheet.Open (string file);
//Using Workbook,
spreadsheet.Open(IWorkbook workbook);
```

C#

```
spreadsheet.Open (@"..\\..\\Data\\Outline.xlsx");
```

Outline.xlsx

FILEHOMEDATAVIEW

CutCopyPaste

Arial9A+

B

I

U

A-

Font

Alignment

General

\$ % + - . /

Number

Conditional Formatting

Format as Table

Cell Styles

Insert

Delete

Format

C7

12

A

B

C

D

E

F

G

H

I

J

K

L

M

N

Hours & Expense Tracking

START DATE:

Graphic Design Institute

Building Expansion Project

APR 20 - MAY 11

MAY 18 - JUN 8

RATE

4-20-2015

4-27-2015

5-4-2015

5-11-2015

TOTAL

5-18-2015

5-25-2015

6-1-2015

6-8-2015

TOTAL

RESOURCE 1

Initiation / Planning

\$50.00

12.0

3.0

25.0

7.0

47.0

25.0

35.0

40.0

70.0

170.0

Install / Development / Testing

\$45.00

20.0

15.0

12.0

18.0

65.0

20.0

20.0

38.0

65.0

143.0

Deployment

\$50.00

30.0

36.0

27.0

38.0

131.0

0.0

18.0

20.0

30.0

189.0

Operational / Maintenance

\$50.00

78.0

50.0

30.0

25.0

183.0

20.0

10.0

10.0

5.0

208.0

Total Hours

140.0

104.0

94.0

88.0

426.0

65.0

83.0

108.0

170.0

426.0

T & L Expenses

\$0.00

\$0.00

T & L Expenses - Capitalized

\$0.00

\$0.00

RESOURCE 2

Initiation / Planning

\$50.00

12.0

3.0

25.0

7.0

47.0

25.0

35.0

40.0

70.0

170.0

Install / Development / Testing

\$45.00

78.0

50.0

30.0

25.0

183.0

20.0

20.0

38.0

65.0

346.0

Deployment

\$50.00

10.0

36.0

27.0

38.0

111.0

0.0

18.0

20.0

30.0

189.0

Operational / Maintenance

\$50.00

12.0

3.0

25.0

7.0

47.0

20.0

10.0

10.0

5.0

92.0

Total Hours

112.0

92.0

107.0

77.0

388.0

65.0

83.0

108.0

170.0

426.0

Project Cost Tracker

Opening Excel File in SfSpreadsheet**Saving the Excel Workbook**

The Excel workbook can be saved in SfSpreadsheet using [Save](#) method. If the workbook already exists in the system drive, it will be saved in the same location, otherwise Save Dialog box opens to save the workbook in user specified location.

C#

```
spreadsheet.Save();
```

You can also use [SaveAs](#) method directly to save the existing excel file with modifications.

The [SaveAs](#) method in SfSpreadsheet can be used in various ways,

C#

```
//Using Stream,
spreadsheet.SaveAs (Stream file);
//Using String,
```

```
spreadsheet.SaveAs (string file);  
//For Dialog box,  
spreadsheet.SaveAs();
```

Displaying Charts and Sparklines

For importing charts and sparklines in SfSpreadsheet, add the following assembly as reference into the application.

Assembly: **Syncfusion.SfSpreadsheetHelper.WPF.dll**

Charts

Create an instance of Syncfusion.UI.Xaml.SpreadsheetHelper.[GraphicChartCellRenderer](#) and add that renderer into [GraphicCellRenderers](#) collection by using the helper method [AddGraphicChartCellRenderer](#) which is available under the namespace Syncfusion.UI.Xaml.Spreadsheet.GraphicCells.

C#

```
public MainWindow()  
{  
    InitializeComponent();  
    //For importing charts,  
    this.spreadsheet.AddGraphicChartCellRenderer(new  
        GraphicChartCellRenderer());  
}
```

Sparklines

Create an instance of Syncfusion.UI.Xaml.SpreadsheetHelper.[SparklineCellRenderer](#) and add that renderer into the Spreadsheet by using the helper method [AddSparklineCellRenderer](#) which is available under the namespace Syncfusion.UI.Xaml.Spreadsheet.GraphicCells.

C#

```
public MainWindow()  
{  
    InitializeComponent();  
    //For importing sparklines,  
    this.spreadsheet.AddSparklineCellRenderer(new SparklineCellRenderer());  
}
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Working With Spreadsheet in WPF Spreadsheet (SfSpreadsheet)

This section explains about accessing the Worksheet, Grid and the events associated with it.

Accessing the Worksheet

A **workbook** is an excel document in the SfSpreadsheet. It is an object that exposes the [IWorkbook](#) interface. Currently loaded workbook in the Spreadsheet can be accessed by using the [Workbook](#) property of SfSpreadsheet.

A workbook consists of one or more worksheets stored within the worksheet collection. Accessing the worksheets in the collection, can be done by the following ways,

C#

```
//By Specifying the index as,
spreadsheet.Workbook.Worksheets[0]
//By Specifying the sheet name as,
spreadsheet.Workbook.Worksheets["sheet1"]
//Access the Active worksheet as,
spreadsheet.ActiveSheet
```

For more information regarding working with worksheets, you can refer the [XlsIO UG](#) link

Note: `ActiveGrid` and `ActiveSheet` property can be accessed only after the `WorkbookLoaded` Event of `SfSpreadsheet` is triggered

Accessing the Grid

Each worksheet in the workbook is loaded into the view as `SpreadsheetGrid` in `SfSpreadsheet`.

When the workbook is loaded in the `SfSpreadsheet`, the `WorkbookLoaded` Event is invoked and when the workbook is removed from `SfSpreadsheet`, the `WorkbookUnloaded` Event is invoked.

When the worksheet is added into the `SfSpreadsheet`, the `WorksheetAdded` Event is invoked and when the worksheet is removed in the `SfSpreadsheet`, `WorksheetRemoved` Event is invoked.

Hence you can access the `ActiveGrid` either in the `WorkbookLoaded` or `WorksheetAdded` Event.

C#

```
spreadsheet.WorksheetAdded += spreadsheet_WorksheetAdded;
spreadsheet.WorksheetRemoved += spreadsheet_WorksheetRemoved;
void spreadsheet_WorksheetAdded(object sender, WorksheetAddedEventArgs args)
{
    //Access the Active SpreadsheetGrid and hook the events associated with it.
    var grid = spreadsheet.ActiveGrid;
    grid.CurrentCellActivated += grid_CurrentCellActivated;
}
void spreadsheet_WorksheetRemoved(object sender, WorksheetRemovedEventArgs
args)
{
    //Access the Active SpreadsheetGrid and unhook the events associated with it
    var grid = spreadsheet.ActiveGrid;
    grid.CurrentCellActivated -= grid_CurrentCellActivated;
}
```

You can also access the each `SpreadsheetGrid` in the `SfSpreadsheet` either by passing the particular sheet name in the `GridCollection` or by invoking `WorkbookLoaded` Event of `SfSpreadsheet`.

By using Sheet Name

For your reference, setting the row and column count dynamically for the second sheet in the Workbook

C#

```
var sheet = spreadsheet.Workbook.Worksheets[1];
spreadsheet.GridCollection[sheet.Name].RowCount = 50;
```

```
spreadsheet.GridCollection[sheet.Name].ColumnCount = 12;
```

By using Event

C#

```
spreadsheet.WorkbookLoaded += spreadsheet_WorkbookLoaded;
spreadsheet.WorkbookUnloaded += spreadsheet_WorkbookUnloaded;
void spreadsheet_WorkbookLoaded(object sender, WorkbookLoadedEventArgs args)
{
    //Hook the events here
    foreach (var grid in args.GridCollection)
    {
        grid.QueryRange += grid_QueryRange;
    }
}
void spreadsheet_WorkbookUnloaded(object sender, WorkbookUnloadedEventArgs args)
{
    //Unhook the events here
    foreach (var grid in args.GridCollection)
    {
        grid.QueryRange -= grid_QueryRange;
    }
}
```

Note: SfSpreadsheet supports virtual mode, which lets you dynamically provide data to the grid by handling an event, [QueryRange](#), for example. In virtual mode, data will be dynamically loaded into the SpreadsheetGrid on demand or when users need to view the data.

Setting the ActiveSheet programmatically

SfSpreadsheet allows you to set the [ActiveSheet](#) programmatically by specifying the sheet name in the [SetActiveSheet](#) method of [SfSpreadsheet](#).

C#

```
spreadsheet.SetActiveSheet("Sheet5");
```

Accessing the cell or range of cells

SfSpreadsheet allows to access a single cell or range of cells in the workbook using [IRange](#) interface.

The following code shows the several ways of accessing a single cell or range of cells in the [Worksheet](#),

C#

```
// Access a cell by specifying cell address.
var cell = spreadsheet.Workbook.Worksheets[0].Range["A3"];
// Access a cell by specifying cell row and column index.
var cell1 = spreadsheet.Workbook.Worksheets[0].Range[3, 1];
// Access a cells by specifying user defined name.
var cell2 = spreadsheet.Workbook.Worksheets[0].Range["Namerange"];
// Accessing a range of cells by specifying cell's address.
var cell3 = spreadsheet.Workbook.Worksheets[0].Range["A5:C8"];
// Accessing a range of cells specifying cell row and column index.
var cell4 = spreadsheet.Workbook.Worksheets[0].Range[15, 1, 15, 3];
```


For more reference regarding accessing the range, refer [XlsIO](#) UG.

Note: If the user has made any modifications with XlsIO range in SfSpreadsheet, then they should [refresh the view](#) to update the modifications in `SpreadsheetGrid`.

Accessing the value of a cell

SfSpreadsheet allows you to access the value of a cell by using `Value` property of `IRange` and to get the value of the cell along with its format, `DisplayText` property can be used.

C#

```
// Access a cell value by using "Value" Property,
var cellValue = spreadsheet.Workbook.Worksheets[1].Range["A3"].Value
// Access a cell value by using "DisplayText" Property.
var displayValue = spreadsheet.Workbook.Worksheets[1].Range[4,
1].DisplayText;
```

Setting the value or formula to a cell

In SfSpreadsheet, to update the cell value and formula programmatically, `SetCellValue` method of `SpreadsheetGrid` should be invoked and then invalidate that cell to update the view.

C#

```
var range = spreadsheet.ActiveSheet.Range[2, 2];
spreadsheet.ActiveGrid.SetCellValue(range, "cellValue");
spreadsheet.ActiveGrid.InvalidateCell(2, 2);
```

Clearing the value or formatting from a cell

SfSpreadsheet allows you to delete the contents of a cell or delete the contents along with its formatting (comments, Conditional formats,..) also.

The following code illustrates the different way of deleting the value from a cell,

C#

```
//To clear the contents in the range alone,
spreadsheet.Workbook.Worksheets[0].Range[3, 3].Clear();
//To clear the contents along with its formatting in the range,
spreadsheet.Workbook.Worksheets[0].Range[3, 3].Clear(true);
//To clear the range with specified ExcelClearOptions,
spreadsheet.Workbook.Worksheets[0].Range[3,
3].Clear(ExcelClearOptions.ClearDataValidations);
```

Note: `ExcelClearOptions` is an enum which specifies the possible directions to clear the cell formats, content, comments, conditional format, data validation or clear all of them.

Refreshing the view

SfSpreadsheet allows you to invalidate or refresh the view either by specifying the specific range or full range.

The following code demonstrates the different ways of refreshing the view,

C#

```
//Invalidates the mentioned cell in the grid,
spreadsheet.ActiveGrid.InvalidateCell(3, 3);
//Invalidates the range ,
var range = GridRangeInfo.Cells(5, 4, 6, 7);
spreadsheet.ActiveGrid.InvalidateCell(range);
//Invalidates all the cells in the grid,
spreadsheet.ActiveGrid.InvalidateCells();
//Invalidates the measurement state(layout) of grid,
spreadsheet.ActiveGrid.InvalidateVisual();
//Invalidates the cell borders in the range,
var range = GridRangeInfo.Cells(2, 4, 6, 4);
spreadsheet.ActiveGrid.InvalidateCellBorders(range);
```

Scrolling the Grid programmatically

SfSpreadsheet allows the user to scroll the grid into mentioned cell, by using [ScrollInView](#) method of `SpreadsheetGrid`.

C#

```
spreadsheet.ActiveGrid.ScrollInView(new RowColumnIndex(5, 5));
```

Formula Bar

The Formula Bar is located above the worksheet area of the SfSpreadsheet. The formula bar displays the data or formula stored in the active cell.

Users can set the visibility state of Formula Bar using [FormulaBarVisibility](#) property of `SfSpreadsheet`.

XML

```
<syncfusion:SfSpreadsheet x:Name="spreadsheet"
FormulaBarVisibility="Collapsed"/>
```

C#

```
spreadsheet.FormulaBarVisibility = System.Windows.Visibility.Collapsed;
```

Identify whether the workbook is modified or not

`IsCellModified` property of `WorkbookImpl` is used to identify whether any cell modified in a workbook or not after importing. Since it is an internal property, access it using Reflection.

C#

```
var workbook = spreadsheet.Workbook as WorkbookImpl;
BindingFlags bindFlags = BindingFlags.Instance | BindingFlags.Public |
BindingFlags.NonPublic | BindingFlags.Static;
var value = typeof(WorkbookImpl).GetProperty("IsCellModified",
bindFlags).GetValue(workbook);
```

Suppress message boxes in Spreadsheet

In Spreadsheet, warning messages, error alerts are displayed while performing some actions like Excel. If you want to avoid those alerts, then set the [DisplayAlerts](#) property to `false`.

C#

```
//To Suppress message boxes in Spreadsheet
spreadsheet.DisplayAlerts = false;
```

Suspend and resume formula calculation

Spreadsheet provides support to suspend the formula calculation and resume it whenever needed using the [SuspendFormulaCalculation](#) and [ResumeFormulaCalculation](#) method.

Resuming formula calculation will recalculate all the formula cells in a workbook. This would be helpful to improve the performance when you are updating the value of more number of cells by skipping the dependent cells recalculation on each cell value changed.

C#

```
//Resumes the automatic formula calculation
spreadsheet.ResumeFormulaCalculation();
//Suspends the automatic formula calculation
spreadsheet.SuspendFormulaCalculation();
```

Close the popup programmatically

In SfSpreadsheet, popup windows are used to display the options like copy paste option, fill series option, etc. which will be closed automatically on certain actions. However you can also able to close the popup programmatically by using the [ShowHidePopup](#) method of [SpreadsheetGrid](#).

C#

```
//To close the popup
spreadsheet.ActiveGrid.ShowHidePopup(false);
//To show the closed popup, if needed.
spreadsheet.ActiveGrid.ShowHidePopup(true);
```

Identify when the active sheet is changed

SfSpreadsheet provides support to identify when the active sheet is changed by using [PropertyChanged](#) event of SfSpreadsheet like below.

C#

```
Spreadsheet.PropertyChanged += Spreadsheet_PropertyChanged;
void Spreadsheet_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
{
    // when the worksheets in the workbook changed
    if(e.PropertyName == "ActiveSheet")
    {
        //Implement code
    }
}
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Selection in WPF Spreadsheet (SfSpreadsheet)

This section explains about the Selection behavior in SfSpreadsheet.

The SfSpreadsheet control provides support for selection in grid by using mouse, keyboard and touch interactions.

By default, Selection behavior will be enabled in SfSpreadsheet, but if you want to disable the selection in SfSpreadsheet, then set the [AllowSelection](#) Property to be false.

C#

```
void spreadsheet_WorkbookLoaded(object sender, WorkbookLoadedEventArgs args)
{
    spreadsheet.ActiveGrid.AllowSelection = false;
}
```

Accessing the Current cell

SfSpreadsheet allows the user to access the active cell by using the [CurrentCell](#) property of [SelectionController](#) Class.

C#

```
var cell= spreadsheet.ActiveGrid.SelectionController.CurrentCell;
```

Accessing the Selected ranges

SfSpreadsheet allows the user to access the selected ranges of the [SpreadsheetGrid](#) using [SelectedRanges](#) property of [SpreadsheetGrid](#).

C#

```
var rangeList = spreadsheet.ActiveGrid.SelectedRanges;
```

Note: To get the active range in the selected ranges list, use [ActiveRange](#) property of [GridRangeInfoList](#) class.

Adding or Clearing the Selection

SfSpreadsheet allows the user to add and clear the selection in the Active [SpreadsheetGrid](#) for the given range.

C#

```
//To Add the Selection for range,
spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Cells(
4, 6, 5, 8));
//To Add the Selection for particular row,
spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Row(4)
);
//To Add the Selection for multiple rows,
spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Rows(4
, 9));
//To Add the Selection for particular column,
spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Col(5)
);
//To Add the Selection for multiple columns,
spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Cols(5
, 10));
//To Clear the Selection,
```

```
spreadsheet.ActiveGrid.SelectionController.ClearSelection();
```

Move Current Cell

SfSpreadsheet allows the user to move the current cell to the mentioned cell in [SpreadsheetGrid](#).

C#

```
//Moves current cell to the mentioned row and column index of cell,
spreadsheet.ActiveGrid.CurrentCell.MoveCurrentCell(5, 5);
//For moving the current cell to a different sheet,
spreadsheet.SetActiveSheet("Sheet2");
spreadsheet.ActiveGrid.CurrentCell.MoveCurrentCell(6, 5);
```

Converting GridRangeInfo into IRange

SfSpreadsheet allows the user to convert the [GridRangeInfo](#) into the equivalent [IRange](#) by using [ConvertGridRangeToExcelRange](#) method of [GridExcelHelper](#) class.

C#

```
var excelRange =
GridExcelHelper.ConvertGridRangeToExcelRange(GridRangeInfo.Cell(4, 5),
spreadsheet.ActiveGrid);
```

Tips: Users can also convert the [IRange](#) into equivalent [GridRangeInfo](#) by using [ConvertExcelRangeToGridRange](#) method of [GridExcelHelper](#) class.

Properties, Methods and Events

Below table lists the events associated with selection behavior,

Events	Description
CellClick	Occurs when you click on the cell.
CurrentCellActivating	Occurs when the current cell is going to be activated. This event allow to cancel the current cell activation.
CurrentCellActivated	Occurs after the current cell is activated.
SelectionChanging	Occurs when the selection is going to be changed. This event allows to cancel the selection change.
SelectionChanged	Occurs after the selection is changed.

Below table lists the properties associated with selection,

Properties	Description
SelectedRanges	Gets or sets the collection of selected ranges from grid.
SelectionBrush	Gets or sets the selected area brush.

SelectionBorderBrush	Gets or sets the selection border brush.
SelectionBorderThickness	Gets or sets the thickness of selection border.
SelectionController	Gets the Selection Controller which provides the selection of content when the user drags the pressed mouse to an edge of the control.
AllowSelection	Gets or Sets the value whether to allow the selection in the ActiveGrid or not.
ShowTouchIndicator	Determines whether the touch indicator will be shown or not.
TouchHitTestPrecision	Gets or sets the distance of touch precision point from touch indicator.

Below table lists the properties associated with [CurrentCell](#) of SpreadsheetGrid,

Properties	Description
CellRowColumnIndex	Gets the row and column index of the CurrentCell.
RowIndex	Gets the row index of the CurrentCell.
ColumnIndex	Gets the column index of the CurrentCell.
Range	Gets the range of the CurrentCell.
HasCurrentCell	Gets the value indicating whether the Grid has CurrentCell or not.
PreviousRowColumnIndex	Gets or sets the row and column index of old CurrentCell.

Below table lists the methods associated with selection,

Methods	Description
AddSelection	Adds/Extends the Selection to the mentioned range .
ClearSelection	Clears the Selection.
MoveCurrentCell	Move the Current cell to mentioned row and column index.

Key Navigation

Below table lists the key combinations associated with selection,

Key Combination	Description
HOME	Moves to the first cell of the current row .
END	Moves to the last cell of the current row .
UPARROW	Moves to one cell up of the current cell in the worksheet.
DOWNARROW	Moves to one cell down of the current cell in the worksheet.
LEFTARROW	Moves to one cell left of the current cell in the worksheet.

RIGHTARROW	Moves to one cell right of the current cell in the worksheet.
PAGEUP	Moves to the first visible cell of the current column.
PAGEDOWN	Moves to the last visible cell of the current column.
CTRL+HOME	Moves to the beginning cell of a worksheet.
CTRL+END	Moves to the last cell of a worksheet.
ALT+PAGE UP	Moves one screen to the left in a worksheet.
ALT+PAGE DOWN	Moves one screen to the right in a worksheet.
CTRL + ARROW KEYS	Moves to the first/last cell of the current cell based on Arrow directions .
ENTER	Moves the active current cell to one cell down in the selection.
SHIFT+ENTER	Moves the active current cell to one cell up in the selection.
TAB	Moves the active current cell in one cell to the right of the selection.
SHIFT+TAB	Moves the active current cell in one cell to the left of the selection.
BACKSPACE	Begins the edit operation for the active cell in the selection.
CTRL+A	Selects the entire worksheet.
SHIFT+ARROW KEYS	Extends the selection by one cell based on the arrow direction.
CTRL+SHIFT+ARROW KEYS	Extend the selection to the last cell in a row or column.
SHIFT+HOME	Extends the selection to the first column from the active cell.
CTRL+SHIFT+HOME	Extends the selection from the active cell to the first cell.
SHIFT+PAGE DOWN	Extends the selection down in a worksheet.
SHIFT+PAGE UP	Extends the selection up in a worksheet.

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Editing in WPF Spreadsheet (SfSpreadsheet)

This section explains about the Editing behavior, Data Validation and Hyperlinks in SfSpreadsheet.

Editing

The SfSpreadsheet control provides support for editing, you can modify and commit the cell values in the workbook.

By default, Editing will be enabled in SfSpreadsheet, but if you want to disable the editing in SfSpreadsheet, then set the [AllowEditing](#) Property to be false.

C#

```
void spreadsheet_WorkbookLoaded(object sender, WorkbookLoadedEventArgs args)
{
    spreadsheet.ActiveGrid.AllowEditing = false;
}
```

Editing a cell programmatically

Start Editing

User can edit a cell programmatically by using [BeginEdit](#) method.

C#

```
spreadsheet.ActiveGrid.CurrentCell.BeginEdit(true);
```

End Editing

User can end the editing of a cell programmatically in any of the following way,

- [ValidateAndEndEdit](#) - Validates and ends the edit operation for the current cell. if the cancel is "true", then the current cell remains in edit mode else if the validation is true, commits the new value and moved to next cell or else if the validation is false, it reverts the old value and moved to next cell.
- [EndEdit](#) - Commits and ends the edit operation for the current cell, if it is passed with parameter "true", commits the new changes for the cell, else reverts the old value.

C#

```
//Validates and end the edit operation,
spreadsheet.ActiveGrid.CurrentCell.ValidateAndEndEdit();
//Commits the value and end the edit operation,
spreadsheet.ActiveGrid.CurrentCell.EndEdit(true);
```

Locking or Unlocking a cell

Locking cells allows you to disable editing and formatting the cells when the sheet is protected. By default, every cells are locked in the worksheet.

But while in protect mode, if you want to edit or format a cell, you can unlock the cells.

C#

```
var worksheet = spreadsheet.ActiveSheet;
var excelStyle = worksheet.Range["A2"].CellStyle;
//To unlock a cell,
excelStyle.Locked = false;
//To lock a cell,
excelStyle.Locked = true;
```

Properties, Methods and Events

The order of events when editing and committing a cell value in SfSpreadsheet,

Events	Description
--------	-------------

CurrentCellBeginEdit	Occurs when the current cell enters into edit mode. This event allows to cancel entering the edit mode.
CurrentCellValueChanged	Occurs when the current cell value is changed in edit mode.
CurrentCellValidating	Occurs when the current cell value is going to be validated. It allows you to validate and cancel the end editing.
CurrentCellValidated	Occurs after the current cell is validated.
CurrentCellEndEdit	Occurs when the current cell leaves from edit mode.

Below table list the properties associated with Editing.

Properties	Description
AllowEditing	Gets or sets the value indicating whether to allow the editing operation or not.
EditorSelectionBehavior	Gets or sets a value indicating whether editor select all the value or move last position.
EditTrigger	Gets or sets a value indicating any of the trigger options will cause cells to enter Edit Mode.
IsEditing	Gets whether the current cell is in edit mode or not.

Below table list the methods associated with Editing.

Methods	Description
BeginEdit	Begins the editing operation of the current cell and returns true if the current cell enters into edit mode.
EndEdit	Commits and ends the edit operation of the current cell.
ValidateAndEndEdit	Validates and ends the edit operation of the current cell.
Validate	Validates the current cell in the SpreadsheetGrid.

Data Validation

Data Validation is a list of rules to limit the type of data or the values that can be entered in the cell.

Applying Data Validation at runtime

SfSpreadsheet allows the user to apply the data validation rules at runtime for particular cell or range using `IDataValidation` interface.

C#

```
//Number Validation
IDataValidation validation =
spreadsheet.ActiveSheet.Range["A5"].DataValidation;
validation.AllowType = ExcelDataType.Integer;
```

```
validation.CompareOperator = ExcelDataValidationComparisonOperator.Between;
validation.FirstFormula = "4";
validation.SecondFormula = "15";
validation.ShowErrorBox = true;
validation.ErrorBoxText = "Accepts values only between 4 to 15";
//Date Validation
IDataValidation validation =
spreadsheet.ActiveSheet.Range["B4"].DataValidation;
validation.AllowType = ExcelDataType.Date;
validation.CompareOperator = ExcelDataValidationComparisonOperator.Greater;
validation.FirstDateTime = new DateTime(2016, 5, 5);
validation.ShowErrorBox = true;
validation.ErrorBoxText = "Enter the date value which is greater than
05/05/2016";
//TextLength Validation
IDataValidation validation =
spreadsheet.ActiveSheet.Range["A3:B3"].DataValidation;
validation.AllowType = ExcelDataType.TextLength;
validation.CompareOperator =
ExcelDataValidationComparisonOperator.LessOrEqual;
validation.FirstFormula = "4";
validation.ShowErrorBox = true;
validation.ErrorBoxText = "Text length should be lesser than or equal 4
characters";
//List Validation
IDataValidation validation =
spreadsheet.ActiveSheet.Range["D4"].DataValidation;
validation.ListOfValues = new string[] { "10", "20", "30" };
//Custom Validation
IDataValidation validation =
spreadsheet.ActiveSheet.Range["D4"].DataValidation;
validation.AllowType = ExcelDataType.Formula;
validation.FirstFormula = "=A1+A2>0";
validation.ErrorBoxText = "Sum of the values in A1 and A2 should be greater
than zero";
```

For more reference, please go through the [XlsIO](#) UG.

Tips: If you want to load ComboBox to a cell in SfSpreadsheet, you can apply List Validation to that cell.

Hyperlink

The Hyperlink is a convenient way to access the web pages, files and browse the data within a worksheet or other worksheets in a workbook. SfSpreadsheet provides support to add, edit and remove the Hyperlinks in the workbook.

Add a Hyperlink to a cell

SfSpreadsheet provides support to add hyperlink to a cell and it can be added in the hyperlinks collection using [IHyperlinks](#) interface.

SfSpreadsheet allows you to add below types of the hyperlink.

- Web URL
- Cell or range in workbook
- E-mail
- External files

C#

```
// Creating a Hyperlink for e-mail,
var range = spreadsheet.ActiveSheet.Range["A5"];
IHyperLink hyperlink1 = spreadsheet.ActiveSheet.HyperLinks.Add(range);
hyperlink1.Type = ExcelHyperLinkType.Url;
hyperlink1.Address = "mailto:Username@syncfusion.com";
hyperlink1.TextToDisplay="Send Mail";
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Cell(5, 1));
// Creating a Hyperlink for Opening Files,
var range1 = spreadsheet.ActiveSheet.Range["D5"];
IHyperLink hyperlink2 = spreadsheet.ActiveSheet.HyperLinks.Add(range1);
hyperlink2.Type = ExcelHyperLinkType.File;
hyperlink2.Address = @"C:\Samples\Local";
hyperlink2.TextToDisplay = "File Location";
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Cell(5, 4));
//Creating a Hyperlink to refer another cell in the workbook,
var range2 = spreadsheet.ActiveSheet.Range["C13"];
IHyperLink hyperlink3 = spreadsheet.ActiveSheet.HyperLinks.Add(range);
hyperlink3.Type = ExcelHyperLinkType.Workbook;
hyperlink3.Address = "Sheet2!C23";
hyperlink3.TextToDisplay = "Sample";
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Cell(13, 3));
```

Edit or Remove a Hyperlink

SfSpreadsheet provides support to edit or remove the hyperlinks from the range by accessing Hyperlinks collection.

C#

```
//To Edit a hyperlink in a cell,
var hyperlink = spreadsheet.ActiveSheet.Range["A5"].Hyperlinks[0];
hyperlink.TextToDisplay = "Sample";
hyperlink.Address = "http://help.syncfusion.com";
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Cell(5, 1));
//To remove a hyperlink in a cell,
var hyperlink = spreadsheet.ActiveSheet.Range["A5"].Hyperlinks.RemoveAt(0);
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Cell(5, 1));
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Formatting in WPF Spreadsheet (SfSpreadsheet)

This section explains about the formatting options similar to excel in SfSpreadsheet.

Styles and formats defined in an Excel file are automatically imported. Users can also apply these settings to cells during run time. The following are the formatting attributes for the cell.

- Cell font settings (font name, size, color, style, etc.)
- Cell background
- Cell content alignment (vertical and horizontal alignment, indent and text wrapping)
- Cell borders
- Number Formatting

- Merge Cells
- Built-in Styles
- Table Formats

Cell Background

For applying background color for the cells at runtime in SfSpreadsheet, set the color index for the particular XlsIO range and invalidate the range in order to update the view in [SpreadsheetGrid](#).

For single cell

C#

```
IRange range = spreadsheet.ActiveSheet.Range["A5"];
range.CellStyle.ColorIndex = Syncfusion.XlsIO.ExcelKnownColors.Blue;
spreadsheet.ActiveGrid.InvalidateCell(range.Row, range.Column);
```

For selected range of cells,

C#

```
var selectedRanges = spreadsheet.ActiveGrid.SelectedRanges;
foreach (var range in selectedRanges)
{
    string cell = GridExcelHelper.ConvertGridRangeToExcelRange(range,
        spreadsheet.ActiveGrid);
    spreadsheet.ActiveSheet.Range[cell].CellStyle.ColorIndex =
        ExcelKnownColors.Blue;
    spreadsheet.ActiveGrid.InvalidateCell(range, true);
}
```

Font

SfSpreadsheet allows the user to apply the font settings such as font color, font name, font size etc., for a particular cell or a range of cells.

C#

```
IRange range = spreadsheet.Workbook.Worksheets[0].Range["A1:B5"];
var gridRange = GridExcelHelper.ConvertExcelRangeToGridRange(range);
//Setting the Font Family Name,
range.CellStyle.Font.FontName = "Arial Black";
//Setting the Font Styles,
range.CellStyle.Font.Bold = true;
range.CellStyle.Font.Italic = true;
//Setting the Font Size,
range.CellStyle.Font.Size = 18;
//Setting the Font Effects,
range.CellStyle.Font.Strikethrough = true;
//Setting the Underline Types,
range.CellStyle.Font.Underline = ExcelUnderline.Single;
//Setting the Font Color,
range.CellStyle.Font.Color = ExcelKnownColors.Blue;
//Invalidating the range, to update in view,
spreadsheet.ActiveGrid.InvalidateCell(gridRange, true);
```

Cell Borders

SfSpreadsheet allows the user to apply the borders at runtime for particular cell or range of cells,

C#

```
//For a single cell,
IRange range = spreadsheet.Workbook.Worksheets[0].Range["A5"];
range.Borders.LineStyle = ExcelLineStyle.Dash_dot;
range.Borders.Color = ExcelKnownColors.Gold;
spreadsheet.ActiveGrid.InvalidateCell(range.Row, range.Column);
//For a range of cells,
IRange excelRange = spreadsheet.Workbook.Worksheets[0].Range["C3:D8"];
excelRange.BorderAround(ExcelLineStyle.Double, ExcelKnownColors.Green);
excelRange.BorderInside(ExcelLineStyle.Dotted, ExcelKnownColors.Tan);
var gridRange = GridExcelHelper.ConvertExcelRangeToGridRange(excelRange);
spreadsheet.ActiveGrid.InvalidateCell(gridRange, true);
```

Cell Alignment

SfSpreadsheet allows the user to align the content of the cell. The alignment options includes Horizontal Alignment, Vertical Alignment, Indentation, Orientation etc.,

C#

```
//Applying Horizontal Alignment for the cell "A2",
spreadsheet.Workbook.Worksheets[0].Range["A2"].CellStyle.HorizontalAlignment = ExcelHAlign.HAlignCenter;
spreadsheet.ActiveGrid.InvalidateCell(2,1);
//Applying Vertical Alignment for the cell "B2",
spreadsheet.Workbook.Worksheets[0].Range["B2"].CellStyle.VerticalAlignment = ExcelVAlign.VAlignBottom;
spreadsheet.ActiveGrid.InvalidateCell(2,2);
//Applying Orientation for the selected cell or ranges,
spreadsheet.FormatOrientation(90);
//For Indentation,
//Increase the indent for the selected ranges or cell,
spreadsheet.FormatIndent(true);
//Decrease the indent for the selected ranges or cell,
spreadsheet.FormatIndent(false);
//Level of indent for selected ranges or cell,
spreadsheet.FormatIndentLevel(3);
```

Wrap Text

SfSpreadsheet allows the user to wrap the text in the cell, if the text is too large.

C#

```
spreadsheet.ActiveSheet.Range["C4"].Text = "Wrapping the content in the cell";
spreadsheet.ActiveSheet.Range["C4"].WrapText = true;
spreadsheet.ActiveSheet.AutofitRow(4);
spreadsheet.ActiveGrid.SetRowHeight(4, 4,
spreadsheet.ActiveSheet.GetRowHeightInPixels(4));
spreadsheet.ActiveGrid.InvalidateCell(4, 3);
```

Merge Cells

Merge

SfSpreadsheet provides support to merge two or more cells. When a group of cells is merged, the contents of the upper-left cell will be taken as the content of the merged cell, rest will be deleted.

For merging the cells in SfSpreadsheet, you need to add the [CoveredCellInfo](#) into [CoveredCells](#) collection of SpreadsheetGrid and merge the range using [Merge](#) method in XlsIO. Also to update the view, you need to invalidate the cells in the SpreadsheetGrid

C#

```
var gridRange = spreadsheet.ActiveGrid.SelectedRanges.ActiveRange;
var excelRange =
gridRange.ConvertGridRangeToExcelRange(spreadsheet.ActiveGrid);
var coverCell = new CoveredCellInfo(gridRange.Top, gridRange.Left,
gridRange.Bottom, gridRange.Right);
spreadsheet.ActiveGrid.CoveredCells.Add(coverCell);
spreadsheet.ActiveSheet.Range[excelRange].Merge();
spreadsheet.ActiveGrid.InvalidateCell(gridRange, true);
```

Unmerge

You can also unmerge the merged cells in SfSpreadsheet.

For unmerging the cells in SfSpreadsheet, you need to clear the [CoveredCells](#) from the SpreadsheetGrid and unmerge the range using [UnMerge](#) method in XlsIO. Also to update the view, you need to invalidate the cells in the SpreadsheetGrid

C#

```
var gridRange = spreadsheet.ActiveGrid.SelectedRanges.ActiveRange;
var excelRange =
gridRange.ConvertGridRangeToExcelRange(spreadsheet.ActiveGrid);
spreadsheet.ActiveGrid.CoveredCells.Clear(gridRange);
spreadsheet.ActiveSheet.Range[excelRange].UnMerge();
spreadsheet.ActiveGrid.InvalidateCell(gridRange, true);
```

Number Format

SfSpreadsheet allows the user to view the numbers in the cells with different formats which includes currency, percentage, datetime, scientific etc.,

C#

```
//Applying Percentage format for the selected ranges at runtime,
spreadsheet.Workbook.ActiveSheet.Range["C3"].NumberFormat = "0.00%";
spreadsheet.ActiveGrid.InvalidateCell(3, 3);
//Applying Date format for the selected ranges at runtime,
spreadsheet.Workbook.ActiveSheet.Range["D1"].NumberFormat = "m/d/yyyy";
spreadsheet.ActiveGrid.InvalidateCell(1, 4);
//Applying Time format for the selected ranges at runtime,
spreadsheet.Workbook.ActiveSheet.Range["D4"].NumberFormat = "[$-F400]h:mm:ss
AM/PM";
spreadsheet.ActiveGrid.InvalidateCell(3, 4);
//Applying Text format for the selected ranges at runtime,
spreadsheet.Workbook.ActiveSheet.Range["D5"].NumberFormat = "@";
spreadsheet.ActiveGrid.InvalidateCell(4, 4);
```

The different types of number formats with its notation are

Formats	Notation
General	General
Number	0.00
Currency	\$* #,##0.00
Accounting	\$ (#,##0.00);\$ -?;@
Short Date	m/d/yyyy
Long Date	[\$-F800]dddd, mmmm dd, yyyy
Time	[\$-F400]h:mm:ss AM/PM
Percentage	0.00%
Fraction	#?/?
Scientific	0.00E+00
Text	@

Built-in Styles

SfSpreadsheet supports some predefined built in styles of XlsIO. [BuiltInStyles](#) is an enum which contains different styles for formatting a cell or range of cells.

C#

```
spreadsheet.Workbook.ActiveSheet.Range["A3"].BuiltInStyle =
BuiltInStyles.Heading2;
spreadsheet.ActiveGrid.InvalidateCell(3, 1);
```

Format as Table

SfSpreadsheet allows the users to format a table with built in styles of table (i.e.) [TableBuiltInStyles](#) of XlsIO

C#

```
// Creating a table
IListObject table =
spreadsheet.Workbook.ActiveSheet.ListObjects.Create("Table1",
spreadsheet.Workbook.ActiveSheet.Range["C1:G5"]);
// Formatting table with a built-in style
table.BuiltInTableStyle = TableBuiltInStyles.TableStyleLight6;
spreadsheet.ActiveGrid.InvalidateCells();
```

For more information regarding formatting options, please go through [XlsIO](#)

Note: Users need to [refresh the view](#) after the formatting is applied on the XlsIO range to update the styles in `SpreadsheetGrid`.

Clear formatting

SfSpreadsheet provides support to clear the contents of a cell along with its formatting or by specifying the required clear options using [ExcelClearOptions](#) enum which specifies the possible directions to clear the cell formats, content, comments, conditional format, data validation or clear all of them.

C#

```
//To clear the contents along with its formatting in the range,
spreadsheet.Workbook.Worksheets[0].Range[4, 5].Clear(true);
//To clear the range with specified ExcelClearOptions,
spreadsheet.Workbook.Worksheets[0].Range[4,
5].Clear(ExcelClearOptions.ClearConditionalFormats);
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Conditional Formatting in WPF Spreadsheet (SfSpreadsheet)

This section explains about how to apply conditional formatting rules programmatically at run time in SfSpreadsheet.

In SfSpreadsheet, to apply conditional format for a cell or range of cells, add [IConditionalFormat](#) to that range by using [AddCondition](#) method.

C#

```
var worksheet = spreadsheet.Workbook.Worksheets[0];
IConditionalFormats condition = worksheet.Range["A1"].ConditionalFormats;
IConditionalFormat condition1 = condition.AddCondition();
```

Highlight Cell Rules

Based on CellValue

To format the cells based on cell value, define the conditional format type as **CellValue** and other formatting options such as formula, operator, background color etc., to the specified cell or range. Finally, invalidate the cells to refresh the view.

C#

```
var worksheet = spreadsheet.Workbook.Worksheets[0];
IConditionalFormats condition =
worksheet.Range["A1:A100"].ConditionalFormats;
IConditionalFormat condition1 = condition.AddCondition();
condition1.FormatType = ExcelCfType.CellValue;
condition1.Operator = ExcelComparisonOperator.Greater;
condition1.FirstFormula = "10";
condition1.BackColor = ExcelKnownColors.Light_orange;
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Col(1));
```


Based on Formula or Cell References

To format the cells based on Formula or Cell References, define the conditional format type as **Formula** and other formatting options such as formula, background color etc., to the specified cell or range. Finally, invalidate the cells to refresh the view.

C#

```
var worksheet = spreadsheet.Workbook.Worksheets[0];
IConditionalFormats condition =
worksheet.Range["A1:A100"].ConditionalFormats;
IConditionalFormat condition1 = condition.AddCondition();
condition1.FormatType = ExcelCFTType.Formula;
condition1.FirstFormula = "= (B1+B2) > 50";
condition1.BackColor = ExcelKnownColors.Brown;
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Col(1));
```

Based on SpecificText

To format the cells based on specified text, define the conditional format type as **SpecificText** and other formatting options such as the particular text, operator, background color etc., to the specified cell or range. Finally, invalidate the cells to refresh the view.

C#

```
var worksheet = spreadsheet.Workbook.Worksheets[0];
IConditionalFormats condition =
worksheet.Range["A1:A100"].ConditionalFormats;
IConditionalFormat condition1 = condition.AddCondition();
condition1.FormatType = ExcelCFTType.SpecificText;
condition1.Text = "SYNC";
condition1.Operator = ExcelComparisonOperator.ContainsText;
condition1.BackColor = ExcelKnownColors.Light_orange;
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Col(1));
```

Based on TimePeriod

To format the cells based on time period, define the conditional format type as **TimePeriod** and other formatting options such as the time periods for the date, operator, background color etc., to the specified cell or range. Finally, invalidate the cells to refresh the view.

C#

```
var worksheet = spreadsheet.Workbook.Worksheets[0];
IConditionalFormats condition =
worksheet.Range["A1:A100"].ConditionalFormats;
IConditionalFormat condition1 = condition.AddCondition();
condition1.FormatType = ExcelCFTType.TimePeriod;
condition1.TimePeriodType = CFTTimePeriods.Today;
condition1.BackColor = ExcelKnownColors.Light_orange;
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Col(1));
```

Sample Output

	A	B	D	E
1		HighlightCellRules		
2				
3		GreaterThan >10	12	
4				
5		LessThan <10	2	
6				
7		Between 10 to 20	12	
8				
9		Equal to 10	10	
10				
11		Text Contains	Syncfusion	
12				
13		Dates Occuring "Today"	1/20/2016	




















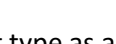
Data Bars

To apply the conditional format based on data bars, define the conditional format type as a **DataBar** and specify the properties associated with DataBars such as bar color, MinPoint, MaxPoint etc., to the specified cell or range. Finally, invalidate that cells to update the view.

C#

```
var worksheet = spreadsheet.Workbook.Worksheets[0];
var conditionalFormats = worksheet.Range["B1:B100"].ConditionalFormats;
var conditionalFormat = conditionalFormats.AddCondition();
conditionalFormat.FormatType = ExcelCFTType.DataBar;
conditionalFormat.DataBar.BarColor = Color.FromArgb(255, 214, 0, 123);
conditionalFormat.DataBar.MinPoint.Type = ConditionValueType.LowestValue;
conditionalFormat.DataBar.MaxPoint.Type = ConditionValueType.HighestValue;
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Col(2));
```

Sample Output

	A	B	D	E	F	G	H	I
1	Mountain	Location	Default Value	Percent	Percentile	Bar only	Two column bar chart	
2	Aconcagua	Argentina	6,960	6,960	6,960		6,960	
3	Annapurna I	Nepal	8,078	8,078	8,078		8,078	
4	Ararat	Turkey	5,122	5,122	5,122		5,122	
5	Citlaltepetl	Mexico	5,700	5,700	5,700		5,700	
6	Cook	New Zealand	3,764	3,764	3,764		3,764	
7	Everest	India-Nepal	8,848	8,848	8,848		8,848	
8	Grand Teton	Wyoming, U.S.	4,196	4,196	4,196		4,196	
9	Huascaran	Peru	6,768	6,768	6,768		6,768	
10	Jaja	New Guinea	5,029	5,029	5,029		5,029	
11	Jebel Toubkal	Morocco	4,164	4,164	4,164		4,164	

Color Scales

To apply the conditional format based on color scales, define the conditional format type as a **ColorScale** and specify the other properties associated with ColorScale such as condition count,color criteria etc.,to the specified cell or range. Finally,invalidate that cells to update the view.

C#

```
var worksheet = spreadsheet.Workbook.Worksheets[0];
var conditionalFormats = worksheet.Range["C2:C100"].ConditionalFormats;
var conditionalFormat = conditionalFormats.AddCondition();
conditionalFormat.FormatType = ExcelCFTType.ColorScale;
conditionalFormat.ColorScale.SetConditionCount(2);
conditionalFormat.ColorScale.Criteria[0].FormatColorRGB =
Color.FromArgb(255, 99, 190, 123);
conditionalFormat.ColorScale.Criteria[1].FormatColorRGB =
Color.FromArgb(255, 90, 138, 198);
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Col(3));
```

Sample Output

	A	B	C	D	E	F
1	Country	Region	Dairy	Produce	Grain	Total Sales
2	Finland	North	\$1	\$1,000	\$10,000	\$11,001
3	Finland	South	\$2	\$2,000	\$20,000	\$22,002
4	Spain	North	\$3	\$3,000	\$30,000	\$33,003
5	Spain	South	\$4	\$4,000	\$40,000	\$44,004
6	France	East	\$5	\$5,000	\$50,000	\$55,005
7	France	West	\$6	\$6,000	\$60,000	\$66,006
8	Norway	North	\$7	\$7,000	\$70,000	\$77,007
9	Norway	South	\$8	\$8,000	\$80,000	\$88,008
10	Denmark	East	\$9	\$9,000	\$90,000	\$99,009
11	Denmark	West	\$10	\$10,000	\$100,000	\$110,010




Icon Sets

To apply the conditional format for Icon sets, define the conditional format type as **IconSet** and the properties associated with IconSet such as the type of the icon,criteria etc., to the specified cell or range. Finally, invalidate that cells to update the view.

C#

```
vvar worksheet = spreadsheet.Workbook.Worksheets[0];
var conditionalFormats = worksheet.Range["D2:D100"].ConditionalFormats;
var conditionalFormat = conditionalFormats.AddCondition();
conditionalFormat.FormatType = ExcelCfType.IconSet;
conditionalFormat.IconSet.IconSet = ExcelIconSetType.ThreeSymbols;
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Col(4));
```

Sample Output

	A	B	C	D	E	F
2		BID #	DESCRIPTION	VAL1	VAL2	VAL3
3		1	Bid number 1	 10	 1	 100
4		2	Bid number 2	 20	 2	 200
5		3	Bid number 3	 30	 3	 300
6		4	Bid number 4	 40	 4	 400
7		5	Bid number 5	 50	 5	 500
8		6	Bid number 6	 60	 6	 600
9		7	Bid number 7	 70	 7	 700
10		8	Bid number 8	 80	 8	 800

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Themes in WPF Spreadsheet (SfSpreadsheet)

SfSpreadsheet supports the following built-in themes.

MaterialLight *MaterialDark* *Office2019Colorful* *Office2019Black* *Office2019White*
Office2019DarkGray *FluentLight* *FluentDark* * *SystemTheme*

Refer to the below links to apply themes for the SfSpreadsheet,

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

	A	B	C	D	E	F
	Employee ID	Employee Name	Gender	Title	Birth Date	Sick Leave
1						
2	1001	Teresa Atkinson	Male	Engineering Manager	5/19/2015	
3	1002	Mae Anderson	Male	Production Technician - WC45	3/17/2013	
4	1003	Ronald Adina	Female	Information Services Manager	3/21/2005	
5	1004	Chris Ashton	Female	Production Technician - WC50	7/27/2014	
6	1005	Kim Akers	Male	Production Supervisor - WC50	8/22/2011	
7	1006	Marvin Allen	Male	Production Technician - WC50	8/8/2013	
8	1007	Francois Ferrier	Male	Shipping and Receiving Clerk	1/14/2012	
9	1008	James Aguilar	Female	Human Resources Manager	10/15/2010	
10	1009	Francois Ferrier	Male	Recruiter	10/6/2010	
11	1010	Sabria Appelbaum	Male	Marketing Assistant	1/17/2015	
12	1011	Stephen Ayers	Male	Tool Designer	4/15/2005	
13	1012	Cecil Allison	Male	Production Technician - WC50	3/12/2012	
14	1013	Phyllis Allen	Male	Senior Tool Designer	4/25/2011	

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Sorting and Filtering in WPF Spreadsheet (SfSpreadsheet)

This section explains about sorting and filtering functionalities in the SfSpreadsheet.

Filtering

By default, Filtering support will be enabled in the SfSpreadsheet, but if you want to disable the Filtering in SfSpreadsheet, set the `AllowFiltering` property of the `SfSpreadsheet` to be false.

XML

```
<syncfusion:SfSpreadsheet x:Name="spreadsheet" AllowFiltering="False"/>
```

C#

```
spreadsheet.AllowFiltering = false;
```

Programmatic Sorting and Filtering

Sorting

Programmatically sort the data while importing the workbook by using `XlsIO` in the `WorkbookLoaded` event of `SfSpreadsheet`.

C#

```
spreadsheet.WorkbookLoaded += spreadsheet_WorkbookLoaded;
void spreadsheet_WorkbookLoaded(object sender, WorkbookLoadedEventArgs args)
{
    IRange filterRange = spreadsheet.Workbook.ActiveSheet.Range["A1:D9"];
}
```

```
spreadsheet.Workbook.ActiveSheet.AutoFilters.FilterRange = filterRange;  
IDataSort sorter = spreadsheet.Workbook.CreateDataSorter();  
sorter.SortRange = spreadsheet.ActiveSheet.Range["A1:D9"];  
ISortField sortField = sorter.SortFields.Add(1, SortOn.Values,  
OrderBy.Ascending);  
sorter.Sort();  
}
```

Filtering

Programmatically filter the data while importing the workbook by using XlsIO in the **WorkbookLoaded** event of SfSpreadsheet.

C#

```
spreadsheet.WorkbookLoaded += spreadsheet_WorkbookLoaded;  
void spreadsheet_WorkbookLoaded(object sender, WorkbookLoadedEventArgs args)  
{  
    IRange filterRange = spreadsheet.Workbook.ActiveSheet.Range["A1:D9"];  
    spreadsheet.Workbook.ActiveSheet.AutoFilters.FilterRange = filterRange;  
    IAutoFilter filter = spreadsheet.Workbook.ActiveSheet.AutoFilters[0];  
    filter.AddTextFilter("1");  
}
```

For more reference, please go through the [XlsIO](#) UG documentation.

Note: If sorting or filtering has been applied programmatically at runtime, then [refresh the view](#) to update it in **SpreadsheetGrid**. But this changes will not be updated in Filter popup even after refreshing the view.

Unsupported Features

Currently SfSpreadsheet does not have support for following features.

- Advanced filtering
- Table filtering
- Multi-column sorting

Limitations

Sorting

- In Microsoft Excel, sorting label should be varied in filter popup based on the type of values in a column (For e.g, "Sort Smallest to Largest" for numeric values, "Sort A to Z" for string values, etc.). But in Spreadsheet, sort label should not be varied based on values due to improve the loading performance of filter popup.
- Sort Ascending or Sort Descending label is not checked in the filter popup, if the column is sorted in Microsoft Excel. Because currently XlsIO do not have support to fetch the sorted order while importing the workbook.

Filtering

- If the filter applied in Microsoft Excel, then the filter will be cleared from all columns while clearing the filter from any one column. Because unable to fetch the filtering order.
- While importing the workbook, checked and unchecked items are only displayed in the right most filtered column. Because unable to fetch the filtering order.

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Formulas in WPF Spreadsheet (SfSpreadsheet)

SfSpreadsheet calculation engine offers automated calculation over a formula, expression, or cross sheet references. SfSpreadsheet calculation engine is preloaded with 409 formulas covering a broad range of business functions.

Adding Formula into cell

To add formulas into a cell programmatically, use [SetCellValue](#) method of [SpreadsheetGrid](#) should be invoked and then invalidate that cell to update the view.

C#

```
var range = spreadsheet.ActiveSheet.Range["A2"];
spreadsheet.ActiveGrid.SetCellValue(range, "=SUM(B1:B2)");
spreadsheet.ActiveGrid.InvalidateCell(2, 1);
```

Named Ranges

Named Ranges are the defined names that represents a cell, range of cells, formula, or constant value or table. Each name have a scope of either to a specific worksheet or to the entire workbook.

Define named ranges at runtime

SfSpreadsheet allows the user to define/add the named ranges at runtime by using [AddNamedRange](#) method.

C#

```
spreadsheet.AddNamedRange("SampleName", "A3:B3", "Sheet1");
```

Edit or remove named ranges at runtime

SfSpreadsheet allows the user to edit the named ranges at runtime by [EditNamedRange](#) method and remove the named ranges at runtime by [DeleteNamedRange](#) method

C#

```
//To Edit the named ranges,
IName name = spreadsheet.Workbook.Names["Sample"];
spreadsheet.EditNamedRange("Test", "A3:B3", name);
//To remove the named ranges,
IName name = spreadsheet.Workbook.Names["Sample"];
spreadsheet.DeleteNamedRange(name);
```


Supported functions

Following is a list of functions that are supported by SfSpreadsheet

Database Functions

Name	Description
DCOUNT	Returns the number of cells containing numbers in a field of a list or database that satisfy specified conditions
DCOUNTA	Returns the number of non-blank cells in a field of a list or database, that satisfy specified conditions
DAVERAGE	Calculates the average of values in a field of a list or database, that satisfy specified conditions
DGET	Returns a single value from a field of a list or database, that satisfy specified conditions
DMAX	Returns the maximum value from a field of a list or database, that satisfy specified conditions
DMIN	Returns the minimum value from a field of a list or database, that satisfy specified conditions
DSTDEVP	Calculates the standard deviation (based on an entire population) of values in a field of a list or database, that satisfy specified conditions
DSTEVP	Calculates the standard deviation (based on a sample of a population) of values in a field of a list or database, that satisfy specified conditions
DVARP	Calculates the variance (based on an entire population) of values in a field of a list or database, that satisfy specified conditions
DVAR	Calculates the variance (based on a sample of a population) of values in a field of a list or database, that satisfy specified conditions

Date and Time Functions

Name	Description
DATE	Returns a date, from a user-supplied year, month and day
DATEVALUE	Converts a text string showing a date, to an integer that represents the date in Excel's date-time code
DAY	Returns the day (of the month) from a user-supplied date

DAYS360	Calculates the number of days between 2 dates, based on a 360-day year (12 x 30 months)
HOUR	Returns the hour part of a user-supplied time
MINUTE	Returns the minute part of a user-supplied time
SECOND	Returns the seconds part of a user-supplied time
MONTH	Returns the month from a user-supplied date
NOW	Returns the current date & time
TIME	Returns a time, from a user-supplied hour, minute and second
TIMEVALUE	Converts a text string showing a time, to a decimal that represents the time in Excel
TODAY	Returns today's date
WEEKDAY	Returns an integer representing the day of the week for a supplied date
YEAR	Returns the year from a user-supplied date
DAYS	Calculates the number of days between 2 dates
EDATE	Returns a date that is the specified number of months before or after an initial supplied start date
EOMONTH	Returns a date that is the last day of the month which is a specified number of months before or after an initial supplied start date
ISOWEEKNUM	Returns the ISO week number of the year for a given date
NETWORKDAYS.INTL	Returns the number of whole network days (excluding weekends & holidays), between two supplied dates, using parameters to specify weekend days
WEEKNUM	Returns an integer representing the week number (from 1 to 53) of the year from a user-supplied date
WORKDAY	Returns a date that is a supplied number of working days (excluding weekends & holidays) ahead of a given start date
WORKDAY.INTL	Returns a date that is a supplied number of working days (excluding weekends & holidays) ahead of a given start date, using supplied parameters to specify weekend days

YEARFRAC	Calculates the fraction of the year represented by the number of whole days between two dates
----------	---

Engineering Functions

Name	Description
DEC2BIN	Converts a decimal number to binary
DCE2OCT	Converts a binary number to octal
DEC2HEX	Converts a decimal number to hexadecimal
BIN2DEC	Converts a binary number to hexadecimal
BIN2OCT	Converts a binary number to octal
BIN2HEX	Converts a binary number to hexadecimal
HEX2BIN	Converts a hexadecimal number to binary
HEX2DEC	Converts a hexadecimal number to a decimal
HEX2OCT	Converts a hexadecimal number to octal
OCT2BIN	Converts octal number to binary
OCT2DEC	Converts octal number to a decimal
OCT2HEX	Converts octal number to hexadecimal
IMABS	Returns the absolute value (the modulus) of a complex number
IMAGINARY	Returns the imaginary coefficient of a complex number
IMREAL	Returns the real coefficient of a complex number
COMPLEX	Converts user-supplied real and imaginary coefficients into a complex number
IMSUM	Calculates the sum of two complex numbers
IMSUB	Subtracts two complex numbers
IMPRODUCT	Returns the product of up to 255 supplied complex numbers

IMDIV	Returns the quotient of two supplied complex numbers
IMCONJUGATE	Returns the complex conjugate of a complex number
IMSQRT	Returns the square root of a complex number
IMARGUMENT	Returns the argument $\hat{\sim}$ (an angle expressed in radians) of a complex number
IMSIN	Returns the sine of a complex number
IMCSC	Returns the cosecant of a complex number
IMCOS	Returns the cosine of a complex number
IMSEC	Returns the secant of a complex number
IMTAN	Returns the tangent of a complex number
IMCOT	Returns the cotangent of a complex number
IMSINH	Returns the hyperbolic sine of a complex number
IMCSCH	Returns the hyperbolic cosecant of a complex number
IMCOSH	Returns the hyperbolic cosine of a complex number
IMSECH	Returns the hyperbolic secant of a complex number
IMLOG10	Returns the base-10 logarithm of a complex number
IMLOG2	Returns the base-2 logarithm of a complex number
IMLN	Returns the natural logarithm of a complex number
IMEXP	Returns the exponential of a complex number
IMPOWER	Calculates a complex number raised to a supplied power
GESTEP	Tests whether a number is greater than a supplied threshold value
DELTA	Tests whether two supplied numbers are equal
BITAND	Returns a Bitwise 'And' of two numbers
BITOR	Returns a Bitwise 'Or' of two numbers

BITXOR	Returns a Bitwise 'Exclusive Or' of two numbers
BITLSHIFT	Returns a number shifted left by a specified number of bits
BITRSHIFT	Returns a number shifted right by a specified number of bits
ERF	Returns the error function integrated between two supplied limits
ERF.PRECISE	Returns the error function integrated between 0 and a supplied limit
ERFC.PRECISE	Returns the complementary error function integrated between a supplied lower limit and infinity
BESSELI	Calculates the modified Bessel function $I_n(x)$
BESSELJ	Calculates the Bessel function $J_n(x)$
BESSELY	Calculates the modified Bessel function $Y_n(x)$
BESSELK	Calculates the modified Bessel function $K_n(x)$
CONVERT	Converts a number from one measurement system to another

Financial Functions

Name	Description
DB	Calculates the depreciation of an asset for a specified period, using the fixed-declining balance method
DDB	Calculates the depreciation of an asset for a specified period, using the double-declining balance method, or some other user-specified method
FV	Calculates the future value of an investment with periodic constant payments and a constant interest rate
IPMT	Calculates the interest payment for a given period of an investment, with periodic constant payments and a constant interest rate
IRR	Calculates the internal rate of return for a series of cash flows
XIRR	Calculates the internal rate of return for a schedule of cash flows
ISPMT	Returns the interest paid during a specified period of an investment

MIRR	Calculates the internal rate of return for a series of periodic cash flows, considering the cost of the investment and the interest on the reinvestment of cash
NPER	Returns the number of periods for an investment with periodic constant payments and a constant interest rate
NPV	Calculates the net present value of an investment, based on a supplied discount rate, and a series of future payments and income
PMT	Calculates the payments required to reduce a loan, from a supplied present value to a specified future value
PPMT	Calculates the payment on the principal for a given investment, with periodic constant payments and a constant interest rate
PV	Calculates the present value of an investment (i.e. the total amount that a series of future payments is worth now)
RATE	Calculates the interest rate required to pay off a specified amount of a loan, or reach a target amount on an investment over a given period
SLN	Returns the straight-line depreciation of an asset for one period
SYD	Returns the sum-of-years' digits depreciation of an asset for a specified period
VDB	Returns the depreciation of an asset for a specified period, (including partial periods), using the double-declining balance method or another user-specified method
DOLLARDE	Converts a dollar price expressed as a fraction, into a dollar price expressed as a decimal
DOLLARFR	Converts a dollar price expressed as a decimal, into a dollar price expressed as a fraction
DURATION	Calculates the Macaulay duration of a security with an assumed par value of \$100
RRI	Calculates an equivalent interest rate for the growth of an investment
FVSCHEDULE	Calculates the future value of an initial principal, after applying a series of compound interest rates
DISC	Calculates the discount rate for a security
INTRATE	Calculates the interest rate for a fully invested security
CUMIPMT	Calculates the cumulative interest paid between two specified periods
CUMPRINC	Calculates the cumulative principal paid on a loan, between two specified periods

RECEIVED	Calculates the amount received at maturity for a fully invested Security
----------	--

Information Functions

Name	Description
ISERROR	Checks whether the value is an error and returns true or false
ISNUMBER	Checks whether the value is number and returns true or false
ISLOGICAL	Checks whether a value is logical value(TRUE/FALSE) and returns true or false
ISNA	Checks whether a value is #N/A and returns true or false
ISERR	Checks whether the value is an error except #N/A and returns true or false
ISBLANK	Checks whether the reference is to an empty cell and returns true or false
ISTEXT	Checks whether the value is text and returns true or false
ISNONTEXT	Checks whether the value is not text(blank cells are not text) and returns true or false
ISEVEN	Returns true if number is even
CONCATENATE	Joins together two or more text strings
DOLLAR	Converts a number to text using currency format
LEN	Returns the length of a supplied text string
FIXED	Rounds a supplied number to a specified number of decimal places, and then converts this into text
ISODD	Returns true if number is odd
ERROR.TYPE	Tests a supplied value and returns an integer relating to the supplied value's error type
N	Converts a non-number value to a number, a date to a serial number, the logical valueÂ TRUEÂ to 1 and all other values to 0
NA	Returns the Excel #N/A error
CELL	Returns information about the contents, formatting or location of a given cell
INFO	Returns information about the the current operating environment

TYPE	Returns information about the data type of a supplied value
ISFORMULA	Tests if a supplied cell contains a formula and if so, returns TRUE; Otherwise, returns FALSE

Logical Functions

Name	Description
AND	Tests a number of user-defined conditions and returnsÂ TRUEÂ ifÂ ALLÂ of the conditions evaluate to TRUE, orFALSEÂ otherwise
OR	Tests a number of user-defined conditions and returnsÂ TRUEÂ ifÂ ANYÂ of the conditions evaluate to TRUE, orFALSEÂ otherwise
IF	Tests a user-defined condition and returns one result if the condition is TRUE, and another result if the condition is FALSE
IFERROR	Tests if an initial supplied value (or expression) returns an error, and if so, returns a supplied value; Otherwise the function returns the initial value.
FALSE	Simply returns the logical valueÂ FALSE
TRUE	Simply returns the logical valueÂ TRUE
NOT	Returns a logical value that is the opposite of a user supplied logical value or expression

Lookup & Reference Functions

Name	Description
OFFSET	Returns a reference to a range of cells that is a specified number of rows and columns from an initial supplied range
HLOOKUP	Looks up a supplied value in the first row of a table, and returns the corresponding value from another row
VLOOKUP	Looks up a supplied value in the first column of a table, and returns the corresponding value from another column
MATCH	Finds the relative position of a value in a supplied array
COLUMN	Returns the column number of a supplied range, or of the current cell
ROW	Returns the row number of a supplied range, or of the current cell

INDIRECT	Returns a cell or range reference that is represented by a supplied text string
AREAS	Returns the number of areas in a supplied range
COLUMNS	Returns the number of columns in a supplied range
FORMULATEXT	Returns a formula as a string
HYPERLINK	Creates a hyperlink to a document in a supplied location
ROW	Returns the row number of a supplied range, or of the current cell
ROWS	Returns the number of rows in a supplied range
SHEET	Returns the sheet number of the referenced sheet
TRANSPOSE	Performs a transpose transformation on a range of cells (i.e. transforms a horizontal range of cells into a vertical range and vice versa)
SHEETS	Returns the number of sheets in reference

Math & Trigonometry functions

Name	Description
ABS	Returns the absolute value of a number
ACOS	Returns the arccosine of a number
ACOSH	Returns the inverse hyperbolic cosine of a number
ASIN	Returns the arcsine of a number
ASINH	Returns the inverse hyperbolic sine of a number
ATAN	Returns the arctangent of a number
ATAN2	Returns the arctangent from x- and y-coordinates
ATANH	Returns the inverse hyperbolic tangent of a number
SUM	Adds its arguments
PI	Returns the value of pi

POWER	Returns the result of a number raised to a power
POW	Returns the result of a number raised to a power
SUBTOTAL	Returns a subtotal in a list or database
COS	Returns the cosine of a number
SIN	Returns the sine of the given angle
COSH	Returns the hyperbolic cosine of a number
SINH	Returns the hyperbolic sine of a number
TANH	Returns the hyperbolic tangent of a number
TAN	Returns the tangent of a number
ACOT	Returns the arc cotangent of a number, in radians in the range 0 to Pi
ACOTH	Returns the inverse hyperbolic cotangent of a number
SIGN	Returns the sign of a number
SQRT	Returns a positive square root
ROUND	Rounds a number to a specified number of digits
LOG	Returns the logarithm of a number to a specified base
LOG10	Returns the base-10 logarithm of a number
EXP	Returns e raised to the power of a given number
CEILING	Rounds a number to the nearest integer or to the nearest multiple of significance
CEILING.MATH	Returns the RoundUp of the given number to the given significance
COLUMNS	Returns the number of columns of the passed in cell reference
FLOOR	Rounds a number down, toward zero
PRODUCT	Multiplies its arguments
MOD	Returns the remainder from division

TRUNC	Truncates a number to an integer
INT	Rounds a number down to nearest integer
ISEVEN	Returns true if the number is even
SUMPRODUCT	Returns the sum of the products of corresponding array components
EXP	Returns e raised to the power of a given number
INT	Rounds a number down to the nearest integer
RAND	Returns an evenly distributed random number ≥ 0 and < 1
COMBIN	Returns the number of combinations for a given number of objects
DEGREES	Converts radians to degrees
EVEN	Rounds a number up to the nearest even integer
FACT	Returns the factorial of a number
LN	Returns the natural logarithm of a number
ODD	Rounds a number up to the nearest odd integer
RADIANS	Converts degrees to radians
ROUND DOWN	Rounds a number down, toward zero
ROUND UP	Rounds a number up, away from zero
MROUND	Returns a number rounded to the desired multiple
MULTINOMIAL	Returns the multinomial of a set of numbers
QUOTIENT	Returns the integer portion of a division
FACTDOUBLE	Returns the double factorial of a number
GCD	Returns the greatest common divisor
LCM	Returns the least common multiple
SQRTPI	Returns the square root of (number * pi)

ROMAN	Converts an Arabic numeral to Roman, as text
SUMSQ	Returns the sum of the squares of the arguments
SUMX2MY2	Returns the sum of the difference of squares of corresponding values in two arrays
SUMX2PY2	Returns the sum of the sum of squares of corresponding values in two arrays
SUMXMY2	Returns the sum of squares of differences of corresponding values in two arrays
SUMIFS	Adds the cells specified by a given set of conditions or criteria
SEC	Returns the secant of an angle
SECH	Returns the hyperbolic secant of an angle
COT	Returns the cotangent of an angle
COTH	Returns the hyperbolic cotangent of a number
CSC	Returns the cosecant of an angle
CSCH	Returns the hyperbolic cosecant of an angle
TRUNCATE	Truncates a number to an integer
COMBINA	Returns the number of combinations for a given number of objects
BASE	Converts number into text representation
DECIMAL	Converts text representation of a number in a given base into decimal number
ARABIC	Converts a roman numeral to Arabic
CEILING.MATH	Rounds a number to the nearest integer or to the nearest multiple of significance
MDETERM	Returns the matrix determinant of an array
MMULT	Returns the matrix product of two arrays
MINVERSE	Returns the matrix inverse of an array
MUNIT	Returns the unit matrix for the specified dimension

Statistical functions

Name	Description
AVG	Returns the average of its arguments
AVERAGE	Returns the average of its arguments
MAX	Returns the maximum value in a list of arguments
MIN	Returns the minimum value in a list of arguments
MAXA	Returns the maximum value in a list of arguments, including numbers, text, and logical values
MINA	Returns the smallest value in a list of arguments, including numbers, text, and logical values
MEDIAN	Returns the median of the given numbers
CONFIDENCE.T	Returns the confidence interval for a population mean
SKEW.P	Returns the skewness of a distribution
COVARIANCE.P	Returns population covariance, the average of the products deviation for each data point pair in two data sets.
COVARIANCE.S	Returns the sample covariance, the average of the products deviation for each data point pair in two data sets.
PERCENTILE.EXC	Returns the Kth percentile of the values in a range, where K is in the range 0É.1 exclusive
PERCENTILE.INC	Returns the Kth percentile of the values in a range, where K is in the range 0É.1 inclusive
PERCENTRANK.EXC	Returns the rank of value in dataset as a percentage of the data set as percentage (0É.1, exclusive) of the dataset
PERCENTRANK.INC	Returns the rank of value in dataset as a percentage of the data set as percentage (0É.1, inclusive) of the dataset
STDEV.P	Calculates standard deviation based on the entire population
STDEV.S	Estimates standard deviation based on a sample
PERMUTATIONA	Returns the number of permutations for a given number of objects

NORM.DIST	Returns the normal cumulative distribution
NORM.INV	Returns the inverse of the normal cumulative distribution
NORM.S.DIST	Returns the standard normal cumulative distribution
NORM.S.INV	Returns the inverse of the standard normal cumulative distribution
WEIBULL.DIST	Returns the Weibull distribution
EXPON.DIST	Returns the exponential distribution
GAMMA.DIST	Returns the gamma distribution
GAMMA.INV	Returns the inverse of the gamma cumulative distribution
GAMMALN.PRECISE	Returns the natural logarithm of the gamma function, $\Gamma(x)$
T.INV	Returns the left-tailed inverse of the Student's t-distribution
F.INV.RT	Returns the inverse of the right-tailed F probability distribution for two data sets
BINOM.INV	Returns the smallest value for which the cumulative binomial distribution is greater than or equal to a criterion value
HYPGEOM.DIST	Returns the hypergeometric distribution
LOGNORM.DIST	Returns the cumulative log-normal distribution
LOGNORM.INV	Returns the inverse of the lognormal distribution
CONFIDENCE.NORM	Returns the confidence interval for a population mean, using a normal distribution
CHISQ.DIST.RT	Returns the right-tailed probability of the chi-squared distribution
F.DIST	Returns the F probability distribution
F.DIST.RT	Returns the right-tailed F probability distribution for two data sets
CHISQ.TEST	Returns the chi-squared statistical test for independence
CHISQ.INV	Returns the inverse of the left-tailed probability of the chi-squared distribution
CHISQ.INV.RT	Returns the inverse of the right-tailed probability of the chi-squared distribution

BINOM.DIST	Returns the individual term binomial distribution probability
Z.TEST	Returns the one-tailed probability value of a z-test
RANK.AVG	Returns the statistical rank of a given value, within a supplied array of values (if more than one value has same rank, the average rank is returned)
RANK.EQ	Returns the Mode (the most frequently occurring value) of a list of supplied numbers (if more than one value has same rank, the top rank of that set is returned)
NEGBINOM.DIST	Returns the negative binomial distribution
POISSON.DIST	Returns the Poisson distribution
QUARTILE.EXC	Returns the specified quartile of a set of supplied numbers, based on percentile value 0 - 1 (exclusive)
QUARTILE.INC	Returns the specified quartile of a set of supplied numbers, based on percentile value 0 - 1 (inclusive)
AVEDEV	Returns the average of the absolute deviations of data points from their mean
AVERAGEA	Returns the Average of a list of supplied numbers, counting text and the logical value FALSE as the value 0 and counting the logical value TRUE as the value 1
GAMMALN	Calculates the natural logarithm of the gamma function for a supplied value
GAMMADIST	Returns the gamma distribution
GAMMAINV	Returns the inverse gamma cumulative distribution
GEOMEAN	Returns the geometric mean of a set of supplied numbers
HARMEAN	Returns the harmonic mean of a set of supplied numbers
HYPGEOMDIST	Returns the hypergeometric distribution
INTERCEPT	Calculates the best fit regression line, through a supplied series of x- and y- values and returns the value at which this line intercepts the y-axis
BINOMDIST	Returns the individual term binomial distribution probability
CHIDIST	Returns the right-tailed probability of the chi-squared distribution

CHIINV	Returns the inverse of the right-tailed probability of the chi-squared distribution
CHITEST	Returns the chi-squared statistical test for independence
NORMDIST	Returns the normal cumulative distribution
NORMINV	Returns the inverse of the normal cumulative distribution
NORMSINV	Returns the inverse of the standard normal cumulative distribution
NORMSDIST	Returns the standard normal cumulative distribution
CONFIDENCE	Returns the confidence interval for a population mean, using a normal distribution
CORREL	Returns the correlation coefficient between two sets of values
COUNT	Returns the number of numerical values in a supplied set of cells or values
COUNTA	Returns the number of non-blanks in a supplied set of cells or values
COUNTBLANK	Returns the number of blank cells in a supplied range
COUNTIF	Returns the number of cells (of a supplied range), that satisfy a given criteria
COVAR	Returns population covariance (i.e. the average of the products of deviations for each pair within two supplied data sets)
CRITBINOM	Returns the smallest value for which the cumulative binomial distribution is greater than or equal to a criterion value
DEVSQ	Returns the sum of the squares of the deviations of a set of data points from their sample mean
EXPONDIST	Returns the exponential distribution
FDIST	Returns the F probability distribution (probability density or cumulative distribution function)
FINV	Returns the inverse of the right-tailed F probability distribution for two data sets
FISHER	Returns the Fisher transformation
FISHERINV	Returns the inverse of the Fisher transformation

FORECAST	Predicts a future point on a linear trend line fitted to a supplied set of x- and y-values
KURT	Returns the kurtosis of a data set
LARGE	Returns the Kth LARGEST value from a list of supplied numbers, for a given value K
LOGNORMDIST	Returns the cumulative log-normal distribution
LOGINV	Returns the inverse of the lognormal distribution
MODE	Returns the Mode (the most frequently occurring value) of a list of supplied numbers
NEGBINOMDIST	Returns the negative binomial distribution
PEARSON	Returns the Pearson product moment correlation coefficient
PERCENTILE	Returns the K'th percentile of values in a supplied range, where K is in the range 0 - 1 (inclusive)
PERCENTILERANK	Returns the rank of a value in a data set, as a percentage (0 - 1 inclusive)
PERMUT	Returns the number of permutations for a given number of objects
POISSON	Returns the Poisson distribution
PROB	Returns the probability that values in a supplied range are within given limits
QUARTILE	Returns the specified quartile of a set of supplied numbers, based on percentile value 0 - 1 (inclusive)
RANQ	Returns the Mode (the most frequently occurring value) of a list of supplied numbers (if more than one value has same rank, the top rank of that set is returned)
RSQ	Returns the square of the Pearson product moment correlation coefficient
SKEW	Returns the skewness of a distribution
SLOPE	Returns the slope of the linear regression line through a supplied series of x- and y- values
SMALL	Returns the Kth SMALLEST value from a list of supplied numbers, for a given value K

STANDARDIZE	Returns a normalized value
STDEV	Returns the standard deviation of a supplied set of values (which represent a sample of a population)
STDEVA	Returns the standard deviation of a supplied set of values (which represent a sample of a population), counting text and the logical value FALSE as the value 0 and counting the logical value TRUE as the value 1
STDEVP	Returns the standard deviation of a supplied set of values (which represent an entire population)
STDEVPA	Returns the standard deviation of a supplied set of values (which represent an entire population), counting text and the logical value FALSE as the value 0 and counting the logical value TRUE as the value 1
STEYX	Returns the standard error of the predicted y-value for each x in the regression line for a set of supplied x- and y- values
TRIMMEAN	Returns the mean of the interior of a supplied set of values
VAR	Returns the variance of a supplied set of values (which represent a sample of a population)
VARA	Returns the variance of a supplied set of values (which represent a sample of a population), counting text and the logical value FALSE as the value 0 and counting the logical value TRUE as the value 1
VARP	Returns the variance of a supplied set of values (which represent an entire population)
VARPA	Returns the variance of a supplied set of values (which represent an entire population), counting text and the logical value FALSE as the value 0 and counting the logical value TRUE as the value 1
WEIBULL	Returns the Weibull distribution
ZTEST	Returns the one-tailed probability value of a z-test

Text Functions

Name	Description
LEFT	Returns a specified number of characters from the start of a supplied text string

LEN	Returns the length of a supplied text string
TRUNC	Truncates a number to an integer removing decimal part or fractional part
MID	Returns a specified number of characters from the middle of a supplied text string
RIGHT	Returns a specified number of characters from the end of a supplied text string
VALUE	Converts a text string into a numeric value
DOLLAR	Converts a supplied number into text, using a currency format
FIXED	Rounds a supplied number to a specified number of decimal places, and then converts this into text
LOWER	Converts all characters in a supplied text string to lower case
UPPER	Converts all characters in a supplied text string to upper case
TEXT	Converts a supplied value into text, using a user-specified format
TRIM	Removes duplicate spaces, and spaces at the start and end of a text string
CONCATENATE	Joins together two or more text strings
SUBSTITUTE	Substitutes all occurrences of a search text string, within an original text string, with the supplied replacement text
T	Tests whether a supplied value is text and if so, returns the supplied text; If not, returns an empty text string.
CODE	Returns the numeric code for the first character of a supplied string
FINDB	Returns the position of a supplied character or text string from within a supplied text string (case-sensitive)
LEFTB	Returns a specified number of characters from the start of a supplied text string
LENB	Returns the length of a supplied text string
MINB	Returns the smallest value in a set of values. does not ignore logical text and values
RIGHTB	Returns a specified number of characters from the end of a supplied text string
NUMBERVALUE	Converts text to a number, in a locale-independent way

PROPER	Converts all characters in a supplied text string to proper case (i.e. letters that do not follow another letter are upper case and all other characters are lower case)
REPLACE	Replaces all or part of a text string with another string (from a user supplied position)
REPT	Returns a string consisting of a supplied text string, repeated a specified number of times
SEARCHB	Returns the position of a supplied character or text string from within a supplied text string (non-case-sensitive)
UNICHAR	Returns the Unicode character that is referenced by the given numeric value
UNICODE	Returns the number (code point) corresponding to the first character of a supplied text string

Web Functions

Name	Description
ENCODEURL	Returns a URL-encoded string
FILTERXML	Returns data from XML content, using a specified XPath
WEBSERVICE	Returns data from a web service on the Internet or Intranet

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Rows and Columns in WPF Spreadsheet (SfSpreadsheet)

This section explains about the operations related with rows and columns in SfSpreadsheet

Insert Rows and Columns

SfSpreadsheet provides support for dynamically inserting rows and columns into a worksheet.

C#

```
//For Inserting Rows
spreadsheet.ActiveSheet.InsertRow(2, 3);
spreadsheet.ActiveGrid.Model.InsertRows(2, 3);
//For Inserting Cols
spreadsheet.ActiveSheet.InsertColumn(3, 2);
spreadsheet.ActiveGrid.Model.InsertColumns(3, 2);
```

Events

Below events of [SpreadsheetGridModel](#) are triggered while inserting the rows and columns.

- [RowsInserted](#)
- [ColumnsInserted](#)

C#

```
//To notify when rows are inserted
spreadsheet.ActiveGrid.Model.RowsInserted += Model_RowsInserted;
void Model_RowsInserted(object sender, GridRangeInsertedEventArgs e)
{
}
//To notify when Columns are inserted
spreadsheet.ActiveGrid.Model.ColumnsInserted += Model_ColumnsInserted;
void Model_ColumnsInserted(object sender, GridRangeInsertedEventArgs e)
{
}
```

Delete Rows and Columns

SfSpreadsheet provides support for deleting rows and columns from a worksheet.

C#

```
//For Deleting Rows
spreadsheet.ActiveSheet.DeleteRow(5, 2);
spreadsheet.ActiveGrid.Model.RemoveRows(5, 2);
//For Deleting Cols
spreadsheet.ActiveSheet.DeleteColumn(3, 2);
spreadsheet.ActiveGrid.Model.RemoveColumns(3, 2);
```

Events

Below events of [SpreadsheetGridModel](#) are triggered while deleting the rows and columns.

- [RowsRemoved](#)
- [ColumnsRemoved](#)

C#

```
//To notify when rows are deleted
spreadsheet.ActiveGrid.Model.RowsRemoved += Model_RowsRemoved;
void Model_RowsRemoved(object sender, GridRangeRemovedEventArgs e)
{
}
//To notify when columns are deleted
spreadsheet.ActiveGrid.Model.ColumnsRemoved += Model_ColumnsRemoved;
void Model_ColumnsInserted(object sender, GridRangeInsertedEventArgs e)
{
}
```

Hide Rows and Columns

SfSpreadsheet provides support to hide rows/columns and this can be done by [HideRow](#) and [HideColumn](#) method

C#

```
//For Hiding Rows,  
spreadsheet.ActiveSheet.HideRow(5);  
spreadsheet.ActiveGrid.RowHeights.SetHidden(5, 5, true);  
//For Hiding Cols,  
spreadsheet.ActiveSheet.HideColumn(4);  
spreadsheet.ActiveGrid.ColumnWidths.SetHidden(4, 4, true);
```

Unhide Rows and Columns

Unhide the rows/columns in SfSpreadsheet can be done by [ShowRow](#) and [ShowColumn](#) methods.

C#

```
//For Unhiding Rows,  
spreadsheet.ActiveSheet.ShowRow(5, true);  
spreadsheet.ActiveGrid.RowHeights.SetHidden(5, 5, false);  
//For Unhiding Cols,  
spreadsheet.ActiveSheet.ShowColumn(4, true);  
spreadsheet.ActiveGrid.ColumnWidths.SetHidden(4, 4, false);
```

Row Height and Column Width

SfSpreadsheet provides support to adjust the row height and column width. And also can import the adjusted row height and column width from Excel. SfSpreadsheet provides support to fit the row and column based on its contents.

C#

```
//For setting RowHeight for 4th Row  
spreadsheet.ActiveGrid.SetRowHeight(4, 4, 30);  
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Row(4), true);  
//For setting ColumnWidth for 5th Column  
spreadsheet.ActiveGrid.SetColumnWidth(5, 5, 22);  
spreadsheet.ActiveGrid.InvalidateCell(GridRangeInfo.Col(5), true);
```

Note: In case if you insert/delete and hide/unhide the rows/columns inside the Grouping, [RefreshOutlines](#) method must be invoked to refresh/update the Outlines of the Group.

Freeze Rows and Columns

SfSpreadsheet provides support for Freeze panes to keep an area of a worksheet visible while you scroll to another area of the worksheet.

C#

```
//Freeze panes  
//To Freeze 4 rows and 4 columns  
spreadsheet.Workbook.ActiveSheet.Range[4, 4].FreezePanes();  
spreadsheet.ActiveGrid.FrozenRows = 5;  
spreadsheet.ActiveGrid.FrozenColumns = 5;
```

Unfreeze Rows and Columns

SfSpreadsheet provides support to unfreeze the freeze panes in the worksheet of SfSpreadsheet.

C#

```
//Unfreeze panes
```

```
//To Unfreeze 4 rows and 4 columns
spreadsheet.Workbook.ActiveSheet.RemovePanes();
spreadsheet.ActiveGrid.FrozenRows = 1;
spreadsheet.ActiveGrid.FrozenColumns = 1;
```

Auto Fit Rows and Columns

SfSpreadsheet provides support to fit the rows or columns based on its content at run time.

You can fit the rows/columns by calling [AutoFitRows](#) and [AutoFitColumns](#) methods of XlsIO's `IRange`. Also set the adjusted row height and column width into the grid by using [SetRowHeight](#) and [SetColumnWidth](#) methods of `SpreadsheetGrid`.

C#

```
//To AutoFit a single column,
spreadsheet.ActiveSheet.AutofitColumn(2);
spreadsheet.ActiveGrid.SetColumnWidth(2, 2, spreadsheet.ActiveSheet.GetColumnWidthInPixels(2));
//To AutoFit multiple columns,
spreadsheet.ActiveSheet["A1:D100"].AutofitColumns();
for(int i = 1; i <= 4 ; i++)
{
    spreadsheet.ActiveGrid.SetColumnWidth(i, i, spreadsheet.ActiveSheet.GetColumnWidthInPixels(i));
}
//To AutoFit a single row,
spreadsheet.ActiveSheet.AutofitRow(3);
spreadsheet.ActiveGrid.SetRowHeight(3, 3, spreadsheet.ActiveSheet.GetRowHeightInPixels(3));
//To AutoFit multiple rows,
spreadsheet.ActiveSheet["B1:B5"].AutofitRows();
for(int i = 1; i <= 5 ; i++)
{
    spreadsheet.ActiveGrid.SetRowHeight(i, i, spreadsheet.ActiveSheet.GetRowHeightInPixels(i));
}
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Worksheet Management in WPF Spreadsheet (SfSpreadsheet)

This section explains about the operations that are performed with the worksheet.

Insert and Delete worksheet

SfSpreadsheet provides support to insert and delete the worksheets in a workbook.

C#

```
//Insert Sheet
spreadsheet.AddSheet();
//Insert sheet with name
spreadsheet.AddSheet("Sheet4", 3);
//Delete Sheet
spreadsheet.RemoveSheet("Sheet2");
```

Hide and Unhide worksheets

SfSpreadsheet provides support to hide and unhide the worksheets in a workbook.

C#

```
//Hide Sheet
spreadsheet.HideSheet("Sheet 2");
//Unhide Sheet
spreadsheet.UnhideSheet("Sheet 2");
```

Hide or unhide sheet tabs

Spreadsheet provides support to hide and unhide the all worksheet tabs in the workbook using the [ShowSheetTabs](#) property. The Default value is `true`.

C#

```
private void SpreadsheetControl_Loaded(object sender, RoutedEventArgs e)
{
    spreadsheetControl.ShowSheetTabs = false;
}
```

XML

```
<syncfusion:SfSpreadsheet x:Name="spreadsheetControl" ShowSheetTabs="False" />
```

Rename a worksheet

SfSpreadsheet provides support to rename a worksheet in the workbook by using [RenameSheet](#) method. After invoking this method, the sheet tab enters into editing mode and now the users can change the name of the sheet in the tab.

C#

```
//Rename sheet
spreadsheet.RenameSheet("Sheet1");
```

Rename a worksheet programmatically

SfSpreadsheet provides support to rename a worksheet in the workbook programmatically by using [RenameSheet](#) method.

C#

```
//To Rename a sheet programmatically
spreadsheet.RenameSheet("ExistingSheetName", "NewSheetName");
```

Worksheet Protection

Protecting a worksheet

SfSpreadsheet provides support to protect the worksheet with or without password. This helps to prevent a user from modifying the contents of the worksheet. The protection of worksheet can be done with [ExcelSheetProtection](#) options also.

The Protect sheet options are

- LockedCells - Allows the users to select the locked cells of the protected worksheet.
- UnLockedCells - Allows the users to select the unlocked cells of the protected worksheet.
- FormattingCells - Allows the users to format any cell on a protected worksheet.
- FormattingRows - Allows the users to format any row on a protected worksheet.
- FormattingColumns - Allows the users to format any column on a protected worksheet.
- InsertingRows - Allows the users to insert rows on the protected worksheet.
- InsertingColumns - Allows the users to insert columns on the protected worksheet.
- InsertingHyperlinks - Allows the users to insert hyperlinks on the protected worksheet.
- DeletingRows - Allows the users to delete rows on the protected worksheet.
- DeletingColumns - Allows the users to delete columns on the protected worksheet.
- Objects - Allows the users to edit the objects such as Graphic cells like charts,rich textbox, etc.

C#

```
//Protect the sheet with password
spreadsheet.ProtectSheet(spreadsheet.ActiveSheet, "123");
//Protect the sheet with Protection options
spreadsheet.ProtectSheet(spreadsheet.ActiveSheet, "123",
ExcelSheetProtection.FormattingCells);
//Unprotect the sheet
spreadsheet.UnProtectSheet(spreadsheet.ActiveSheet, "123");
```

Protecting a workbook

SfSpreadsheet provides support to protect the structure and windows of a workbook. By protecting the structure, prevent a user from adding or deleting worksheets or from displaying hidden worksheets. By protecting the windows in the workbook, you can control the size of the workbook, etc.

C#

```
// To Protect the Workbook
spreadsheet.Protect(true, true, "123");
//To Unprotect the Workbook
spreadsheet.Unprotect("123");
```

Gridlines

SfSpreadsheet provides support to control the visibility and color of the Gridlines in a worksheet.

C#

```
//To show GridLines
spreadsheet.SetGridLinesVisibility(true);
//To hide GridLines
spreadsheet.SetGridLinesVisibility(false);
```

Headings

SfSpreadsheet provides support to control the visibility of row and column headers in a worksheet

C#

```
//To hide the Header cells visibility
```

```
spreadsheet.SetRowColumnHeadersVisibility(false);
```

Zooming

SfSpreadsheet provides support to zoom in and zoom out of a worksheet view. The property [AllowZooming](#) determines whether to allow zooming or not.

C#

```
//zoom factor
spreadsheet.SetZoomFactor("Sheet1", 200);
```

The Events associated with the Zooming are

- . [ZoomFactorChanged](#)
- . [ZoomFactorChanging](#)

Events

Events	Description
WorkbookCreating	Occurs when the workbook is to be created in SfSpreadsheet.
WorkbookLoaded	Occur when the workbook is loaded in SfSpreadsheet.
WorksheetAdding	Occurs when the worksheet is to be added in SfSpreadsheet.
WorksheetAdded	Occurs when the worksheet is added in SfSpreadsheet.
WorksheetRemoving	Occurs when the worksheet is to be removed from SfSpreadsheet.
WorksheetRemoved	Occurs when the worksheet is removed from SfSpreadsheet.
WorkbookUnloaded	Occurs when the workbook is unloaded or removed from the SfSpreadsheet.
ZoomFactorChanged	Occurs when the zoom factor in SfSpreadsheet is changed.
ZoomFactorChanging	Occurs when the zoom factor in SfSpreadsheet is to be changed.
ResizingColumns	Occurs when performing the resizing columns in SfSpreadsheet.
ResizingRows	Occurs when performing the resizing rows in SfSpreadsheet.

CellCommentOpening	Occurs when opening the comments in the cells of SfSpreadsheet.
CellTooltipOpening	Occurs when opening the tool tips of cells in SfSpreadsheet.
CellContextMenuOpening	Occurs when opening the context menu of the cell in SfSpreadsheet.
QueryRange	Occurs when grid queries for <code>IRange</code> information about a specific cell while rendering.

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Data Management in WPF Spreadsheet (SfSpreadsheet)

This section explains about how to import and export the DataTable in SfSpreadsheet.

Import from DataTable

SfSpreadsheet provides support to import the data. The following list of data can be imported into the worksheet

- Data Table
- Data Column
- Data View
- Business Objects
- Array

To import the data from a data table, you can use [ImportDataTable](#)

method

C#

```
spreadsheet.ActiveSheet.ImportDataTable(data_table, true, 1, 1);
spreadsheet.ActiveGrid.InvalidateCells();
```

For more details regarding importing of data, please refer the [XlsIO UG](#)

Export to DataTable

SfSpreadsheet provides support to export the data. To Export the data from a data table, you can use [ExportDataTable](#)

method

C#

```
IWorksheet sheet = spreadsheet.Workbook.Worksheets[0];
IRange range = sheet.Range["A1:K50"];
DataTable data_table = sheet.ExportDataTable(range,
ExcelExportDataTableOptions.ColumnNames);
```

For more details regarding exporting of data, please refer the [XlsIO UG](#)

Interactive Features in WPF Spreadsheet (SfSpreadsheet)

This section explains about the interactive operations with SfSpreadsheet

Clipboard Operations

SfSpreadsheet provides support for all the clipboard operations to with all the format settings when copied within a workbook.

You can use the following shortcut keys for Clipboard operations like Excel

Operations	Shortcut Keys
Cut	Ctrl + X
Copy	Ctrl + C
Paste	Ctrl + V

The following are a list of paste options used while performing paste operation,

Options	Description
Paste	To paste with all the format options in the source range
Formula	To paste the formulas alone
Keep Source Formatting	To maintain the source range's formatting
Value	To paste the values alone
Format	To paste only the formats alone without pasting the values.
Value & Source Formatting	To maintain the source range original format and paste only values

Note: When the content is copied from external source, SfSpreadsheet does not support the format settings (paste options).

For [Cut](#) Operation,

C#

```
//To perform cut operation for selected ranges
var range = spreadsheet.ActiveGrid.SelectedRanges.ActiveRange;
spreadsheet.ActiveGrid.CopyPaste.Copy(range, true);
//To perform cut operation
spreadsheet.ActiveGrid.CopyPaste.Cut();
```

For [Copy](#) Operation,

C#

```
//To perform copy operation for selected ranges
var range = spreadsheet.ActiveGrid.SelectedRanges.ActiveRange;
spreadsheet.ActiveGrid.CopyPaste.Copy(range, false);
```

```
//To perform Copy operation
spreadsheet.ActiveGrid.CopyPaste.Copy();
```

For [Paste](#) Operation,

C#

```
//To perform paste operation
spreadsheet.ActiveGrid.CopyPaste.Paste();
//To perform paste operation with range and Paste Options
var copyPaste = spreadsheet.ActiveGrid.CopyPaste as SpreadsheetCopyPaste;
copyPaste.Paste(range);
copyPaste.Paste(range, PasteOptions.Paste);
```

Tips: Users can also set their default [PasteOptions](#) while pasting in SfSpreadsheet, by using [DefaultPasteOption](#) property.

Undo/Redo

SfSpreadsheet provides support for Undo/Redo functionality like Microsoft Excel.

The shortcut keys used for Undo/Redo Operations

Operations	Shortcut Keys
Undo	Ctrl + Z
Redo	Ctrl + Y

SfSpreadsheet has [History Manager](#) class that supports the implementation of undo/ redo operations

By default, Undo/Redo operations in SfSpreadsheet is enabled. To disable the Undo/Redo operations, set the [Enabled](#) property of [History Manager](#) to be false.

C#

```
spreadsheet.HistoryManager.Enabled = false;
```

To programmatically, invoke the Undo/Redo operations,

C#

```
spreadsheet.HistoryManager.Enabled = true;
spreadsheet.HistoryManager.Undo();
spreadsheet.HistoryManager.Redo();
```

Context menu

Context menu in SfSpreadsheet is customizable menu which can be used for various functionalities

[TabItem Context menu](#)

TabItem Context menu opens when the user right-click on the sheet tab and contains the menus related to worksheet operations.

By default, TabItem Context menu is enabled in SfSpreadsheet. To disable the TabItem context menu, set the [AllowTabItemContextMenu](#) property to false.

C#

```
spreadsheet.AllowTabItemContextMenu = false;
```

Default TabItem context menu has options like Insert, Delete, Hide/Unhide and Protect sheet. You can also customize the TabItem Context menu by setting [IsCustomTabItemContextMenuEnabled](#) property to be true and you can add your customized menu items.

C#

```
spreadsheet.IsCustomTabItemContextMenuEnabled = true;
spreadsheet.TabItemContextMenu = CustomTabItemContextMenu();
//Custom TabItem ContextMenus
public ContextMenu CustomTabItemContextMenu()
{
    var contextMenu = new ContextMenu();
    var insertRowIcon = new Image() { Source = new BitmapImage(new
    Uri(@"..\..\Icon\insertRow.png", UriKind.Relative)) };
    var insertRow = new MenuItem() { Header = "InsertRow" };
    insertRow.Icon = insertRowIcon;
    insertRow.Click += insertRow_Click;
    var deleteRowIcon = new Image() { Source = new BitmapImage(new
    Uri(@"..\..\Icon\deleteRow.png", UriKind.Relative)) };
    var deleteRow = new MenuItem() { Header = "DeleteRow" };
    deleteRow.Icon = deleteRowIcon;
    deleteRow.Click += deleteRow_Click;
    contextMenu.Items.Add(insertRow);
    contextMenu.Items.Add(deleteRow);
    return contextMenu;
}
```

Cell Context menu

Cell Context menu opens when the user right-click on a worksheet cell or selection of cells in SfSpreadsheet.

By default, Cell Context menu is enabled in SfSpreadsheet. To disable the Cell Context menu, set the [AllowCellContextMenu](#) property as false.

C#

```
spreadsheet.AllowCellContextMenu = false;
```

Users can also customize the Cell Context menu of SfSpreadsheet by using [CellContextMenuOpening](#) Event of [SpreadsheetGrid](#).

Adding the customized menu items in the CellContextMenuOpening Event,

C#

```
spreadsheet.ActiveGrid.CellContextMenuOpening +=
ActiveGrid_CellContextMenuOpening;
void ActiveGrid_CellContextMenuOpening(object sender,
CellContextMenuOpeningEventArgs e)
{
    //Adding Customized Menu item
    MenuItem PasteSpecial = new MenuItem();
```

```
PasteSpecial.Header = "Pastespecial";  
Image paste = new Image() { Source = new BitmapImage(new  
Uri(@"..\..\Icon\paste.png", UriKind.Relative)) };  
PasteSpecial.Icon = paste;  
spreadsheet.ActiveGrid.CellContextMenu.Items.Add(PasteSpecial);  
//Remove the existing Context menu  
spreadsheet.ActiveGrid.CellContextMenu.Items.RemoveAt(2);  
}
```

Tips: Custom Cell Context menu can also be added by assigning the customized menu items to the [CellContextMenu](#) property of SpreadsheetGrid. For your reference, [CustomContextMenu](#)

Cell Comments

SfSpreadsheet provides support for cell comments like in excel to give the reader additional context for the data it contains. You can set the comment height and color for the particular comments at runtime by invoking [CellCommentOpening](#) Event of SpreadsheetGrid

To enable the comment in SfSpreadsheet, set the [ShowComment](#) property of SpreadsheetGrid to true.

C#

```
spreadsheet.ActiveGrid.ShowComment = true;
```

To set the comments for particular cell at run time,

C#

```
spreadsheet.ActiveSheet.Range["E5"].AddComment().Text = "Sample Comment";  
spreadsheet.ActiveGrid.InvalidateCell(5, 5);
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Find and Replace in WPF Spreadsheet (SfSpreadsheet)

This section explains about Find and Replace operations in SfSpreadsheet.

Find

Searches for specific data such as particular number or text according to specified options and returns an IRange representing the cell or null if no cell is found. The various options in Find operation are

- [FindAll](#)
- [FindNext](#)
- [FindConditionalFormatting](#)
- [FindConstants](#)
- [FindFormulas](#)
- [FindDataValidation](#)

The common parameters to be passed in Find functions are,

- The option to specify whether the search can be done within the Workbook([IWorkbook](#)) or Worksheet([IWorksheet](#)).

- The text to be searched.
- The option to specify the direction whether the search can be done either by row wise or column wise using [SearchBy](#) enum.
- The type to specify whether the search can be done either in formulas or values using [ExcelFindType](#) enum.
- For a case sensitive search, pass the parameter as true otherwise you can pass the parameter as false.
- For matching the entire cell content with the search text, pass the parameter as true otherwise you can pass the parameter as false.

Find All

Searches every occurrence of specific data based on the criteria that you are searching for and returns an [IRange](#) list representing the cells in [SfSpreadsheet](#)

C#

```
//Search the entire workbook
var list = spreadsheet.SearchManager.FindAll(spreadsheet.Workbook, "sample",
SearchBy.ByRows, ExcelFindType.Text, false, true);
// To select the matched cell content ranges,
foreach (var cell in list)
{
    spreadsheet.
    ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Cell(cell.Row,
    cell.Column));
}
//Search the particular worksheet
var list =
spreadsheet.SearchManager.FindAll(spreadsheet.Workbook.Worksheets[0],
"sample", SearchBy.ByRows, ExcelFindType.Text, false, true);
// To select the matched cell content ranges,
foreach (var cell in list)
{
    spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Cell(c
    ell.Row, cell.Column));
}
```

Find Next

Searches the first occurrence of specific data which matches the conditions and returns the matched [IRange](#) from the current range that represents the cell.

C#

```
//Search the text in entire workbook in column wise,
var cell = spreadsheet.SearchManager.FindNext(spreadsheet.Workbook,
"sample", SearchBy.ByColumns, ExcelFindType.Text, false, true);
// To move the current cell to matched cell content range,
spreadsheet.ActiveGrid.CurrentCell.MoveCurrentCell(cell.Row, cell.Column);
//Search the formula in particular worksheet in row wise,
var cell =
spreadsheet.SearchManager.FindNext(spreadsheet.Workbook.Worksheets[0],
"sum", SearchBy.ByRows, ExcelFindType.Text, false, false);
// To move the current cell to matched cell content range,
spreadsheet.ActiveGrid.CurrentCell.MoveCurrentCell(cell.Row, cell.Column);
```


Find Conditional Formatting

Searches and returns the **IRange** list which have conditional formatting within the specified worksheet.

C#

```
//Searches the conditional formatting within the worksheet,
var list =
spreadsheet.SearchManager.FindConditionalFormatting(spreadsheet.Workbook.Worksheets[0]);
// To select the matched cell content ranges,
foreach (var cell in list)
{
spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Cell(cell.Row, cell.Column));
}
```

Find Constants

Searches and returns the **IRange** list which have constants within the specified worksheet.

C#

```
//Searches the constants within the worksheet,
var list =
spreadsheet.SearchManager.FindConstants(spreadsheet.Workbook.Worksheets[0]);
// To select the matched cell content ranges,
foreach (var cell in list)
{
spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Cell(cell.Row, cell.Column));
}
```

Find Formulas

Searches and returns the **IRange** list which have formulas within the specified worksheet.

C#

```
//Searches the formulas within the worksheet,
var list =
spreadsheet.SearchManager.FindFormulas(spreadsheet.Workbook.Worksheets[0]);
// To select the matched cell content ranges,
foreach (var cell in list)
{
spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Cell(cell.Row, cell.Column));
}
```

Find Data Validation

Searches and returns the **IRange** list which have data validation within the specified worksheet.

C#

```
//Searches the data validation within the worksheet,
```

```
var list =
spreadsheet.SearchManager.FindDataValidation(spreadsheet.Workbook.Worksheets
[0]);
// To select the matched cell content ranges,
foreach (var cell in list)
{
spreadsheet.ActiveGrid.SelectionController.AddSelection(GridRangeInfo.Cell(c
ell.Row, cell.Column));
}
```

Replace All

Searches and replaces all the texts either in the workbook or worksheet based on the given option.

The parameters to be passed in [ReplaceAll](#) function is,

- The option to specify whether the search can be done within the Workbook([IWorkbook](#)) or Worksheet([IWorksheet](#)) in SfSpreadsheet.
- The text to be searched.
- The text to be replaced.
- For a case sensitive search, pass the parameter as true otherwise you can pass the parameter as false.
- For matching the entire cell content with the search text, pass the parameter as true otherwise you can pass the parameter as false.

C#

```
//Replaces the text in the entire workbook
spreadsheet.SearchManager.ReplaceAll(spreadsheet.Workbook, "sample", "Sync",
false, false);
//Replaces the text in the particular worksheet
spreadsheet.SearchManager.ReplaceAll(spreadsheet.Workbook.Worksheets[0],
"sample", "sync", false, true);
```

Replace

Searches for the text or numbers that you want to change using [FindNext](#) method and once the immediate matched cell has been found, use [SetCellValue](#) method to replace it with specified text or numbers in SfSpreadsheet.

C#

```
//Searches the given text and replaces it with specified text
var cell = spreadsheet.SearchManager.FindNext(spreadsheet.Workbook,
"sample", SearchBy.ByColumns, ExcelFindType.Text, false, true);
spreadsheet.ActiveGrid.SetCellValue(cell, "sync");
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Outline in WPF Spreadsheet (SfSpreadsheet)

SfSpreadsheet provides support for outlines like in excel which makes your data easier to view. You can group or ungroup the data's either by rows or columns.

Group rows and columns

SfSpreadsheet provides support to group the specified range in a worksheet.

To [Group](#) the rows/columns

C#

```
//Group rows,
var gridRange = GridRangeInfo.Rows(4, 8);
spreadsheet.Group(spreadsheet.ActiveSheet, gridRange, ExcelGroupBy.ByRows);
//Group columns,
var gridRange = GridRangeInfo.Cols(4, 8);
spreadsheet.Group(spreadsheet.ActiveSheet, gridRange,
ExcelGroupBy.ByColumns);
```

Ungroup rows and columns

SfSpreadsheet provides support to ungroup the specified range in a worksheet.

To [Ungroup](#) the rows/columns

C#

```
//Ungroup rows,
var gridRange = GridRangeInfo.Rows(4, 8);
spreadsheet.Ungroup(spreadsheet.ActiveSheet, gridRange,
ExcelGroupBy.ByRows);
//Ungroup columns,
var gridRange = GridRangeInfo.Cols(4, 8);
spreadsheet.Ungroup(spreadsheet.ActiveSheet, gridRange,
ExcelGroupBy.ByColumns);
```

Collapse or Expand Group

Groups can be Expanded by [ExpandGroup](#) method and Collapsed by [CollapseGroup](#) method of **XlsIO**.

C#

```
//Expand Rows,
spreadsheet.ActiveSheet.Range["A4:A8"].ExpandGroup(ExcelGroupBy.ByRows);
spreadsheet.ActiveGrid.RowHeights.SetHidden(4, 8, false);
spreadsheet.RefreshOutlines(true, false);
//Expand Columns,
spreadsheet.ActiveSheet.Range["A3:F3"].ExpandGroup(ExcelGroupBy.ByColumns);
spreadsheet.ActiveGrid.ColumnWidths.SetHidden(1, 6, false);
spreadsheet.RefreshOutlines(false, true);
//Collapse Rows,
spreadsheet.ActiveSheet.Range["A4:A8"].CollapseGroup(ExcelGroupBy.ByRows);
spreadsheet.ActiveGrid.RowHeights.SetHidden(4, 8, true);
spreadsheet.RefreshOutlines(true, false);
//Collapse Columns,
spreadsheet.ActiveSheet.Range["A3:F3"].CollapseGroup(ExcelGroupBy.ByColumns);
;
spreadsheet.ActiveGrid.ColumnWidths.SetHidden(1, 6, true);
```

```
spreadsheet.RefreshOutlines(false, true);
```

Note: [RefreshOutlines](#) method is invoked to refresh/update the Outlines of the Group in SfSpreadsheet.

Change Outline Settings

In SfSpreadsheet, users can change the outline settings by changing the display of summary rows to either below or above the details and summary columns to either left or right of the details in Outlines Group.

C#

```
spreadsheet.ActiveSheet.PageSetup.IsSummaryRowBelow = false;  
spreadsheet.ActiveSheet.PageSetup.IsSummaryColumnRight = false;  
spreadsheet.RefreshOutlines(true, true);
```

Clear Outlines

SfSpreadsheet provides support to clear all the Outlines of the Grouped range.

C#

```
var sheet = spreadsheet.Workbook.Worksheets[0] as WorksheetImpl;  
foreach (OutlineWrapper outline in sheet.OutlineWrappers)  
{  
    outline.OutlineRange.Ungroup(outline.GroupBy);  
}  
spreadsheet.RefreshOutlines(true, true);
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Shapes in WPF Spreadsheet (SfSpreadsheet)

This section explains how to import charts, sparklines, pictures and textboxes in SfSpreadsheet.

Charts

SfSpreadsheet provides support to import charts from excel which are used to represent numeric data in graphical format to make it easier to understand large quantities of data.

For importing charts in SfSpreadsheet, add the following assembly as reference into the application.

Assembly: **Syncfusion.SfSpreadsheetHelper.WPF.dll**

Create an instance of Syncfusion.UI.Xaml.SpreadsheetHelper.[GraphicChartCellRenderer](#) and add that renderer into [GraphicCellRenderers](#) collection by using the helper method [AddGraphicChartCellRenderer](#) which is available under the namespace Syncfusion.UI.Xaml.Spreadsheet.GraphicCells.

C#

```
public MainWindow()  
{  
    InitializeComponent();  
    //For importing charts,  
    this.spreadsheet.AddGraphicChartCellRenderer(new  
        GraphicChartCellRenderer());  
}
```

```
}
```

Adding the Charts at Runtime

For adding the Charts in SfSpreadsheet at runtime, use [AddChart](#) method, also you can resize and reposition the chart.

C#

```
var chart = spreadsheet.AddChart(spreadsheet.ActiveSheet);
object[] Y_values = new object[] { 200, 100, 100 };
object[] X_values = new object[] { "Total Income", "Expenses", "Profit" };
IChartSeries series = chart.Series.Add(ExcelChartType.Pie);
// Enters the X and Y values directly
series.EnteredDirectlyValues = Y_values;
series.EnteredDirectlyCategoryLabels = X_values;
var shape = chart as ShapeImpl;
// Re-Positioning Chart
shape.Top = 200;
shape.Left = 200;
//Re-sizing a Chart
shape.Height = 300;
shape.Width = 300;
```

Sparklines

For importing sparklines in SfSpreadsheet, add the following assembly as reference into the application.

Assembly: **Syncfusion.SfSpreadsheetHelper.WPF.dll**

Create an instance of Syncfusion.UI.Xaml.SpreadsheetHelper.[SparklineCellRenderer](#) and add that renderer into the Spreadsheet by using the helper method [AddSparklineCellRenderer](#) which is available under the namespace `Syncfusion.UI.Xaml.Spreadsheet.GraphicCells`.

C#

```
public MainWindow()
{
    InitializeComponent();
    //For importing sparklines,
    this.spreadsheet.AddSparklineCellRenderer(new SparklineCellRenderer());
}
```

Pictures

SfSpreadsheet provides support to import images in SpreadsheetGrid and to add an image at run time, use [AddImage](#) method and also you can resize and reposition the image.

C#

```
var worksheet = spreadsheet.ActiveSheet;
var stream =
typeof(MainWindow).Assembly.GetManifestResourceStream("GraphicCellDemo.Data.
Sample.jpg");
var shape = spreadsheet.AddImage(worksheet, new RowColumnIndex(5, 5),
stream);
// Re-Positioning Picture
shape.Top = 200;
```

```
shape.Left = 200;
//Re-sizing a Picture
shape.Height = 200;
shape.Width = 200;
```

TextBoxes

SfSpreadsheet provides support to import RichText Box in `SpreadsheetGrid` and to add the rich text box at run time, use [AddTextBox](#) method

C#

```
var rtfText =
"{\rtf1\ansi\ansicpg1252\deff0\deflang1033{\fonttbl{\f0\fnil\fchars
et1 Calibri;}}{\f1\fnil\fcharset1
Calibri;}}{\colortbl;\red0\green0\blue0;\red255\green0\blue0;}{\f0\
fs22\b\cf1\u83*\u121*\u110*\u99*\u102*\u117*\u115*\u105*\u111*\u
110*\u32*\b0}
{\f1\fs22\cf2\u83*\u111*\u102*\u116*\u119*\u97*\u114*\u101*\u32*
}{\f1\fs22\cf1\u80*\u118*\u116*\u46*\u32*\u76*\u116*\u100*}}";
var textBox = spreadsheet.AddTextBox(spreadsheet.ActiveSheet, new
RowColumnIndex(5, 5), new Size(200, 200), rtfText) as TextBoxShapeImpl;
// Re-positioning RichTextBox
textBox.Left = 200;
textBox.Top = 200;
```

Accessing the selected Shapes

SfSpreadsheet allows the user to access the selected shapes and modify the properties associated with it in `SpreadsheetGrid`.

C#

```
var selectedShape = spreadsheet.ActiveGrid.GraphicModel.SelectedShapes;
for(int i = 0; i < selectedShape.Count ; i++)
{
    if(ExcelShapeType.Chart == selectedShape[i].ShapeType)
    {
        var chart = selectedShape[i] as IChart;
        chart.ChartArea.Fill.FillType = ExcelFillType.Gradient;
        chart.ChartArea.Fill.ForeColor = Color.Blue;
    }
    else if(ExcelShapeType.Picture == selectedShape[i].ShapeType)
    {
        var picture = selectedShape[i] as ShapeImpl;
        picture.Height = 100;
        picture.Width = 100;
    }
}
spreadsheet.ActiveGrid.GraphicModel.InvalidateGraphicObjects();
spreadsheet.ActiveGrid.GraphicModel.InvalidateGraphicVisual();
```

Select a Shape Programmatically

Users can select a shape programmatically by using [AddSelectedShapes](#) method of [GraphicModel](#) class.

C#

```
var shape = spreadsheet.ActiveSheet.Shapes[2] as ShapeImpl;  
spreadsheet.ActiveGrid.GraphicModel.AddSelectedShapes(shape);
```

Clear a Selection

Users can clear the selection from the shapes and move the selection to the grid using [ClearSelection](#) method of [GraphicModel](#) class.

C#

```
spreadsheet.ActiveGrid.GraphicModel.ClearSelection();
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Conversion in WPF Spreadsheet (SfSpreadsheet)

This section explains about the conversion of workbook in SfSpreadsheet into image, PDF and HTML

Convert to Image

SfSpreadsheet provides support to convert a worksheet in to an image of type Bitmap or Metafile based on the input range of rows and columns with all basic formats preserved, By using the [ConvertToImage](#) method, worksheet can be converted into an image.

C#

```
IWorksheet sheet = spreadsheet.Workbook.ActiveSheet;  
sheet.UsedRangeIncludesFormatting = false;  
int lastRow = sheet.UsedRange.LastRow + 1;  
int lastColumn = sheet.UsedRange.LastColumn + 1;  
System.Drawing.Image image = sheet.ConvertToImage(1, 1, lastRow, lastColumn,  
ImageType.Bitmap, null);  
image.Save("Sample.png", ImageFormat.Png);  
System.Diagnostics.Process.Start("Sample.png");
```

Convert to PDF

SfSpreadsheet provides support to export the Excel workbook to PDF using ExcelToPdfConverter.

For converting the Excel sheet to PDF, “Syncfusion.ExcelToPDFConverter.Base.dll” and “Syncfusion.Pdf.Base.dll” references should be added.

Export the Excel workbook as PDF document using [Convert](#) method of [ExcelToPdfConverter](#) class which is available under the name space “Syncfusion.ExcelToPdfConverter”

C#

```
ExcelToPdfConverter converter = new  
ExcelToPdfConverter(spreadsheet.Workbook);  
//Initialize the PdfDocument  
PdfDocument pdfDoc = new PdfDocument();  
//Initialize the ExcelToPdfConverter Settings  
ExcelToPdfConverterSettings settings = new ExcelToPdfConverterSettings();  
settings.LayoutOptions = LayoutOptions.NoScaling;  
//Assign the PdfDocument to the templateDocument property of  
ExcelToPdfConverterSettings
```

```
settings.TemplateDocument = pdfDoc;  
settings.DisplayGridLines = GridLinesDisplayStyle.Invisible;  
//Convert Excel Document into PDF document  
pdfDoc = converter.Convert(settings);  
//Save the PDF file  
pdfDoc.Save("Sample.pdf");  
System.Diagnostics.Process.Start("Sample.pdf");
```

Convert to HTML

SfSpreadsheet provides support to convert the excel workbook into HTML page.

C#

```
spreadsheet.Workbook.SaveAsHtml("Sample.html", HtmlSaveOptions.Default);  
System.Diagnostics.Process.Start("Sample.html");
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Localization in WPF Spreadsheet (SfSpreadsheet)

Localization is the process of configuring the application to a specific language. SfSpreadsheet provides support to localize all the static text in a Ribbon and all dialogs to any desired language. Localization can be done by adding resource file and setting the specific culture in the application.

SfSpreadsheet allows you to set custom resource using Resx file. You can define your string values in resource file for a specific culture and set the culture in your application.

Set Current UI Culture to the Application

To set the CultureInformation in the Application, set the `CurrentUICulture` before the `InitializeComponent()` method is called.

Setting of the culture information,

C#

```
public MainWindow()  
{  
    System.Threading.Thread.CurrentThread.CurrentUICulture = new  
    CultureInfo("ja-JP");  
    InitializeComponent();  
}
```

Now, the Application is set to the Japanese Culture info.

Localization using Resource file

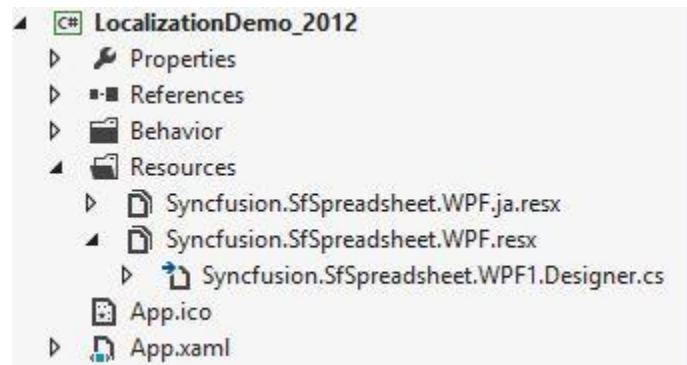
The following steps show how to implement the localization in SfSpreadsheet,

- Create a folder and name it as 'Resources' in your application.
- Add the default resource[English("en-US")] file of `SfSpreadsheet` in the 'Resources' folder named as `Syncfusion.SfSpreadsheet.WPF.resx`.

You can download the Resx file [here](#)

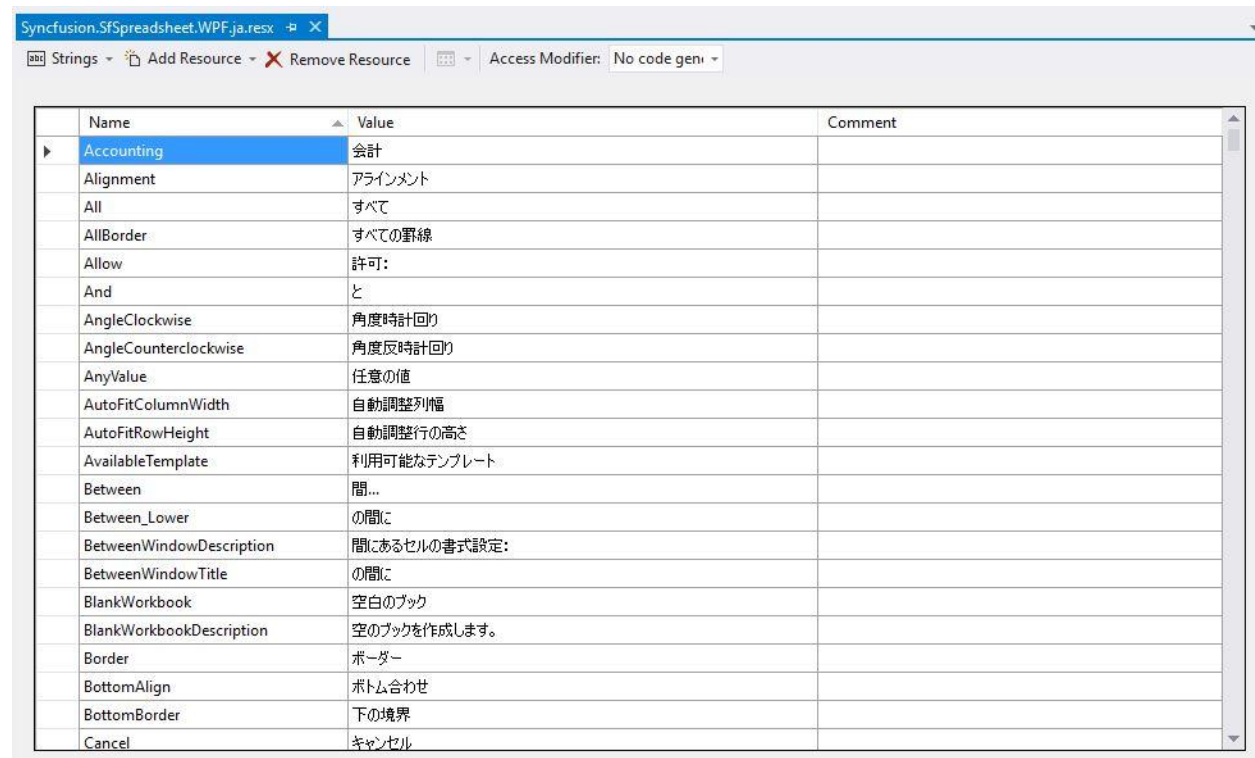
- Create Resx(resource) file under the 'Resources' folder and name it as Syncfusion.SfSpreadsheet.WPF.[Culture name].resx.

For example, Syncfusion.SfSpreadsheet.WPF.ja.resx for Japanese culture.



- Add the resource key such as name and its corresponding localized value in Resource Designer of Syncfusion.SfSpreadsheet.WPF.ja.resx file.

For your reference, you can download the Japanese("ja-JP") Resx file [here](#)



The following screenshot shows you the localization in SfSpreadsheet,



Modifying the localized strings in Resource file

Users can modify the default localized strings in Resource file by adding the default [Resx](#) (resource) file of SfSpreadsheet in the 'Resources' folder of your application and name it as Syncfusion.SfSpreadsheet.WPF.resx.

Now, the default localized strings can be modified by changing the Name/Value pair in the Syncfusion.SfSpreadsheet.WPF.resx file.

Name	Value	Comment
Accounting	Accounting	
Alignment	Alignment	
All	All	
AllBorder	All Borders	
Allow	Allow :	
And	and	
AngleClockwise	Angle Clockwise	
AngleCounterclockwise	Angle Counterclockwise	
AnyValue	Any value	
AutoFitColumnWidth	AutoFit Column Width	
AutoFitRowHeight	AutoFit Row Height	
AvailableTemplate	Available Template	
Between	Between...	
Between_Lower	between	
BetweenWindowDescription	Format cells that are BETWEEN:	
BetweenWindowTitle	Between	
BlankWorkbook	Blank workbook	
BlankWorkbookDescription	Creates an Empty Workbook	
Border	Border	
BottomAlign	BottomAlign	
BottomBorder	Bottom Border	
Cancel	Cancel	

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Printing in WPF Spreadsheet (SfSpreadsheet)

SfSpreadsheet control allows you to print the data in the workbook with the help of PDF Conversion. To provide the printing support in SfSpreadsheet, you need to convert the workbook into PDF document using ExcelToPdfConverter.

For Conversion of Excel Workbook in SfSpreadsheet to PDF document, use [Convert](#) method of [ExcelToPdfConverter](#).

For viewing the PDF document, you can use [PdfViewerControl](#) to load the saved PDF stream.

C#

```
//Create the pdf viewer for load the document.
PdfViewerControl pdfViewer = new PdfViewerControl();
//Create Memory Stream to save pdf document
MemoryStream pdfStream = new MemoryStream();
ExcelToPdfConverter converter = new ExcelToPdfConverter
(spreadsheet.Workbook);
//Initialize the ExcelToPdfConverter Settings
ExcelToPdfConverterSettings settings = new ExcelToPdfConverterSettings();
settings.LayoutOptions = LayoutOptions.NoScaling;
```

For print preview you can load the PDF stream into viewer and for direct printing use **Print** method in PdfViewerControl which is available under the namespace "Syncfusion.PdfViewer.Wpf"

C#

```
//Initialize the PdfDocument
PdfDocument pdfDoc = new PdfDocument ();
//Assign the PdfDocument to the templateDocument property of
ExcelToPdfConverterSettings
settings.TemplateDocument = pdfDoc;
settings.DisplayGridLines = GridLinesDisplayStyle.Invisible;
//Convert Excel Document into PDF document
pdfDoc = converter.Convert(settings);
//Save the PDF file
pdfDoc.Save(pdfStream);
//Load the document to pdf viewer
pdfViewer.Load(pdfStream);
//Print the doc
pdfViewer.Print(true);
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Ribbon Customization in WPF Spreadsheet (SfSpreadsheet)

Ribbon Customization can be done in two ways,

Using Control Template

User can customize the ribbon items by overriding the template of [SfSpreadsheetRibbon](#).

Using Event

By invoking SfSpreadsheetRibbon Loaded Event, User can add/delete the ribbon menu items.

Adding a RibbonTab

To add a custom ribbon tab with user defined menu options in SfSpreadsheetRibbon,

XML

```
<syncfusion:SfSpreadsheetRibbon x:Name="ribbon" DataContext="{Binding  
ElementName=spreadsheet}" />
```

C#

```
ribbon.Loaded += ribbon_Loaded;  
void ribbon_Loaded(object sender, RoutedEventArgs e)  
{  
    var ribbon1 = GridUtil.GetVisualChild<Ribbon>(sender as FrameworkElement);  
    if (ribbon1 != null)  
    {  
        RibbonTab ribbonTab = new RibbonTab();  
        ribbonTab.Caption = "OTHER";  
        RibbonButton Button1 = new RibbonButton();  
        Button1.Label = "PRINT";  
        Button1.SmallIcon = new BitmapImage(new Uri("../Icon/Icons_Print.png",  
UriKind.Relative));  
        Button1.Click += Button1_Click;  
        RibbonButton Button2 = new RibbonButton();  
        Button2.Label = "PRINT PREVIEW";  
        Button2.SmallIcon = new BitmapImage(new Uri("../Icon/Icons_Print.png",  
UriKind.Relative));  
        Button2.Click += Button2_Click;  
        var customRibbonBar = new RibbonBar();  
        customRibbonBar.Header = "Printing Options";  
        customRibbonBar.Items.Add(Button1);  
        customRibbonBar.Items.Add(Button2);  
        customRibbonBar.IsLauncherButtonVisible = false;  
        ribbonTab.Items.Add(customRibbonBar);  
        ribbon1.Items.Add(ribbonTab);  
    }  
}
```

Adding a Ribbon Items in Existing Tab

To add a ribbon items in already existing tab,

XML

```
<syncfusion:SfSpreadsheetRibbon x:Name="ribbon" DataContext="{Binding  
ElementName=spreadsheet}" />
```

C#

```
ribbon.Loaded += ribbon_Loaded;  
void ribbon_Loaded(object sender, RoutedEventArgs e)  
{  
    var ribbon1 = GridUtil.GetVisualChild<Ribbon>(sender as FrameworkElement);
```

```
// To add the ribbon button in View tab,
if (ribbon1 != null)
{
    var ribbonTab = ribbon1.Items[2] as RibbonTab;
    RibbonButton Button1 = new RibbonButton();
    Button1.Label = "PRINT";
    Button1.SmallIcon = new BitmapImage(new Uri("../Icon/Icons_Print.png",
    UriKind.Relative));
    Button1.Click += Button1_Click;
    ribbonTab.Items.Add(Button1);
}
}
```

Removing a RibbonTab

To remove the ribbon tab in the SfSpreadsheetRibbon,

XML

```
<syncfusion:SfSpreadsheetRibbon x:Name="ribbon" DataContext="{Binding
ElementName=spreadsheet}" />
```

C#

```
ribbon.Loaded += ribbon_Loaded;
void ribbon_Loaded(object sender, RoutedEventArgs e)
{
    var ribbon1 = GridUtil.GetVisualChild<Ribbon>(sender as FrameworkElement);
    //To remove the Data tab from the ribbon,
    if (ribbon1 != null)
    {
        var item = ribbon1.Items[1];
        ribbon1.Items.Remove(item);
    }
}
```

Removing a Ribbon Items in a RibbonTab

To remove the ribbon menu items in the ribbon tab of SfSpreadsheetRibbon,

XML

```
<syncfusion:SfSpreadsheetRibbon x:Name="ribbon" DataContext="{Binding
ElementName=spreadsheet}" />
```

C#

```
ribbon.Loaded += ribbon_Loaded;
void ribbon_Loaded(object sender, RoutedEventArgs e)
{
    var ribbon1 = GridUtil.GetVisualChild<Ribbon>(sender as FrameworkElement);
    // To remove the Freeze panes menu group in View tab,
    if (ribbon1 != null)
    {
        var ribbonTab = ribbon1.Items[2] as RibbonTab;
        ribbonTab.Items.Remove(ribbonTab.Items[1]);
    }
}
```

```
}
```

Canceling a Ribbon commands

You can cancel particular action of SpreadsheetRibbon commands by handling [CommandExecuting](#) event.

XML

```
<syncfusion:SfSpreadsheetRibbon x:Name="ribbon" DataContext="{Binding
ElementName=spreadsheet}" />
```

C#

```
this.ribbon.Commands.CommandExecuting += Commands_CommandExecuting;
void Commands_CommandExecuting(object sender, CommandExecutingEventArgs
args)
{
    //stops copy button command execution.
    if(args.CommandName == "Copy")
    {
        //set the bool value is true.
        //the operation is not performed as you mentioned in CommandName
        args.cancel = true;
    }
}
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Custom Formula in WPF Spreadsheet (SfSpreadsheet)

SfSpreadsheet allows you to add custom formulas into its function library. You can add the custom formula into the SfSpreadsheet by using the [AddFunction](#) method of [FormulaEngine](#),

C#

```
spreadsheet.WorkbookLoaded += spreadsheet_WorkbookLoaded;
void spreadsheet_WorkbookLoaded(object sender, WorkbookLoadedEventArgs args)
{
    foreach (var grid in args.GridCollection)
        AddCustomFormula(grid);
    //Computing the formula at runtime
    var range = spreadsheet.ActiveSheet.Range["B2"];
    spreadsheet.ActiveGrid.SetCellValue(range, "=Find(sample)");
}
private void AddCustomFormula(SpreadsheetGrid grid)
{
    // Add a formula named Find to the Library.
    grid.FormulaEngine.AddFunction("Find", new
    FormulaEngine.LibraryFunction(ComputeLength));
}
//Implementation of formula
public string ComputeLength(string range)
{
```

```
//Used to calculate the length of the string
return range.Length.ToString();
}
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Cell Customization in WPF Spreadsheet (SfSpreadsheet)

SfSpreadsheet provides support to customize the cell with Data template. It allows to load any WPF control or custom control into the cell.

In order to customize the cell, please follow the below steps

- Create a DataTemplate
- Override the SpreadsheetColumn class
- Create the Custom Cell Renderer
- Associating the Custom Cell Renderer to SpreadsheetGrid

Create a DataTemplate

Create a custom DataTemplate (For ex: Button Template) in the Main window.xaml file,

XML

```
<DataTemplate x:Key="ButtonTemplate" >
<Expander x:Name="Button" ExpandDirection="Right" IsExpanded="True"
Expanded="Button_Expanded" Collapsed="Button_Collapsed"/>
</DataTemplate>
```

Override the SpreadsheetColumn class

Create an extension class SpreadsheetColumnExt by overriding the [SpreadsheetColumn](#) Class which holds all the operations related with cells.

Now override the function `OnUpdateColumn` which updates the column properties (cell types, renderer, cell element, etc.) and get the data template which is used for displaying in the particular cell.

C#

```
public class SpreadsheetColumnExt : SpreadsheetColumn
{
    public SpreadsheetColumnExt(SpreadsheetGrid grid) : base(grid)
    {
    }
    // Gets or sets the DataTemplate for the cell
    public DataTemplate CellItemTemplate
    {
        get;
        set;
    }
    // Gets or sets the DataTemplate for the cell
    public DataTemplate CellEditTemplate
    {
        get;
```

```

set;
}
//Update the cell with the defined template
protected override void OnUpdateColumn(out FrameworkElement oldElement)
{
    if (RowIndex == 3 && ColumnIndex == 6)
    {
        this.CellItemTemplate =
        Application.Current.MainWindow.Resources["ButtonTemplate"] as DataTemplate;
        this.CellEditTemplate =
        Application.Current.MainWindow.Resources["ButtonTemplate"] as DataTemplate;
    }
    base.OnUpdateColumn(out oldElement);
}
}

```

Create the Custom Cell Renderer

Create a SpreadsheetTemplateCellRenderer class by overriding the [SpreadsheetVirtualizingCellRendererBase](#) class to display the defined custom renderer element.

For initializing the display element, set the content template in `OnInitializeDisplayElement` method and for editing, set the content template in `OnInitializeEditElement` method otherwise it will load the default display and edit element in the cells

C#

```

public class SpreadsheetTemplateCellRenderer :
    SpreadsheetVirtualizingCellRendererBase<ContentControl, ContentControl>
{
    public SpreadsheetTemplateCellRenderer()
    {
        SupportDrawingOptimization = false;
    }
    //Update the cell style for display element
    protected override void OnUpdateCellStyle(RowColumnIndex cellRowColumnIndex,
        ContentControl uiElement, SpreadsheetColumn column)
    {
        if (uiElement.ContentTemplate == null)
        {
            uiElement.ContentTemplate = (column as
            SpreadsheetColumnExt).CellItemTemplate;
        }
        base.OnUpdateCellStyle(cellRowColumnIndex, uiElement, column);
    }
    //Update the cell style for edit element
    protected override void OnUpdateEditCellStyle (RowColumnIndex
        cellRowColumnIndex, ContentControl uiElement, SpreadsheetColumn column)
    {
        if (uiElement.ContentTemplate == null)
        {
            uiElement.ContentTemplate = (column as
            SpreadsheetColumnExt).CellEditTemplate;
        }
        base.OnUpdateEditCellStyle (cellRowColumnIndex, uiElement, column);
    }
}

```



```

//To initialize the display element on the cell
protected override void OnInitializeDisplayElement(RowColumnIndex
rowColumnIndex, ContentControl uiElement, SpreadsheetColumn column)
{
    uiElement.ContentTemplate = (column as
    SpreadsheetColumnExt).CellItemTemplate;
}
//To initialize the edit element on the cell
protected override void OnInitializeEditElement(RowColumnIndex
rowColumnIndex, ContentControl uiElement, SpreadsheetColumn column)
{
    uiElement.ContentTemplate = (column as
    SpreadsheetColumnExt).CellEditTemplate;
}
}

```

Note: If you want to load the default edit element, then no need to override the `OnInitializeEditElement` method.

Associating the Custom Cell Renderer to SpreadsheetGrid

To associate the custom cell renderer in [SpreadsheetGrid](#), invoke the [WorkbookLoaded](#) Event of SfSpreadsheet and initialize the `SpreadsheetTemplateCellRenderer` and add it to the renderer collection.

Invoke the [QueryRange](#) Event of `SpreadsheetGrid` and set the `CellType` of particular range to be "DataTemplate" to load the user defined template.

C#

```

void spreadsheet_WorkbookLoaded(object sender, WorkbookLoadedEventArgs args)
{
    var grid = spreadsheet.ActiveGrid;
    grid.CreateGridColumn = CreateSpreadsheetColumnExt;
    var renderer = new SpreadsheetTemplateCellRenderer();
    grid.CellRenderers.Add("DataTemplate", renderer);
    grid.QueryRange += grid_QueryRange;
}
//To access the SpreadsheetColumnExt
public GridColumn CreateSpreadsheetColumnExt(SfCellGrid grid)
{
    return new SpreadsheetColumnExt(grid as SpreadsheetGrid);
}
//To update the cell type
void grid_QueryRange(object sender, SpreadsheetQueryRangeEventArgs e)
{
    if (e.Cell.ColumnIndex == 6)
    {
        e.CellType = "DataTemplate";
        e.CellValue = "0";
        e.Handled = true;
    }
}

```

For more reference, please find the [customization](#) sample.

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

Limitations in WPF Spreadsheet (SfSpreadsheet)

Release memory held by AutomationPeer

SfSpreadsheet holds some instance in memory even after disposing the spreadsheet or removed the sheets from the spreadsheet. Because, the **AutomationPeer** for WPF Components holds some memory and it needs to be released manually. This can be done by using the following steps.

Create a class derived from **WindowAutomationPeer** and override its **GetChildrenCore** method and returns "null" value that clears the **AutomationPeer** item from memory as follows

C#

```
public class FakeWindowsPeer : WindowAutomationPeer
{
    public FakeWindowsPeer (Window window): base(window)
    { }
    protected override List<AutomationPeer> GetChildrenCore()
    {
        return null;
    }
}
```

Now override the **OnCreateAutomationPeer** of the window and it returns the class as follows.

C#

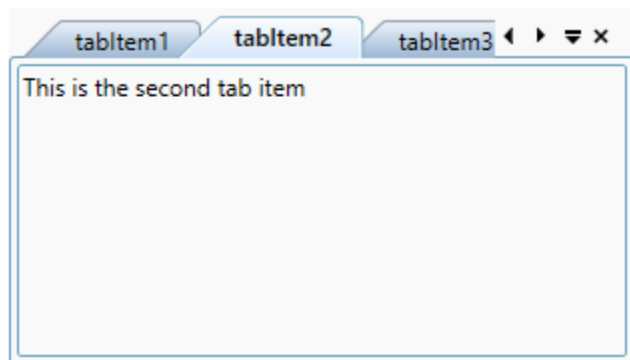
```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
    protected override AutomationPeer OnCreateAutomationPeer()
    {
        return new FakeWindowsPeer(this);
    }
}
```

Note: You can refer to our [WPF Spreadsheet](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Spreadsheet example](#) to know how to render and configure the spreadsheet.

TabControlExt

WPF TabControl (TabControlExt) Overview

The [TabControl](#) is similar to the dividers in a notebook or the labels in a file cabinet. By using **TabControl**, an application can define multiple pages for the same area of a window. **TabControl** contains the [TabItemExt](#), which is used to define Tab Items for **TabControl**. By clicking a tab item header, the data corresponding to that particular tab item will be displayed.



Key features

Tab Orientation - Provides support to position the tabs horizontally at the top or bottom and vertically at the left or right.

Editable header - Provides support to edit the headers interactively in UI by pressing the F2 key or by double-clicking a tab header.

Display mode - Provides support to customize the display mode of the Close button.

Layout - Provides different layout types for enhanced usage to the control. The types are SingleLine, MultiLine, and MultiLineWithFillWidth.

Pin and UnPin - Provides support to pinning tabs for quick access and allows users to interactively pin and unpin the tabs.

Selection - Provides support to select the tabs quickly through keyboard or mouse interaction.

Styles - Provides a rich set of built-in themes and customizes the style of each part of **TabControl**.

Drag and drop - Provides support to reorder the tabs by dragging and dropping headers and change the color of drag marker while dragging the tab page in **TabControl**.

Scrolling - Provides support to scroll the tab items to next, previous, first, and last in **TabControl**.

Images - Provides support to add images to the tab header and also aligns the header image to any position.

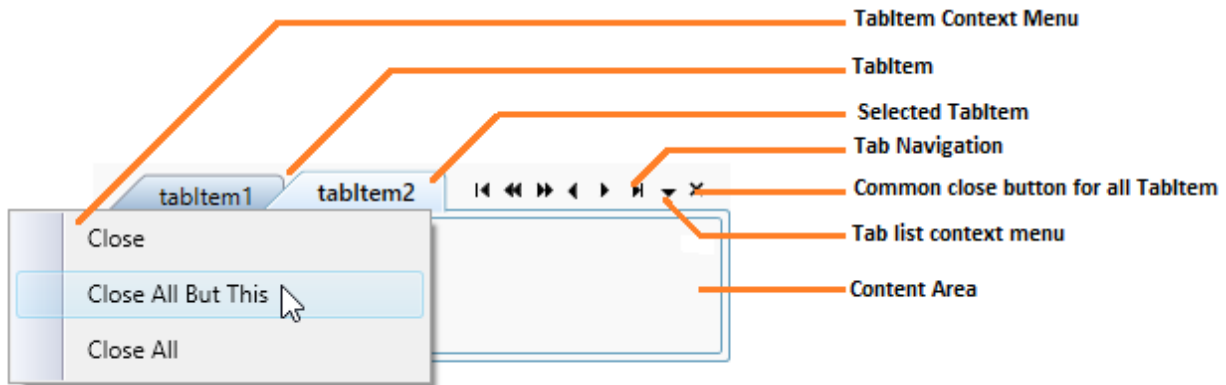
Context menu - Provides support to built-in context menu option for tab list and tab item.

Getting Started with WPF TabControl (TabControlExt)

This section explains how to create a WPF [TabControl](#) and explains about its structure.

Structure of TabControl

The various elements of **TabControl** are illustrated in the following images.



Assembly deployment

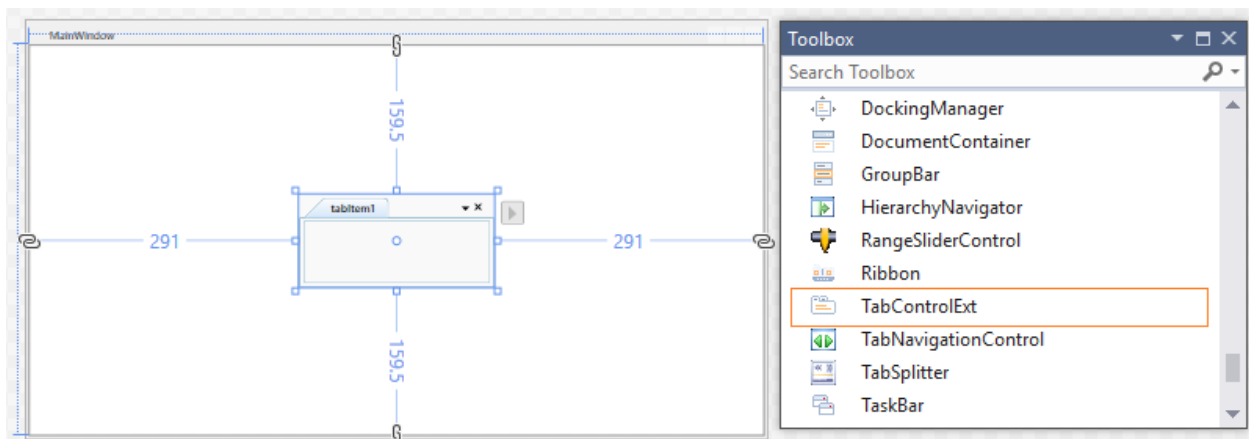
Refer to the [Control Dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

Refer to this [documentation](#) to find more details about installing nuget packages in a WPF application.

Adding WPF TabControl via designer

You can add the **TabControl** 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.Tools.WPF
- Syncfusion.Shared.WPF



Adding WPF TabControl via XAML

To add the **TabControl** 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.Tools.WPF* *Syncfusion.Shared.WPF* 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the **TabControl** in XAML page.

XML

```
<Window x:Class="TabControlExt_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:TabControlExt_sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid Name="grid">
<syncfusion:TabControlExt Name="tabControl" Height="100" Width="280" />
</Grid>
```

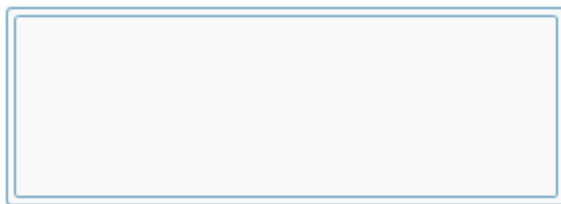
Adding WPF TabControl via C#

To add the [TabControl](#) control manually in C#, follow these steps:

1. Create a new WPF application via Visual Studio.
2. Add the following assembly references to the project,
 - o Syncfusion.Shared.WPF
3. Include the required namespace and create an instance of `TabControl` and add it to the window.
4. Declare the `TabControl` control using C#.

C#

```
using Syncfusion.Windows.Tools.Controls;
public partial class MainWindow : Window {
public MainWindow() {
InitializeComponent();
// Creating an instance of the TabControl
TabControlExt tabControlExt = new TabControlExt();
// Setting height and width to TabControl
tabControlExt.Height = 100;
tabControlExt.Width = 280;
//Adding TabControl as window content
this.Content = tabControlExt;
}
}
```



Adding TabItem

You can add the tab item using the `Items` property of `TabControl`. You can set the tab item name using `TabItemExt` property and add the content to each tab item by using `TabItemExt.Content` property.

XML

```
<syncfusion:TabControlExt Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1">
<TextBlock Name="textBlock" Text="This is the first tab item." />
</syncfusion:TabItemExt>
<syncfusion:TabItemExt Header="tabItem2">
```

```

<TextBlock Name="textBlock1" Text="This is the second tab item." />
</syncfusion:TabItemExt>
<syncfusion:TabItemExt Header="tabItem3">
<TextBlock Name="textBlock2" Text="This is the third tab item." />
</syncfusion:TabItemExt>
</syncfusion:TabControlExt>

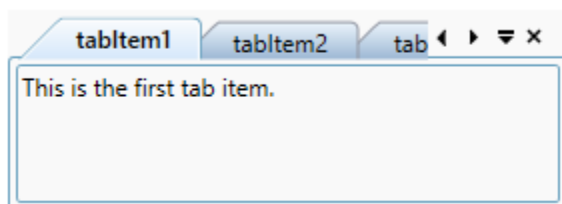
```

C#

```

// Creating an instances of tabitems and adding header & content
TabItemExt tabItemExt1 = new TabItemExt()
{
    Header= "tabItem1",
    Content= new TextBlock() { Text= "This is the first tab item" }
};
TabItemExt tabItemExt2 = new TabItemExt()
{
    Header= "tabItem2",
    Content= new TextBlock() { Text= "This is the second tab item" }
};
TabItemExt tabItemExt3 = new TabItemExt()
{
    Header= "tabItem3",
    Content= new TextBlock() { Text= "This is the third tab item" }
};
// Creating an instances of TabControl and adding the tabitems into the
TabControl
TabControlExt tabControlExt = new TabControlExt();
tabControlExt.Items.Add(tabItemExt1);
tabControlExt.Items.Add(tabItemExt2);
tabControlExt.Items.Add(tabItemExt3);

```



Please refer [DataBinding](#) page to know about how to add a tab item using Data Binding.

Note: View [Sample](#) in GitHub

Placement of TabItem

You can place the tab item header at any of the four sides of **TabControl** using the [TabStripPlacement](#) property.

XML

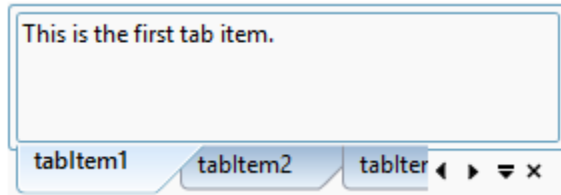
```

<syncfusion:TabControlExt TabStripPlacement="Bottom"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
<syncfusion:TabItemExt Header="tabItem3" />
</syncfusion:TabControlExt>

```

C#

```
tabControlExt.TabStripPlacement = Dock.Bottom;
```

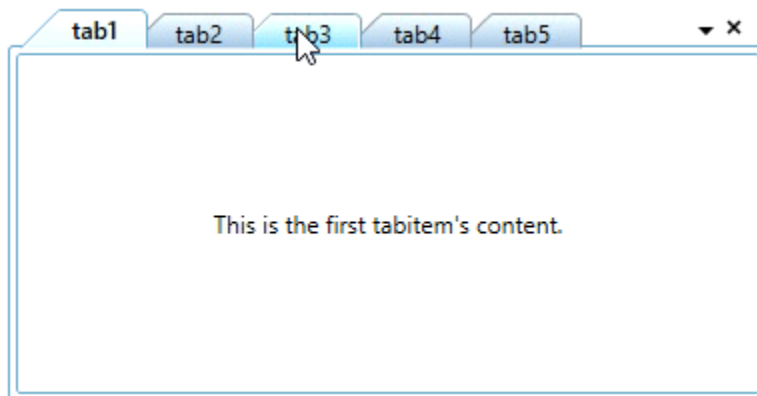


Note: View [Sample](#) in GitHub

Selecting TabItem

You can select a particular tab item by using the mouse click on the tab header. You can use the **Ctrl + Tab** key to select a tab item when control not in focused state. You also use the **Left-Arrow** and **Right-Arrow** key , to select the previous tab item or next tab item of current selected tab item when control in focused . You can get the selected item by using the **SelectedItem** property. By default, the first tab item is selected.

Note: You can select only one tab item at a time.



Note: View [Sample](#) in GitHub

Tab selection changed notification

The **TabControl** notifies that the selected tab item is changed by user through the **SelectedItemChangedEvent** event. You can use the **OldSelectedItem** and **NewSelectedItem** properties to get the old and new selected tabitem in the **SelectedItemChangedEvent** event.

XML

```
<syncfusion:TabControlExt  
    SelectedItemChangedEvent="TabControlExt_SelectedItemChangedEvent"  
    Name="tabControlExt" />
```

C#

```
TabControlExt tabControlExt = new TabControlExt();  
tabControlExt.SelectedItemChangedEvent +=  
    TabControlExt_SelectedItemChangedEvent;
```

You can handle the event as follows:

C#

```
private void TabControlExt_SelectedItemChangedEvent(object sender,  
    SelectedItemChangedEventArgs e) {  
    var newTabItem = e.NewSelectedItem.Header;  
    if (e.OldSelectedItem != null) {  
        var oldTabItem = e.OldSelectedItem.Header;  
    }  
    else {  
        var oldTabItem = string.Empty;  
    }  
}
```

Closing the tab item

TabControl allows end-users to close the tabs using close button. The close button can be displayed in Tabcontrol using [TabControlExt.CloseButtonType](#) property.

The following options are supported to show close the button in TabControl.

- **Common** - Only, TabControl shows the close button.
- **Individual** - The close button displayed only in the headers of tab items.
- **Both** - TabControl and tab items displays the close button.
- **Hide** - The close button is not visible.
- **IndividualOnMouseOver** - The close button displayed only when the mouse is over the tab item.
- **Extended** - The close button displayed only for the selected tabitem and the remaining tab item displays close button while mouse is over the tab header.

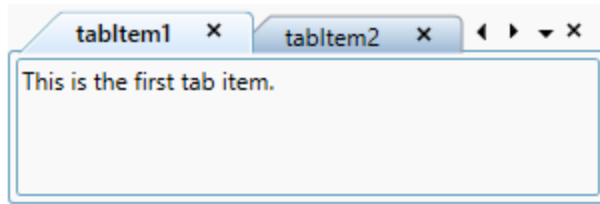
For example, when you set the TabControlExt.CloseButtonType as Both, then the close button is displayed in both TabItem and TabControl.

XML

```
<syncfusion:TabControlExt Name="tabControlExt" CloseButtonType="Both">  
    <syncfusion:TabItemExt Header="tabItem1" />  
    <syncfusion:TabItemExt Header="tabItem2" />  
</syncfusion:TabControlExt>
```

C#

```
//set `Both` option to `CloseButtonType` property.  
tabControlExt.CloseButtonType = CloseButtonType.Both;
```

Note: View [Sample](#) in GitHub

Disable particular tab close button

You can disable the users from closing on particular tab item by setting [TabItemExt.CanClose](#) property as `false`.

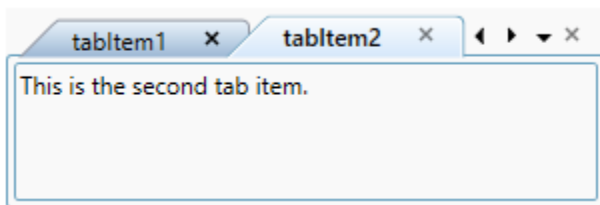
XML

```
<syncfusion:TabControlExt CloseButtonType="Both"
Name="tabControlExt" >
<syncfusion:TabItemExt Header="tabItem1" Name="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" Name="tabItem2"
CanClose="False" />
</syncfusion:TabControlExt>
```

C#

```
//set `Both` option to `CloseButtonType` property.
tabControlExt.CloseButtonType = CloseButtonType.Both;
//Disable the close button.
tabItem2.CanClose = false;
```

In the below screenshot, second tabitem `TabItemExt.CanClose` property is `false`. So, close button in tabitem header and `TabControl` is in disabled state for second tabitem.



Note: View [Sample](#) in GitHub

Hide particular tab close button

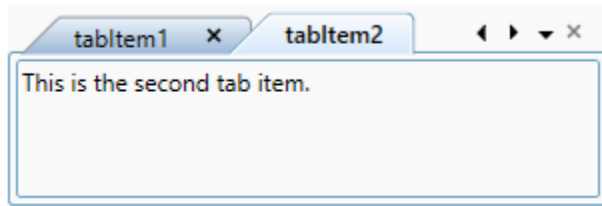
If you want to hide the visibility of close button in particular tab item, then you can collapse using [TabItemExt.CloseButtonState](#) property value as `Collapsed`.

XML

```
<syncfusion:TabControlExt CloseButtonType="Both"
Name="tabControlExt" >
<syncfusion:TabItemExt Header="tabItem1" Name="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" Name="tabItem2"
CloseButtonState="Collapsed"
CanClose="False" />
</syncfusion:TabControlExt>
```

C#

```
//set `Both` option to `CloseButtonType` property.
tabControlExt.CloseButtonType = CloseButtonType.Both;
//Disable the close button.
tabItem2.CanClose = false;
//Collapse the close button.
tabItem2.CloseButtonState = Collapsed;
```



Here, `tabItem2` close button is collapsed.

Note: View [Sample](#) in GitHub

Add new `TabItem` using new button

You can add the new tab item by clicking the New button. You can show the new tab button by setting the `TabControlExt.IsNewButtonEnabled` property to true in `TabControl`. The `NewButtonClick` event handles the click action of new tab button to add new tab item in `TabControl`.

XML

```
<syncfusion:TabControlExt IsNewButtonEnabled="True"
NewButtonClick="TabControlExt_NewButtonClick"
Name="tabControlExt" >
  <syncfusion:TabItemExt Header="tabItem1" />
  <syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

```
tabControlExt.IsNewButtonEnabled = true;
```

```
tabControlExt.NewButtonClick += TabControlExt_NewButtonClick;
```

C#

```
private void TabControlExt_NewButtonClick(object sender, EventArgs e) {
    TabItemExt itemExt = new TabItemExt();
    itemExt.Header = "tabItem" + (tabControlExt.Items.Count + 1);
    //Add a new item in a TabControl
    tabControlExt.Items.Add(itemExt);
}
```



Note: View [Sample](#) in GitHub

Tab list menu for switching tabs

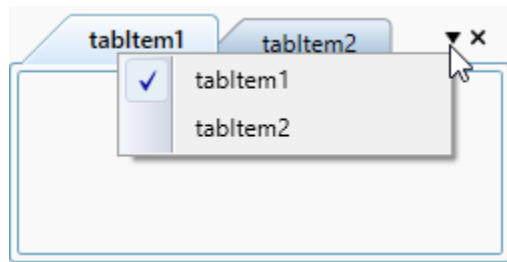
You can easily navigate to any tab item by using the tab list menu which is placed in the top-right corner of the tab header panel. The header of all tab item's are shown as a menu item in the tab list menu. You can hide this tab list menu by using the [ShowTabListContextMenu](#) property value as `false`. The default value of `ShowTabListContextMenu` property is `true`.

XML

```
<syncfusion:TabControlExt ShowTabListContextMenu="True"
Name="tabControlExt" >
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.ShowTabListContextMenu = true;
```



Note: View [Sample](#) in GitHub

Show hidden tab items

By default, all the tab items except hidden items are listed in the tab list context menu. If you want to show the hidden tab items into tab list context menu to navigate, use the [TabListContextMenuOptions](#) property value as `Default`, `ShowHiddenItems`. You can set single or multiple options for the `TabListContextMenuOptions` to show single or different types of tab items into the context menu. The default value of `TabListContextMenuOptions` property is `Default`.

XML

```
<syncfusion:TabControlExt TabListContextMenuOptions="Default,
ShowHiddenItems"
ShowTabListContextMenu="True">
```

```
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" Visibility="Collapsed"/>
<syncfusion:TabItemExt Header="tabItem2"/>
<syncfusion:TabItemExt Header="tabItem3" Visibility="Collapsed"/>
<syncfusion:TabItemExt Header="tabItem4" IsEnabled="False"/>
<syncfusion:TabItemExt Header="tabItem5"/>
<syncfusion:TabItemExt Header="tabItem6" Visibility="Collapsed"/>
</syncfusion:TabControlExt>
```

C#

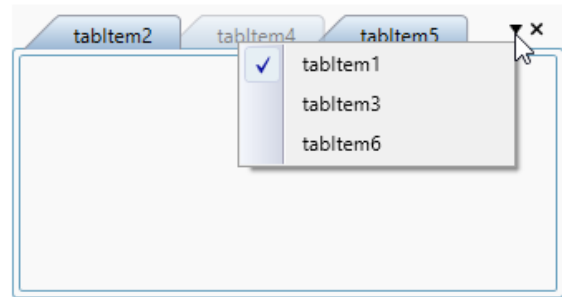
```
tabControlExt.ShowTabListContextMenu = true;
tabControlExt.TabListContextMenuOptions = TabListContextMenuOptions.Default
|
TabListContextMenuOptions.ShowHiddenItems;
```

Single options

TabListContextMenuOptions = "Default"



TabListContextMenuOptions = "ShowHiddenItems"



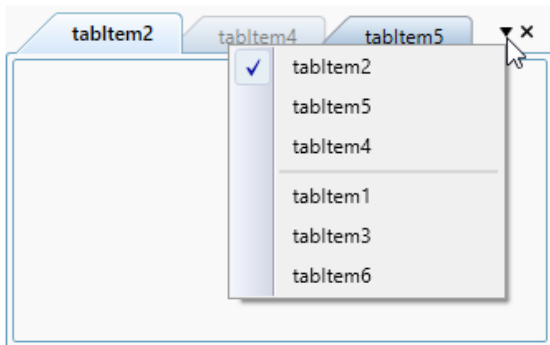
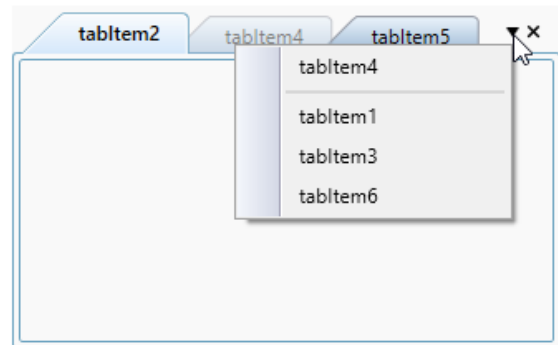
TabListContextMenuOptions = "ShowEnabledItems"



TabListContextMenuOptions = "ShowDisabledItems"



Multiple options

TabListContextMenuOptions= "Default,
ShowHiddenItems"TabListContextMenuOptions="ShowDisabledItems,
ShowHiddenItems"**Note:** View [Sample](#) in GitHub

Enable or disable tab navigation bar

You can enable or disable the the tab navigation bar in `TabControl` by using the `TabScrollButtonVisibility` property value as `Visible` or `Hidden`. You can show the different tab navigation style in `TabControl` using the `TabScrollStyle` property.

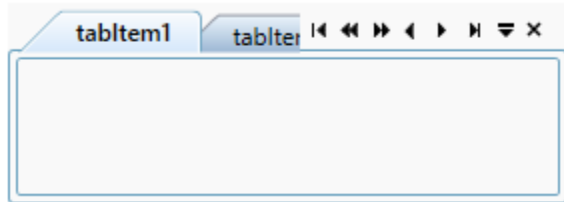
XML

```
<syncfusion:TabControlExt TabScrollButtonVisibility="Visible"
TabScrollStyle="Extended"
Name="tabControlExt" >
<syncfusion:TabItemExt Header="tabItem1" />
```

```
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.TabScrollButtonVisibility = TabScrollButtonVisibility.Visible;
tabControlExt.TabScrollStyle = TabScrollStyle.Extended;
```



Note: View [Sample](#) in GitHub

Show or hide built-in context menu

You can show the built-in context menu of the tab item by setting the [ShowTabItemContextMenu](#) property to **true** in **TabControl**.

The built-in context menu of the tab item has following menu items:

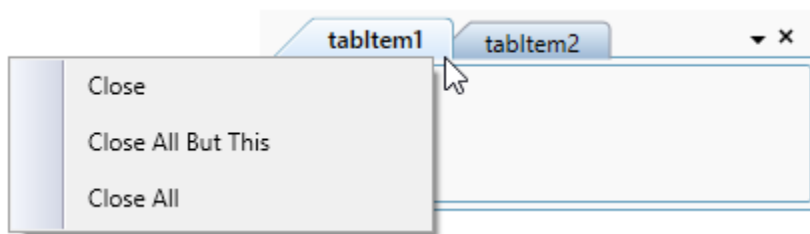
- **Close** - Closes the current or selected tab item.
- **Close All But This** - Closes all the tab items, except the current or selected tab item.
- **Close All** - Closes all the tab items.

XML

```
<syncfusion:TabControlExt ShowTabItemContextMenu="True"
Name="tabControlExt" >
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
//Changing the visibility of context menu
tabControlExt.ShowTabItemContextMenu = true;
```



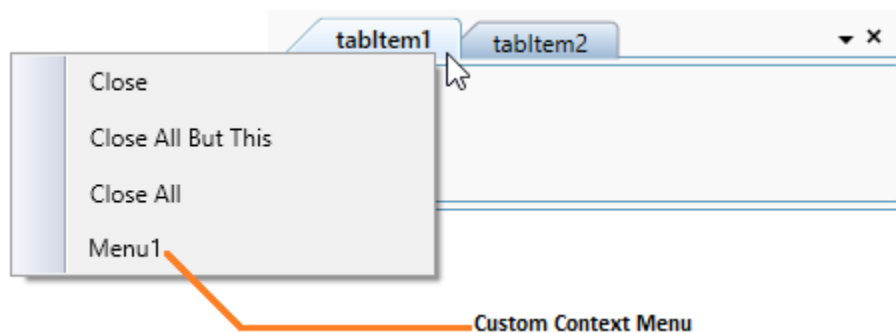
You can add custom context menu item using the [TabItemExt.ContextMenuItems](#) property in **TabControl**. This property is enabled by setting the [IsCustomTabItemContextMenuEnabled](#) property to **true**.

XML

```
<syncfusion:TabControlExt IsCustomTabItemContextMenuEnabled="True"
Name="tabControlExt">
  <syncfusion:TabItemExt Header="tabItem1" Name="tabItem1">
    <syncfusion:TabItemExt.ContextMenuItems>
      <syncfusion:CustomMenuItem Header="Menu1" />
    </syncfusion:TabItemExt.ContextMenuItems>
  </syncfusion:TabItemExt>
  <syncfusion:TabItemExt Header="tabItem2" Name="tabItem2">
    <syncfusion:TabItemExt.ContextMenuItems>
      <syncfusion:CustomMenuItem Header="Menu2" />
    </syncfusion:TabItemExt.ContextMenuItems>
  </syncfusion:TabItemExt>
</syncfusion:TabControlExt>
```

C#

```
// Enabling custom tabitem context menu
tabControlExt.IsCustomTabItemContextMenuEnabled = true;
// Adding custom context menu for the first tabitem
CustomMenuItem customMenuItem = new CustomMenuItem() { Header="Menu1" };
tabItem1.ContextMenuItems.Add(customMenuItem);
// Adding custom context menu for the second tabitem
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "Menu1" };
tabItem2.ContextMenuItems.Add(customMenuItem1);
```



Note: View [Sample](#) in GitHub

Localization support

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

Refer the following links to know more about how provide a localization support for the TabControl,

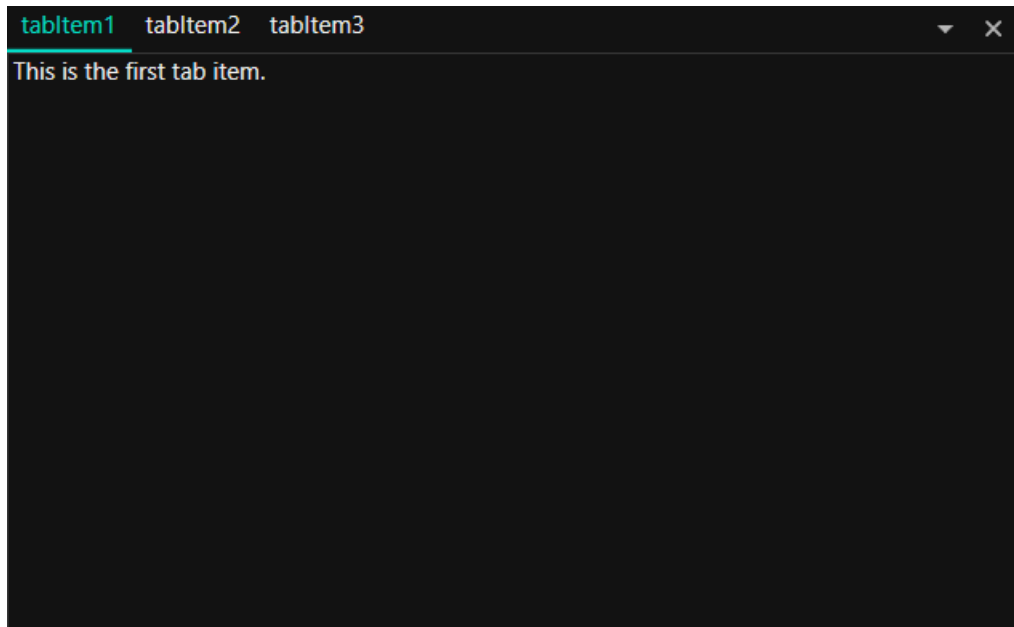
- <https://help.syncfusion.com/wpf/localization>
- <https://github.com/syncfusion/wpf-controls-localization-resx-files>

Theme

The WPF TabControl supports various built-in themes. Refer to the below links to apply themes for the TabControl,

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

Note: View [Sample](#) in GitHub



DataBinding in WPF TabControl (TabControlExt)

You can add a tab item using data binding in the WPF [TabControl](#).

Adding tab items using data binding

The `TabControl` can bound to an external source to auto create tabs and display the data using `ItemsSource` property. When you are auto generating tabitem using `ItemsSource`, you need to set `HeaderTemplate` property in `ItemContainerStyle` or `ItemTemplate` to define header and use `ContentTemplate` for display the content of the tab item.

If the data source implements `INotifyCollectionChanged` interface, then `TabControl` will automatically refresh the UI when item is added, removed or cleared in the collection. When an item is added or removed in `ObservableCollection`, `TabControl` automatically refresh the UI as `ObservableCollection` implements `INotifyCollectionChanged`. But when an item is added or removed in `List`, `TabControl` will not refresh the UI automatically.

Note: To bind `ItemsSource` to `TabControl`, you need to have collection with data object which holds header and content details.

Here, `Model` class defined with `Header` and `Content` properties and `ViewModel` class has `ItemsSource` property of type `ObservableCollection<Model>`.

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

C#

```
// Model.cs
public class Model {
    public string Header { get; set; }
```



```

public string Content { get; set; }
public Model() {
}
}
//ViewModel.cs
public class ViewModel : NotificationObject {
private ObservableCollection<Model> tabItems;
public ObservableCollection<Model> TabItems {
get { return tabItems; }
set {
tabItems = value;
this.RaisePropertyChanged("TabItems");
}
}
public ViewModel() {
tabItems = new ObservableCollection<Model>();
PopulateCollection();
}
public void PopulateCollection() {
Model model1 = new Model() {
Header = "tab1",
Content = "This is the content of first tabitem."
};
Model model2 = new Model() {
Header = "tab1",
Content = "This is the content of second tabitem."
};
//Adding the tab items into the collection
tabItems.Add(model1);
tabItems.Add(model2);
}
}

```

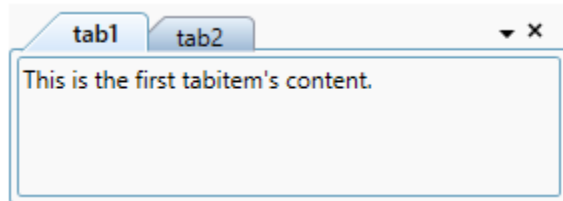
XML

```

<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
<syncfusion:TabControlExt ItemsSource="{Binding TabItems}"
Name="tabControlExt">
<!--Binding the header text for the Tab item-->
<syncfusion:TabControlExt.ItemContainerStyle>
<Style TargetType="syncfusion:TabItemExt">
<Setter Property="HeaderTemplate">
<Setter.Value>
<DataTemplate>
<TextBlock Text="{Binding Header, Mode=TwoWay}" />
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:TabControlExt.ItemContainerStyle>
<!--Binding the content for the Tab item-->
<syncfusion:TabControlExt.ContentTemplate>
<DataTemplate>

```

```
<TextBlock Text="{Binding Content}" />
</DataTemplate>
</syncfusion:TabControlExt.ContentTemplate>
</syncfusion:TabControlExt>
</Grid>
```



Note: View [Sample](#) in GitHub

TabItem Header

You can define tab item header using [ItemTemplate](#) or [HeaderTemplate](#) in [ItemContainerStyle](#) properties. Otherwise, Tab Item header will display the data object class name associated with tab item.

Note: The data object associated with tab item is the [BindingContext](#) for both [ItemContainerStyle.HeaderTemplate](#) and [ItemTemplate](#).

XML

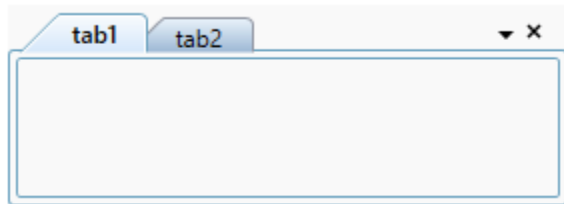
```
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
<syncfusion:TabControlExt x:Name="tabControlExt"
ItemsSource="{Binding TabItems}">
<syncfusion:TabControlExt.ItemContainerStyle>
<Style TargetType="syncfusion:TabItemExt">
<Setter Property="HeaderTemplate">
<Setter.Value>
<DataTemplate>
<TextBlock Text="{Binding Header, Mode=TwoWay}" />
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:TabControlExt.ItemContainerStyle>
</syncfusion:TabControlExt>
</Grid>
```

Below code is used to define the tabitem header using [ItemTemplate](#) property.

XML

```
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
<syncfusion:TabControlExt x:Name="tabControl"
ItemsSource="{Binding TabItems}">
<syncfusion:TabControlExt.ItemTemplate>
```

```
<DataTemplate>
<TextBlock Text="{Binding Header}"/>
</DataTemplate>
</syncfusion:TabControlExt.ItemTemplate>
</syncfusion:TabControlExt>
</Grid>
```



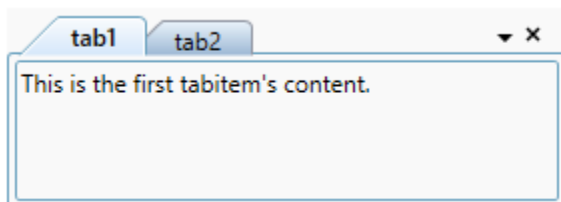
Note: View [Sample](#) in GitHub

TabItem content

You can define the tab item content using `ContentTemplate` property. Otherwise, Tab item content will display the data object class name which is associated with tab item.

XML

```
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
<syncfusion:TabControlExt x:Name="tabControl"
ItemsSource="{Binding TabItems}">
<syncfusion:TabControlExt.ContentTemplate>
<DataTemplate>
<TextBlock Text="{Binding Content}" />
</DataTemplate>
</syncfusion:TabControlExt.ContentTemplate>
</syncfusion:TabControlExt>
</Grid>
```



Editing tab header

By default, built-in `TextBox` is used as editor for the tab item header editing. You can customize the editing tab item header appearance for the each tab items by using the [EditHeaderTemplate](#) property.

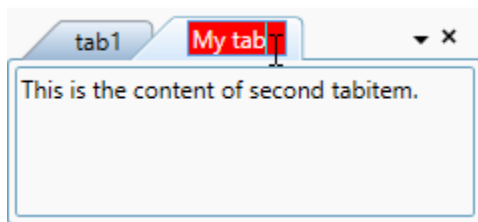
XML

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

```

<syncfusion:TabControlExt ItemsSource="{Binding TabItems}"
Name="tabControlExt"
Margin="20">
<syncfusion:TabControlExt.ItemContainerStyle>
<Style TargetType="syncfusion:TabItemExt">
<Setter Property="HeaderTemplate">
<Setter.Value>
<DataTemplate>
<TextBlock Text="{Binding Header}" />
</DataTemplate>
</Setter.Value>
</Setter>
<Setter Property="Content"
Value="{Binding Content}" />
</Style>
</syncfusion:TabControlExt.ItemContainerStyle>
<!--Custom UI for edit header template-->
<syncfusion:TabControlExt.EditHeaderTemplate>
<DataTemplate>
<TextBox Text="{Binding Header, Mode=TwoWay}"
Background="Red"
Foreground="White" />
</DataTemplate>
</syncfusion:TabControlExt.EditHeaderTemplate>
</syncfusion:TabControlExt>
</Grid>

```



Note: View [Sample](#) in GitHub

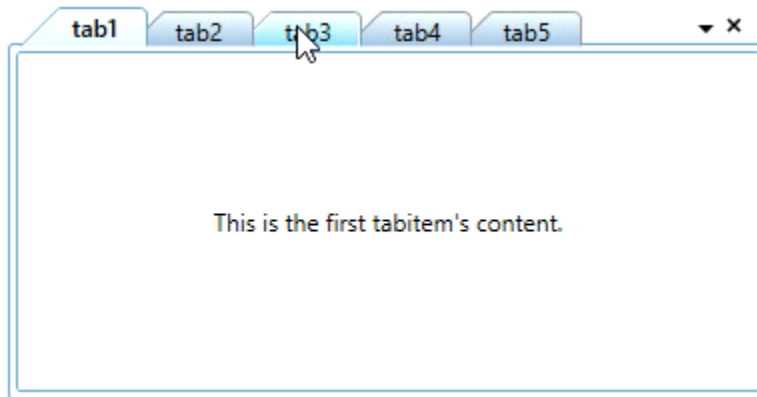
Select tab in WPF TabControl (TabControlExt)

This section explains how to select tab item and selection functionalities in the [TabControl](#).

Select tab item using mouse or keyboard

You can select a particular tab item by using the mouse click on the tab header. You can use the **Ctrl + Tab** key to select a next tab item when control not in focused state. You also use the **Left-Arrow** and **Right-Arrow** key , to select the previous tab item or next tab item of current selected tab item when control in focused state. You can get the selected item by using the **SelectedItem** property. By default, the first tab item is selected.

Note: You can select only one tab item at a time.



Tab selection changed notification

The `TabControl` notifies that the selected tabitem is changed by user through the [SelectedItemChangedEvent](#) event. You can use the [OldSelectedItem](#) and [NewSelectedItem](#) properties to get the old and new selected tabitem in the `SelectedItemChangedEvent` event.

XML

```
<syncfusion:TabControlExt
    SelectedItemChangedEvent="TabControlExt_SelectedItemChangedEvent"
    Name="tabControlExt" />
```

C#

```
TabControlExt tabControlExt = new TabControlExt();
tabControlExt.SelectedItemChangedEvent +=
    TabControlExt_SelectedItemChangedEvent;
```

You can handle the event as follows:

C#

```
private void TabControlExt_SelectedItemChangedEvent(object sender,
    SelectedItemChangedEventArgs e) {
    var newTabItem = e.NewSelectedItem.Header;
    if (e.OldSelectedItem != null) {
        var oldTabItem = e.OldSelectedItem.Header;
    }
    else {
        var oldTabItem = string.Empty;
    }
}
```

Load the previously selected tab item content

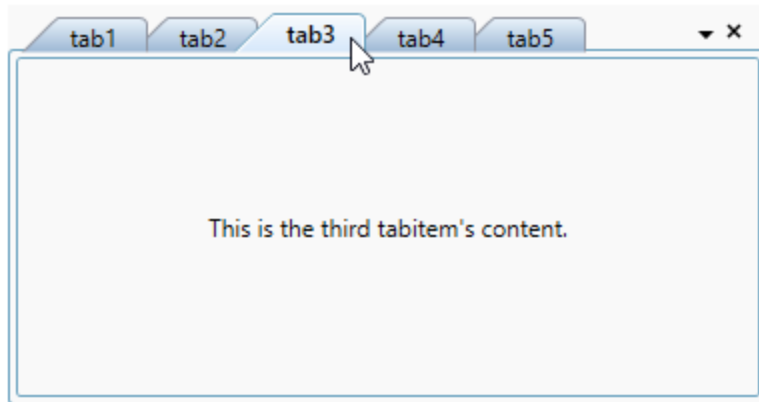
If you want to load the previously selected tab item in background after new tab item is selected, use the [IsDisableUnloadTabItemExtContent](#) property value as `true`. The default value of `IsDisableUnloadTabItemExtContent` property is `false`.

XML

```
<syncfusion:TabControlExt IsDisableUnloadTabItemExtContent="True"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tab1" Name="tabItemExt1"/>
<syncfusion:TabItemExt Header="tab2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.IsDisableUnloadTabItemExtContent = true;
```



Note: View [Sample](#) in GitHub

Display mode of the selected tab item

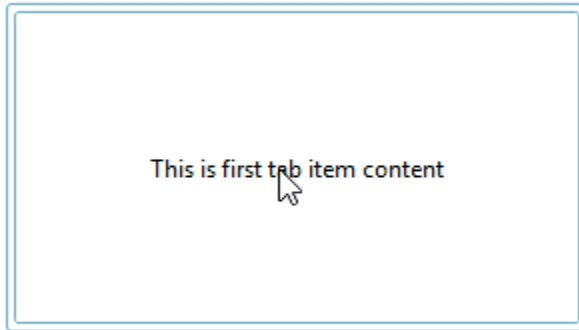
If you want to show tab items without its headers in the **TabControl**, use the [FullScreenMode](#) property. If you set **FullScreenMode** property value as **ControlMode**, then it will auto hide headers and show it only on when hover the mouse on respective tab header placed area. You can also display the **TabControl** in the full window by setting the **FullScreenMode** property value as **WindowMode**. The default value of **FullScreenMode** property is **None**.

XML

```
<syncfusion:TabControlExt FullScreenMode="ControlMode"
TabStripPlacement="Bottom"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
<syncfusion:TabItemExt Header="tabItem3" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.FullScreenMode = FullScreenMode.ControlMode;
tabControlExt.TabStripPlacement = Dock.Bottom;
```



Customize selected tab item header

You can change the selected tab item header font weight, background and foreground.

Change selected tab header font weight

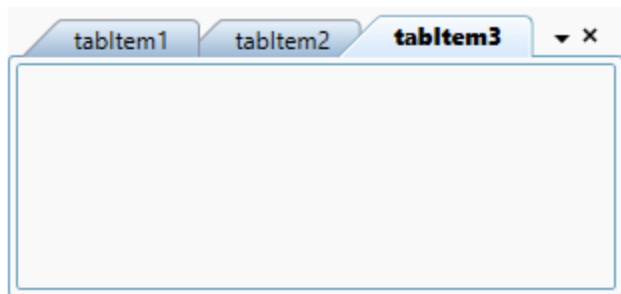
If you want to highlight the selected tab header, change the font of tab header from semi bold to extra bold by using the [SelectedItemFontWeight](#) property. The default value of `SelectedItemFontWeight` property is `SemiBold`.

XML

```
<syncfusion:TabControlExt SelectedItemFontWeight="ExtraBold"
Name="tabControlExt">
  <syncfusion:TabItemExt Header="tabItem1" Name="tabItemExt1"/>
  <syncfusion:TabItemExt Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.SelectedItemFontWeight = FontWeights.ExtraBold;
```



Note: View [Sample](#) in GitHub

Change selected tab header background and foreground

You can change the highlighting background and foreground of the selected tab item by using the [TabItemSelectedForeground](#) and [TabItemSelectedBackground](#) properties. The default value of `TabItemSelectedForeground` property is `Black` and `TabItemSelectedBackground` property is `Lavender`.

XML

```
<syncfusion:TabControlExt TabItemSelectedForeground="Red"
TabItemSelectedBackground="Green"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" Name="tabItemExt1"/>
<syncfusion:TabItemExt Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.TabItemSelectedForeground = Brushes.Red;
tabControlExt.TabItemSelectedBackground = Brushes.Green;
```



Note: View [Sample](#) in GitHub

Close tabs in WPF TabControl (TabControlExt)

This section explains how to closing the tab item and its functionalities in the [TabControl](#).

Closing tab item

You can close the selected tab by clicking the close button which is placed top-right corner of the tab panel.



Closing tab item using mouse middle click

You can close any tab item by clicking mouse middle button on the tab item header. You can enable it by setting the [CloseTabOnMiddleClick](#) property value as `true`. The default value of `CloseTabOnMiddleClick` property is `false`.

XML

```
<syncfusion:TabControlExt CloseTabOnMiddleClick="True"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" Name="tabItemExt1"/>
<syncfusion:TabItemExt Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```



```
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.CloseTabOnMiddleClick = true;
```

You can also close the tab items using the default tab item context menu. Refer this [page](#) to know more about the closing the tab item using context menu.

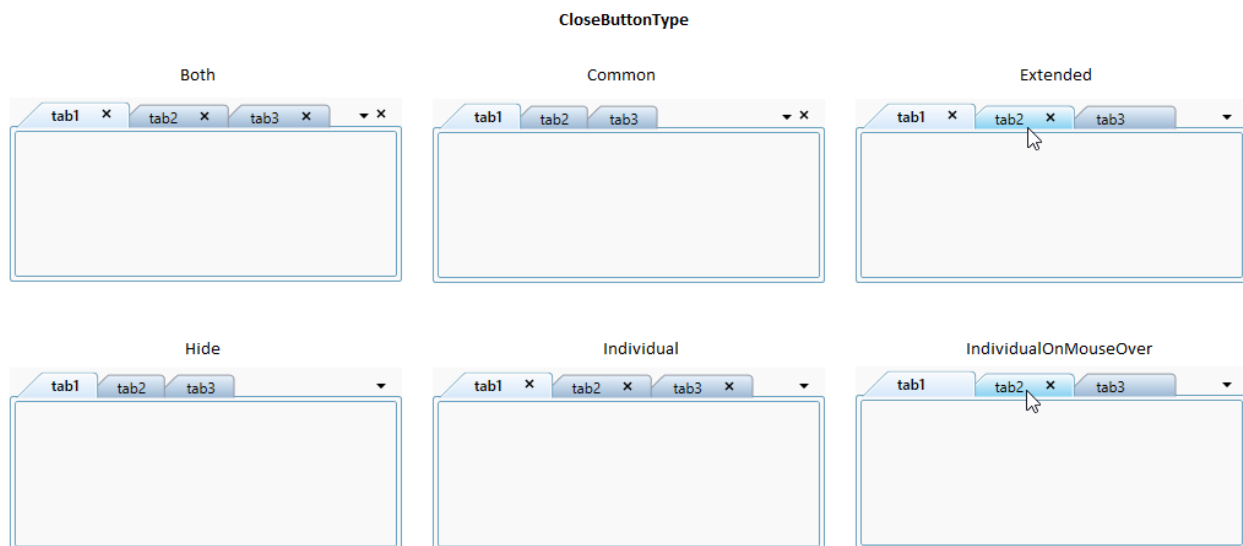
Note: View [Sample](#) in GitHub

Show or hide close button

You can show or hide the close button in each tabs and tab panel by using the [CloseButtonType](#) property.

The following Close button modes are supported by the **TabControl**.

- Common – The single close button is displayed commonly for all tab items in the tab panel.
- Individual – The close button is displayed individually for each tab items in the tab panel.
- Both – The close button is displayed individually for each tab item and also displayed in the tab panel.
- Hide – All close buttons is hidden in the tab panel.
- IndividualOnMouseOver - The close button is displayed when the mouse hovers over the tab item in the tab panel.
- Extended - The Close button is displayed in the selected tab and mousing hovering tab in the tab panel.

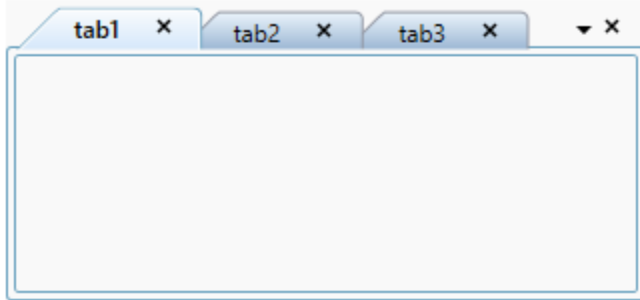


XML

```
<syncfusion:TabControlExt CloseButtonType="Both"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tab1" Name="tabItemExt1"/>
<syncfusion:TabItemExt Header="tab2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.CloseButtonType = CloseButtonType.Both;
```



Note: View [Sample](#) in GitHub

Show or hide close button for specific tab item

You can show or hide close button for particular tab item by using the [TabItemExt.CloseButtonState](#) property. The `TabItemExt.CloseButtonState` is effective only when the `CloseButtonType` property is set to one among `Individual`, `Extended`, or `Both`.

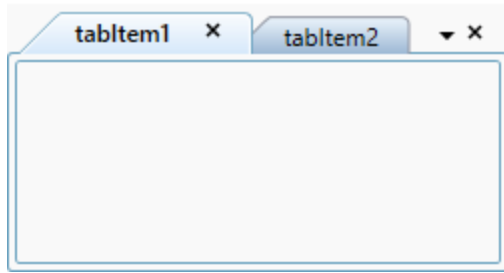
CloseButtonState	Description
Visible	The close button of TabItemExt is visible
Collapsed	The close button of TabItemExt is collapsed

XML

```
<syncfusion:TabControlExt CloseButtonType="Both"
Name="tabControlExt">
  <syncfusion:TabItemExt CloseButtonState="Visible"
Header="tabItem1" Name="tabItemExt1"/>
  <syncfusion:TabItemExt CloseButtonState="Collapsed"
Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.CloseButtonType = CloseButtonType.Both;
//Changing the particular item close button visibility
tabItemExt1.CloseButtonState = Visibility.Visible;
tabItemExt2.CloseButtonState = Visibility.Collapsed;
```



Note: View [Sample](#) in GitHub

Restrict or allow closing the tab item

You can restrict or allow the tab item closing by using either property or event.

Restrict closing the tab item using property

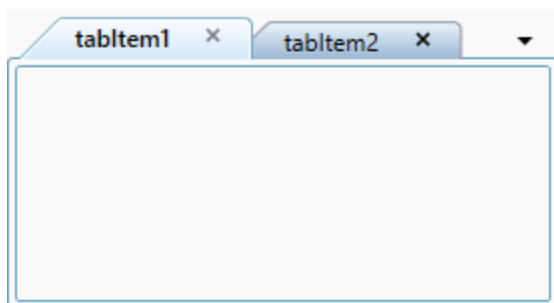
You can restrict the closing functionality of specific tab item using the [TabItemExt.CanClose](#) property. When the `TabItemExt.CanClose` property is set to `false`, the corresponding tab item will be non-closable. The default value of `TabItemExt.CanClose` property is `true`.

XML

```
<syncfusion:TabControlExt CloseButtonType="Individual"
    Name="tabControlExt">
  <syncfusion:TabItemExt CanClose="False"
    Header="tabItem1" Name="tabItemExt1"/>
  <syncfusion:TabItemExt CanClose="True"
    Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.CloseButtonType = CloseButtonType.Individual;
//Restrict and allow the particular item close button
tabItemExt1.CanClose = false;
tabItemExt2.CanClose = true;
```



Here, `tabItem1` cancel button is disabled.

Note: View [Sample](#) in GitHub

Restrict closing the tab item using event

The closing of tab item can be restricted by setting `e.Cancel` to `true` in [OnCloseButtonClick](#) event. `e` represents the event argument [CloseTabEventArgs](#) for `OnCloseButtonClick` event. The default value of `e.Cancel` is `false`.

XML

```
<syncfusion:TabControlExt
  OnCloseButtonClick="TabControlExt_OnCloseButtonClick" >
  <syncfusion:TabItemExt Header="tabItem1" />
  <syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.OnCloseButtonClick += TabControlExt_OnCloseButtonClick;
```

You can handle the event as follows:

C#

```
private void TabControlExt_OnCloseButtonClick(object sender,
  CloseTabEventArgs e) {
  if(e.TargetTabItem.Header.ToString() == "tabItem1") {
    e.Cancel = true;
  }
}
```

Here, `tabItem1` cannot be closed.

Note: View [Sample](#) in GitHub

Hide or delete item when closing a tab

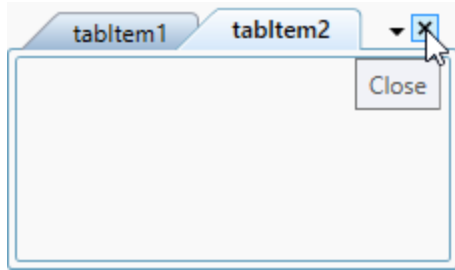
You can decide whether the tab item can be only hidden from the view or removed from the items collection of `TabControl` by using the [CloseMode](#) property while closing it. If you set `CloseMode` property as `Hide`, the tab item will be hidden and the selection will be moved to previous index while hiding it. Also, if the property `CloseMode` is `Delete`, the tab item will be removed from the items collection and the selection will be retained in the same index while removing it. The default value of the `CloseMode` property is `Hide`.

XML

```
<syncfusion:TabControlExt CloseMode="Delete"
  Name="tabControlExt">
  <syncfusion:TabItemExt Header="tabItem1" />
  <syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.CloseMode = CloseMode.Delete;
```



Note: View [Sample](#) in GitHub

Tab item closed notification

When the tab item is closed, it will be notified by using the [TabClosed](#) event. You can get the details about the closed tab item from [CloseTabEventArgs](#).

XML

```
<syncfusion:TabControlExt TabClosed="TabControlExt_TabClosed" >
  <syncfusion:TabItemExt Header="tabItem1" />
  <syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.TabClosed += TabControlExt_TabClosed;
```

You can handle the event as follows:

C#

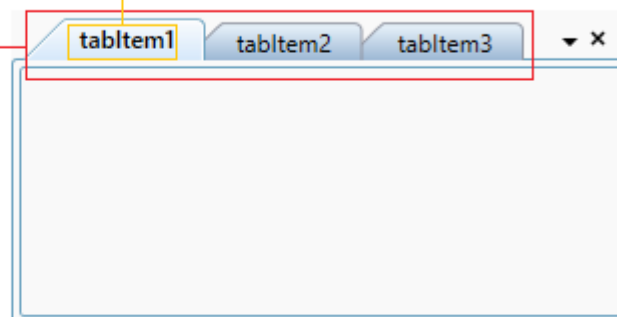
```
private void TabControlExt_TabClosed(object sender, CloseTabEventArgs e) {
    var deletedItem = e.TargetTabItem;
}
```

TabItem Header in WPF TabControl (TabControlExt)

This section explains how to set header text and UI customization of the tab header in the [TabControl](#).

Tab header text

Tabs



Setting tab item header

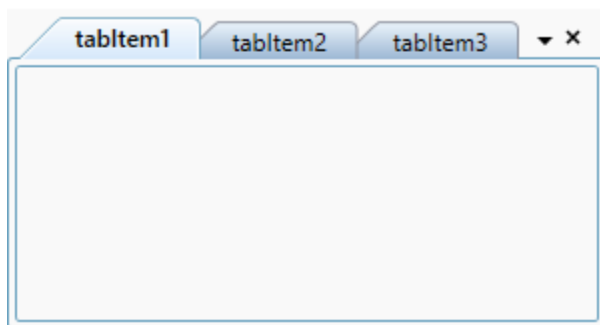
You can add a text for the each tab headers by using the `TabItemExt.Header` property.

XML

```
<syncfusion:TabControlExt Name="tabControlExt">
  <syncfusion:TabItemExt Header="tabItem1" Name="tabItemExt1"/>
  <syncfusion:TabItemExt Header="tabItem2" Name="tabItemExt2"/>
  <syncfusion:TabItemExt Header="tabItem3" Name="tabItemExt3"/>
</syncfusion:TabControlExt>
```

C#

```
// Creating an instances of tabitems and adding header
TabItemExt tabItemExt1 = new TabItemExt() { Header = "tabItem1" };
TabItemExt tabItemExt2 = new TabItemExt() { Header = "tabItem2" };
TabItemExt tabItemExt3 = new TabItemExt() { Header = "tabItem3" };
// Creating an instances of TabControl and adding the tabitems into the
TabControl
TabControlExt tabControlExt = new TabControlExt();
tabControlExt.Items.Add(tabItemExt1);
tabControlExt.Items.Add(tabItemExt2);
tabControlExt.Items.Add(tabItemExt3);
```

**Edit tab item header at runtime**

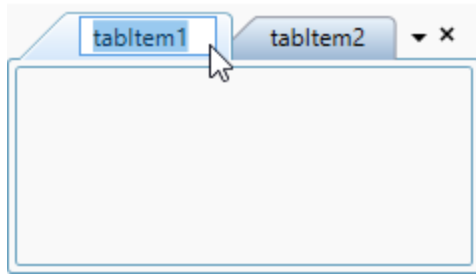
You can edit the text of the tab header at runtime by double clicking the tab header or selected a tab and pressing **Ctrl + F2** key. You can restrict all the tab header editing by using the [EnableLabelEdit](#) property value as **false**. The default value of **EnableLabelEdit** property is **true**.

XML

```
<syncfusion:TabControlExt EnableLabelEdit="True"
  Name="tabControlExt">
  <syncfusion:TabItemExt Header="tabItem1"/>
  <syncfusion:TabItemExt Header="tabItem2"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.EnableLabelEdit = true;
```



Note: View [Sample](#) in GitHub

Restrict header editing for specific tab item

If you want to restrict header editing for specific tab item, handle the [BeforeLabelEdit](#) and setting the [EnableLabelEdit](#) property as `true` for that specific tab item.

XML

```
<syncfusion:TabControlExt BeforeLabelEdit="tabControlExt_BeforeLabelEdit"
EnableLabelEdit="True"
Name="tabControlExt">
  <syncfusion:TabItemExt Header="tabItem1"/>
  <syncfusion:TabItemExt Header="tabItem2"/>
</syncfusion:TabControlExt>
```

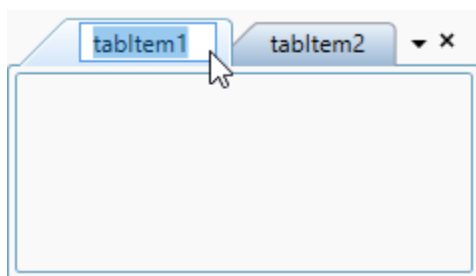
C#

```
tabControlExt.BeforeLabelEdit += tabControlExt_BeforeLabelEdit;
tabControlExt.EnableLabelEdit = true;
```

You can restrict the specific tab item as follows,

C#

```
private void tabControlExt_BeforeLabelEdit(object sender,
BeforeLabelEditEventArgs e) {
  //Restrict header editing for tabItem1
  if( e.HeaderBeforeEdit.ToString() == "tabItem1" ) {
    e.Cancel = true;
  }
}
```



Custom UI for the edit tab item header

You can customize the editing tab item header appearance for the each tab items by using the [EditHeaderTemplate](#) property.

Please refer [Editing tab header](#) topic to know more details about `EditHeaderTemplate`.

Note: View [Sample](#) in GitHub

Setting size and alignment of tab header

You can set a size of the each tabs by using the `TabItemExt.Width` and `TabItemExt.Height` properties. You can also align the header content horizontally and vertically by using the `TabItemExt.HorizontalContentAlignment` and `TabItemExt.VerticalContentAlignment` properties. The default value of `TabItemExt.HorizontalContentAlignment` and `TabItemExt.VerticalContentAlignment` properties is `Stretch`.

XML

```
<syncfusion:TabControlExt Name="tabControlExt">
  <syncfusion:TabItemExt Width="200" Height="30"
    HorizontalContentAlignment="Left"
    VerticalContentAlignment="Center"
    Header="tabItem1"
    Name="tabItemExt1"/>
  <syncfusion:TabItemExt Width="100" Height="30"
    VerticalContentAlignment="Bottom"
    HorizontalContentAlignment="Right"
    Header="tabItem2"
    Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
//Adding size and alignment of tabItemExt1 header
tabItemExt1.Width = 200;
tabItemExt1.Height = 300;
tabItemExt1.HorizontalContentAlignment = HorizontalAlignment.Left;
tabItemExt1.VerticalContentAlignment = VerticalAlignment.Center;
//Adding size and alignment of tabItemExt2 header
tabItemExt1.Width = 100;
tabItemExt1.Height = 30;
tabItemExt1.HorizontalContentAlignment = HorizontalAlignment.Right;
tabItemExt1.VerticalContentAlignment = VerticalAlignment.Bottom;
```



Setting image for tab item header

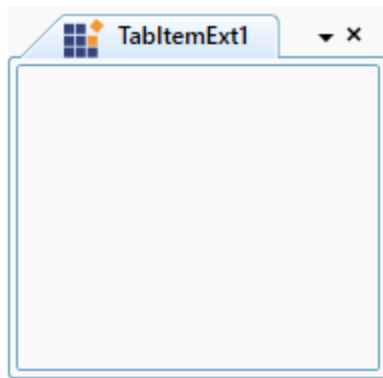
You can add images to the tab item headers by using the [TabItemExt.Image](#) property. You can change the height and width of the header image by using the [TabItemExt.ImageHeight](#) and [TabItemExt.ImageWidth](#) properties.

XML

```
<!-- Adding TabControlExt -->
<syncfusion:TabControlExt Name="tabControlExt">
  <!-- Adding TabItemExt with image -->
  <syncfusion:TabItemExt Image="sync_icon.ico"
    ImageWidth="20"
    ImageHeight="20"
    Header="TabItemExt1"
    Name="tabItemExt1">
  </syncfusion:TabItemExt>
</syncfusion:TabControlExt>
```

C#

```
//Adding header image for tabItemExt1
tabItemExt1.Image = new BitmapImage(
new Uri(@"sync_icon.ico", UriKind.RelativeOrAbsolute));
//Setting height and width for the tab header image
tabItemExt1.ImageWidth = 20;
tabItemExt1.ImageHeight = 20;
```



Note: View [Sample](#) in GitHub

Tab header image alignment

You can align the tab item header image by using the [TabItemExt.ImageAlignment](#) property. You can set a margin for the image by using the [TabItemExt.IconMargin](#) property. The default value of [TabItemExt.ImageAlignment](#) property is `LeftOfText` and [TabItemExt.IconMargin](#) property is `0,0,0,4`. You can align the image to any one of the following positions.

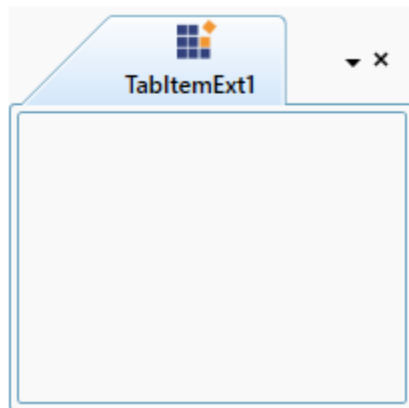
- `AboveText` - Images are placed above the tab item header text.
- `BelowText` - Images are placed below the tab item header text.
- `LeftOfText` - Images are placed to the left of the tab item header text.
- `RightOfText` - Images are placed to the right of the tab item header text.

XML

```
<!-- Adding TabControlExt -->
<syncfusion:TabControlExt Name="tabControlExt">
  <!-- Adding TabItemExt with image with alignment -->
  <syncfusion:TabItemExt Image="sync.png"
    ImageAlignment="AboveText"
    IconMargin= "2"
    Header="TabItemExt1"
    Name="tabItemExt1">
  </syncfusion:TabItemExt>
</syncfusion:TabControlExt>
```

C#

```
//Setting alignment for the tab header image
tabItemExt1.ImageAlignment = ImageAlignment.AboveText;
tabItemExt1.IconMargin = new Thickness(2);
```



Note: View [Sample](#) in GitHub

Setting tooltip

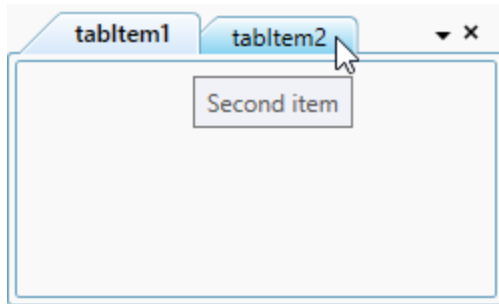
ToolTip is used to show the information about the segment, when you mouse over on the segment. You can add a tooltip information for the each tab items by using the [TabItemExt.ItemToolTip](#) property and show it by hovering the mouse on the respective header of the tab item.

XML

```
<syncfusion:TabControlExt Name="tabControlExt">
  <syncfusion:TabItemExt ItemToolTip="First item"
    Header="tabItem1" Name="tabItemExt1"/>
  <syncfusion:TabItemExt ItemToolTip="Second item"
    Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
tabItemExt1.ItemToolTip = "First item";
tabItemExt2.ItemToolTip = "Second item";
```



Hide tab header when there is single tab item

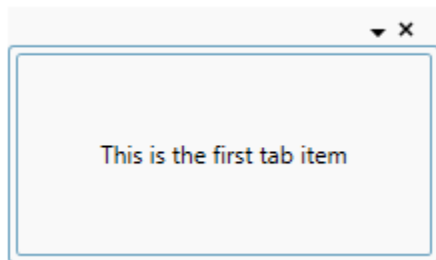
You can hide the header of tab item only on when single tab item present in the `TabControl`. You can enable it by using the [HideHeaderOnSingleChild](#) property value as `true`. The Default value of `HideHeaderOnSingleChild` property is `false`.

XML

```
<syncfusion:TabControlExt HideHeaderOnSingleChild="True"
Name="tabControlExt">
  <syncfusion:TabItemExt Content="This is the first tab item"
Header="tabItem1"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.HideHeaderOnSingleChild = true;
```



Note: View [Sample](#) in GitHub

Customize tab item header

You can change the tab item header background and foreground.

Change tab item background

If you want to change the tab item and its header panel background, use the `Background` and [TabPanelBackground](#) properties. You can change the hover background of all tab headers by using the [TabItemHoverBackground](#) property. You can also change the individual tab item header background and its hover background by using the `TabItemExt.Background` and [TabItemExt.HoverBackground](#) properties.

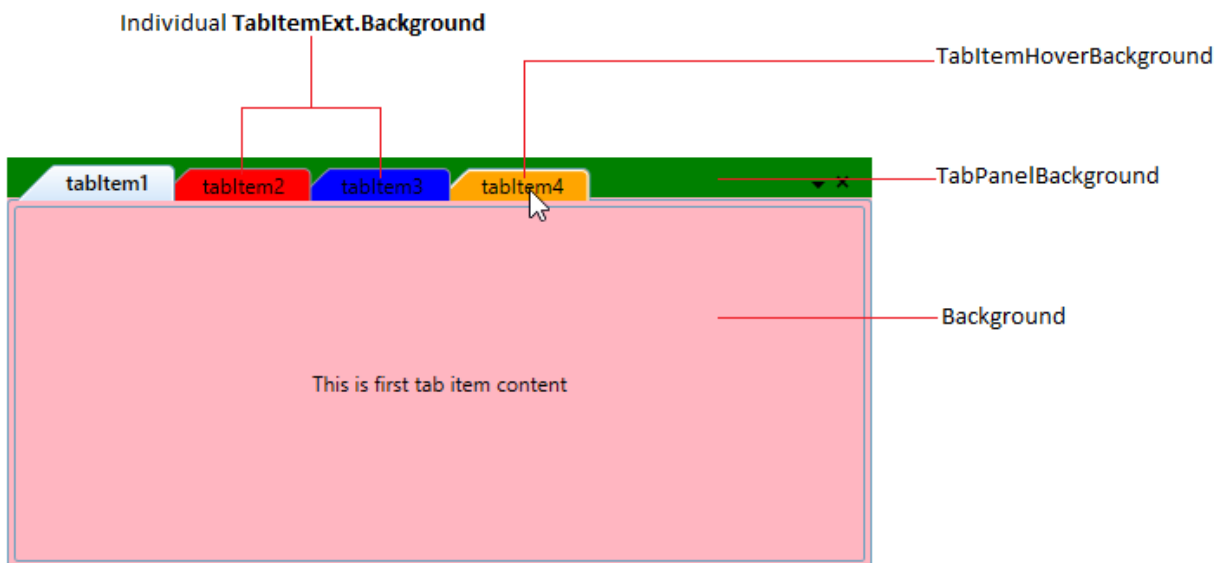
Note: If you use both `TabItemHoverBackground` and `TabItemExt.HoverBackground` for the tab item, the `TabItemExt.HoverBackground` have higher priority for that tab item.

XML

```
<syncfusion:TabControlExt Background="LightPink"
  TabPanelBackground="Green"
  TabItemHoverBackground="Orange"
  Name="tabControlExt">
  <syncfusion:TabItemExt HoverBackground="LightPink"
    Background="Red"
    Header="tabItem1"
    Name="tabItemExt1"/>
  <syncfusion:TabItemExt HoverBackground="Green"
    Background="Blue"
    Header="tabItem2"
    Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
//Setting the tab items panel, headers and its hover background
tabControlExt.Background = Brushes.LightPink;
tabControlExt.TabPanelBackground = Brushes.Green;
tabControlExt.TabItemHoverBackground = Brushes.Orange;
//Setting the header and hover background for the particular tab item
tabItemExt1.Background = Brushes.Red;
tabItemExt2.Background = Brushes.Blue;
tabItemExt1.HoverBackground = Brushes.LightPink;
tabItemExt2.HoverBackground = Brushes.Green;
```



Change tab item foreground

You can change the hover foreground of all tab headers by using the [TabItemHoverForeground](#) property. You can also change the individual tab item header foreground by using the `TabItemExt.Foreground` property. The default value of `TabItemHoverForeground` and `TabItemExt.Foreground` properties is `Black`.

XML

```
<syncfusion:TabControlExt TabItemHoverForeground="Green">
```

```

Name="tabControlExt">
<syncfusion:TabItemExt Foreground="Red"
Header="tabItem1"
Name="tabItemExt1"/>
<syncfusion:TabItemExt Foreground="Blue"
Header="tabItem2"
Name="tabItemExt2"/>
</syncfusion:TabControlExt>

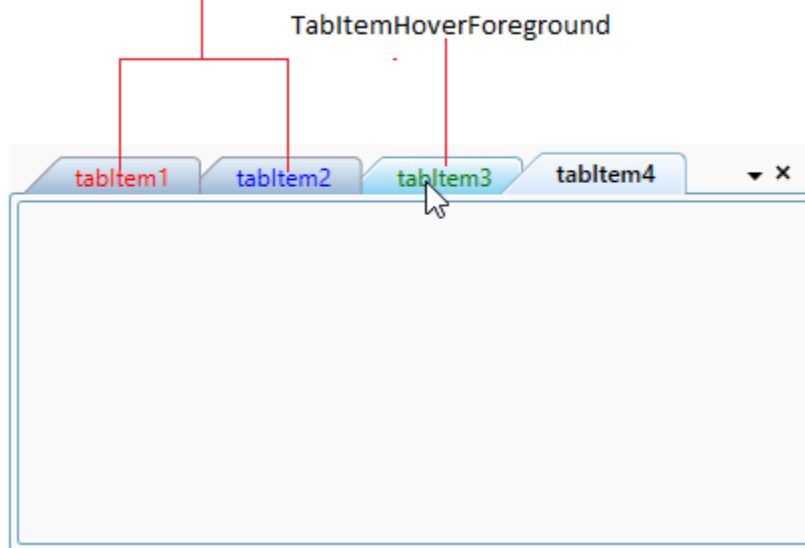
```

C#

```

//Setting the tab items hover foreground
tabControlExt.TabItemHoverForeground = Brushes.Green;
//Setting the header and item foreground for the particular tab item
tabItemExt1.Foreground = Brushes.Red;
tabItemExt2.Foreground = Brushes.Blue;

```

Individual TabItemExt.Foreground**Arrange tabs in WPF TabControl (TabControlExt)**

This section explains how to arrange the tab item and its alignment functionalities in the [TabControl](#).

Rearrange position of tab items

If you want to rearrange the tab items position, drag that item and drop to anywhere you want to place it in the tab panel. You can restrict it by setting the [AllowDragDrop](#) property value as `false`. The default value of `AllowDragDrop` property is `true`. The drag marker will preview the location, where you drop the dragged tab item.

XML

```

<syncfusion:TabControlExt AllowDragDrop="True" >
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
<syncfusion:TabItemExt Header="tabItem3" />

```

```
<syncfusion:TabItemExt Header="tabItem4" />
<syncfusion:TabItemExt Header="tabItem5" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.AllowDragDrop = true;
```

Note: View [Sample](#) in GitHub

*Rearrange position of tab items with auto scrolling*

You can easily move or rearrange tab items when there are several tab items by setting the `EnableAutoScroll` property as `true` and dragging the respective item over the overflow button (with three dots) or tab scroll buttons to autoscroll.

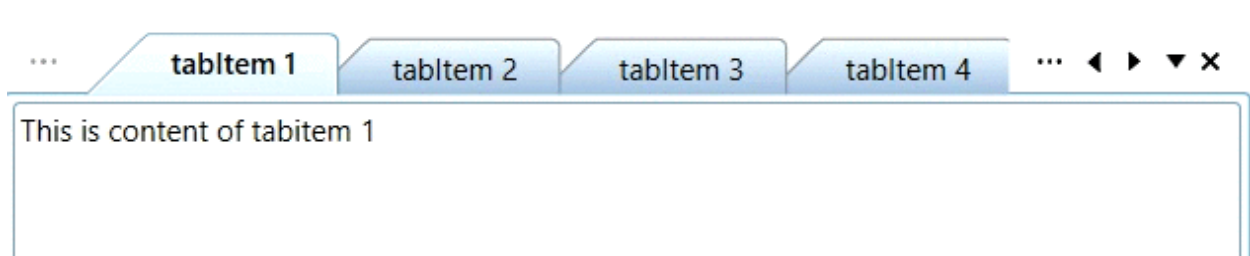
The default value of `EnableAutoScroll` property is `false`.

XML

```
<syncfusion:TabControlExt EnableAutoScroll="True">
<syncfusion:TabItemExt Header="tabItem 1" />
<syncfusion:TabItemExt Header="tabItem 2" />
<syncfusion:TabItemExt Header="tabItem 3" />
<syncfusion:TabItemExt Header="tabItem 4" />
<syncfusion:TabItemExt Header="tabItem 5" />
<syncfusion:TabItemExt Header="tabItem 6" />
<syncfusion:TabItemExt Header="tabItem 7" />
</syncfusion:TabControlExt>
```

C#

```
TabControlExt tabControlExt = new TabControlExt();
tabControlExt.EnableAutoScroll = true;
```



change drag marker color

You can change the drag marker color by setting the color value for the [DragMarkerColor](#) property. The default value of [DragMarkerColor](#) property is [Black](#).

XML

```
<syncfusion:TabControlExt DragMarkerColor="Red"
AllowDragDrop="True">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.DragMarkerColor = Brushes.Red;
tabControlExt.AllowDragDrop = true;
```



Note: View [Sample](#) in GitHub

Tab item order changed notification

You will be notified when the tab item's order is changed by using the [TabOrderChanged](#) event. You can get the order changed item with its old and new index values by using the [TargetItem](#), [OldIndex](#) and [NewIndex](#) properties.

Note: The [TabOrderChanged](#) event occurs only during drag and drop operation. Not occurs when add or remove items interactively or using code behind.

XML

```
<syncfusion:TabControlExt TabOrderChanged="TabControlExt_TabOrderChanged"
AllowDragDrop="True" >
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
<syncfusion:TabItemExt Header="tabItem3" />
</syncfusion:TabControlExt>
```

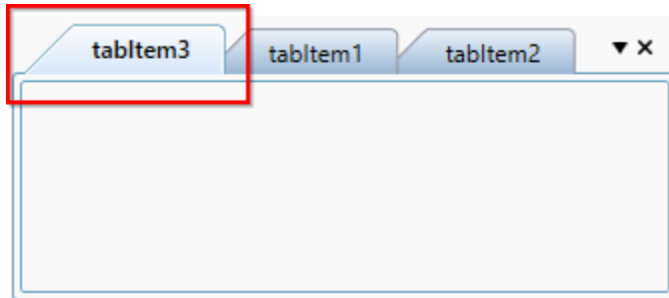
C#

```
tabControlExt.AllowDragDrop = true;
tabControlExt.TabOrderChanged += TabControlExt_TabOrderChanged;
```

You can handle the event as follows,

C#

```
private void TabControlExt_TabOrderChanged(object sender,
Syncfusion.Windows.Tools.Controls.TabOrderChangedEventArgs e)
{
    var drag_Drop_Item = e.TargetItem;
    var oldIndex = e.OldIndex;
    var newIndex = e.NewIndex;
}
```



Note: [View Sample in GitHub](#)

Restrict tab item reordering

If you want to restrict the user to reordering the tab item by drag and drop operation, use the [TabOrderChanging](#) event and set **Cancel** property value as **true**.

Note: The **TabOrderChanging** event occurs only during drag and drop operation. Not occurs when add or remove items using code behind.

XML

```
<syncfusion:TabControlExt TabOrderChanging="TabControlExt_TabOrderChanging"
AllowDragDrop="True" >
    <syncfusion:TabItemExt Header="tabItem1" />
    <syncfusion:TabItemExt Header="tabItem2" />
    <syncfusion:TabItemExt Header="tabItem3" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.AllowDragDrop = true;
tabControlExt.TabOrderChanging += TabControlExt_TabOrderChanging;
```

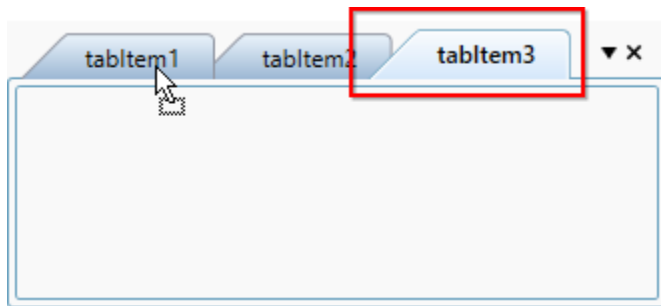
You can handle the event as follows,

C#

```
private void TabControlExt_TabOrderChanging(object sender,
Syncfusion.Windows.Tools.Controls.TabOrderChangedEventArgs e)
{
    // Restrict the Tab item order changing
    e.Cancel = true;
    var dragged_Item = e.TargetItem;
```



```
var oldIndex = e.OldIndex;  
var newIndex = e.NewIndex;  
}
```



Note: [View Sample in GitHub](#)

Tab items alignment

Tab items can be aligned to any side of the `TabControl` by using the `TabStripPlacement` property. The default value of `TabStripPlacement` property is `Top`.

The following `TabStrip` placement options are supported by the `TabControl`.

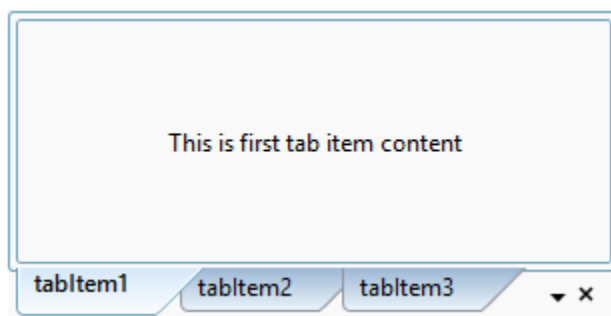
- `Top` - Tab items are placed at top of the `TabControl`.
- `Bottom` - Tab items are placed at bottom of the `TabControl`.
- `Left` - Tab items are placed at left side of the `TabControl`.
- `Right` - Tab items are placed at right side of the `TabControl`.

XML

```
<syncfusion:TabControlExt TabStripPlacement="Bottom">  
  <syncfusion:TabItemExt Header="tabItem1" />  
  <syncfusion:TabItemExt Header="tabItem2" />  
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.TabStripPlacement = Dock.Bottom;
```



Note: View [Sample](#) in GitHub

Rotating the tab items

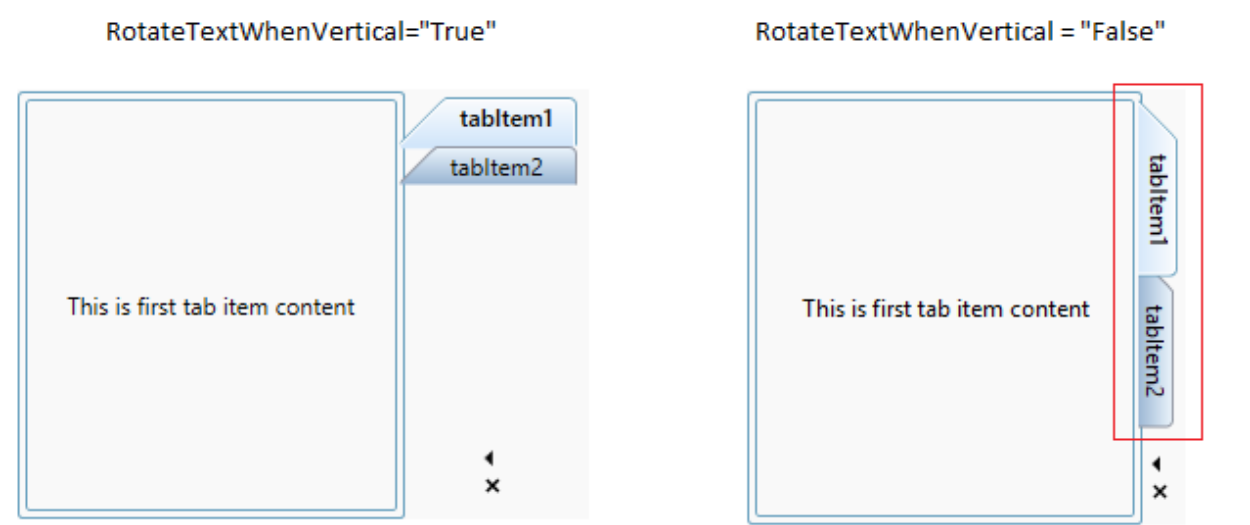
Whenever the `TabStripPlacement` is set to `Left` or `Right`, the tab headers are vertically arranged. To improve the user readability, the tab items can be rotated by using the `RotateTextWhenVertical` property as `true`. The default value of `RotateTextWhenVertical` property is `false`.

XML

```
<syncfusion:TabControlExt RotateTextWhenVertical="True"
TabStripPlacement="Right">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.RotateTextWhenVertical = true;
tabControlExt.TabStripPlacement = Dock.Right;
```







Note: View [Sample](#) in GitHub

Arrange tab item on single or multiple lines

You can arrange the tab item in single-line or multi-line by using the `TabItemLayout` property. By default, the tab items are arranged in a single line. The default value of `TabItemLayout` property is `SingleLine`.

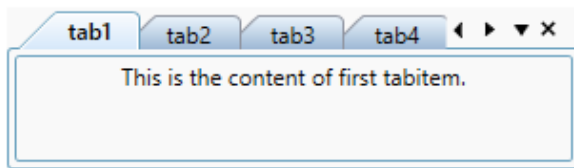
`TabControl` provides the following layout types.

TabItemLayout	Description	Image
SingleLine	Tabs are displayed in single line with default width. If you add more tabs and they exceed the TabControl size, then they are in hidden state and you can navigate to that hidden tabs only by using the scroll button or mouse scrolling. In this mode, each tab size not changed based on the TabControl size.	

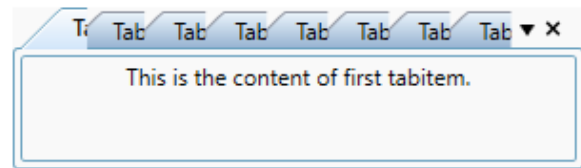
SingleLineStar	All the tabs are visible and arranged in single line. TabControl width is equally divided by the tabs. In this mode, each tab size changed based on the TabControl size.	
MultiLine	If you want to arrange tabs with equal width in multiple tab rows , use "MultiLine" mode.	
MultiLineWithFullWidth	To eliminate the empty space at the end of the row,"MultiLineWithFullWidth" is used. Here, the tabs arranged like in multiline except they expand to fill up the empty space at last row.	
MultiLineStar	If you want to arrange the multiple tabs with equal with and without any unwanted space at last tab, use the "MultiLineStar" mode. In this mode, multiple tabs are arranged in last row to fill the empty space	

Note: MultiLineStar mode is preferable for the TabControl which contains larger width.

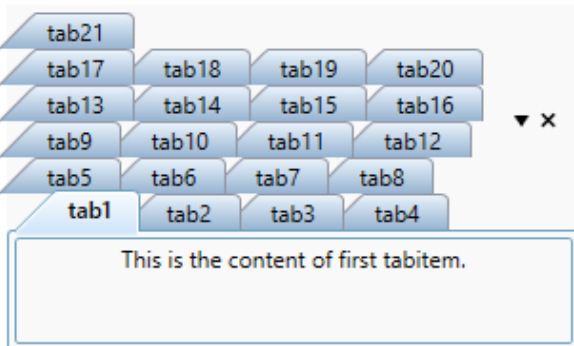
SingleLine



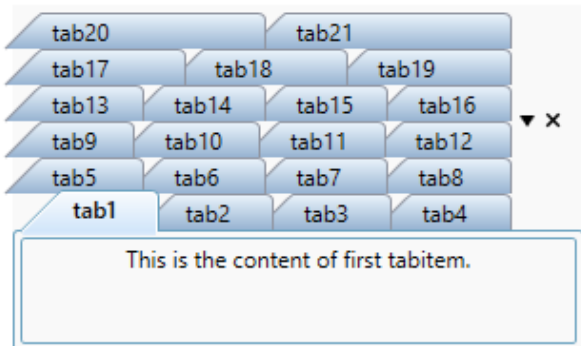
SingleLineStar



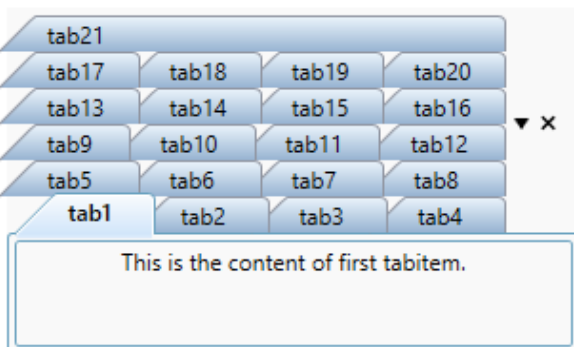
MultiLine



MultiLineStar



MultiLineWithFullWidth



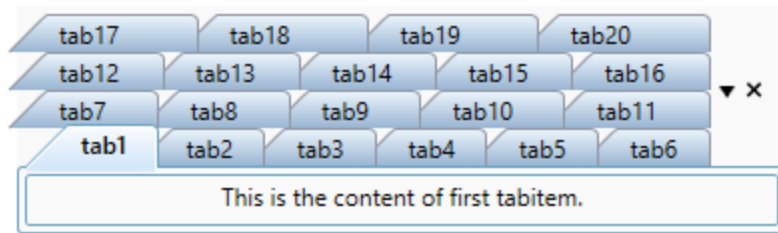
Note: View [Sample](#) in GitHub

XML

```
<syncfusion:TabControlExt TabItemLayout="MultiLineStar"
Width="300"
Name="tabControlExt">
  <syncfusion:TabItemExt Header="tabItem1" />
  <syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.TabItemLayout = TabItemLayoutType.MultiLineStar;
```



Note: View [Sample](#) in GitHub

Restrict size of a TabItem

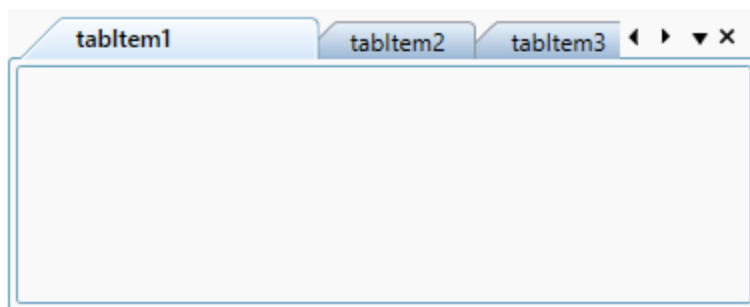
By default, size of the tab item is based on its content. When content size changes, tab item size also changes. You can restrict it for any `TabItemLayout` modes by using the `TabItemExt.Width`, `TabItemExt.MinWidth` and `TabItemExt.MaxWidth` properties. You can resize the tab item only within the `TabItemExt.MinWidth` and `TabItemExt.MaxWidth` values.

XML

```
<syncfusion:TabControlExt TabItemLayout="SingleLine"
Margin="30"
Name="tabControlExt">
<syncfusion:TabItemExt MinWidth="100"
Header="tabItem1"/>
<syncfusion:TabItemExt Header="tabItem2" />
<syncfusion:TabItemExt Header="tabItem3" />
</syncfusion:TabControlExt>
```

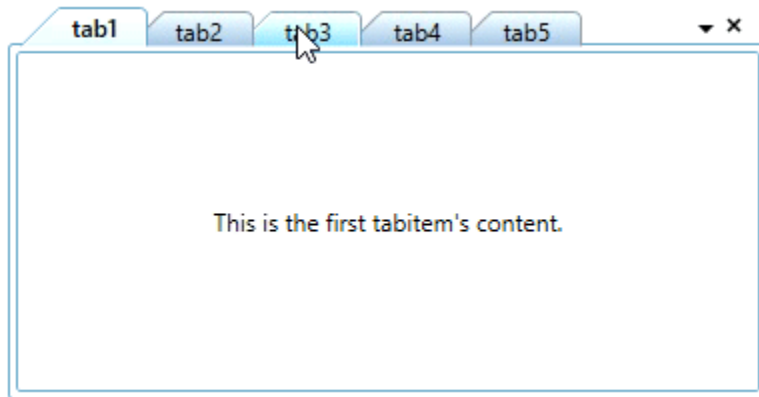
C#

```
tabControlExt.TabItemLayout = TabItemLayoutType.SingleLine;
```



Navigate using mouse or keyboard

You can navigate from one tab to any other tab by using the mouse click on the tab header. You can use the `Ctrl + Tab` key to navigate to the next tab item when control not in focused state. You also use the `Left-Arrow` and `Right-Arrow` key, to navigate the previous tab item or next tab item from the current tab item.



Navigate using scroll button

If you add more tab items, then some tab headers are collapsed. If you navigate to the collapsed tab items, click the scroll button which is placed in the top-right corner of the tab header panel. You can auto visible or hide the scroll button by using the [TabScrollButtonVisibility](#) property value as **Auto** or **Hidden**.

The following **TabScrollStyle** supported by the **TabControl** control.

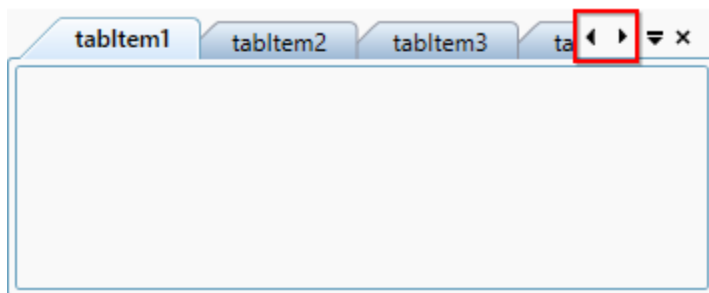
- Extended Mode - Provides the Next, Previous, Last and First navigation options
- Normal Mode – Provides the Next and Previous navigation options only

XML

```
<syncfusion:TabControlExt TabScrollButtonVisibility="Auto"
Width="300"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.TabScrollButtonVisibility = TabScrollButtonVisibility.Auto;
```



Navigate using scroll button in extended mode

You can easily navigate to first or last or next or previous tab items by setting the [TabScrollStyle](#) property value as `Extended`, then it will show the `First`, `Last`, `Next`, `Previous` button options in the scroll button. The default value of the `TabScrollStyle` property is `Normal`.

XML

```
<syncfusion:TabControlExt TabScrollButtonVisibility="Auto"
TabScrollStyle="Extended"
Width="300"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.TabScrollStyle = TabScrollStyle.Extended;
```



Note: View [Sample](#) in GitHub

Navigate using tab list menu

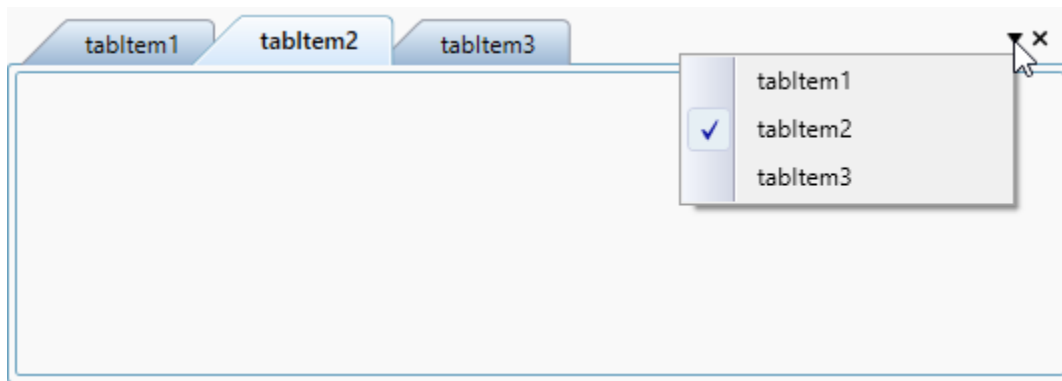
You can easily navigate to any tab item by using the tab list menu which is placed in the top-right corner of the tab header panel. The header of all tab item's are shown as a menu item in the tab list menu. You can hide this tab list menu by using the [ShowTabListContextMenu](#) property value as `false`. The default value of `ShowTabListContextMenu` property is `true`.

XML

```
<syncfusion:TabControlExt ShowTabListContextMenu="True"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.ShowTabListContextMenu = true;
```



Note: View [Sample](#) in GitHub

Scroll items using mouse wheel

You can scroll the tab items in `TabItemLayout.SingleLine` layout mode, by mouse over tab header and scroll the mouse backward or forward direction.



Pin and Unpin TabItems in WPF TabControl (TabControlExt)

This section explains the pin and unpin tab items in [TabControl](#).

Enabling pin and unpin behaviors

If you want to pin or unpin the tab items, use the `TabItemExt.AllowPin` property as `true`. When this property is set to `false`, the pin and unpin behaviors of tab item will be disabled. The default value of the `TabItemExt.AllowPin` property is `false`.

XML

```
<syncfusion:TabControlExt Name="tabControlExt">
  <syncfusion:TabItemExt AllowPin="True"
    Header="tabItem1" Name="tabItemExt1"/>
  <syncfusion:TabItemExt AllowPin="True"
    Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
tabItemExt1.AllowPin = true;
tabItemExt2.AllowPin = true;
```

Note: View [Sample](#) in GitHub

Functionality of PinButton

When the corresponding tab item is pinned, it will be inserted at first position of the tab header panel(if its not have any pinned tab. Otherwise, the pinned tab item will be added next to last pinned item). When the tab item is unpinned, it will be placed after to the pinned tab items.

Pin and Unpin tab items using PinButton

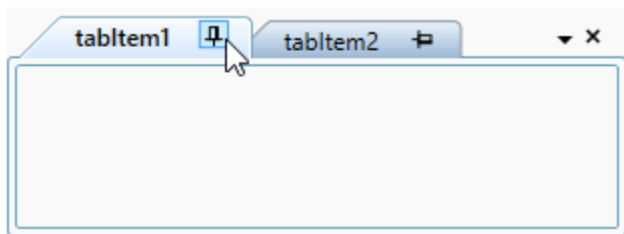
If you want to pin or unpin the tab item using the pin button, use the [TabItemExt.ShowPin](#) property value as `true` and set the [TabItemExt.AllowPin](#) property value as `true`. The default value of [TabItemExt.ShowPin](#) property is `false`, so the PinButton is collapsed from header panel of the tab item.

XML

```
<syncfusion:TabControlExt Name="tabControlExt">
  <syncfusion:TabItemExt ShowPin="True"
    AllowPin="True"
    Header="tabItem1" Name="tabItemExt1"/>
  <syncfusion:TabItemExt ShowPin="True"
    AllowPin="True"
    Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
//Showing the pin buttons
tabItemExt1.ShowPin = true;
tabItemExt2.ShowPin = true;
//Enabling the pin buttons
tabItemExt1.AllowPin = true;
tabItemExt2.AllowPin = true;
```



Note: If the [ShowPin](#) property is `true`, and the [AllowPin](#) property is `false`, the PinButton will be displayed as a disabled button.

Note: View [Sample](#) in GitHub

Pin and Unpin the tab items programmatically

You can pin or unpin the tab items programmatically by using the [TabItemExt.IsPinned](#) property. If the [TabItemExt.IsPinned](#) property is set to `true`, the corresponding item will be pinned. Also, if the [TabItemExt.IsPinned](#) property is set as `false` and the item is pinned, then it will be unpinned. The default value of [TabItemExt.IsPinned](#) property is `false`.

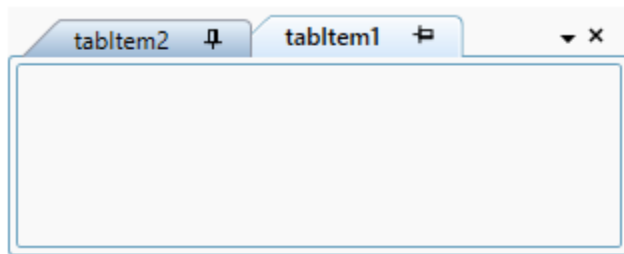
XML

```
<syncfusion:TabControlExt Name="tabControlExt">
```

```
<syncfusion:TabItemExt ShowPin="True"
AllowPin="True"
IsPinned="False"
Header="tabItem1" Name="tabItemExt1"/>
<syncfusion:TabItemExt ShowPin="True"
AllowPin="True"
IsPinned="True"
Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
//Pin or unpin the tab items programmatically
tabItemExt1.IsPinned = false;
tabItemExt2.IsPinned = true;
//Showing the pin buttons
tabItemExt1.ShowPin = true;
tabItemExt2.ShowPin = true;
//Enabling the pin buttons
tabItemExt1.AllowPin = true;
tabItemExt2.AllowPin = true;
```



Note: View [Sample](#) in GitHub

Pin and Unpin tab items using ContextMenu

You can pin or unpin the tab items using the context menu. You can enable it by setting the `TabItemExt.AllowPin` property value as `true`. If the `TabItemExt.AllowPin` property is `true`, and the tab item is not pinned, "Pin Tab" option will be visible. If the tab item is pinned already, "Unpin Tab" option will be visible in the context menu.

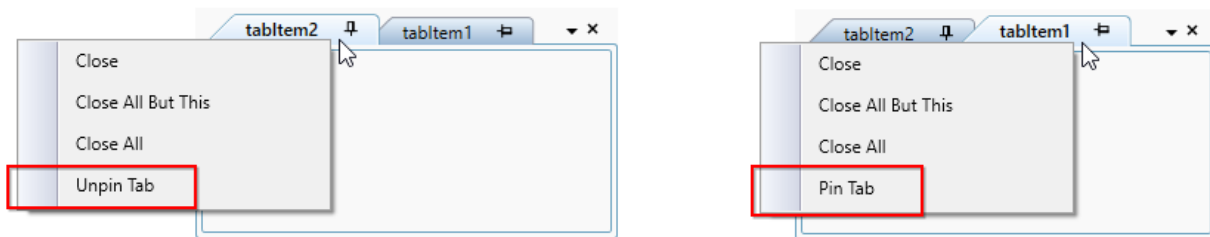
Note: If you want to show the pin and unpin buttons, use the `TabItemExt.ShowPin` value as `true`.

XML

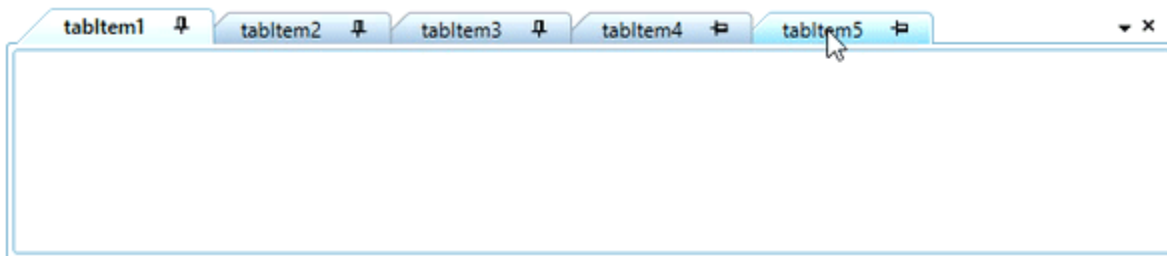
```
<syncfusion:TabControlExt Name="tabControlExt">
<syncfusion:TabItemExt ShowPin="True"
AllowPin="True"
IsPinned="False"
Header="tabItem1" Name="tabItemExt1"/>
<syncfusion:TabItemExt ShowPin="True"
AllowPin="True"
IsPinned="True"
Header="tabItem2" Name="tabItemExt2"/>
</syncfusion:TabControlExt>
```

C#

```
//Pin or unpin the tab items programmatically
tabItemExt1.IsPinned = false;
tabItemExt2.IsPinned = true;
//Showing the pin buttons
tabItemExt1.ShowPin = true;
tabItemExt2.ShowPin = true;
//Enabling the pin buttons
tabItemExt1.AllowPin = true;
tabItemExt2.AllowPin = true;
```

**Re-order pinned tabs**

You can re-order the pinned item within the pinned items and re-order the un-pinned item within the unpinned, but re-ordering between pinned and unpinned or unpinned and pinned has been restricted. If the pinned tab item is dropped inside the unpinned tab items, the dragged item will be inserted at the last position of pinned tab item and if the unpinned tab item is dropped inside pinned tab items, the dragged item will be inserted after the last pinned item. You can disable this reordering by using the `AllowDragDrop` property as `false`.

**NewButton Feature in WPF TabControl (TabControlExt)**

This section explains how to create new tab items using new button and its UI customization in the [TabControl](#).

Adding New tab button and new tab item

You can add a new tab item at runtime by clicking the new button. You can enable it by using the `IsNewButtonEnabled` property. You should handle the click action of the New Button and add a new tab items to `TabControl` by using the `NewButtonClick` event. The new button can be aligned first or last position in the tab item header panel by using the `NewButtonAlignment` property. The default value of `IsNewButtonEnabled` property is `false` and `NewButtonAlignment` property is `Last`.

XML

```
<syncfusion:TabControlExt NewButtonClick="tabControlExt_NewButtonClick"
IsNewButtonEnabled="True"
```

```

NewButtonAlignment="First"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>

```

C#

```

tabControlExt.NewButtonClick += tabControlExt_NewButtonClick;
tabControlExt.IsNewButtonEnabled = true;
tabControlExt.NewButtonAlignment = NewButtonAlignment.First;

```

You can handle the event as follows:

C#

```

private void tabControlExt_NewButtonClick(object sender, EventArgs e) {
    TabItemExt tabItemExt1 = new TabItemExt()
    {
        Header = "tabItem"+ (tabControlExt.Items.Count + 1),
        Content = new TextBlock() { Text = "This is the content area of TabItem" }
    };
    //Adding new tab item into the TabControl.
    tabControlExt.Items.Add(tabItemExt1);
}

```



Note: View [Sample](#) in GitHub

Select a new tab item while creating it by new button

If you want to select the recently creating tab item as the selected item that is created by the new button click, use [SelectOnCreatingNewItem](#) property value as `true`. You can restrict it by setting the [SelectOnCreatingNewItem](#) property value as `false`. The default value of [SelectOnCreatingNewItem](#) property is `true`.

XML

```

<syncfusion:TabControlExt SelectOnCreatingNewItem="True"
NewButtonClick="tabControlExt_NewButtonClick"
IsNewButtonEnabled="True"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>

```

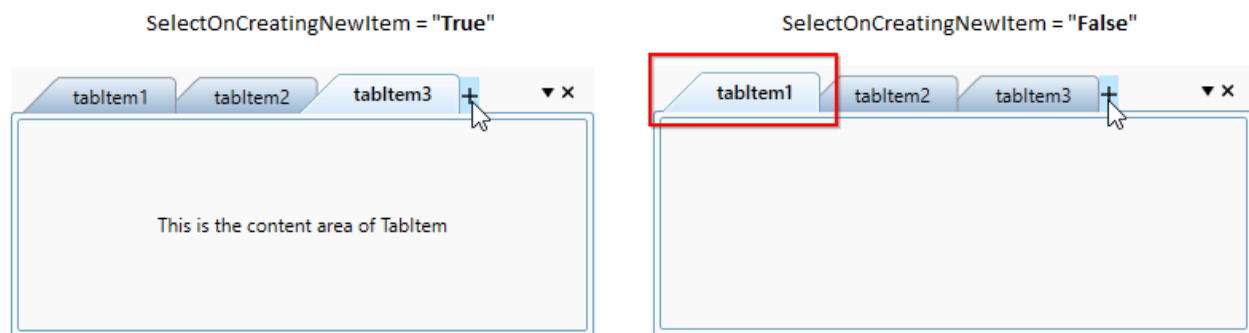
C#

```
tabControlExt.NewButtonClick += tabControlExt_NewButtonClick;
tabControlExt.IsNewButtonEnabled = true;
tabControlExt.SelectOnCreatingNewItem = true;
```

You can handle the event as follows:

C#

```
private void tabControlExt_NewButtonClick(object sender, EventArgs e) {
    TabItemExt tabItemExt1 = new TabItemExt()
    {
        Header = "tabItem" + (tabControlExt.Items.Count + 1),
        Content = new TextBlock() { Text = "This is the content area of TabItem" }
    };
    //Adding new tab item into the TabControl.
    tabControlExt.Items.Add(tabItemExt1);
}
```



Note: View [Sample](#) in GitHub

Auto hide new button when no child tab item

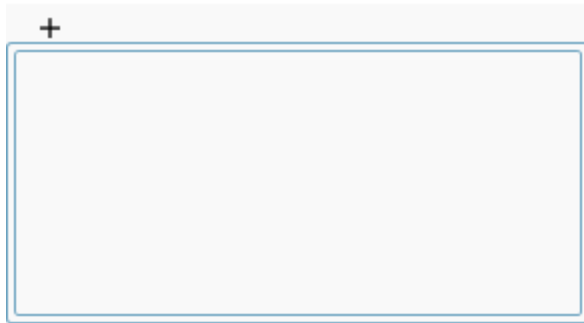
The **TabControl** automatically hides the new button on when no child tab item present in the **TabControl**. If you want to always show the new button, use the [IsNewButtonClosedonNoChild](#) property value as **false**. The default value of **IsNewButtonClosedonNoChild** property is **true**.

XML

```
<syncfusion:TabControlExt IsNewButtonClosedonNoChild="False"
IsNewButtonEnabled="True"
Name="tabControlExt">
<syncfusion:TabItemExt Content="This is the first tab item"
Header="tabItem1"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.IsNewButtonClosedonNoChild = false;
tabControlExt.IsNewButtonEnabled = true;
```



Note: View [Sample](#) in GitHub

Custom template for the new button

If you want to change the UI for the new button, use the [NewTabButtonTemplate](#) property.

XML

```
<syncfusion:TabControlExt IsNewButtonEnabled="True"
x:Name="tabControlExt">
  <syncfusion:TabControlExt.NewTabButtonTemplate>
    <DataTemplate>
      <TextBlock Background="Red"
        FontSize="14"
        TextAlignment="Center"
        Width="25" Height="25"
        Text=" + " />
    </DataTemplate>
  </syncfusion:TabControlExt.NewTabButtonTemplate>
  <syncfusion:TabItemExt Header="tabItem1" Name="tabItemExt1" />
  <syncfusion:TabItemExt Header="tabItem2" Name="tabItemExt2" />
</syncfusion:TabControlExt>
```



Note: View [Sample](#) in GitHub

Change background and border thickness of new button

If you want to change the background and border thickness of the new button, use the [NewButtonBackground](#) and [NewButtonBorderThickness](#) properties. The default value of [NewButtonBackground](#) property is `null` and property is .

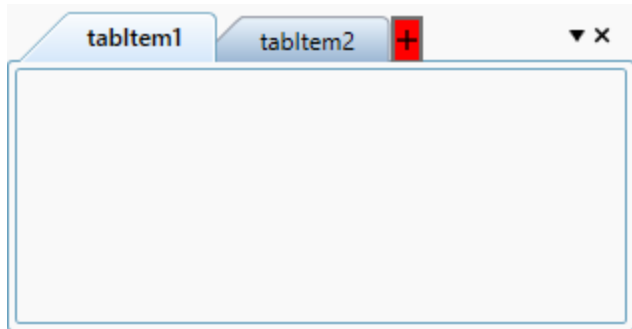
XML

```
<syncfusion:TabControlExt NewButtonBackground="Red"
  NewButtonBorderThickness="2">
```

```
Name="tabControlExt">
<syncfusion:TabItemExt Content="This is the first tab item"
Header="tabItem1"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.NewButtonBackground = Brushes.Red;
tabControlExt.NewButtonBorderThickness = new Thickness(2);
```



Note: View [Sample](#) in GitHub

Context menu in WPF TabControl (TabControlExt)

This section explains how to show the tab list and tab item context menu and add custom context menu in [TabControl](#).

Default tab item context menu

You can close the one or more tab items by using the context menu. You can enable it by using the [ShowTabItemContextMenu](#) property to true. You can open this context menu by right click on tab header. The default value of ShowTabItemContextMenu property is false.

The built-in tabitem context menu has the following menu items:

- **Close** - Closes the selected tab item.
- **Close All But This** - Closes all the tab items, except the selected tab item.
- **Close All** - Closes all the tab items.

Note: The [CloseMode](#) in [TabControl](#) is an enum that includes option **Hide** and **Delete**. When you click any default options (Close, Close All and Close All But This) in tabitem context menu and when the [CloseMode](#) is set to **Hide**, the tabitem moves to the hidden state or when the [CloseMode](#) is set to **Delete**, the tabitem is completely deleted from [TabControl](#).

XML

```
<syncfusion:TabControlExt ShowTabItemContextMenu="True"
CloseMode="Hide"
Name="tabControlExt">
<syncfusion:TabItemExt Content="This is the first tab item"
Header="tabItem1"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.ShowTabItemContextMenu = true;
tabControlExt.CloseMode = CloseMode.Hide;
```



Note: View [Sample](#) in GitHub

Tab item context menu events

Closing the tab items using the context menu can be notified by following events.

- Close option - [OnCloseButtonClick](#) event
- Close All But This option - [OnCloseOtherTabs](#) event
- Close All option - [OnCloseAllTabs](#) event.

You can use the [CloseTabEventArgs.ClosingTabItems](#) property to find out which tabitem is closing from the [OnCloseOtherTabs](#) and [OnCloseAllTabs](#) events and use the [CloseTabEventArgs.TargetTabItem](#) property to find out which tabitem is closing from the [OnCloseButtonClick](#) event.

XML

```
<syncfusion:TabControlExt OnCloseAllTabs="TabControlExt_OnCloseAllTabs"
OnCloseButtonClick="TabControlExt_OnCloseButtonClick"
OnCloseOtherTabs="TabControlExt_OnCloseAllTabs"
Name="tabControlExt">
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.OnCloseAllTabs += TabControlExt_OnCloseAllTabs;
tabControlExt.OnCloseButtonClick += TabControlExt_OnCloseButtonClick;
tabControlExt.OnCloseOtherTabs += TabControlExt_OnCloseOtherTabs;
```

You can handle this events as follows,

C#

```
private void TabControlExt_OnCloseAllTabs(object sender, CloseTabEventArgs
e) {
var closingTabItems = e.ClosingTabItems;
}
```



```
private void TabControlExt_OnCloseButtonClick(object sender,
CloseTabEventArgs e) {
    var closingTabItem = e.TargetTabItem;
}
private void TabControlExt_OnCloseOtherTabs(object sender, CloseTabEventArgs
e) {
    var closingTabItems = e.ClosingTabItems;
}
```

Custom context menu item for the tab items

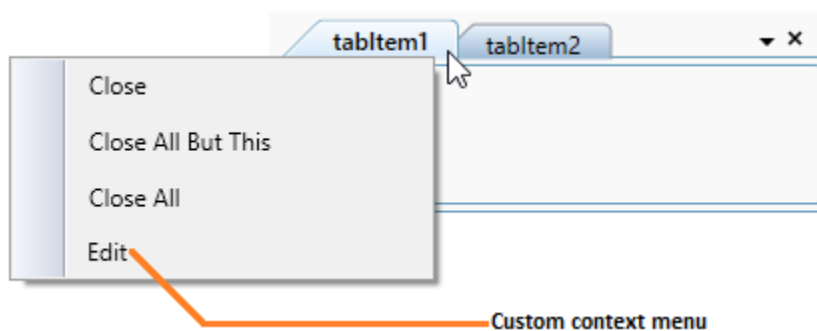
You can add the custom context menu item for the tab item using the [TabItemExt.ContextMenuItems](#) property. You can perform any operation by handling the [CustomMenuItem.Click](#) event. You can enable it by setting the [TabControlExt.IsCustomTabItemContextMenuEnabled](#) property to true. The default value of [TabControlExt.IsCustomTabItemContextMenuEnabled](#) property is false.

XML

```
<syncfusion:TabControlExt IsCustomTabItemContextMenuEnabled="True"
Name="tabControlExt" >
    <syncfusion:TabItemExt Header="tabItem1">
        <!--Adding custom context menu items-->
        <syncfusion:TabItemExt.ContextMenuItems>
            <syncfusion:CustomMenuItem Header="Edit" />
        </syncfusion:TabItemExt.ContextMenuItems>
    </syncfusion:TabItemExt>
</syncfusion:TabControlExt>
```

C#

```
// Enable custom tabitem context menu
tabControlExt.IsCustomTabItemContextMenuEnabled = true;
// Adding custom context menu for the tabitem
CustomMenuItem customMenuItem = new CustomMenuItem();
customMenuItem.Header = "Edit";
tabItemExt1.ContextMenuItems.Add(customMenuItem);
```



Note: View [Sample](#) in GitHub

Check or uncheck the custom context menu items

You can check or uncheck the custom menu items by using [CustomMenuItem.IsChecked](#) property. You can disable the checkable functionalities of the custom context menu item by using the

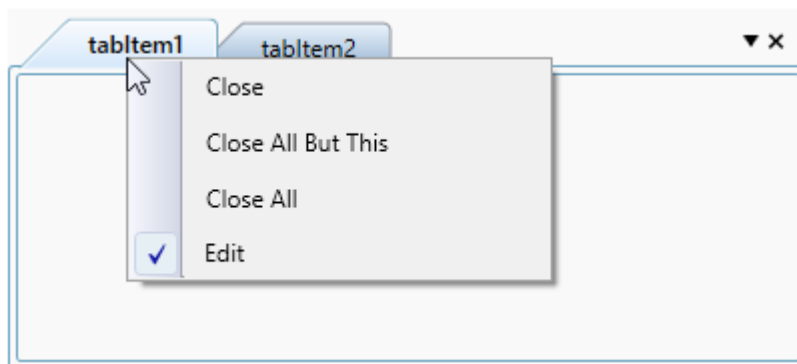
CustomMenuItem.IsCheckedable property value as false. The default value of CustomMenuItem.IsCheckedable property is true and CustomMenuItem.IsChecked property value is false.

XML

```
<syncfusion:TabControlExt IsCustomTabItemContextMenuEnabled="True"
Name="tabControlExt" >
<syncfusion:TabItemExt Header="tabItem1">
<!--Adding custom context menu items-->
<syncfusion:TabItemExt.ContextMenuItems>
<syncfusion:CustomMenuItem Header="Edit"
IsChecked="True"
IsCheckable="True" />
</syncfusion:TabItemExt.ContextMenuItems>
</syncfusion:TabItemExt>
</syncfusion:TabControlExt>
```

C#

```
// Enable custom tabitem context menu
tabControlExt.IsCustomTabItemContextMenuEnabled = true;
// Adding custom context menu for the tabitem
CustomMenuItem customMenuItem = new CustomMenuItem();
customMenuItem.Header = "Edit";
customMenuItem.IsCheckedable = true;
customMenuItem.IsChecked = true;
tabItemExt1.ContextMenuItems.Add(customMenuItem);
```



Note: View [Sample](#) in GitHub

Adding sub menu items for custom context menu item

You can add sub menu items for custom context menu item with any level by adding that sub CustomMenuItem to the parent CustomMenuItem.

XML

```
<syncfusion:TabControlExt IsCustomTabItemContextMenuEnabled="True"
Name="tabControlExt" >
<syncfusion:TabItemExt Name="tabItem1" Header="tabItem1">
<syncfusion:TabItemExt.ContextMenuItems>
<syncfusion:CustomMenuItem Header="Edit">
<!--Adding sub custom context menu items-->
```

```

<syncfusion:CustomMenuItem Header="SubItem0"/>
<syncfusion:CustomMenuItem Header="SubItem1"/>
<syncfusion:CustomMenuItem Header="SubItem2">
<syncfusion:CustomMenuItem Header="Level 2"/>
</syncfusion:CustomMenuItem>
</syncfusion:CustomMenuItem>
</syncfusion:TabItemExt.ContextMenuItems>
</syncfusion:TabItemExt>
<syncfusion:TabItemExt Name="tabItem2" Header="tabItem2">
<syncfusion:TabItemExt.ContextMenuItems>
<syncfusion:CustomMenuItem Header="Edit">
<!--Adding sub custom context menu items-->
<syncfusion:CustomMenuItem Header="SubItem0"/>
<syncfusion:CustomMenuItem Header="SubItem1"/>
<syncfusion:CustomMenuItem Header="SubItem2">
<syncfusion:CustomMenuItem Header="Level 2"/>
</syncfusion:CustomMenuItem>
</syncfusion:CustomMenuItem>
</syncfusion:TabItemExt.ContextMenuItems>
</syncfusion:TabItemExt>
</syncfusion:TabControlExt>

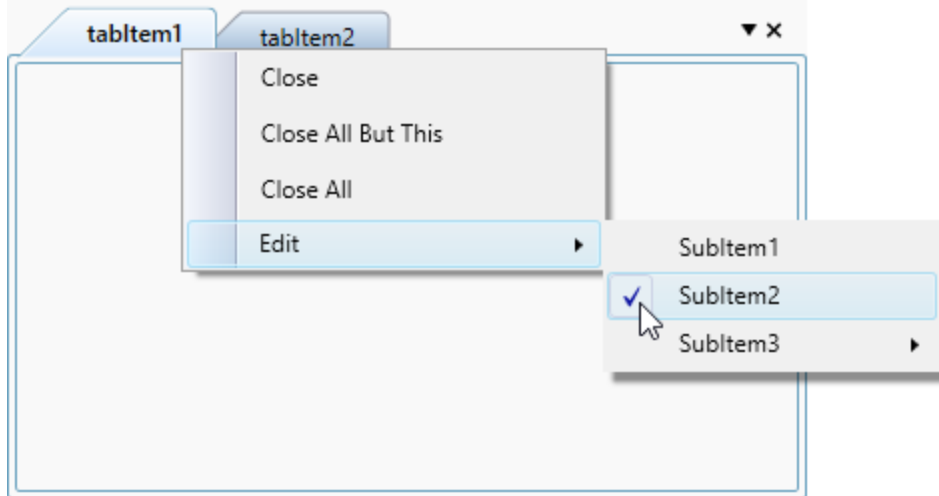
```

C#

```

// Enable custom tabitem context menu
tabControlExt.IsCustomTabItemContextMenuEnabled = true;
CustomMenuItem editMenuItem = new CustomMenuItem() { Header = "Edit" };
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "SubItem1"
};
CustomMenuItem customMenuItem2 = new CustomMenuItem() { Header = "SubItem2"
};
CustomMenuItem customMenuItem3 = new CustomMenuItem() { Header = "SubItem3"
};
//Adding sub menu items for 'SubItem3' custom menu item
CustomMenuItem level2_customMenuItem = new CustomMenuItem() { Header =
"Level 2" };
customMenuItem3.Items.Add(level2_customMenuItem);
//Adding sub menu items
editMenuItem.Items.Add(customMenuItem1);
editMenuItem.Items.Add(customMenuItem2);
editMenuItem.Items.Add(customMenuItem3);
tabItem1.ContextMenuItems.Add(editMenuItem);
tabItem2.ContextMenuItems.Add(editMenuItem);

```



Note: View [Sample](#) in GitHub

Hide default context menu items

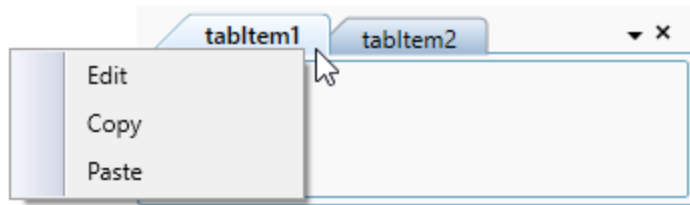
If you want to show only custom context menu items in the default context menu, then you can hide the default context menu item using [TabControlExt.DefaultContextMenuItemsVisibility](#) property value as false. The default value of [TabControlExt.DefaultContextMenuItemsVisibility](#) property is true.

XML

```
<syncfusion:TabControlExt IsCustomTabItemContextMenuEnabled="True"
DefaultContextMenuItemsVisibility="Collapsed"
Name="tabControlExt" >
  <syncfusion:TabItemExt Header="tabItemExt1">
    <syncfusion:TabItemExt.ContextMenuItems>
      <syncfusion:CustomMenuItem Header="Edit" />
      <syncfusion:CustomMenuItem Header="Copy" />
      <syncfusion:CustomMenuItem Header="Paste" />
    </syncfusion:TabItemExt.ContextMenuItems>
  </syncfusion:TabItemExt>
</syncfusion:TabControlExt>
```

C#

```
//Collapse the default contextmenu visibility
tabControlExt.DefaultContextMenuItemsVisibility = Visibility.Collapsed;
//Adding custom context menu for the first tabitem
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "Edit"};
CustomMenuItem customMenuItem2 = new CustomMenuItem() { Header = "Copy"};
CustomMenuItem customMenuItem3 = new CustomMenuItem() { Header = "Paste"};
tabItemExt1.ContextMenuItems.Add(customMenuItem1);
tabItemExt1.ContextMenuItems.Add(customMenuItem2);
tabItemExt1.ContextMenuItems.Add(customMenuItem3);
```



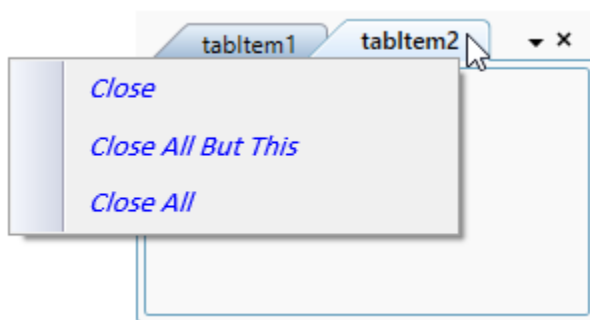
Note: View [Sample](#) in GitHub

Custom template for default tab item context menu

You can customize the default tab item context menu appearance for the each tab items by using the [TabControlExt.TabItemContextMenuItemTemplate](#) property.

XML

```
<Window.Resources>
  <!--Custom data template for the TabItem ContextMenu-->
  <DataTemplate x:Key="TabItemContextMenuItemTemplate"
    DataType="syncfusion:TabItemExt">
    <TextBlock FontFamily="Calibri"
      Foreground="Blue"
      FontStyle="Oblique"
      Text="{Binding}" />
  </DataTemplate>
</Window.Resources>
<syncfusion:TabControlExt ShowTabItemContextMenu="True"
  Name="tabControlExt">
  <syncfusion:TabItemExt TabItemContextMenuItemTemplate=
    "{StaticResource TabItemContextMenuItemTemplate}"
    Header="tabItem1" />
  <syncfusion:TabItemExt TabItemContextMenuItemTemplate=
    "{StaticResource TabItemContextMenuItemTemplate}"
    Header="tabItem2" />
</syncfusion:TabControlExt>
```



Note: View [Sample](#) in GitHub

Tab list context menu for switching tabs

You can easily navigate to any tab item by using the tab list menu which is placed in the top-right corner of the tab header panel. The header of all visible tab item's are shown as a menu item in the tab list menu. You can hide this tab list menu by using the [ShowTabListContextMenu](#) property value as `false`. The default value of `ShowTabListContextMenu` property is `true`.

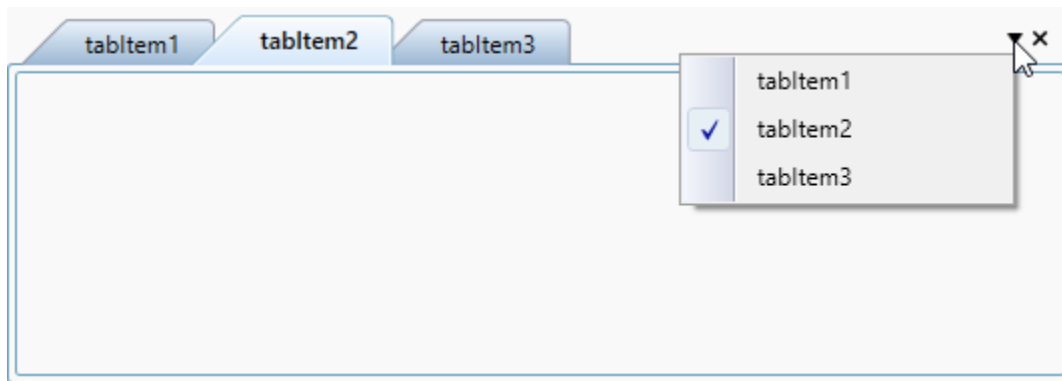
Note: The currently selected tab item has a tick mark on the TabListContextMenu.

XML

```
<syncfusion:TabControlExt ShowTabListContextMenu="True"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.ShowTabListContextMenu = true;
```



Note: View [Sample](#) in GitHub

Show hidden tab items in tab list context menu

By default, all the tab items except hidden items are listed in the tab list context menu. If you want to show the hidden tab items into tab list context menu to navigate, use the [TabListContextMenuOptions](#) property value as `Default, ShowHiddenItems`.

You can set the multiple options for the `TabListContextMenuOptions` to show different types of tab items into the context menu. The default value of `TabListContextMenuOptions` property is `Default`.

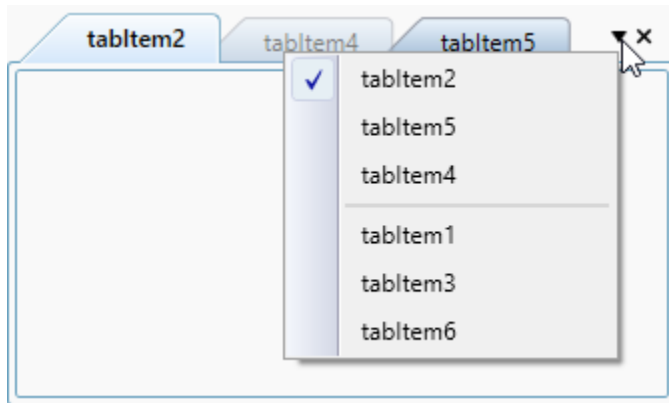
XML

```
<syncfusion:TabControlExt TabListContextMenuOptions="Default,
ShowHiddenItems"
ShowTabListContextMenu="True"
Name="tabControlExt">
<syncfusion:TabItemExt Header="tabItem1" Visibility="Collapsed"/>
<syncfusion:TabItemExt Header="tabItem2"/>
<syncfusion:TabItemExt Header="tabItem3" Visibility="Collapsed"/>
<syncfusion:TabItemExt Header="tabItem4" IsEnabled="False"/>
<syncfusion:TabItemExt Header="tabItem5"/>
<syncfusion:TabItemExt Header="tabItem6" Visibility="Collapsed"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.ShowTabListContextMenu = true;
```

```
tabControlExt.TabListContextMenuOptions = TabListContextMenuOptions.Default
|
TabListContextMenuOptions.ShowHiddenItems;
```



Show specific type tab items in tab list context menu

If you want to show the specific type tab items in tab list context menu, set the single option to the `TabListContextMenuOptions` property. If you want to show only the hidden tab items into tab list context menu to navigate, use the `TabListContextMenuOptions` property value as `ShowHiddenItems`. If you want to show only the enabled tab items into tab list context menu to navigate, use the `TabListContextMenuOptions` property value as `ShowEnabledItems`. If you want to show only the disabled tab items into tab list context menu to navigate, use the `TabListContextMenuOptions` property value as `ShowDisabledItems`.

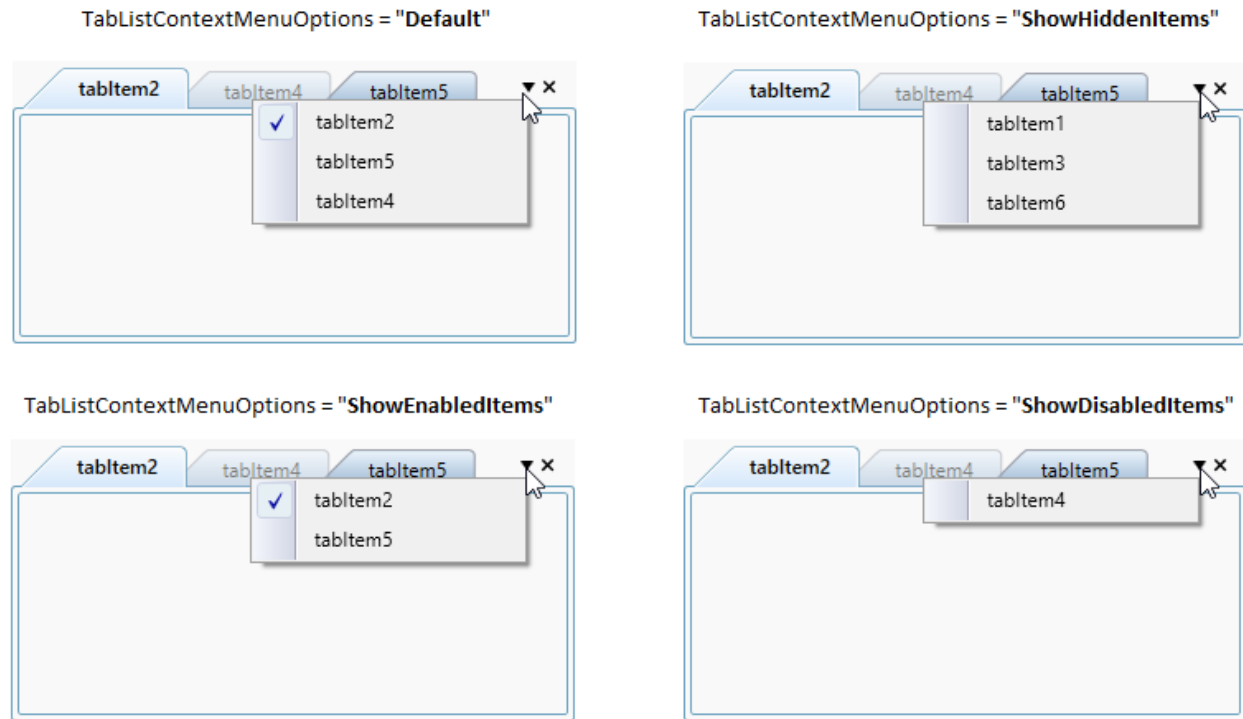
XML

```
<syncfusion:TabControlExt TabListContextMenuOptions="ShowEnabledItems"
ShowTabListContextMenu="True"
Name="tabControlExt">
  <syncfusion:TabItemExt Header="tabItem1" Visibility="Collapsed"/>
  <syncfusion:TabItemExt Header="tabItem2"/>
  <syncfusion:TabItemExt Header="tabItem3" Visibility="Collapsed"/>
  <syncfusion:TabItemExt Header="tabItem4" IsEnabled="False"/>
  <syncfusion:TabItemExt Header="tabItem5"/>
  <syncfusion:TabItemExt Header="tabItem6" Visibility="Collapsed"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.ShowTabListContextMenu = true;
tabControlExt.TabListContextMenuOptions =
TabListContextMenuOptions.ShowEnabledItems;
```

Single options



Note: View [Sample](#) in GitHub

Show multi-type tab items in tab list context menu

If you want to show the multiple type of tab items like enabled, disabled or hidden tab items together in tab list context menu, set the multiple options for the `TabListContextMenuOptions` property.

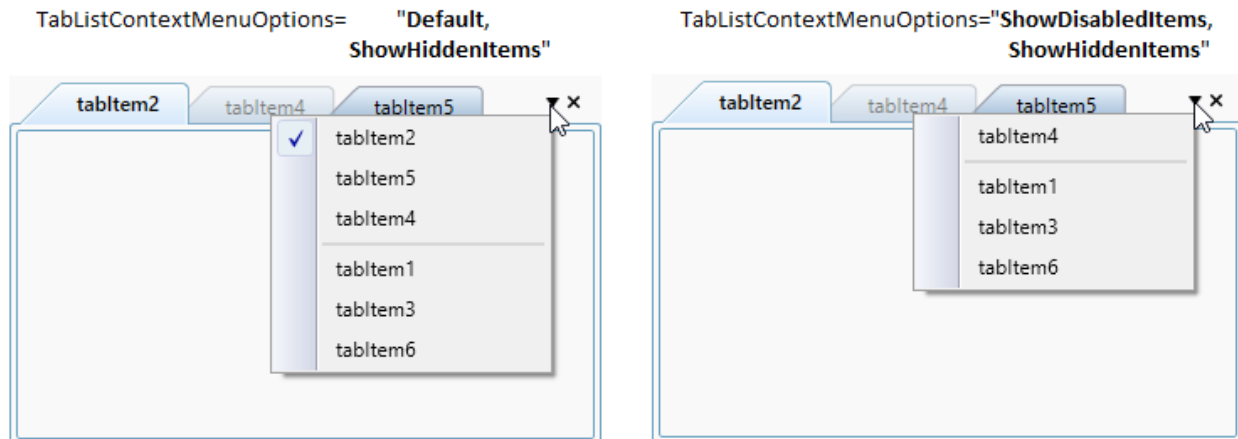
XML

```
<syncfusion:TabControlExt
  TabListContextMenuOptions="ShowDisabledItems, ShowHiddenItems"
  ShowTabListContextMenu="True"
  Name="tabControlExt">
  <syncfusion:TabItemExt Header="tabItem1" Visibility="Collapsed"/>
  <syncfusion:TabItemExt Header="tabItem2"/>
  <syncfusion:TabItemExt Header="tabItem3" Visibility="Collapsed"/>
  <syncfusion:TabItemExt Header="tabItem4" IsEnabled="False"/>
  <syncfusion:TabItemExt Header="tabItem5"/>
  <syncfusion:TabItemExt Header="tabItem6" Visibility="Collapsed"/>
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.ShowTabListContextMenu = true;
tabControlExt.TabListContextMenuOptions =
  TabListContextMenuOptions.ShowDisabledItems |
  TabListContextMenuOptions.ShowHiddenItems
```


Multiple options



Custom tab list context menu item

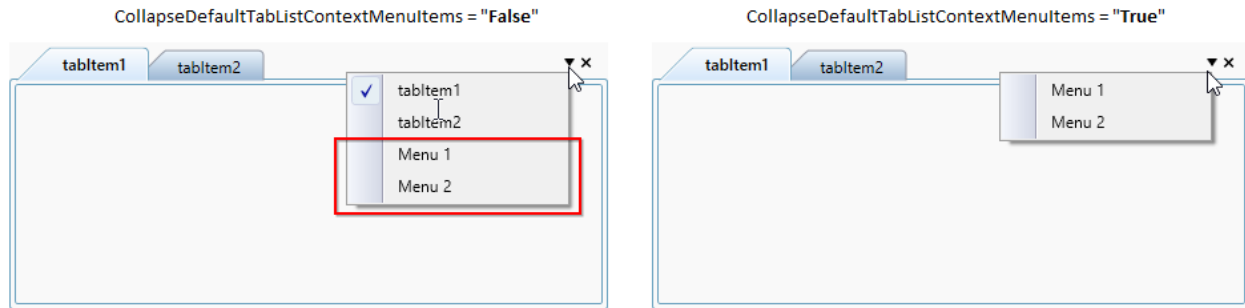
You can add the custom tab list context menu items by using the [TabControlExt.TabListContextMenuItems](#) property. You can show only the custom tab list context menu items by setting the [CollapseDefaultTabListContextMenuItems](#) property to `true`. The default value of [CollapseDefaultTabListContextMenuItems](#) property is `false`.

XML

```
<syncfusion:TabControlExt CollapseDefaultTabListContextMenuItems="True"
Name="tabControlExt" >
  <syncfusion:TabControlExt.TabListContextMenuItems>
    <!--Adding custom context menu items-->
    <syncfusion:CustomMenuItem Header="Menu 1" />
    <syncfusion:CustomMenuItem Header="Menu 2" />
  </syncfusion:TabControlExt.TabListContextMenuItems>
  <syncfusion:TabItemExt Header="tabItem1"/>
  <syncfusion:TabItemExt Header="tabItem2"/>
</syncfusion:TabControlExt>
```

C#

```
// Display only the custom tab list context menu items
tabControlExt.CollapseDefaultTabListContextMenuItems = true;
// Adding custom tab list context menu items
CustomMenuItem customMenuItem1 = new CustomMenuItem();
customMenuItem1.Header = "Menu 1";
CustomMenuItem customMenuItem2 = new CustomMenuItem();
customMenuItem2.Header = "Menu 2";
tabControlExt.TabListContextMenuItems.Add(customMenuItem1);
tabControlExt.TabListContextMenuItems.Add(customMenuItem2);
```



Note: [View Sample in GitHub](#)

Check or uncheck the custom tab list menu items

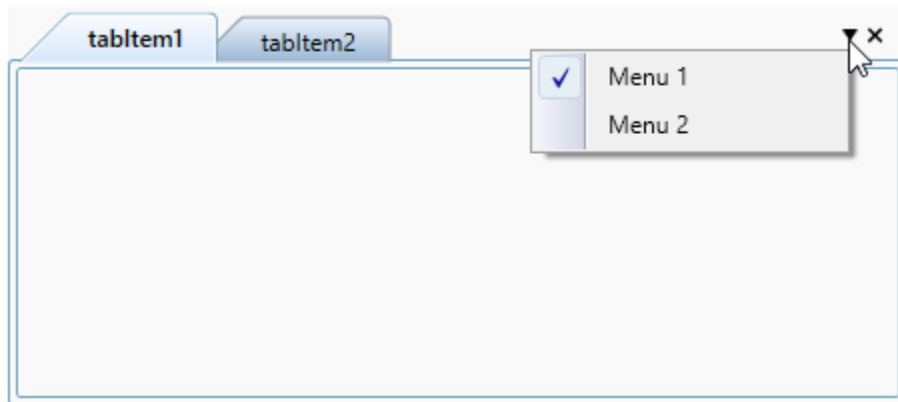
You can check or uncheck the custom tab list menu items by using `CustomMenuItem.IsChecked` property. You can disable the checkable functionalities of the tab list menu items by using the `CustomMenuItem.IsCheckable` property value as `false`. The default value of `CustomMenuItem.IsCheckable` property is `true` and `CustomMenuItem.IsChecked` property value is `false`.

XML

```
<syncfusion:TabControlExt CollapseDefaultTabListContextMenuItems="True"
Name="tabControlExt" >
  <!--Adding custom tab list context menu items-->
  <syncfusion:TabControlExt.TabListContextMenuItems>
    <!--Adding custom context menu items-->
    <syncfusion:CustomMenuItem Header="Menu 1"
IsCheckable="True"
IsChecked="True"/>
    <syncfusion:CustomMenuItem Header="Menu 2"
IsCheckable="False"/>
  </syncfusion:TabControlExt.TabListContextMenuItems>
  <syncfusion:TabItemExt Header="tabItem1"/>
  <syncfusion:TabItemExt Header="tabItem2"/>
</syncfusion:TabControlExt>
```

C#

```
// Display only the custom tab list context menu
tabControlExt.CollapseDefaultTabListContextMenuItems = true;
// Adding custom tab list context menu items
CustomMenuItem customMenuItem1 = new CustomMenuItem();
customMenuItem1.Header = "Menu 1";
customMenuItem1.IsCheckable = true;
customMenuItem1.IsChecked = true;
CustomMenuItem customMenuItem2 = new CustomMenuItem();
customMenuItem2.Header = "Menu 2";
customMenuItem2.IsCheckable = false;
tabControlExt.TabListContextMenuItems.Add(customMenuItem1);
tabControlExt.TabListContextMenuItems.Add(customMenuItem2);
```



Note: [View Sample in GitHub](#)

Adding sub menu items for custom context menu item

You can add sub menu items for custom context menu item with any level by adding that sub `CustomMenuItem` to the parent `CustomMenuItem`.

XML

```
<syncfusion:TabControlExt CollapseDefaultTabListContextMenuItems="True"
Name="tabControlExt" >
  <!--Adding custom tab list context menu items-->
  <syncfusion:TabControlExt.TabListContextMenuItems>
    <!--Adding custom context menu items-->
    <syncfusion:CustomMenuItem Header="Menu 1">
      <!--Adding sub custom context menu items-->
      <syncfusion:CustomMenuItem Header="SubMenu 1"/>
      <syncfusion:CustomMenuItem Header="SubMenu 2">
        <!--Adding sub custom context menu items for 'SubMenu 2'-->
        <syncfusion:CustomMenuItem Header="Level 2"/>
      </syncfusion:CustomMenuItem>
      <syncfusion:CustomMenuItem Header="SubMenu 3"/>
    </syncfusion:CustomMenuItem>
    <syncfusion:CustomMenuItem Header="Menu 2">
      <!--Adding sub custom context menu items-->
      <syncfusion:CustomMenuItem Header="SubMenu 1"/>
      <syncfusion:CustomMenuItem Header="SubMenu 2"/>
      <syncfusion:CustomMenuItem Header="SubMenu 3">
        <!--Adding sub custom context menu items for 'SubMenu 3'-->
        <syncfusion:CustomMenuItem Header="Level 2"/>
      </syncfusion:CustomMenuItem>
    </syncfusion:CustomMenuItem>
  </syncfusion:TabControlExt.TabListContextMenuItems>
  <syncfusion:TabItemExt Header="tabItem1"/>
  <syncfusion:TabItemExt Header="tabItem2"/>
</syncfusion:TabControlExt>
```

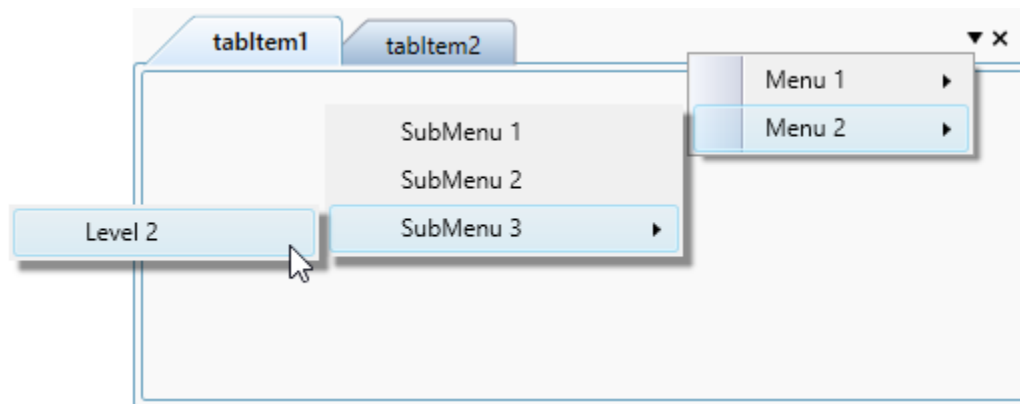
C#

```
// Display only the custom tab list context menu
tabControlExt.CollapseDefaultTabListContextMenuItems = true;
// Adding custom tab list context menu items
CustomMenuItem menu1 = new CustomMenuItem();
menu1.Header = "Menu 1";
```

```

CustomMenuItem menu2 = new CustomMenuItem();
menu2.Header = "Menu 2";
// Creating custom sub menu tab list context menu items
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "SubMenu 1" };
};
CustomMenuItem customMenuItem2 = new CustomMenuItem() { Header = "SubMenu 2" };
};
CustomMenuItem customMenuItem3 = new CustomMenuItem() { Header = "SubMenu 3" };
};
//Adding sub menu items for 'SubMenu 2' and 'SubMenu 3' custom tablist menu item
CustomMenuItem level2_customMenuItem = new CustomMenuItem() { Header = "Level 2" };
customMenuItem3.Items.Add(level2_customMenuItem);
menu2.Items.Add(customMenuItem1);
menu2.Items.Add(customMenuItem2);
menu2.Items.Add(customMenuItem3);
tabControlExt.TabListContextMenuItems.Add(menu1);
tabControlExt.TabListContextMenuItems.Add(menu2);

```



Note: [View Sample in GitHub](#)

CustomMenuItem as Separator

You can display the separator between custom menu items by using [CustomMenuItem.IsSeparator](#) property. If `IsSeparator` property is set to true, the `CustomMenuItem` will act as separator. The default value of `CustomMenuItem.IsSeparator` property is false.

XML

```

<syncfusion:TabControlExt CollapseDefaultTabListContextMenuItems="True"
Name="tabControlExt" >
  <!--Adding custom tab list context menu items-->
  <syncfusion:TabListContextMenuItems>
    <!--Adding custom context menu items-->
    <syncfusion:CustomMenuItem Header="Menu 1">
      <!--Adding sub custom context menu items-->
      <syncfusion:CustomMenuItem Header="SubMenu 1"/>
      <syncfusion:CustomMenuItem Header="SubMenu 2">
        <!--Adding sub custom context menu items for 'SubMenu 2'-->
        <syncfusion:CustomMenuItem Header="Level 2"/>
      </syncfusion:CustomMenuItem>
      <syncfusion:CustomMenuItem Header="SubMenu 3"/>
    </syncfusion:CustomMenuItem>
  </syncfusion:TabListContextMenuItems>

```

```

</syncfusion:CustomMenuItem>
<!--Adding custom context menu item as Seperator-->
<syncfusion:CustomMenuItem IsSeparator="True"
Height="1"/>
<syncfusion:CustomMenuItem Header="Menu 2">
<!--Adding sub custom context menu items-->
<syncfusion:CustomMenuItem Header="SubMenu 1"/>
<syncfusion:CustomMenuItem Header="SubMenu 2"/>
<syncfusion:CustomMenuItem Header="SubMenu 3">
<!--Adding sub custom context menu items for 'SubMenu 3'-->
<syncfusion:CustomMenuItem Header="Level 2"/>
</syncfusion:CustomMenuItem>
</syncfusion:CustomMenuItem>
</syncfusion:TabControlExt.TabListContextMenuItems>
<syncfusion:TabItemExt Header="tabItem1"/>
<syncfusion:TabItemExt Header="tabItem2"/>
</syncfusion:TabControlExt>

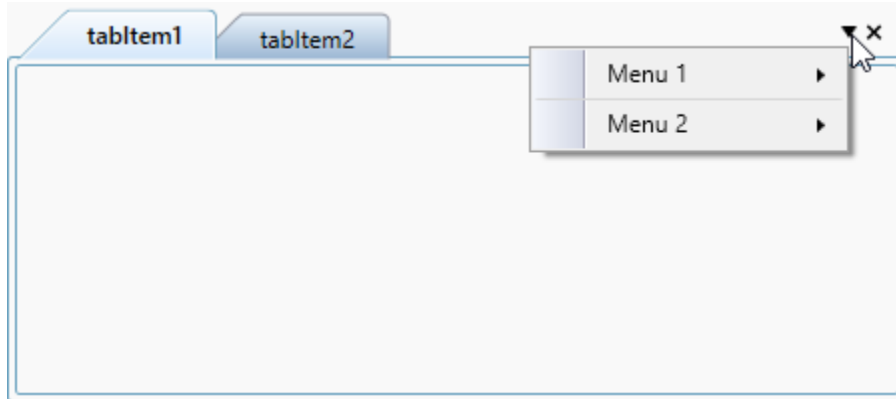
```

C#

```

// Disaply only the custom tab list context menu
tabControlExt.CollapseDefaultTabListContextMenuItems = true;
// Adding custom tab list context menu items
CustomMenuItem menu1 = new CustomMenuItem();
menu1.Header = "Menu 1";
CustomMenuItem menu2 = new CustomMenuItem();
menu2.Header = "Menu 2";
// Adding custom tab list context menu item as Seperator
CustomMenuItem separator = new CustomMenuItem();
separator.Seperator = true;
separator.Height = 1;
// Creating custom sub menu tab list context menu items
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "SubMenu 1"
};
CustomMenuItem customMenuItem2 = new CustomMenuItem() { Header = "SubMenu 2"
};
CustomMenuItem customMenuItem3 = new CustomMenuItem() { Header = "SubMenu 3"
};
//Adding sub menu items for 'SubMenu 2' and 'SubMenu 3' custom tablist menu
item
CustomMenuItem level2_customMenuItem = new CustomMenuItem() { Header =
"Level 2" };
customMenuItem3.Items.Add(level2_customMenuItem);
menu2.Items.Add(customMenuItem1);
menu2.Items.Add(customMenuItem2);
menu2.Items.Add(customMenuItem3);
tabControlExt.TabListContextMenuItems.Add(menu1);
tabControlExt.TabListContextMenuItems.Add(separator);
tabControlExt.TabListContextMenuItems.Add(menu2);

```



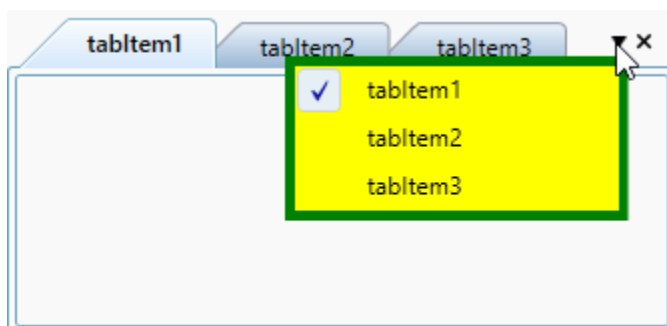
Note: [View Sample in GitHub](#)

Custom template for the tab list context menu

You can customize the tab list menu appearance by using the [TabListContextMenuTemplate](#) property.

XML

```
<syncfusion:TabControlExt ShowTabListContextMenu="True"
Name="tabControlExt" >
<syncfusion:TabControlExt.TabListContextMenuTemplate>
<ControlTemplate>
<Border Background="Green">
<Border Background="Yellow"
Margin="5">
<Grid>
<ItemsPresenter/>
</Grid>
</Border>
</Border>
</ControlTemplate>
</syncfusion:TabControlExt.TabListContextMenuTemplate>
<syncfusion:TabItemExt Header="tabItem1"/>
<syncfusion:TabItemExt Header="tabItem2"/>
<syncfusion:TabItemExt Header="tabItem3"/>
</syncfusion:TabControlExt>
```

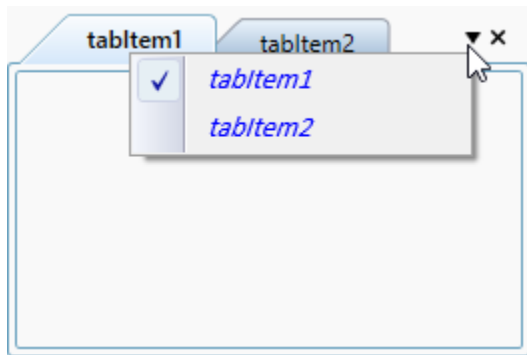


Custom template for the tab list menu item

You can customize the tab list menu items appearance by using the [TabListContextMenuTemplate](#) property.

XML

```
<syncfusion:TabControlExt ShowTabListContextMenu="True"
Name="tabControlExt" >
<syncfusion:TabControlExt.TabListContextMenuItemsTemplate>
<DataTemplate>
<TextBlock FontFamily="Calibri"
Foreground="Blue"
FontStyle="Oblique"
Text="{Binding}" />
</DataTemplate>
</syncfusion:TabControlExt.TabListContextMenuItemsTemplate>
<syncfusion:TabItemExt Header="tabItem1" />
<syncfusion:TabItemExt Header="tabItem2" />
</syncfusion:TabControlExt>
```



Note: View [Sample](#) in GitHub

Appearance in WPF TabControl (TabControlExt)

This section explains different UI customization and theming options available in [TabControl](#).

Change flow direction

You can change the flow direction of the **TabControl** layout from right to left by setting the **FlowDirection** property value as **RightToLeft**. The default value of **FlowDirection** property is **LeftToRight**.

XML

```
<syncfusion:TabControlExt FlowDirection="RightToLeft"
Name="tabControlExt">
<syncfusion:TabItemExt Content="This is the first tab item"
Header="tabItem1" />
</syncfusion:TabControlExt>
```

C#

```
tabControlExt.FlowDirection = FlowDirection.RightToLeft;
```

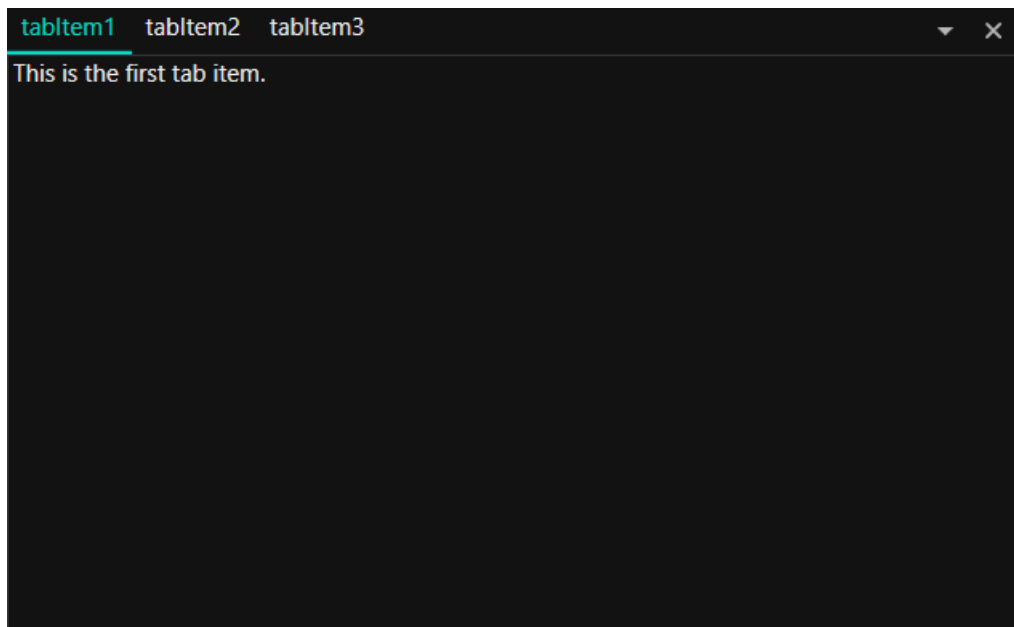


Note: View [Sample](#) in GitHub

Theme

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

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

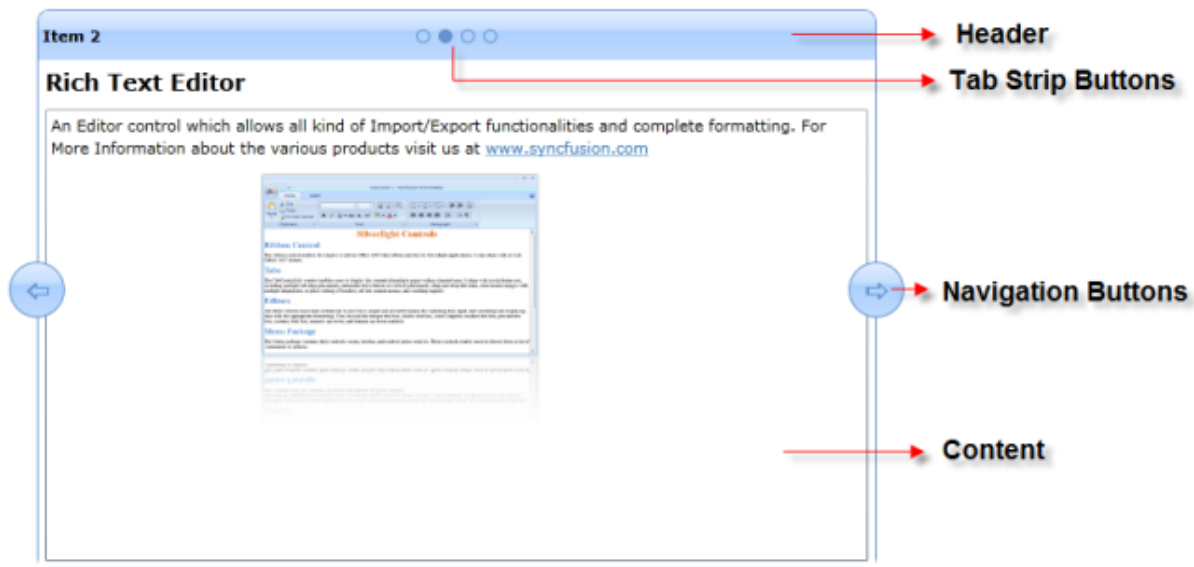


TabNavigation

WPF Tab Navigation Overview

Tab Navigation is a new control for displaying the contents of the control with transition effects. Tab navigation control facilitates the display of pages/items (with transition effects) like an Ad rotator in web applications, thereby enhancing the UI's look and feel. It supports only .NET Framework version 4.0.

Appearance and structure of the control



The following are the key features of this control:

Items source binding

It helps the control to be applicable to all MVVM based Silverlight applications. It supports the following:

- Observable Collection—any modification done to the collection items will get reflected on the UI of the control. Such a collection is bound to the control.
- IList binding—this list is based on all collections such as List, Queue, Stack and so on, that are bound to the control.
- XML binding—the XML data are bound to the control.

Item templates

- Data Templates—this is used to display data visually on the UI in a predefined format.
- Template Selectors—this allows you to choose templates for data items that will be displayed on the UI

Transition effects

This enhances navigation by providing different navigation or transition effects. The following effects are supported:

- Slide

- Fade
- Zoom
- Blur
- Push
- PushIn
- Wipe

Use case scenario

Tab navigation control helps you to use information as banners in web applications such as Picasa. Therefore, it can be used as website banners. As it has virtualization support, it can also be used as tab control.

Getting Started with WPF Tab Navigation

Assembly deployment

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

Creating simple application with TabNavigation

You can create the WPF application with TabNavigation control as follows:

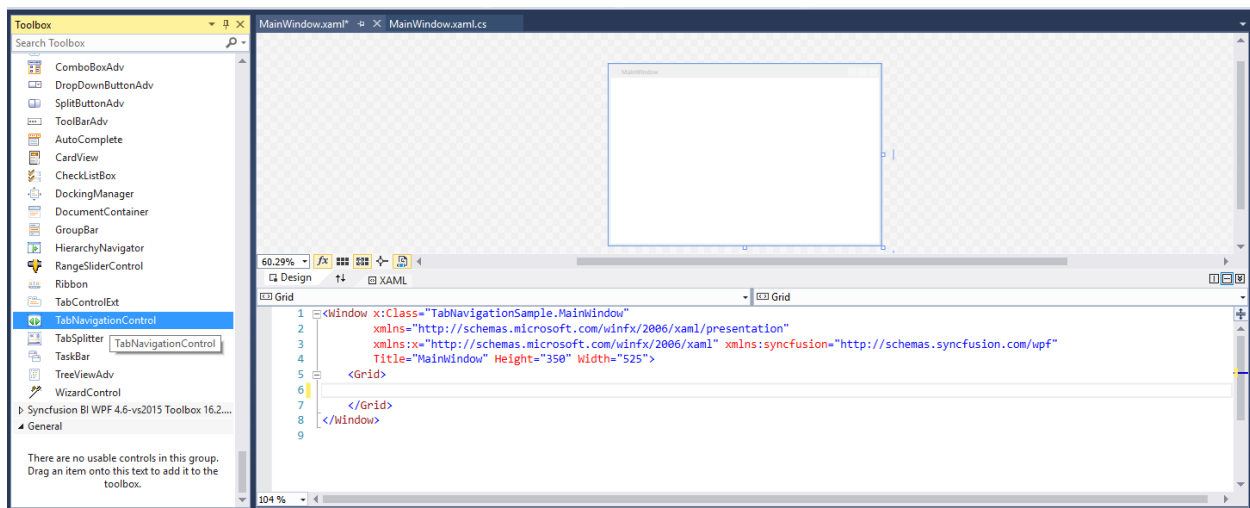
1. [Creating project](#)
2. [Adding control via Designer](#)
3. [Adding control manually in XAML](#)
4. [Adding control manually in C#](#)

Creating the project

Create a new WPF project in the Visual Studio to display the TabNavigation with functionalities.

Adding control via Designer

The TabNavigation control can be added to the application by dragging it from the toolbox and dropping it in a 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.Tools.WPF
 - Syncfusion.Shared.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page.
3. Declare TabNavigation control in XAML page.

XML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="WpfApplication1.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid>
<!-- TabNavigation Control -->
<syncfusion:TabNavigationControl x:Name="tabNavigation"/>
</Grid>
</Window>
```

Adding control manually in C#

In order to add control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.Tools.WPF
 - Syncfusion.Shared.WPF
2. Import TabNavigationControl namespace **using Syncfusion.Windows.Tools.Controls;**
3. Create TabNavigationControl instance and add it to the window.

C#

```
using Syncfusion.Windows.Tools.Controls;
namespace TabNavigationSample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //For adding TabNavigation Control
            TabNavigationControl tabNavigation = new TabNavigationControl();
            grid.Children.Add(tabNavigation);
        }
    }
}
```

Adding Items using TabNavigationItem

You can populate the TabNavigation control by adding objects directly to the [Items](#) collection. Items added to the TabNavigation are wrapped in [TabNavigationItem](#) containers.

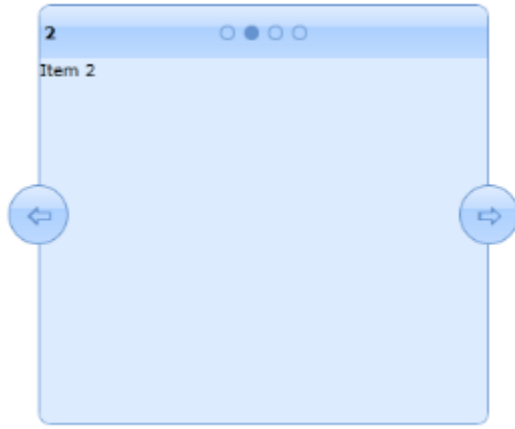
XML

```
<!-- TabNavigationControl -->
<syncfusion:TabNavigationControl x:Name="tabNavigation">
  <!-- TabNavigationItem 1 -->
  <syncfusion:TabNavigationItem Header="1">
    <syncfusion:TabNavigationItem.Content>
      <Grid>
        <TextBlock Text="Item 1"/>
      </Grid>
    </syncfusion:TabNavigationItem.Content>
  </syncfusion:TabNavigationItem>
  <!-- TabNavigationItem 2 -->
  <syncfusion:TabNavigationItem Header="2">
    <syncfusion:TabNavigationItem.Content>
      <Grid>
        <TextBlock Text="Item 2"/>
      </Grid>
    </syncfusion:TabNavigationItem.Content>
  </syncfusion:TabNavigationItem>
  <!-- TabNavigationItem 3 -->
  <syncfusion:TabNavigationItem Header="3">
    <syncfusion:TabNavigationItem.Content>
      <Grid>
        <TextBlock Text="Item 3"/>
      </Grid>
    </syncfusion:TabNavigationItem.Content>
  </syncfusion:TabNavigationItem>
  <!-- TabNavigationItem 4 -->
  <syncfusion:TabNavigationItem Header="4">
    <syncfusion:TabNavigationItem.Content>
      <Grid>
        <TextBlock Text="Item 4"/>
      </Grid>
    </syncfusion:TabNavigationItem.Content>
  </syncfusion:TabNavigationItem>
</syncfusion:TabNavigationControl>
```

C#

```
TabNavigationControl tabNavigation = new TabNavigationControl();
//TabNavigationItem
TabNavigationItem item1 = new TabNavigationItem();
item1.Header = "1";
item1.Content = "Item 1";
TabNavigationItem item2 = new TabNavigationItem();
item2.Header = "2";
item2.Content = "Item 2";
TabNavigationItem item3 = new TabNavigationItem();
item3.Header = "3";
item3.Content = "Item 3";
TabNavigationItem item4 = new TabNavigationItem();
item4.Header = "4";
item4.Content = "Item 4";
//Adding items to TabNavigationControl
tabNavigation.Items.Add(item1);
```

```
tabNavigation.Items.Add(item2);  
tabNavigation.Items.Add(item3);  
tabNavigation.Items.Add(item4);
```



Binding ItemsSource

TabNavigationControl supports binding data to different data sources such as IList Data Source, XML Data Source, Observable Collection Data Source. Refer [Data binding](#) section for more details.

XML

```
<syncfusion:TabNavigationControl TransitionEffect="Slide"  
ItemsSource="{Binding MyCollection}"/>
```

C#

```
namespace TabNavigationSample  
{  
    /// <summary>  
    /// Interaction logic for MainWindow.xaml  
    /// </summary>  
    public partial class MainWindow : Window  
    {  
        TabNavigationItem temp;  
        public MainWindow()  
        {  
            InitializeComponent();  
            MyCollection = new ObservableCollection<TabNavigationItem>();  
            for (int i = 0; i < 10; i++)  
            {  
                temp = new TabNavigationItem();  
                temp.Header = i;  
                temp.Content= "Item " + i.ToString();  
                MyCollection.Add(temp);  
            }  
            this.DataContext = this;  
        }  
        public ObservableCollection<TabNavigationItem> MyCollection  
        {  

```

```

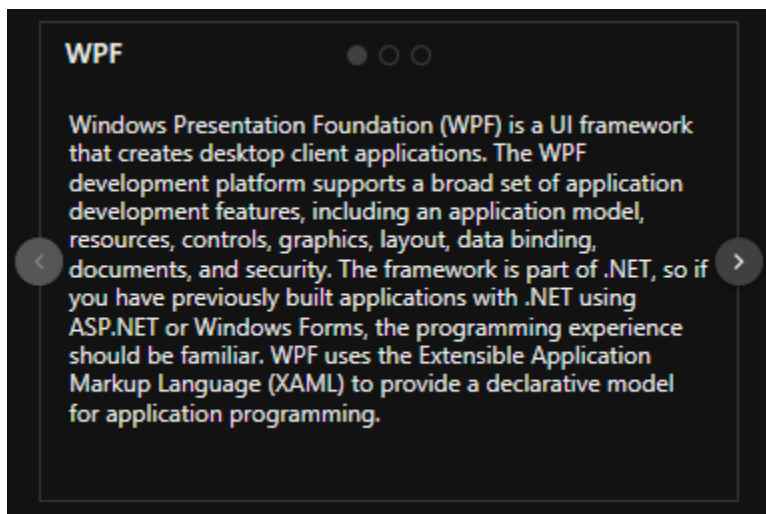
get { return
    (ObservableCollection<TabNavigationItem>)GetValue(MyCollectionProperty); }
set { SetValue(MyCollectionProperty, value); }
}
// Using a DependencyProperty as the backing store for MyCollection. This
enables animation, styling, binding and so on
public static readonly DependencyProperty MyCollectionProperty =
DependencyProperty.Register("MyCollection",
    typeof(ObservableCollection<TabNavigationItem>), typeof(MainWindow), new
PropertyMetadata(null));
}
}

```

Theme

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

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



Data binding in WPF Tab Navigation

Business object collections can be easily bound to the TabNavigation control using [ItemsSource](#) property.

Binding IEnumerable

XML

```

<syncfusion:TabNavigationControl TransitionEffect="Slide"
ItemsSource="{Binding MyCollection}"/>

```

C#

```

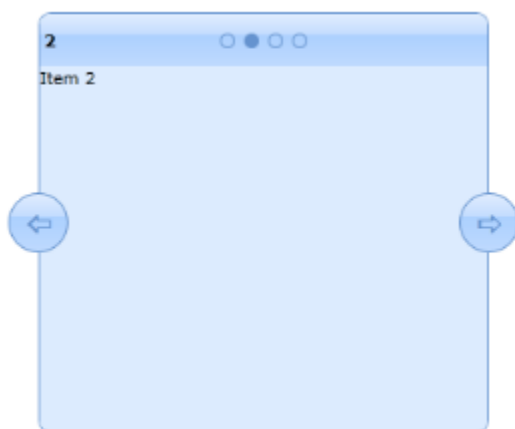
namespace TabNavigationSample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml

```

```

/// </summary>
public partial class MainWindow : Window
{
    TabNavigationItem temp;
    public MainWindow()
    {
        InitializeComponent();
        MyCollection = new ObservableCollection<TabNavigationItem>();
        for (int i = 0; i < 10; i++)
        {
            temp = new TabNavigationItem();
            temp.Header = i;
            temp.Content= "Item " + i.ToString();
            MyCollection.Add(temp);
        }
        this.DataContext = this;
    }
    public ObservableCollection<TabNavigationItem> MyCollection
    {
        get { return
            (ObservableCollection<TabNavigationItem>)GetValue(MyCollectionProperty); }
        set { SetValue(MyCollectionProperty, value); }
    }
    // Using a DependencyProperty as the backing store for MyCollection. This
    // enables animation, styling, binding and so on
    public static readonly DependencyProperty MyCollectionProperty =
        DependencyProperty.Register("MyCollection",
            typeof(ObservableCollection<TabNavigationItem>), typeof(MainWindow), new
            PropertyMetadata(null));
}

```



Binding data from XML

To bind XML data to a TabNavigation control, convert the XML data to a collection like ObservableCollection or IList collection, and then bind the collection by using the ItemsSource property of the TabNavigation control.

- **Create a xml file**

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Books>
<book>
<title>XML Developer's Guide</title>
<description>An indent look at creating applications with XML.</description>
</book>
<book>
<title>Midnight Rain</title>
<description>A former architect battles corporate zombies, an evil
sorceress, and her own childhood to become queen of the world.</description>
</book>
<book>
<title>Oberon's Legacy</title>
<description>In post apocalypse England, the mysterious agent known only as
Oberon helps to create a new life for the inhabitants of London. Sequel to
Mae Ascendant.</description>
</book>
<book>
<title>Lover Birds</title>
<description>When Carla meets Paul at an ornithology conference, tempers fly
as feathers get ruffled.</description>
</book>
<book>
<title>Split Splash</title>
<description>A deep sea diver finds true love twenty thousand leagues
beneath the sea.</description>
</book>
</Books>
```

- **Model.cs**

C#

```
namespace TabNavigationXMLBinding
{
    public class Model : NotificationObject
    {
    }
    public class BookModel : NotificationObject
    {
    }
    private string bookName;
    public string BookName
    {
        get
        {
            return bookName;
        }
    }
}
```



```
set
{
    bookName = value;
    this.RaisePropertyChanged("BookName");
}
}
private string description;
public string Description
{
    get
    {
        return description;
    }
    set
    {
        description = value;
        this.RaisePropertyChanged("Description");
    }
}
}
```

- ViewModel.cs

C#

```
namespace TabNavigationXMLBinding
{
    public class ViewModel : NotificationObject
    {
        private ObservableCollection<Model> modelItems;
        public ObservableCollection<Model> ModelItems
        {
            get
            {
                return modelItems;
            }
            set
            {
                modelItems = value;
            }
        }
        private ObservableCollection<BookModel> bookModelItems;
        public ObservableCollection<BookModel> BookModelItems
        {
            get
            {
                return bookModelItems;
            }
            set
            {
                bookModelItems = value;
            }
        }
        public ViewModel()
    }
}
```

```

{
    modelItems = new ObservableCollection<Model>();
    bookModelItems = new ObservableCollection<BookModel>();
    XDocument xDocument = XDocument.Load(@"assets\Books.xml");
    IEnumerable<XElement> query = from xElement in xDocument.Descendants("book")
    select xElement;
    foreach (XElement xElement in query)
    {
        BookModel bookModel = new BookModel
        {
            BookName = xElement.Element("title").Value,
            Description = xElement.Element("description").Value
        };
        bookModelItems.Add(bookModel);
    }
}
}
}

```

- **MainWindow.Xaml.cs**

C#

```

namespace TabNavigationXMLBinding
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : ChromelessWindow
    {
        public MainWindow()
        {
            InitializeComponent();
            TabNavigationItem tab;
            ViewModel view = new ViewModel();
            BookCollection = new ObservableCollection<TabNavigationItem>();
            for (int i = 0; i < view.BookModelItems.Count; i++)
            {
                tab = new TabNavigationItem();
                tab.Header = view.BookModelItems[i].BookName;
                tab.Content = view.BookModelItems[i].Description;
                BookCollection.Add(tab);
            }
            this.DataContext = this;
        }
        public ObservableCollection<TabNavigationItem> BookCollection
        {
            get { return
                (ObservableCollection<TabNavigationItem>)GetValue(BookCollectionProperty); }
            set
            {
                SetValue(BookCollectionProperty, value);
            }
        }
    }
}

```

```
// Using a DependencyProperty as the backing store for MyCollection. This
// enables animation, styling, binding and so on
public static readonly DependencyProperty BookCollectionProperty =
DependencyProperty.Register("BookCollection",
typeof(ObservableCollection<TabNavigationItem>), typeof(MainWindow), new
PropertyMetadata(null));
}
```

- **MainWindow.Xaml**

XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<!--TabNavigationControl-->
<syncfusion:TabNavigationControl x:Name="TabNavigation" Grid.Column="1"
Grid.Row="1" ItemsSource="{Binding BookCollection}"/>
```

Appearance in WPF Tab Navigation

Show/hide the Header

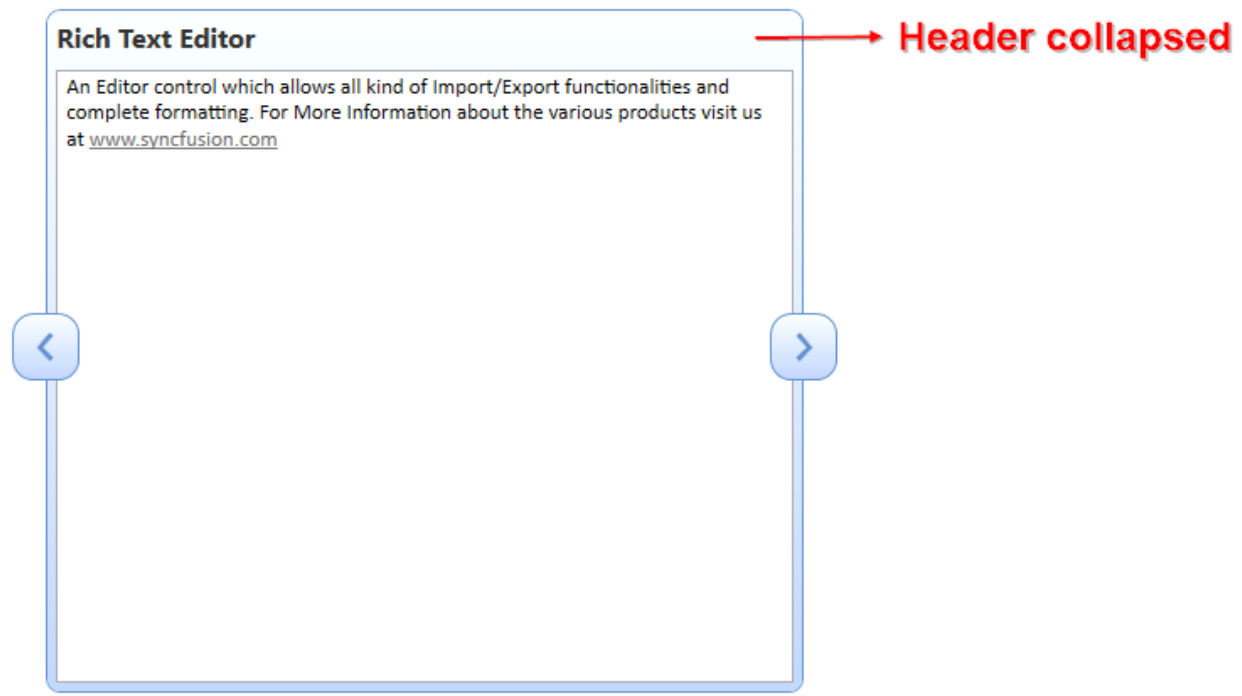
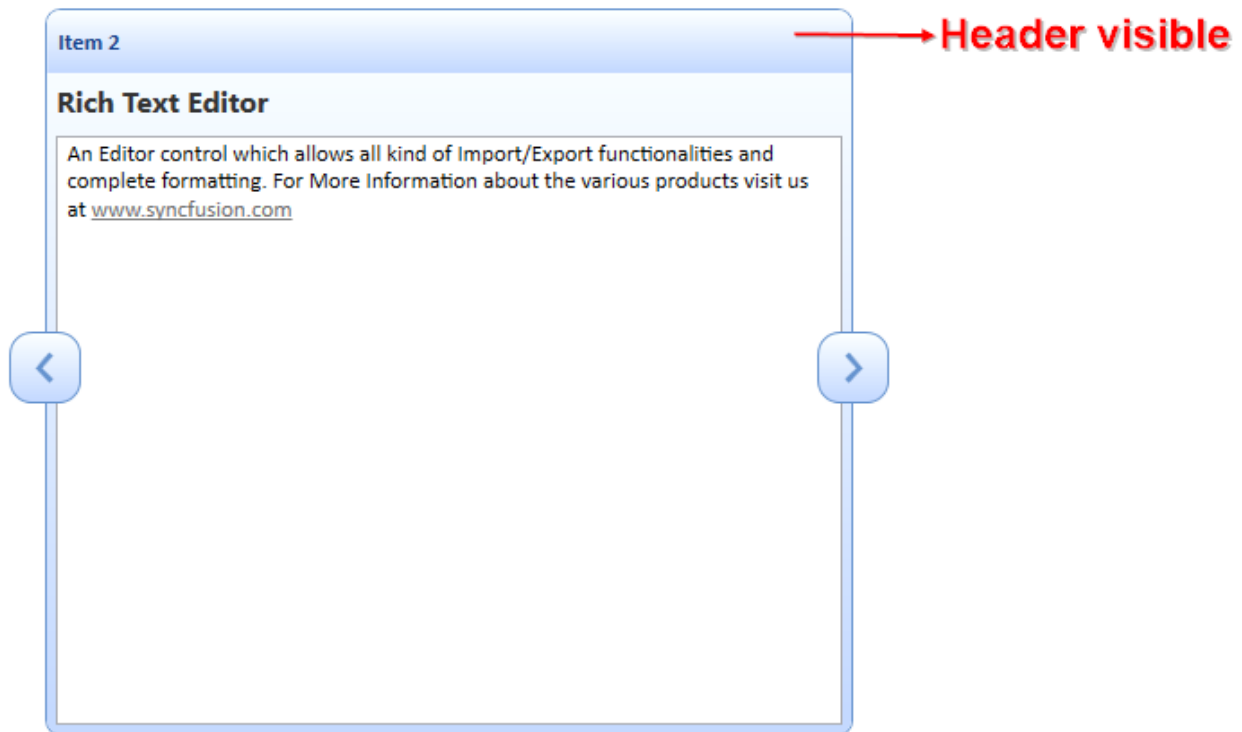
You can enable/disable the visibility of header by setting [HeaderVisibility](#) property of TabNavigationControl.

XML

```
<!-- TabNavigation Control -->
<syncfusion:TabNavigationControl x:Name="TabNavigation"
HeaderVisibility="Collapsed" >
<syncfusion:TabNavigationItem Header="TabItem1" Content="TabNavigationItem
1"/>
<syncfusion:TabNavigationItem Header="TabItem2" Content="TabNavigationItem
2"/>
<syncfusion:TabNavigationItem Header="TabItem3" Content="TabNavigationItem
3"/>
</syncfusion:TabNavigationControl>
```

C#

```
//Hide the header
tabNavigation.HeaderVisibility = Visibility.Collapsed;
```



Show/hide the NavigationButton

You can enable/disable the visibility of navigation button by setting [NavigationButtonVisibility](#) property of TabNavigationControl.

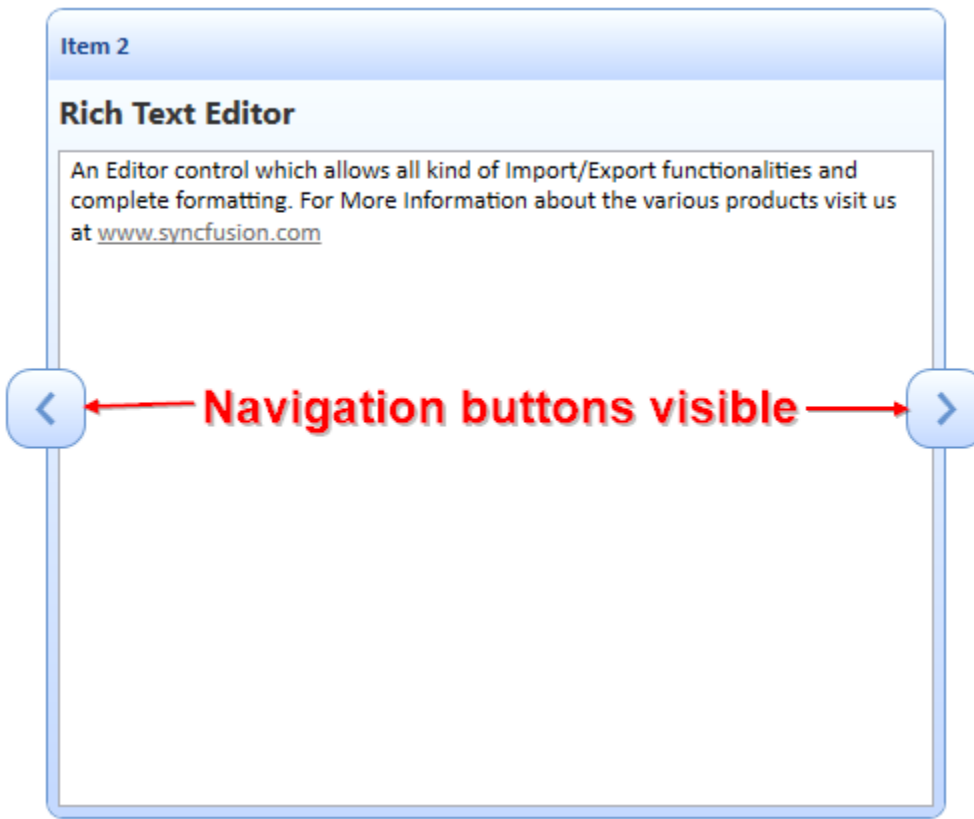
XML

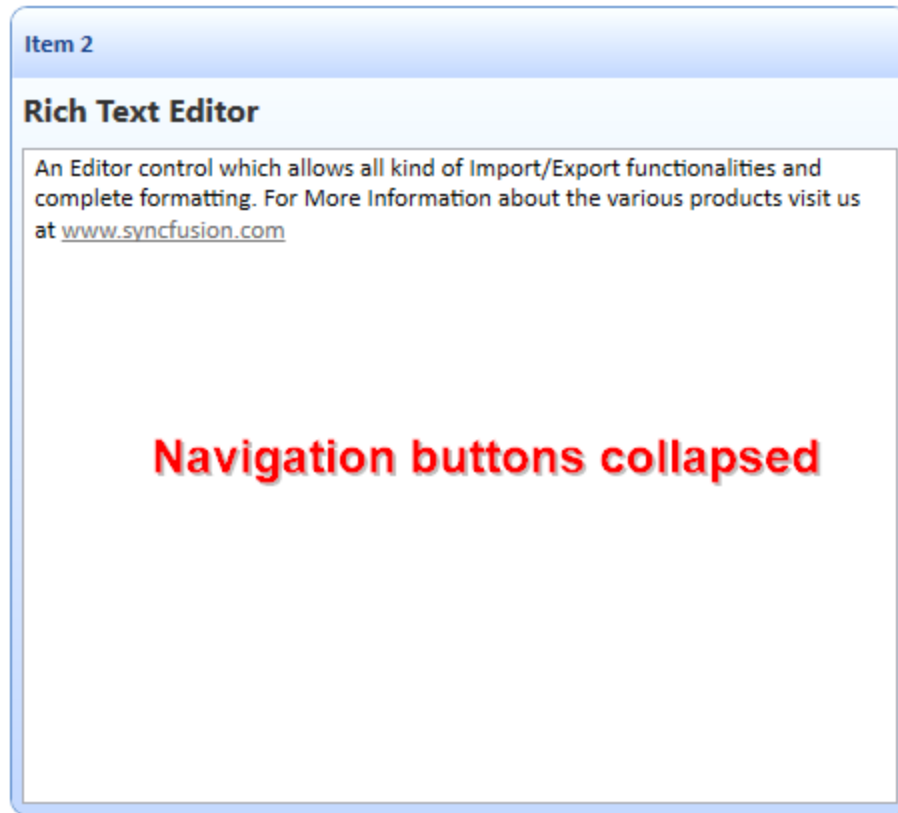
```
<!-- TabNavigation Control -->
```

```
<syncfusion:TabNavigationControl x:Name="TabNavigation"
TabStripVisibility="Visible" NavigationButtonVisibility="Collapsed" >
<syncfusion:TabNavigationItem Header="TabItem1" Content="TabNavigationItem
1"/>
<syncfusion:TabNavigationItem Header="TabItem2" Content="TabNavigationItem
2"/>
</syncfusion:TabNavigationControl>
```

C#

```
//Hide the navigation button
tabNavigation.NavigationButtonVisibility = Visibility.Collapsed;
```





Show/hide the TabStrip

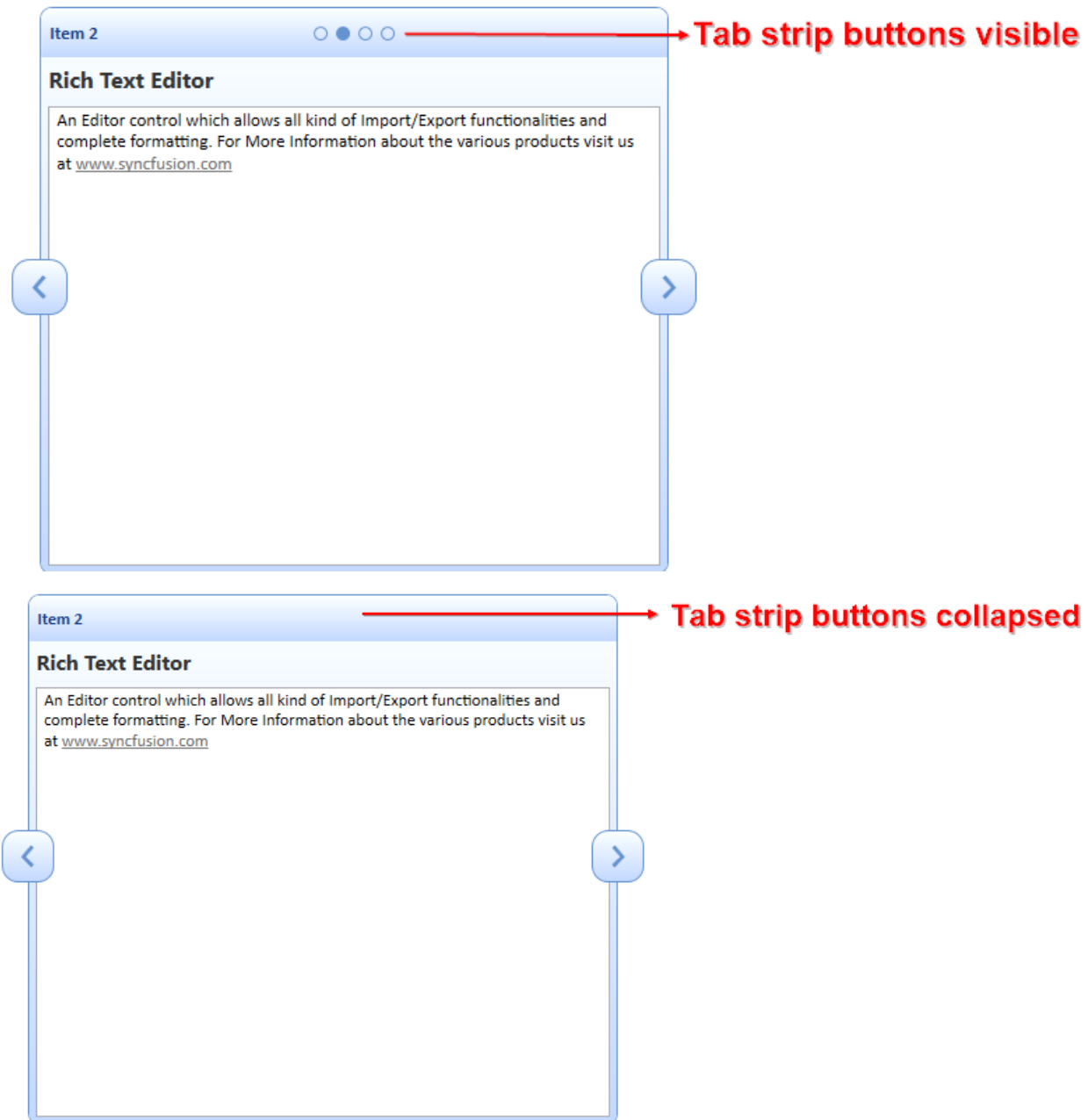
You can enable/disable the visibility of tab strip by setting [TabStripVisibility](#) property of TabNavigationControl.

XML

```
<!-- TabNavigation Control -->
<syncfusion:TabNavigationControl x:Name="TabNavigation"
TabStripVisibility="Visible">
<syncfusion:TabNavigationItem Header="TabItem1" Content="TabNavigationItem
1"/>
<syncfusion:TabNavigationItem Header="TabItem2" Content="TabNavigationItem
2"/>
</syncfusion:TabNavigationControl>
```

C#

```
//Enable the tab strip visibility
tabNavigation.TabStripVisibility = Visibility.Visible;
```



Animation in WPF Tab Navigation

Transition Effect property is used to set the animation effect for the tab navigation control. The Transition effect is an enum that contains five values namely:

- Slide – The item/page navigates with slide effect.

XML

```
<syncfusion:TabNavigationControl TransitionEffect="Slide" ItemsSource="{Binding MyCollection}">
</syncfusion:TabNavigationControl>
```

- Fade – During navigation, the previous item fades out and the new item appears with variation in opacity.

XML

```
<syncfusion:TabNavigationControl TransitionEffect="Fade" ItemsSource="{Binding MyCollection}">
</syncfusion:TabNavigationControl>
```

- Zoom – The new item appears with a zooming effect.

XML

```
<syncfusion:TabNavigationControl TransitionEffect="Zoom" ItemsSource="{Binding MyCollection}">
</syncfusion:TabNavigationControl>
```

- Blur – The new item appears with blur effect.

XML

```
<syncfusion:TabNavigationControl TransitionEffect="Blur" ItemsSource="{Binding MyCollection}">
</syncfusion:TabNavigationControl>
```

- Push – The new item descends from the top

XML

```
<syncfusion:TabNavigationControl TransitionEffect="Push" ItemsSource="{Binding MyCollection}">
</syncfusion:TabNavigationControl>
```

- PushIn – The new item ascends from the bottom

XML

```
<syncfusion:TabNavigationControl TransitionEffect="PushIn" ItemsSource="{Binding MyCollection}">
</syncfusion:TabNavigationControl>
```

- Wipe – The old item gets washed out and the new item appears.

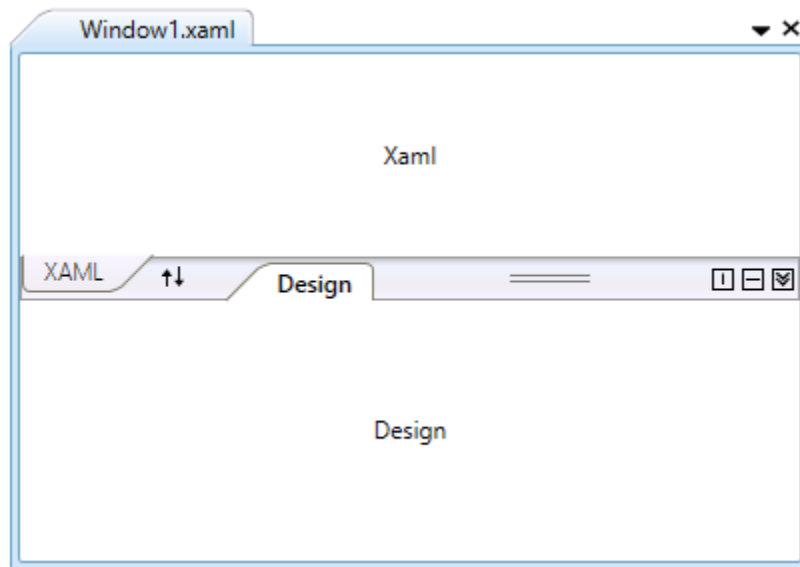
XML

```
<syncfusion:TabNavigationControl TransitionEffect="Wipe"
ItemsSource="{Binding MyCollection}">
</syncfusion:TabNavigationControl>
```


TabSplitter

WPF Tab Splitter Overview

TabSplitter is similar to VS 2008 style Split view of Tabbed Groups.



Features

Swap - Provides support to swapping the two tab groups.

Collapse and Expand - Provides support to collapse and expand the tab groups.

Layout - Provides support to toggle between vertical and horizontal layouts.

Appearance - Provides support to set the Background, Foreground color to TabSplitter.

Getting Started with WPF Tab Splitter

This section explains about how to create [TabSplitter](#) 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.

[Click here](#) to find more details on how to install nuget packages in WPF application.

Adding the TabSplitter control via XAML

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

1) Create a new WPF application via Visual Studio. 2) Add the following assembly reference to this project. *Syncfusion.Shared.WPF* *Syncfusion.Tools.WPF* 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the [TabSplitter](#) control in XAML page.

XML

```
<Window x:Class="TabSplitter.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:TabSplitter"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid Name="grid">
<syncfusion:TabSplitter Name="tabSplitter" Height="280" Width="400" />
</Grid>
```

Adding the TabSplitter control via C#

1) Create a new WPF application via Visual Studio. 2) Add the following required assembly references to the project. *Syncfusion.Shared.WPF* *Syncfusion.Tools.WPF* 3) Include the required namespace.

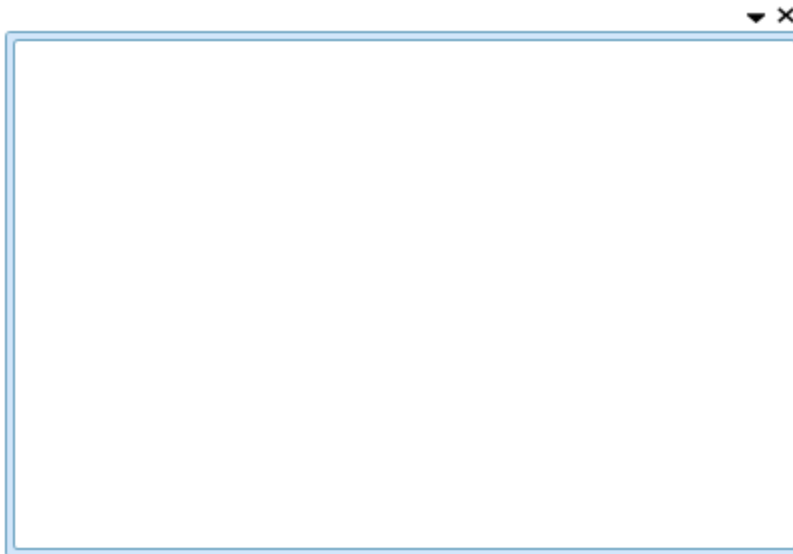
C#

```
using Syncfusion.Windows.Tools.Controls;
```

4) Create an instance of [TabSplitter](#) control and add it to the window.

C#

```
//Instance of TabSplitter
TabSplitter tabSplitter = new TabSplitter();
tabSplitter.Height = 280;
tabSplitter.Width = 400;
//Add control into the window
grid.Children.Add(tabSplitter);
```



Adding splitter item

1) You can add the splitter item into the TabSplitter control by using [Header](#) property. The TabSplitter items contains a collection of panel items. The panel items are [TopPanelItems](#) and [BottomPanelItems](#). You can also split the pages in TabSplitter item by using the [SplitterPage](#).

XML

```
<syncfusion:TabSplitterItem Header="Window1.xaml">
<syncfusion:TabSplitterItem.TopPanelItems>
<syncfusion:SplitterPage Header="XAML" />
</syncfusion:TabSplitterItem.TopPanelItems>
<syncfusion:TabSplitterItem.BottomPanelItems>
<syncfusion:SplitterPage Header="Design" />
</syncfusion:TabSplitterItem.BottomPanelItems>
</syncfusion:TabSplitterItem>
```

C#

```
//Create an instances of splitter page
SplitterPage splitterPage = new SplitterPage();
splitterPage.Header = "XAML";
SplitterPage splitterPage1 = new SplitterPage();
splitterPage1.Header = "Design";
//Create the tab splitter item
TabSplitterItem tabSplitterItem1 = new TabSplitterItem();
tabSplitterItem1.Header = "MainWindow.XAML";
//Add the splitter page into the tabSplitterItem
tabSplitterItem1.TopPanelItems.Add(splitterPage);
tabSplitterItem1.BottomPanelItems.Add(splitterPage1);
//Add tabSplitterItem into the tabSplitter
tabSplitter.Items.Add(tabSplitterItem1);
```

2) Add any one of the control to the splitter page of TabSplitter and add the content into the splitter page.

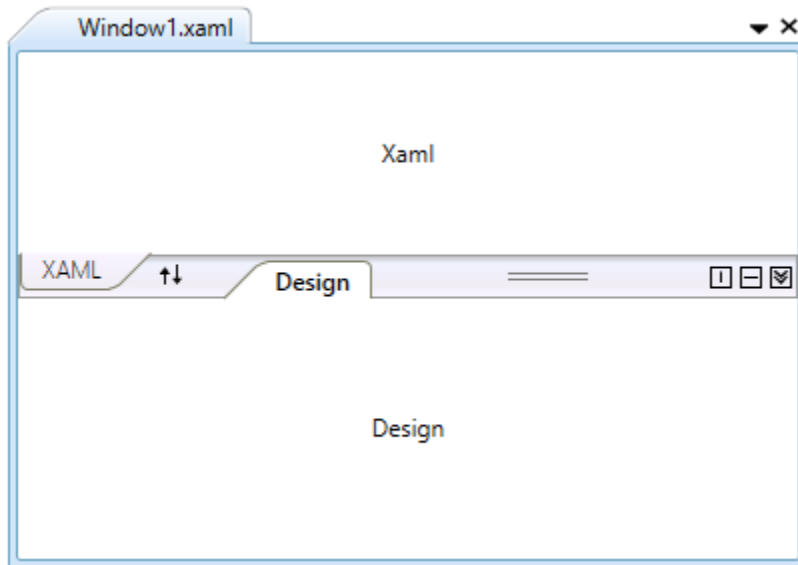
XML

```
<syncfusion:TabSplitterItem Header="Window1.xaml">
<syncfusion:TabSplitterItem.TopPanelItems>
<syncfusion:SplitterPage Header="XAML">
<Label Content="Xaml" HorizontalContentAlignment="Center"
VerticalContentAlignment="Center" />
</syncfusion:SplitterPage>
</syncfusion:TabSplitterItem.TopPanelItems>
<syncfusion:TabSplitterItem.BottomPanelItems>
<syncfusion:SplitterPage Header="Design">
<Label Content="Design" HorizontalContentAlignment="Center"
VerticalContentAlignment="Center" />
</syncfusion:SplitterPage>
</syncfusion:TabSplitterItem.BottomPanelItems>
</syncfusion:TabSplitterItem>
```

C#

```
Label label = new Label();
label.Content = "XAML";
label.HorizontalContentAlignment = HorizontalAlignment.Center;
label.VerticalContentAlignment = VerticalAlignment.Center;
Label label1 = new Label();
label1.Content = "Design";
```

```
label1.HorizontalContentAlignment = HorizontalAlignment.Center;  
label1.VerticalContentAlignment = VerticalAlignment.Center;  
splitterPage.Content = label;  
splitterPage1.Content = label1;
```



Tab orientation

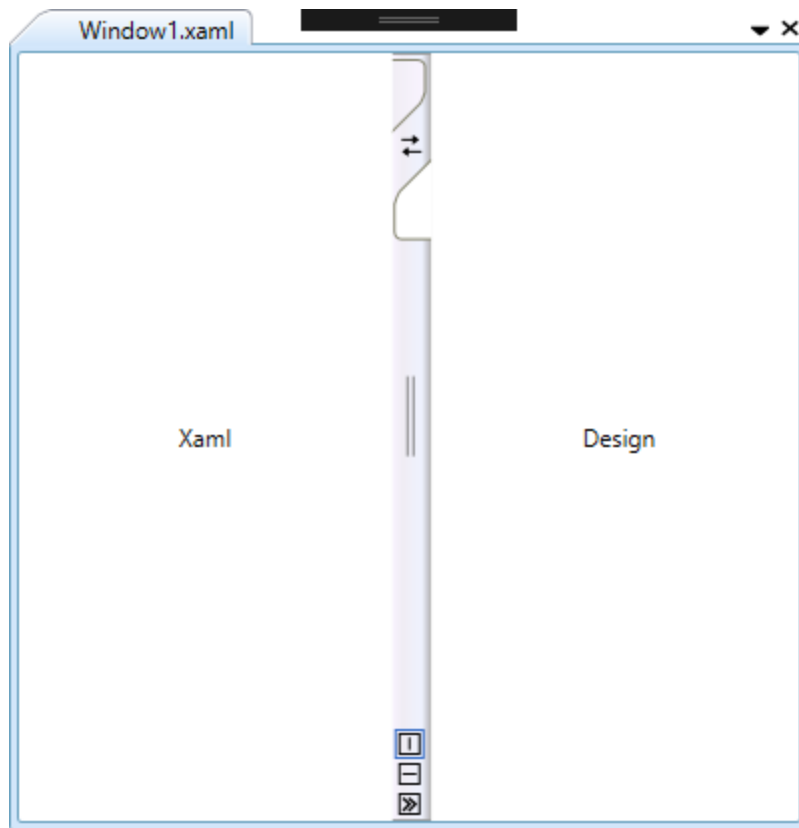
The TabSplitter items are placed horizontally or vertically by using the [Orientation](#) property.

XML

```
<syncfusion:TabSplitterItem Header="Window1.xaml" Orientation="Vertical">  
</syncfusion:TabSplitterItem>
```

C#

```
tabSplitterItem1.Orientation = Orientation.Vertical;
```



Collapsing bottom panel

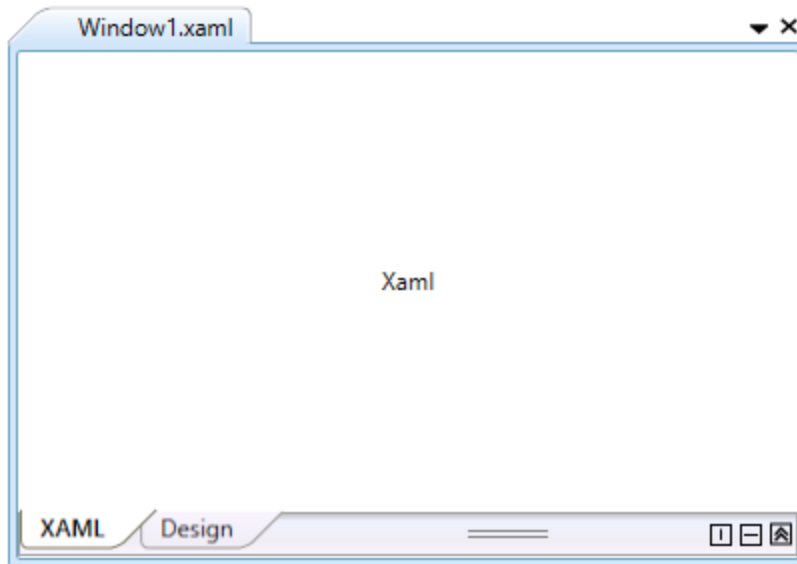
You can collapse or expand the bottom panel by setting the [IsCollapsedBottomPanel](#) property to `true`. The default value is `false`.

XML

```
<syncfusion:TabSplitterItem Header="Window1.xaml"
    IsCollapsedBottomPanel="True">
</syncfusion:TabSplitterItem>
```

C#

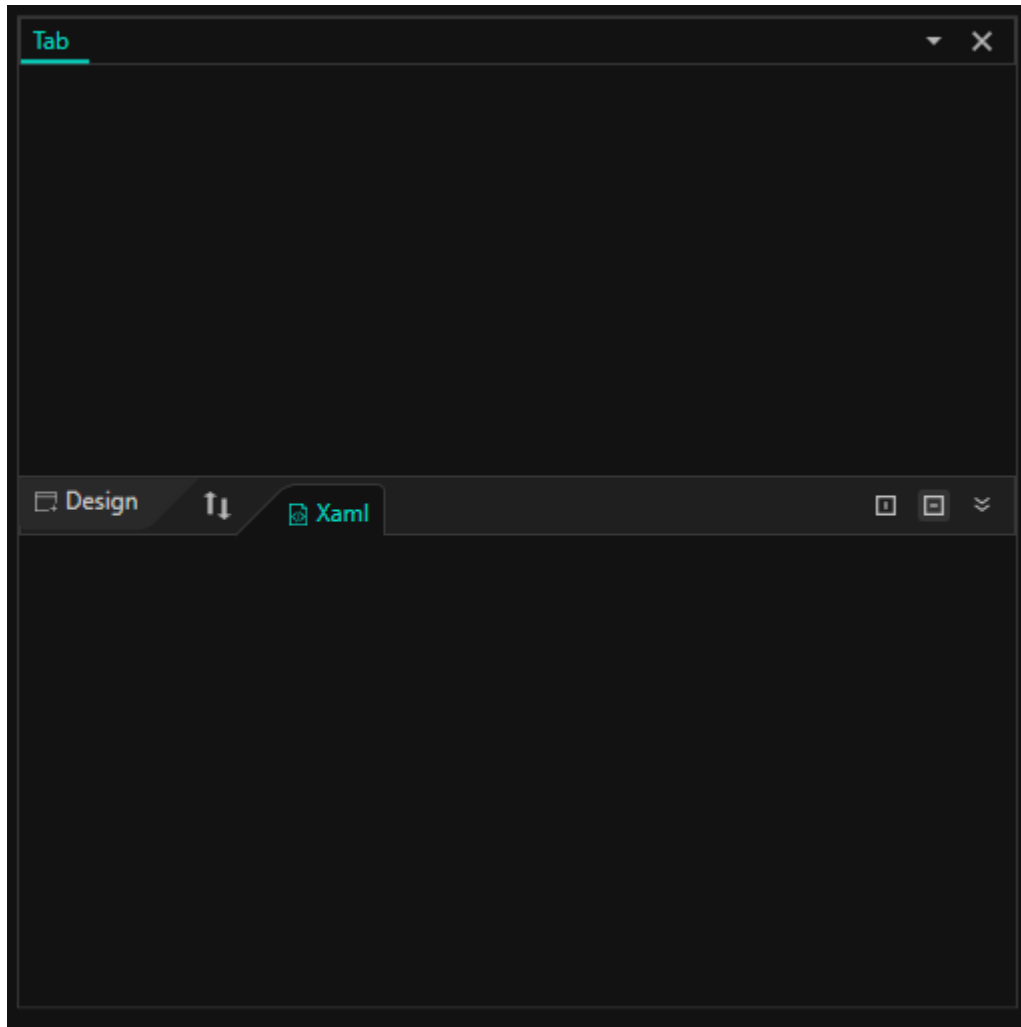
```
tabSplitterItem1.IsCollapsedBottomPanel = true;
```



Theme

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

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



Interactive Features in WPF Tab Splitter

This section illustrates the following interactive features of TabSplitter control.

Adding TabSplitterItem to the TabSplitter Control

TabSplitter contains one or more pages that are defined as TabSplitter Items. Use the following code to add a TabSplitter Item to the TabSplitter control.

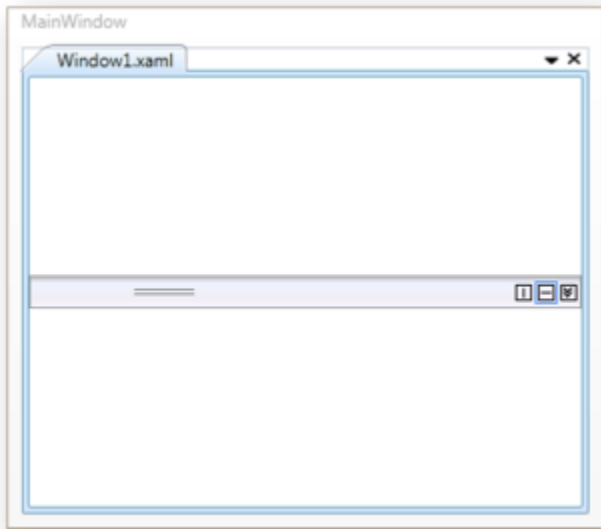
XML

```
<!-- Adding TabSplitter -->
<syncfusion:TabSplitter Name="tabsplitter">
  <!-- Adding TabSplitterItem -->
  <syncfusion:TabSplitterItem Header="Window1.xml" Name="tabSplitterItem1">
  </syncfusion:TabSplitterItem>
</syncfusion:TabSplitter>
```

C#

```
// Creating an instance of TabSplitter
TabSplitter tabSplitter = new TabSplitter();
// Creating an instance of TabSplitterItem
TabSplitterItem tabSplitterItem1 = new TabSplitterItem();
```

```
// Adding header of the TabSplitterItem
tabSplitterItem1.Header = "Window1.xml";
// Adding TabSplitter Item to TabSplitter
tabSplitter.Items.Add(tabSplitterItem1);
// Adding TabSplitter to Window
this.Content = tabsplitter;
```



Panel Items

TabSplitter Items contains a collection of pages. These pages are defined as Panel Items.

There are two types of panel Items:

- [TopPanelItems](#): consists of collection of pages that are placed at the top panel of the TabSplitter
- [BottomPanelItems](#): consists of collection of pages that are placed at the bottom panel of the TabSplitter

The following code example can be used to add Panel Items to the TabSplitter Item:

XML

```
<!-- Adding TabSplitter -->
<syncfusion:TabSplitter Name="tabsplitter">
  <!-- Adding TabSplitterItem -->
  <syncfusion:TabSplitterItem Header="Window1.xml" Name="tabSplitterItem1">
    <!-- Adding TopPanelItems -->
    <syncfusion:TabSplitterItem.TopPanelItems>
      <!-- Adding SplitterPage -->
      <syncfusion:SplitterPage Name="splitterPage1" Header="XAML">
      </syncfusion:SplitterPage>
    </syncfusion:TabSplitterItem.TopPanelItems>
    <!-- Adding BottomPanelItems -->
    <syncfusion:TabSplitterItem.BottomPanelItems>
      <!-- Adding SplitterPage -->
      <syncfusion:SplitterPage Name="splitterPage2" Header="Design">

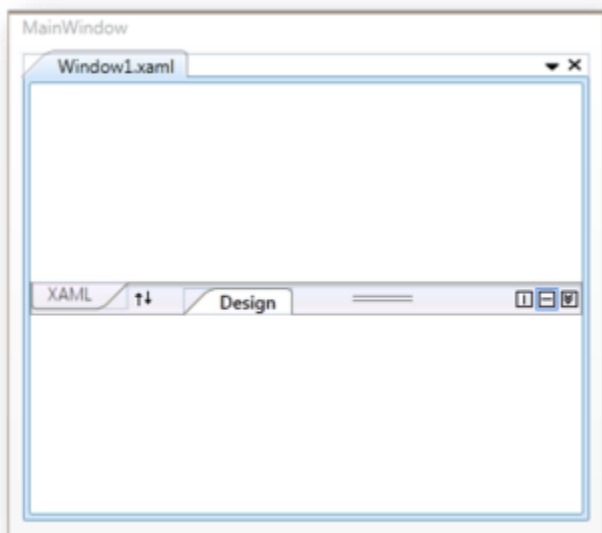
```



```
</syncfusion:SplitterPage>  
</syncfusion:TabSplitterItem.BottomPanelItems>  
</syncfusion:TabSplitterItem>  
</syncfusion:TabSplitter>
```

C#

```
// Creating an instance of TabSplitter  
TabSplitter tabSplitter = new TabSplitter();  
// Creating an instance of TabSplitterItem  
TabSplitterItem tabSplitterItem1 = new TabSplitterItem();  
// Adding header of the TabSplitterItem  
tabSplitterItem1.Header = "Window1.xml";  
// Creating an instance splitter page  
SplitterPage splitterPage1 = new SplitterPage();  
splitterPage1.Header = "XAML";  
// Adding SplitterPage to TopPanelItem  
tabSplitterItem1.TopPanelItems.Add(splitterPage1);  
// Creating an instance SplitterPage  
SplitterPage splitterPage2 = new SplitterPage();  
splitterPage2.Header = "Design";  
// Adding SplitterPage to BottomPanelItem  
tabSplitterItem1.BottomPanelItems.Add(splitterPage2);  
// Adding TabSplitter Item to TabSplitter  
tabSplitter.Items.Add(tabSplitterItem1);  
// Adding TabSplitter to Window  
this.Content = tabSplitter;
```



Splitter Page

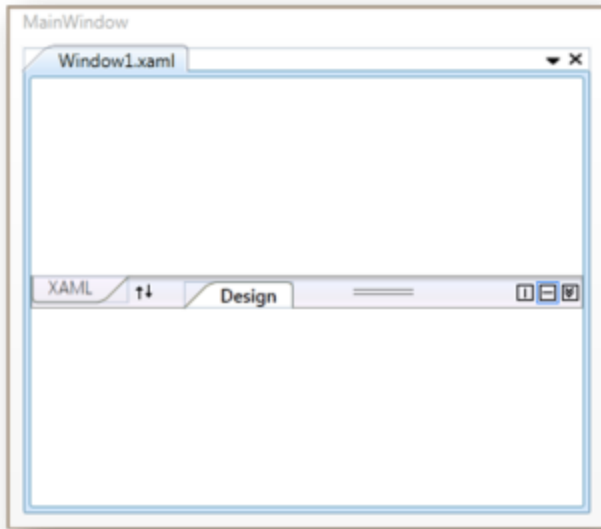
You can split the pages in the TabSplitter Item by using the [SplitterPage](#). The following code example illustrates how to add a SplitterPage to the TabSplitter Item.

XML

```
<!-- Adding TabSplitter -->
<syncfusion:TabSplitter Name="tabsplitter">
  <!-- Adding TabSplitterItem -->
  <syncfusion:TabSplitterItem Header="Window1.xml" Name="tabSplitterItem1">
    <!-- Adding TopPanelItems -->
    <syncfusion:TabSplitterItem.TopPanelItems>
      <!-- Adding SplitterPage -->
      <syncfusion:SplitterPage Name="splitterPage1" Header="XAML">
        </syncfusion:SplitterPage>
      </syncfusion:TabSplitterItem.TopPanelItems>
    <!-- Adding BottomPanelItems -->
    <syncfusion:TabSplitterItem.BottomPanelItems>
      <!-- Adding Splitter Page -->
      <syncfusion:SplitterPage Name="splitterPage2" Header="Design">
        </syncfusion:SplitterPage>
      </syncfusion:TabSplitterItem.BottomPanelItems>
    </syncfusion:TabSplitterItem>
  </syncfusion:TabSplitter>
```

C#

```
// Creating an instance of TabSplitter
TabSplitter tabSplitter = new TabSplitter();
// Creating an instance of TabSplitterItem
TabSplitterItem tabSplitterItem1 = new TabSplitterItem();
// Adding header of the TabSplitterItem
tabSplitterItem1.Header = "Window1.xml";
// Creating an instance SplitterPage
SplitterPage splitterPage1 = new SplitterPage();
splitterPage1.Header = "XAML";
// Adding SplitterPage to TopPanelItem
tabSplitterItem1.TopPanelItems.Add(splitterPage1);
// Creating an instance SplitterPage
SplitterPage splitterPage2 = new SplitterPage();
splitterPage2.Header = "Design";
// Adding SplitterPage to BottomPanelItem
tabSplitterItem1.BottomPanelItems.Add(splitterPage2);
// Adding TabSplitter Item to TabSplitter
tabSplitter.Items.Add(tabSplitterItem1);
// Adding TabSplitter to Window
this.Content = tabSplitter;
```



Collapsing Bottom Panel

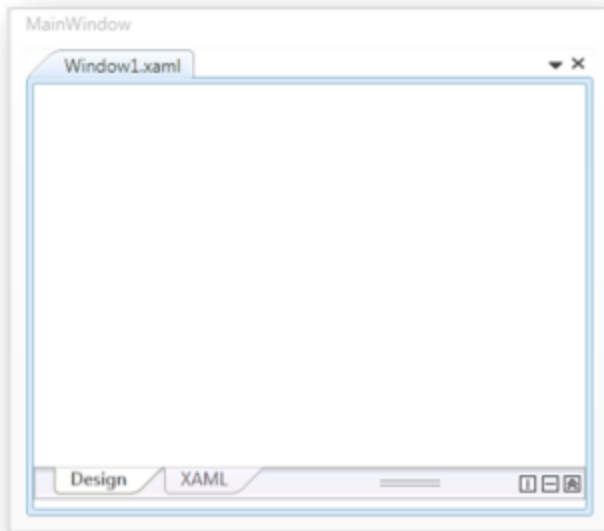
You can collapse or expand the Bottom Panel by using the [IsCollapsedBottomPanel](#) property. The default value is *false*. To collapse the Bottom Panel, refer to the following code snippet:

XML

```
<!-- Adding TabSplitter -->
<syncfusion:TabSplitter Name="tabsplitter">
  <!-- Adding TabSplitterItem -->
  <syncfusion:TabSplitterItem Header="Window1.xml"
    IsCollapsedBottomPanel="True" Name="tabSplitterItem1">
    <!-- Adding TopPanelItems -->
    <syncfusion:TabSplitterItem.TopPanelItems>
      <!-- Adding SplitterPage -->
      <syncfusion:SplitterPage Name="splitterPage1" Header="XAML">
      </syncfusion:SplitterPage>
    </syncfusion:TabSplitterItem.TopPanelItems>
    <!-- Adding BottomPanelItems -->
    <syncfusion:TabSplitterItem.BottomPanelItems>
      <!-- Adding SplitterPage -->
      <syncfusion:SplitterPage Name="splitterPage2" Header="Design">
      </syncfusion:SplitterPage>
    </syncfusion:TabSplitterItem.BottomPanelItems>
  </syncfusion:TabSplitterItem>
</syncfusion:TabSplitter>
```

C#

```
// Enable IsCollapseBottomPanel property.
tabSplitterItem1.IsCollapsedBottomPanel = true;
```



Setting BottomPanelHeight of TabSplitter

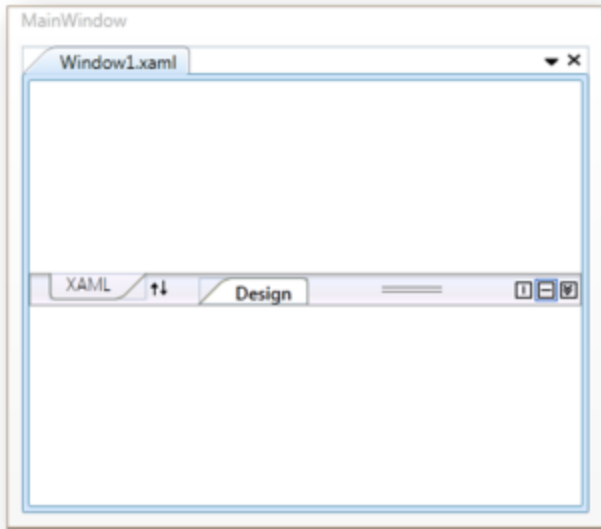
You can set the height of the BottomPanel in TabSplitter using [BottomPanelHeight](#) property. To set the height of the BottomPanel, refer the following code snippet:

XML

```
<!-- Adding TabSplitter -->
<syncfusion:TabSplitter BottomPanelHeight="150">
  <!-- Adding TabSplitterItem -->
  <syncfusion:TabSplitterItem Header="MainWindow.xml">
    <!-- Adding TopPanelItems -->
    <syncfusion:TabSplitterItem.TopPanelItems>
      <syncfusion:SplitterPage Header="Design" />
    </syncfusion:TabSplitterItem.TopPanelItems>
    <!-- Adding BottomPanelItems -->
    <syncfusion:TabSplitterItem.BottomPanelItems>
      <syncfusion:SplitterPage Header="XAML" />
    </syncfusion:TabSplitterItem.BottomPanelItems>
  </syncfusion:TabSplitterItem>
</syncfusion:TabSplitter>
```

C#

```
// Creating an instance of TabSplitter
TabSplitter tabSplitter = new TabSplitter();
//Setting BottomPanelHeight property
tabSplitter.BottomPanelHeight = 150;
```



Orientation in WPF Tab Splitter

TabSplitter Items are placed horizontally or vertically by using the [Orientation](#) property. This is a dependency property which sets the orientation of the TabSplitter Item. It supports the following types of orientation.

- Horizontal: TabSplitter Item in the TabSplitter is placed horizontally
- Vertical: TabSplitter Item in the TabSplitter is placed vertically

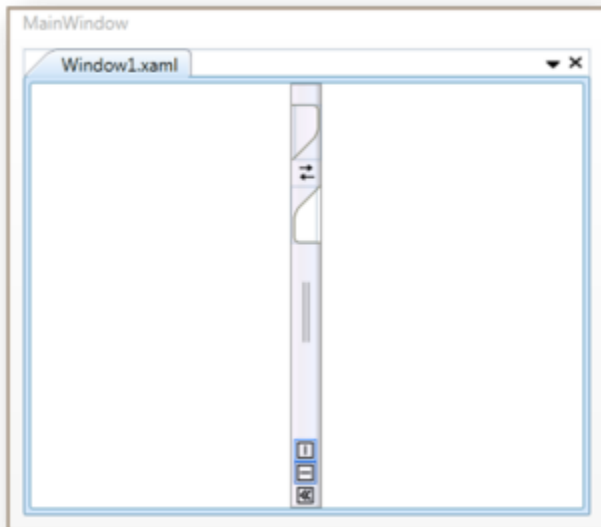
Set the orientation using the code given below:

XML

```
<!-- Adding TabSplitter -->
<syncfusion:TabSplitter Name="tabsplitter">
  <!-- Adding TabSplitterItem -->
  <syncfusion:TabSplitterItem Header="Window1.xml" Orientation="Vertical"
  Name="tabSplitterItem1">
    <!-- Adding TopPanelItems -->
    <syncfusion:TabSplitterItem.TopPanelItems>
      <!-- Adding SplitterPage -->
      <syncfusion:SplitterPage Name="splitterPage1" Header="XAML">
      </syncfusion:SplitterPage>
    </syncfusion:TabSplitterItem.TopPanelItems>
    <!-- Adding BottomPanelItems -->
    <syncfusion:TabSplitterItem.BottomPanelItems>
      <!-- Adding SplitterPage -->
      <syncfusion:SplitterPage Name="splitterPage2" Header="Design">
      </syncfusion:SplitterPage>
    </syncfusion:TabSplitterItem.BottomPanelItems>
  </syncfusion:TabSplitterItem>
</syncfusion:TabSplitter>
```

C#

```
// Set the Orientation
tabSplitterItem1.Orientation = Orientation.Vertical;
```



Selected Page in WPF Tab Splitter

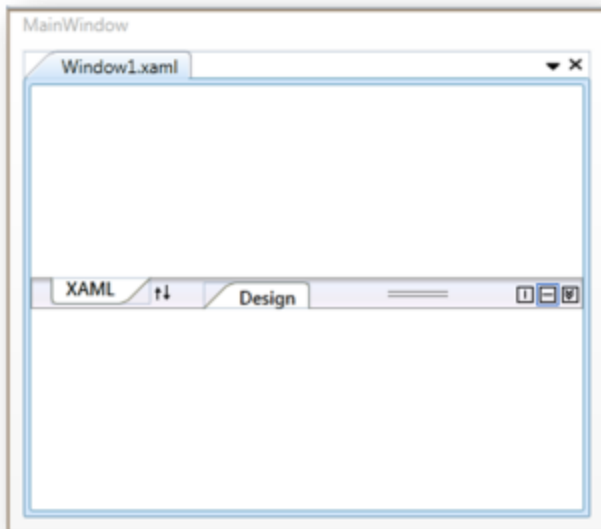
You can set the selected page by using the [IsSelectedPage](#) property. If this property is set to *true*, the page is selected, else it is not selected.

XML

```
<!-- Adding TabSplitter -->
<syncfusion:TabSplitter Name="tabsplitter">
  <!-- Adding TabSplitterItem -->
  <syncfusion:TabSplitterItem Header="Window1.xml" Name="tabSplitterItem1">
    <!-- Adding TopPanelItems -->
    <syncfusion:TabSplitterItem.TopPanelItems>
      <!-- Adding SplitterPage -->
      <syncfusion:SplitterPage IsSelectedPage="True" Name="splitterPage1"
      Header="XAML">
    </syncfusion:SplitterPage>
    </syncfusion:TabSplitterItem.TopPanelItems>
    <!-- Adding BottomPanelItems -->
    <syncfusion:TabSplitterItem.BottomPanelItems>
      <!-- Adding SplitterPage -->
      <syncfusion:SplitterPage Name="splitterPage2" Header="Design">
    </syncfusion:SplitterPage>
    </syncfusion:TabSplitterItem.BottomPanelItems>
  </syncfusion:TabSplitterItem>
</syncfusion:TabSplitter>
```

C#

```
// Enable the IsSelectedPage property.
splitterPage1.IsSelectedPage = true;
```



Layout Related Features in WPF Tab Splitter

This section illustrates the following Layout-related feature of [TabSplitter](#) control.

Customizing the Appearance of TabSplitter

The appearance of the [TabSplitter](#) control is customized by using the appearance properties available in the control. You can set the color for the [MouseOverBackground](#), [MouseOverForeground](#), [SelectedForeground](#) and [SelectedBackground](#) of the TabSplitter control. Here is the code snippet.

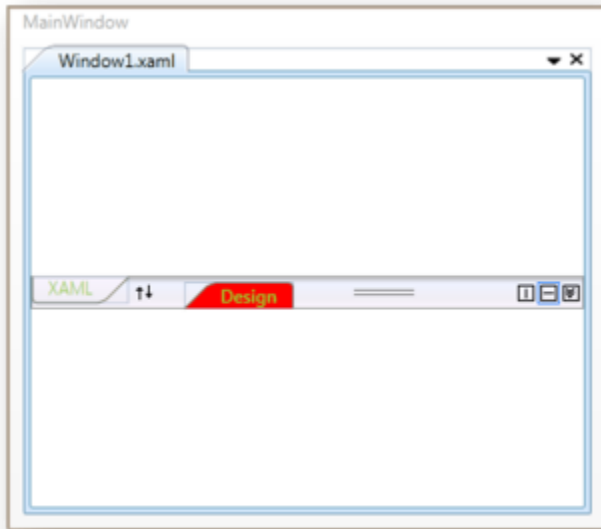
XML

```
<Grid>
  <!-- Adding TabSplitter With Selected Brush -->
  <syncfusion:TabSplitter Name="tabsplitter" MouseOverBackground="Green"
  MouseOverForeground="Yellow" SelectedBackground="Red"
  SelectedForeground="YellowGreen">
    <!-- Adding TabSplitterItem -->
    <syncfusion:TabSplitterItem Header="Window1.xml" Name="tabSplitterItem1">
      <!-- Adding TopPanelItems -->
      <syncfusion:TabSplitterItem.TopPanelItems>
        <syncfusion:SplitterPage Name="splitterPage1" Header="XAML" />
      </syncfusion:TabSplitterItem.TopPanelItems>
      <!-- Adding BottomPanelItems -->
      <syncfusion:TabSplitterItem.BottomPanelItems>
        <syncfusion:SplitterPage Name="splitterPage2" Header="Design" />
      </syncfusion:TabSplitterItem.BottomPanelItems>
    </syncfusion:TabSplitterItem>
  </syncfusion:TabSplitter>
</Grid>
```

C#

```
// Set the selected background.
tabsplitter.SelectedBackground = Brushes.Red;
// Set the selected foreground.
tabsplitter.SelectedForeground = Brushes.YellowGreen;
```

```
// Set the MouseOverBackground.
tabsplitter.MouseOverBackground = Brushes.Green;
// Set the MouseOverForeground.
tabsplitter.MouseOverForeground = Brushes.Yellow;
```



Hide TabSplitterItem header tab in TabSplitter

You can hide the [TabSplitterItem](#) header tab in [TabSplitter](#) by enabling the [HideHeaderOnSingleChild](#) property. This property works only if TabSplitter control has one TabSplitterItem in it.

XML

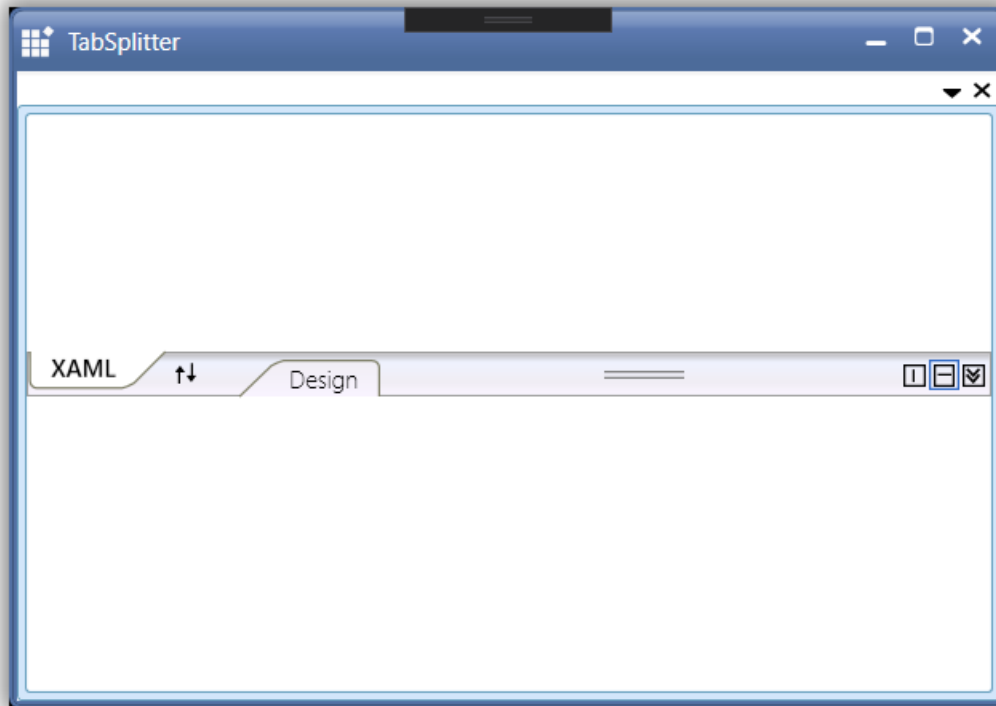
```
<Grid>
<syncfusion:TabSplitter Name="tabsplitter" HideHeaderOnSingleChild="True">
<syncfusion:TabSplitterItem Header="Window1.xml" Name="tabSplitterItem1">
<syncfusion:TabSplitterItem.TopPanelItems>
<syncfusion:SplitterPage Name="splitterPage1" Header="XAML" />
</syncfusion:TabSplitterItem.TopPanelItems>
<syncfusion:TabSplitterItem.BottomPanelItems>
<syncfusion:SplitterPage Name="splitterPage2" Header="Design" />
</syncfusion:TabSplitterItem.BottomPanelItems>
</syncfusion:TabSplitterItem>
</syncfusion:TabSplitter>
</Grid>
```

C#

```
TabSplitter tabSplitter = new TabSplitter() { HideHeaderOnSingleChild =
true};
TabSplitterItem tabSplitterItem = new TabSplitterItem() { Header=
"Window1.xml" };
SplitterPage page1 = new SplitterPage() { Header = "XAML" };
SplitterPage page2 = new SplitterPage() { Header = "Design" };
tabSplitterItem.TopPanelItems.Add(page1);
tabSplitterItem.BottomPanelItems.Add(page2);
```



```
tabSplitter.Items.Add(tabSplitterItem);
```



TaskBar

WPF TaskBar Overview

TaskBar control has the capability to provide an UI that is similar to the Windows Explorer TaskBar. It provides a consistent UI for placing commonly used functionalities as grouped items. You can place any Container Panel control inside the TaskBar. For example, when the customized Grid Panel with other controls are placed inside the TaskBar Item, it will be automatically arranged inside the TaskBar Items collection. TaskBar supports the Office2007Black, Blue, Silver, Office2003 and Blend themes.

Features

- Windows Explorer TaskBar look and feel.
- Horizontal / Vertical layouts for TaskBar Boxes.
- Provides support to customize the background, header size, and collapse / expand the TaskBar Header.
- Keyboard navigation support.
- Ability to specify custom group margin and padding.
- Supports built-in visual styles - Default, Win XP, Zune, Aero, Office2007Blue, Office2007Black, Office2007Silver, Office2003, Blend, LunaRoyale, LunaHomestead and LunaMetallic.

Getting Started with WPF TaskBar

This section guides you on getting started with TaskBar control. It covers the following topics:

Overview of Taskbar control

TaskBar control has the capability to provide a UI that is similar to the Windows Explorer TaskBar. It provides a consistent UI for placing commonly used functionalities as grouped items. You can place any Container Panel control inside the TaskBar. For example, when the customized Grid Panel with other controls are placed inside the TaskBar Item, it will be automatically arranged inside the TaskBar Items collection. TaskBar supports the Office2007Black, Office2007Blue, Office2007Silver, Office2003 and Blend themes.



Why to use our TaskBar control

Here are some highlights of our TaskBar control.

- Windows Explorer TaskBar appearance can be achieved
- Horizontal/Vertical layouts for TaskBar Boxes
- Provides support to customize the background, header size, and collapse / expand the TaskBar Header
- Keyboard navigation support
- Ability to specify custom group margin and padding
- Supports built-in visual styles - Default, Win XP, Zune, Aero, Office2007Blue, Office2007Black, Office2007Silver, Office2003, Blend, LunaRoyale, LunaHomestead and LunaMetallic

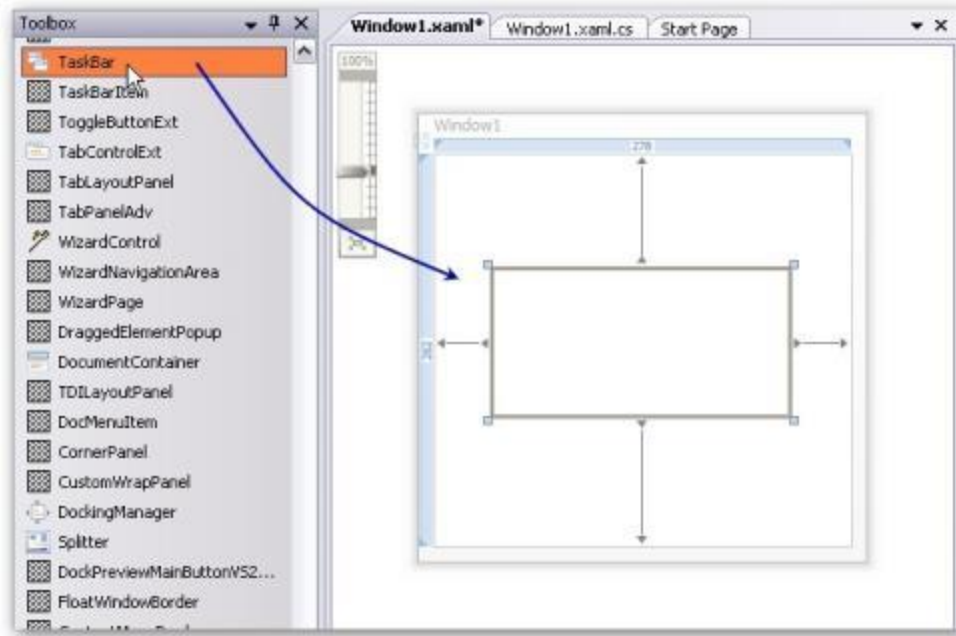
Creating the Taskbar and its children element using Docking Manager

There are two possible ways to create a simple TaskBar control.

Through Designer

To create the TaskBar control through designer, do the following steps:

1. Drag the TaskBar control from the toolbox onto your WPF application.



2. Set the properties for the TaskBar in design mode by using the Smart Tag feature.

Programmatically

TaskBar control is created by using either XAML or C# code. The following lines of code can be used to create a TaskBar control.

XML

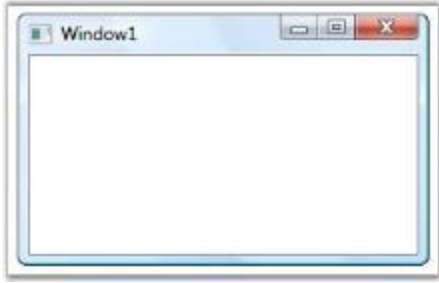
```
<!-- Adding TaskBar -->
<syncfusion:TaskBar Name="taskBar" >
</syncfusion:TaskBar>
```

C#

```
//Creating an instance for TaskBar
TaskBar taskBar = new TaskBar();
//Adding TaskBar as content of window
this.Content = taskBar;
```

Note: To display the TaskBar by using C# code, you must already have a panel in which you are going to add the control. Otherwise, the control cannot be displayed.

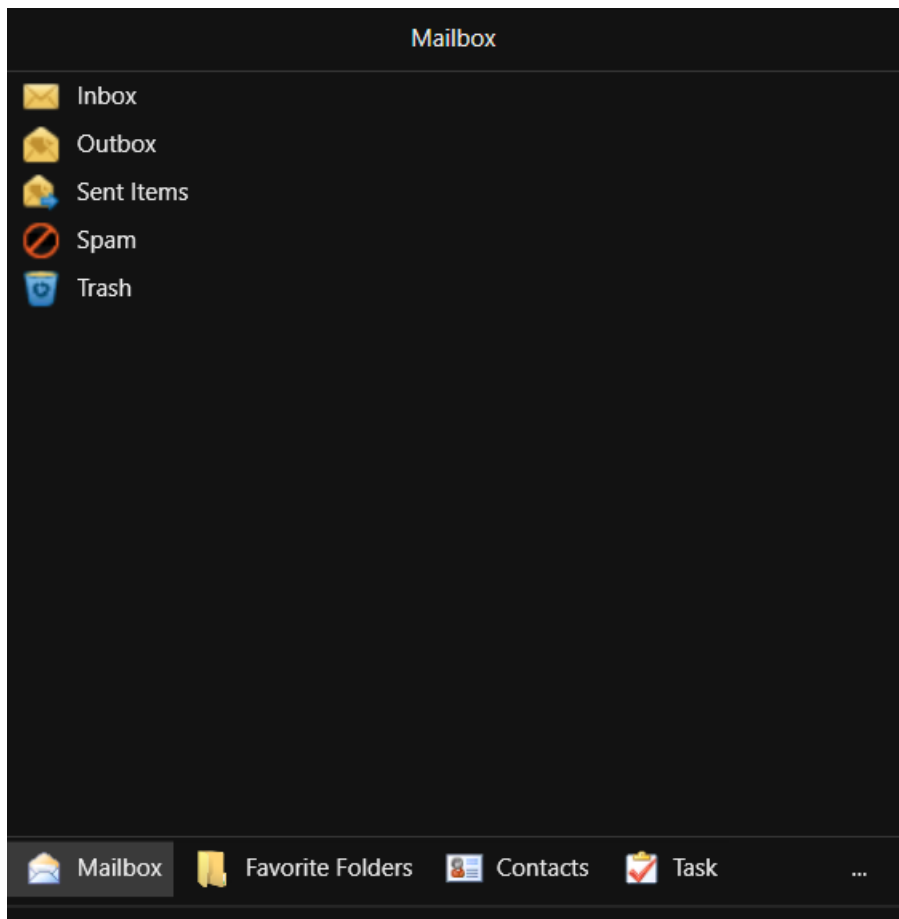
The following screen shot shows the TaskBar control.



Theme

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

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



Adding a TaskBar Item in WPF TaskBar control

The Task Bar Item is added to the Task Bar by using either XAML or C# coding. Use the following code to add a Task Bar Item to the Task Bar control.

XML

```
<!-- Adding TaskBar -->  
<syncfusion:TaskBar Name="taskBar" >
```

```
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
</syncfusion:TaskBarItem>
</syncfusion:TaskBar>
```

C#

```
//Creating an instance for TaskBar
TaskBar taskBar = new TaskBar();
//Creating an instance for TaskBarItem
TaskBarItem taskBarItem1 = new TaskBarItem();
//Setting the header of TaskBarItem1
taskBarItem1.Header = "TaskBarItem1";
//Adding the TaskBarItem to TaskBar
taskBar.Items.Add(taskBarItem1);
//Adding TaskBar as content of window
this.Content = taskBar;
```

The following screen shot shows the TaskBar control with TaskBar Item.

**Adding Content to TaskBar Item**

To add text to the TaskBar Item, use the below code

XML

```
<!-- Adding TaskBar -->
<syncfusion:TaskBar Name="taskBar" >
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
<!-- Adding content to TaskBarItem -->
<StackPanel Margin="10" HorizontalAlignment="Center"
VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap"> This TaskBar that have a
TaskBarItem.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
</syncfusion:TaskBar>
```

C#

```
//Creating an instance for TaskBar
TaskBar taskBar = new TaskBar();
//Creating an instance for TaskBarItem
TaskBarItem taskBarItem1 = new TaskBarItem();
//Setting the header of TaskBarItem1
taskBarItem1.Header = "TaskBarItem1";
// Creating instance for TextBlock
```

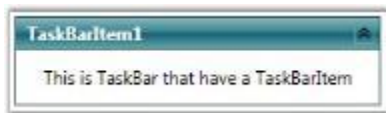
```

TextBlock textBlock1 = new TextBlock();
// Adding text to textblock
textBlock1.Text = "This TaskBar that have a TaskBarItem.";
// Adding textblock to taskbaritem
taskBarItem1.Items.Add(textBlock1);
//Adding the TaskBarItem to TaskBar
taskBar.Items.Add(taskBarItem1);
//Adding TaskBar as content of window
this.Content = taskBar;

```

Note: To display the TaskBar Item, you must already have the TaskBar in which you are going to add the TaskBar Item.

The following screen shot illustrates how text has been added to the TaskBar Item.



Setting Header Image in WPF TaskBar

You can set a custom header image for the TaskBarItem. Use the following code snippet to set an image for the header.

XML

```

<!-- Adding TaskBar that have group orientation as horizontal -->
<syncfusion:TaskBar Name="taskBar" GroupMargin="5">
  <!-- Adding TaskBarItem -->
  <syncfusion:TaskBarItem Name="taskBarItem1">
    <!-- Adding header with image -->
    <syncfusion:TaskBarItem.Header>
      <DockPanel Margin="0">
        <Image Height="16" Width="16" Source="App.ico"/>
        <TextBlock Foreground="White" Margin="5,0,0,0"

          Text="Header" />
      </DockPanel>
    </syncfusion:TaskBarItem.Header>
    <!-- Adding content to taskbaritem -->
    <StackPanel Margin="10" HorizontalAlignment="Center"

      VerticalAlignment="Stretch">
      <TextBlock TextWrapping="Wrap">This taskbar provides an UI similar to that
of Windows XP.</TextBlock>
    </StackPanel>
  </syncfusion:TaskBarItem>
</syncfusion:TaskBar>

```



Group Padding for the TaskBar Item

Setting Group Padding in WPF TaskBar

You can set the padding for the TaskBar Items by using the [GroupPadding](#) property. This is an attached property, which sets the padding for adjusting the TaskBarItems by using the [SetGroupPadding](#) method. You can enhance the appearance of the content in the taskbar items by using this property.

You can also set the value for Left, Right, Bottom and Top group padding.

To set the group padding, use the below code

XML

```
<!-- Adding TaskBar that have group padding as 20 -->
<syncfusion:TaskBar Name="taskBar" GroupMargin="5"

        syncfusion:TaskBar.GroupPadding="20">
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
<!-- Adding content to taskbaritem -->
<StackPanel Margin="10" HorizontalAlignment="Center"
VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap">This taskbar provides an UI similar to that
of Windows XP.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem2" Header="TaskBarItem2">
<!-- Adding content to TaskBarItem -->
<StackPanel Margin="10" HorizontalAlignment="Center"

        VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap">Specify and customize the group
margin.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
</syncfusion:TaskBar>
```

C#

```
//Setting group padding for TaskBar
TaskBar.SetGroupPadding(taskBar, new Thickness(20));
```



TaskBar Item Header Image

Changing the size of Expander Button in WPF TaskBar

You can change the size of the Expander button in the TaskBar control. This is achieved by using the [ButtonSize](#) attached property, which sets the height of the Expander button in the TaskBar by handling the [SetButtonSize](#) method.

To set the size of the Expander button, use the below code

XML

```
<!-- Adding TaskBar that have button size is 20 -->
<syncfusion:TaskBar Name="taskBar" GroupMargin="5"
syncfusion:TaskBar.ButtonSize="20">
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
<!-- Adding content to taskbaritem -->
<StackPanel Margin="10" HorizontalAlignment="Center"

        VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap">This taskbar provides an UI similar to that
of Windows XP.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem2" Header="TaskBarItem2">
<!-- Adding content to taskbaritem -->
<StackPanel Margin="10" HorizontalAlignment="Center"

        VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap">Specify and customize the group
margin.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
</syncfusion:TaskBar>
```

C#

```
// Setting button size
TaskBar.SetButtonSize(taskBar, 20);
```




Collapsing the TaskBar

Collapsing the TaskBar in WPF TaskBar

You can expand or collapse the TaskBar control similar to the Windows Taskbar. This is done by using the [IsOpened](#) property. This is an attached property, which is used to expand or collapse the TaskBarItems by using the [SetIsOpened](#) method.

This property is used in both TaskBar and TaskBarItems.

Use the following code snippet to expand or collapse the TaskBar.

XML

```
<!-- Adding TaskBar that have collapsed TaskBarItem -->
<syncfusion:TaskBar Name="taskBar" GroupMargin="5"
syncfusion:TaskBar.IsOpened="False">
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
<!-- Adding content to TaskBarItem -->
<StackPanel Margin="10" HorizontalAlignment="Center"

    VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap">This taskbar provides an UI similar to that
of Windows XP.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem2" Header="TaskBarItem2">
<!-- Adding content to TaskBarItem -->
<StackPanel Margin="10" HorizontalAlignment="Center"

    VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap">Specify and customize the group
margin.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
</syncfusion:TaskBar>
```

C#

```
// To Collapse the TaskBarItem in TaskBar.
TaskBar.SetIsOpened(taskBar, false);
```



Expander Button Size

Setting Group Margin in WPF TaskBar

You can set the margin for the items in the TaskBar control by using the [GroupMargin](#) property. This is a dependency property, which sets the margin for all TaskBar Items in the TaskBar control. You can enhance the appearance of the TaskBar Items by using this property.

You can also set the value for Left, Right, Bottom and Top group margins.

Use the below code for setting the group margin.

XML

```
<!-- Adding TaskBar that have group margin is 5 -->
<syncfusion:TaskBar Name="taskBar" GroupMargin="5">
  <!-- Adding TaskBarItem -->
  <syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
    <!-- Adding content to TaskBarItem -->
    <StackPanel Margin="10" HorizontalAlignment="Center"
      VerticalAlignment="Stretch">
      <TextBlock TextWrapping="Wrap">
        This taskbar provides an UI similar to that of Windows XP.</TextBlock>
      </StackPanel>
    </syncfusion:TaskBarItem>
    <!-- Adding TaskBarItem -->
    <syncfusion:TaskBarItem Name="taskBarItem2" Header="TaskBarItem2">
      <!-- Adding content to TaskBarItem -->
      <StackPanel Margin="10" HorizontalAlignment="Center"
        VerticalAlignment="Stretch">
        <TextBlock TextWrapping="Wrap">
          Specify and customize the group margin.</TextBlock>
        </StackPanel>
      </syncfusion:TaskBarItem>
    </syncfusion:TaskBar>
```

C#

```
//Creating an instance for TaskBar
TaskBar taskBar = new TaskBar();
//Creating an instance for TaskBarItem
TaskBarItem taskBarItem1 = new TaskBarItem();
//Setting the header of TaskBarItem1
taskBarItem1.Header = "TaskBarItem1";
// Creating instance for TextBlock
TextBlock textBlock1 = new TextBlock();
```

```
// Adding text to textblock
textBlock1.Text = "This taskbar provides an UI similar to that of Windows
XP.";
// Adding textblock to TaskBarItem
taskBarItem1.Items.Add(textBlock1);
//Creating an instance for TaskBarItem
TaskBarItem taskBarItem2 = new TaskBarItem();
//Setting the header of TaskBarItem1
taskBarItem2.Header = "TaskBarItem2";
// Creating instance for TextBlock
TextBlock textBlock2 = new TextBlock();
// Adding text to textblock
textBlock2.Text = "Specify and customize the group margin.";
// Adding textblock to TaskBarItem
taskBarItem2.Items.Add(textBlock2);
//Adding the TaskBarItem to TaskBar
taskBar.Items.Add(taskBarItem1);
taskBar.Items.Add(taskBarItem2);
//Setting Group Margin
taskBar.GroupMargin = new Thickness(5);
//Adding TaskBar as content of window
this.Content = taskBar;
```



Group Width for the TaskBar

Setting Group Width in WPF TaskBar

The width of the TaskBar is customized by using the [GroupWidth](#) property. This is a dependency property, which sets the group width of the TaskBar. It displays all the TaskBar Items in the TaskBar control with the same width.

The following code snippet illustrates how to set the group width.

XML

```
<!-- Adding TaskBar that have group width as 150 -->
<syncfusion:TaskBar Name="taskBar" GroupMargin="5" GroupWidth="150">
  <!-- Adding TaskBarItem -->
  <syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
    <!-- Adding content to TaskBarItem-->
    <StackPanel Margin="10" HorizontalAlignment="Center"
      VerticalAlignment="Stretch">
      <TextBlock TextWrapping="Wrap">This taskbar provides an UI similar to that
of Windows XP.</TextBlock>
    </StackPanel>
  </syncfusion:TaskBarItem>
</syncfusion:TaskBar>
```

```

</syncfusion:TaskBarItem>
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem2" Header="TaskBarItem2">
<!-- Adding content to TaskBarItem-->
<StackPanel Margin="10" HorizontalAlignment="Center"

        VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap">Specify and customize the group
margin.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
</syncfusion:TaskBar>

```

C#

```

//Creating an instance for TaskBar
TaskBar taskBar = new TaskBar();
//Creating an instance for TaskBarItem
TaskBarItem taskBarItem1 = new TaskBarItem();
//Setting the header of TaskBarItem1
taskBarItem1.Header = "TaskBarItem1";
// Creating instance for TextBlock
TextBlock textBlock1 = new TextBlock();
// Adding text to textblock
textBlock1.Text = "This taskbar provides an UI similar to that of Windows
XP.";
// Adding textblock to TaskBarItem
taskBarItem1.Items.Add(textBlock1);
//Creating an instance for TaskBarItem
TaskBarItem taskBarItem2 = new TaskBarItem();
//Setting the header of TaskBarItem1
taskBarItem2.Header = "TaskBarItem2";
// Creating instance for TextBlock
TextBlock textBlock2 = new TextBlock();
// Adding text to textblock
textBlock2.Text = "Specify and customize the group margin.";
// Adding textblock to TaskBarItem
taskBarItem2.Items.Add(textBlock2);
//Adding the TaskBarItem to TaskBar
taskBar.Items.Add(taskBarItem1);
taskBar.Items.Add(taskBarItem2);
//Setting group orientation to TaskBar
taskBar.GroupWidth = 150;
//Adding TaskBar as content of window
this.Content = taskBar;

```



Group Margin for the TaskBar

Changing Orientation of Taskbar in WPF TaskBar

TaskBar control is placed horizontally or vertically by using the [GroupOrientation](#) property. This is a dependency property, which sets the orientation of the TaskBar.

TaskBar control supports the following types of orientation.

- Horizontal–TaskBar Items in the TaskBar are placed horizontally
- Vertical–TaskBar Items in the TaskBar are placed vertically

Use the below code snippet to set the orientation.

XML

```
<!-- Adding TaskBar that have group orientation as horizontal -->
<syncfusion:TaskBar Name="taskBar" GroupMargin="5"
GroupOrientation="Horizontal">
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
<!-- Adding content to TaskBarItem -->
<StackPanel Margin="10" HorizontalAlignment="Center"

        VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap">
This taskbar provides an UI similar to that of Windows XP.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
<!-- Adding TaskBarItem -->
<syncfusion:TaskBarItem Name="taskBarItem2" Header="TaskBarItem2">
<!-- Adding content to TaskBarItem -->
<StackPanel Margin="10" HorizontalAlignment="Center"

        VerticalAlignment="Stretch">
<TextBlock TextWrapping="Wrap">
Specify and customize the group margin.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
</syncfusion:TaskBar>
```

C#

```
//Creating an instance for TaskBar
TaskBar taskBar = new TaskBar();
//Creating an instance for TaskBarItem
TaskBarItem taskBarItem1 = new TaskBarItem();
//Setting the header of TaskBarItem1
taskBarItem1.Header = "TaskBarItem1";
// Creating instance for TextBlock
TextBlock textBlock1 = new TextBlock();
// Adding text to textblock
```

```

textBlock1.Text = "This taskbar provides an UI similar to that of Windows
XP.";
// Adding textblock to TaskBarItem
taskBarItem1.Items.Add(textBlock1);
//Creating an instance for TaskBarItem
TaskBarItem taskBarItem2 = new TaskBarItem();
//Setting the header of TaskBarItem1
taskBarItem2.Header = "TaskBarItem2";
// Creating instance for TextBlock
TextBlock textBlock2 = new TextBlock();
// Adding text to textblock
textBlock2.Text = "Specify and customize the group margin.";
// Adding textblock to TaskBarItem
taskBarItem2.Items.Add(textBlock2);
//Adding the TaskBarItem to TaskBar
taskBar.Items.Add(taskBarItem1);
taskBar.Items.Add(taskBarItem2);
//setting group orientation to TaskBar
taskBar.GroupOrientation = Orientation.Horizontal;
//Adding TaskBar as content of window
this.Content = taskBar;

```



GroupOrientationChanged Event

This [GroupOrientationChanged](#) event is handled when the [GroupOrientation](#) property is changed.

While setting the orientation of the TaskBar, you may want to change the position of the TaskBar for proper alignment. This can be achieved by changing the group margin of the TaskBar along with the orientation.

The following code snippet illustrates handling the GroupOrientationChanged event.

C#

```

/// <summary>
/// Tasks the bar_ group orientation changed.
/// </summary>
/// <param name="d">The d.</param>
/// <param name="e">The <see
cref="System.Windows.DependencyPropertyChangedEventArgs"/> instance
    containing the event data.</param>
private void taskBar_GroupOrientationChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    if (taskBar.GroupOrientation == Orientation.Horizontal)
        taskBar.GroupMargin = new Thickness(5, 0, 0, 0);
    else
        taskBar.GroupMargin = new Thickness(0, 0, 0, 5);
}

```

Changing the flow directions in WPF TaskBar

The flow direction for the TaskBar is set through the [FlowDirection](#) property.

Here is the code for setting this property.

XML

```
<!-- Adding TaskBar -->
<syncfusion:TaskBar Name="taskBar" FlowDirection="RightToLeft">
  <!-- Adding TaskBarItem -->
  <syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
    <!-- Adding content to TaskBarItem -->
    <StackPanel Margin="10" HorizontalAlignment="Center"

      VerticalAlignment="Stretch">
        <TextBlock TextWrapping="Wrap">
          This is TaskBar that have a TaskBarItem</TextBlock>
        </StackPanel>
      </syncfusion:TaskBarItem>
    </syncfusion:TaskBar>
```

C#

```
// Setting flow direction as right to left
taskBar.FlowDirection = FlowDirection.RightToLeft;
```



FlowDirection = "RightToLeft"

Setting the Animation Speed in WPF TaskBar

You can set the animation speed, which controls the time taken to expand or collapse the taskbar items in the TaskBar, using the [Speed](#) property. This is an attached property, which controls the time taken to expand or collapse the TaskBarItems in the TaskBar by using the SetSpeed method.

Use the following code snippet to set this property.

XML

```
<!-- Adding TaskBar that have animation speed as 10 -->
<syncfusion:TaskBar Name="taskBar" GroupMargin="5"
  syncfusion:TaskBar.Speed="10">
  <!-- Adding TaskBarItem -->
  <syncfusion:TaskBarItem Name="taskBarItem1" Header="TaskBarItem1">
    <!-- Adding content to TaskBarItem -->
    <StackPanel Margin="10" HorizontalAlignment="Center"

      VerticalAlignment="Stretch">
        <TextBlock TextWrapping="Wrap">This taskbar provides an UI similar to that
          of Windows XP.</TextBlock>
        </StackPanel>
      </syncfusion:TaskBarItem>
    <!-- Adding TaskBarItem -->
    <syncfusion:TaskBarItem Name="taskBarItem2" Header="TaskBarItem2">
      <!-- Adding content to TaskBarItem -->
```

```
<StackPanel Margin="10" HorizontalAlignment="Center"
    VerticalAlignment="Stretch">
    <TextBlock TextWrapping="Wrap">Specify and customize the group
    margin.</TextBlock>
</StackPanel>
</syncfusion:TaskBarItem>
</syncfusion:TaskBar>
```

C#

```
// Setting the speed
TaskBar.SetSpeed(taskBar, 10);
```

Methods handled with Speed Property

GetSpeed

This [GetSpeed](#) method is used to get the animation speed, which controls the time taken to expand or collapse the TaskBarItems in the TaskBar. It has one argument, which returns the speed value of type, double.

C#

```
double speed;
//Getting speed
speed = TaskBar.GetSpeed(taskBar);
```

SetSpeed

This [SetSpeed](#) method is used to set the animation speed, which controls the time taken to expand or collapse the TaskBarItems in the TaskBar. It has two arguments. The first argument specifies the object, while the second argument specifies the value of speed of type double.

C#

```
//Setting the speed
TaskBar.SetSpeed(taskBar, 10);
```

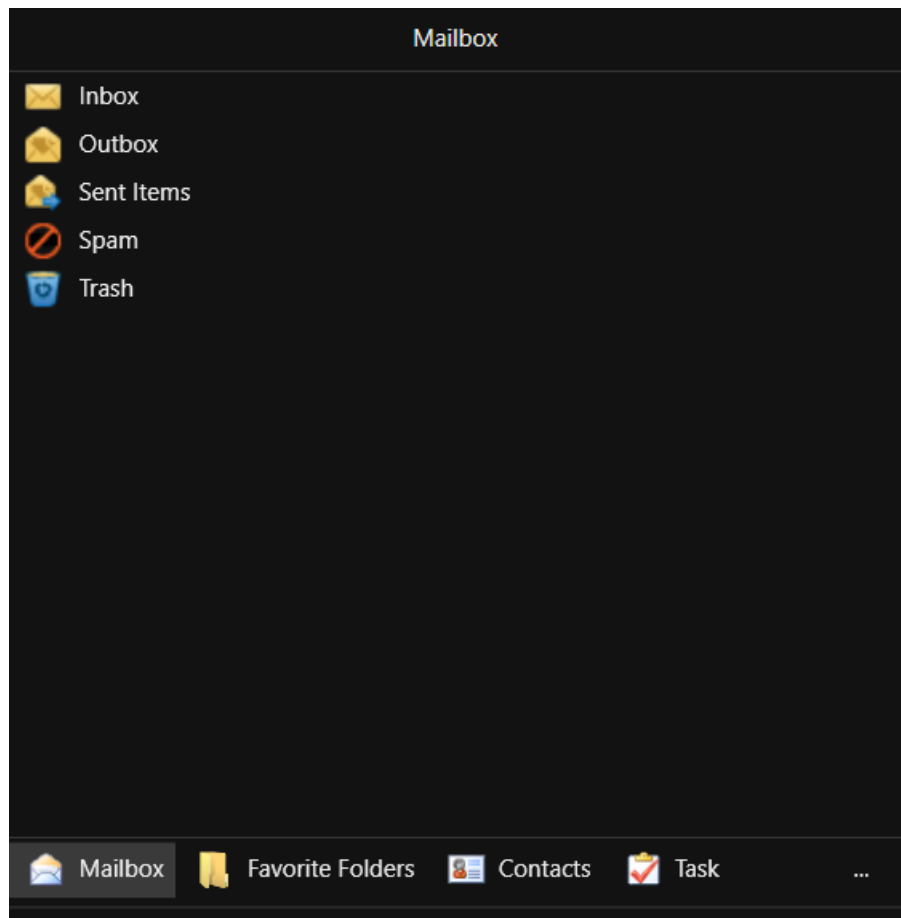
Appearance in WPF TaskBar

This section deals with the appearance of TaskBar control and contains the following topic:

Theme

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

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



SfTextBoxExt

WPF Autocomplete (SfTextBoxExt) Overview

Description

[SfTextBoxExt](#) is an extended version of the [TextBox](#) control. Most of the functionality of [SfTextBoxExt](#) is inherited from the [TextBox](#) class.

Key features

- **AutoComplete modes:** The control provides the suggestions based on a predefined collection while typing.
- **Watermark:** The control will prompt some information when it is not in focus and contains an empty string.
- **Filtering suggestion items:** Suggestions can be filtered in 18 different modes like StartsWith, EndWith, Contains, Equals, and Custom. AutoComplete provides both the case-sensitive and insensitive modes.
- **Popup delay:** The filtering process can be delayed. Displaying filtered suggestions from a drop-down list may also be delayed for a period of time.
- **Minimum prefix characters:** Instead of displaying suggestion list on every character entry, matches can be filtered and displayed after a few character entries.
- **Customization support:** AutoComplete provides the options to customize both text box and drop-down list.

- **Diacritic sense:** The control provides the populating the items from a language with letters containing diacritics, and search for them with English characters from an en-US keyboard.
- **Highlighting Text:** The control provides the highlighting the matching text in the suggestion list based on the input given in it.
- **Custom filter:** AutoComplete provides filter the items in the suggestion list based on their filtering condition.
- **Multi selection support:** Selecting multiple items from a suggestion list.
- **Suggestion box adjustment:** Adjust the position of popup relative to the control.
- **Enable auto size:** Auto sizing can be enabled in AutoComplete control so that the control will extend its layout based on the tokens size in wrap mode.
- **AutoComplete template:** It can be used for own template.
- **NoResults found template:** Set the desire text, if the typing item does not exists in the collection.
- **Retrieving the selection:** Retrieve the selected item, index, and value.

Getting Started with WPF Autocomplete (SfTextBoxExt)

Assembly deployment

Refer to this [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 [How to install nuget packages](#) documentation to find more details about installing the NuGet package in a WPF application.

Creating a simple application

Create a WPF application with [SfTextBoxExt](#) control using the following steps:

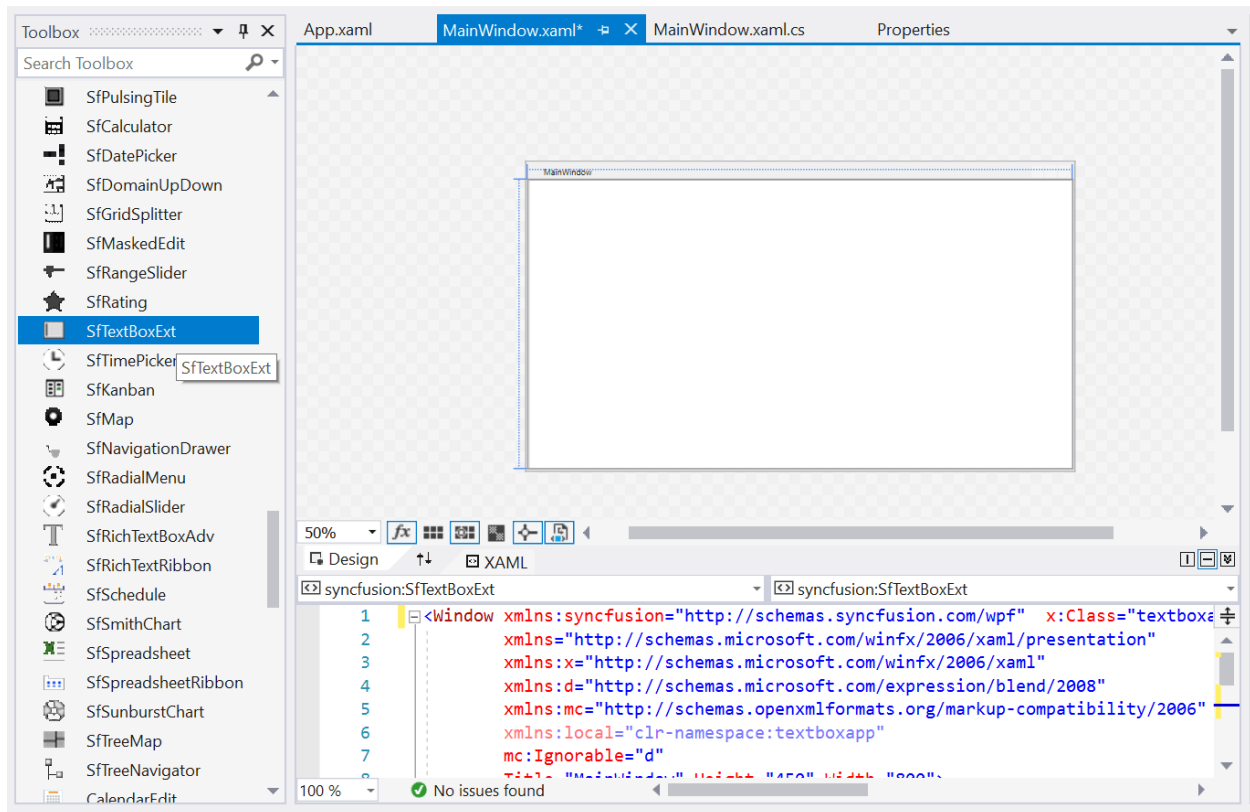
Create a project

Create a new WPF project in Visual Studio to display the [SfTextBoxExt](#) control with their functionalities.

Add a control using the designer

The [SfTextBoxExt](#) control can be added to an application by dragging it from the toolbox to a designer view. The following required assembly references will be added automatically:

- Syncfusion.SfInput.WPF
- Syncfusion.SfShared.WPF



Adding control manually in XAML

To add the control manually in XAML, follow the given steps:

1. Add the following required assembly references to the project:
 - Syncfusion.SfInput.WPF
 - Syncfusion.SfShared.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the [SfTextBoxExt](#) control in the XAML page.

XAML

```

<Window x:Class="TextBoxExt.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:TextBoxExt"
mc:Ignorable="d"
xmlns:editors="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="450" Width="800">
  <editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
HorizontalContentAlignment="Center"
Height="40"
Width="200"
Text="Hello! World..." />
</Window>

```

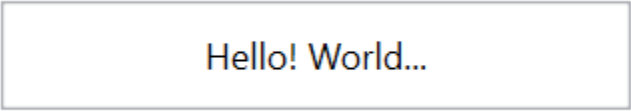
Add a control manually in C#

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 [SfTextBoxExt](#) namespace using **Syncfusion.Windows.Controls.Input**;
3. Create an [SfTextBoxExt](#) instance, and add it to the window.

C#

```
using System.Windows;
using Syncfusion.Windows.Controls.Input;
namespace TextBoxExt
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfTextBoxExt textBoxExt = new SfTextBoxExt();
            textBoxExt.HorizontalContentAlignment = HorizontalAlignment.Center;
            textBoxExt.HorizontalAlignment = HorizontalAlignment.Center;
            textBoxExt.VerticalAlignment = VerticalAlignment.Center;
            textBoxExt.Width = 200;
            textBoxExt.Height = 40;
            textBoxExt.Text = "Hello! World...";
            this.Content = textBoxExt;
        }
    }
}
```



Populating AutoComplete with Data

AutoComplete is a data-bound control. So before create binding to the control, you must create data model for Application.

For illustration, let us create a textbox, which will populate a list of employees.

1. Create data object class named **Employee** and declare properties as shown below,

C#

```
public class Employee
{
    string name;
    string email;
    public string Name
```

```
{
get { return name; }
set { name = value; }
}
public string Email
{
get { return email; }
set { email = value; }
}
}
```

2. Create a **EmployeeViewModel** class with Employees property and Employees property is initialized with several data objects in constructor.

C#

```
public class EmployeeViewModel
{
private List<Employee> employees;
public List<Employee> Employees
{
get { return employees; }
set { employees = value; }
}
public EmployeeViewModel()
{
Employees = new List<Employee>();
Employees.Add(new Employee() { Name = "Eric", Email = "Eric@syncfusion.com"
});
Employees.Add(new Employee() { Name = "James", Email =
"James@syncfusion.com" });
Employees.Add(new Employee() { Name = "Jacob", Email =
"Jacob@syncfusion.com" });
Employees.Add(new Employee() { Name = "Lucas", Email =
"Lucas@syncfusion.com" });
Employees.Add(new Employee() { Name = "Mark", Email = "Mark@syncfusion.com"
});
Employees.Add(new Employee() { Name = "Aldan", Email =
"Aldan@syncfusion.com" });
Employees.Add(new Employee() { Name = "Aldrin", Email =
"Aldrin@syncfusion.com" });
Employees.Add(new Employee() { Name = "Alan", Email = "Alan@syncfusion.com"
});
Employees.Add(new Employee() { Name = "Aaron", Email =
"Aaron@syncfusion.com" });
}
}
```

3. To populate the AutoComplete with data, set the [AutoCompleteSource](#) property to **IEnumerable** implementation.

Bind the collection created in previous step to [AutoCompleteSource](#) property in XAML by setting EmployeeViewModel as **DataContext**.

XML

```

<Window x:Class="TextBoxExt.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:TextBoxExt"
mc:Ignorable="d"
xmlns:editors="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:EmployeeViewModel/>
</Window.DataContext>
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}" />
</Window>

```

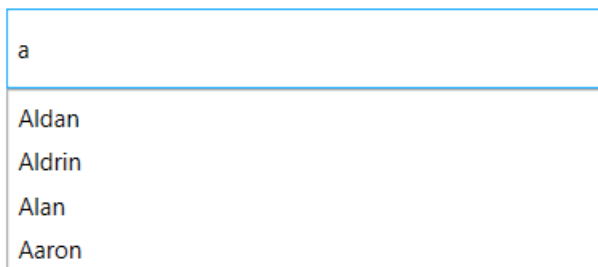
C#

```

EmployeeViewModel viewModel = new EmployeeViewModel();
this.DataContext = viewModel;
SfTextBoxExt textBoxExt = new SfTextBoxExt();
textBoxExt.HorizontalAlignment = HorizontalAlignment.Center;
textBoxExt.VerticalAlignment = VerticalAlignment.Center;
textBoxExt.Width = 200;
textBoxExt.Height = 40;
textBoxExt.SearchItemPath = "Name";
textBoxExt.AutoCompleteMode = AutoCompleteMode.Suggest;
textBoxExt.AutoCompleteSource = viewModel.Employees;
this.Content = textBoxExt;

```

For further details, refer to [AutoComplete source](#).



AutoComplete modes

Suggestions can be shown in number of ways. [SfTextBoxExt](#) supports the following.

AutoCompleteMode	Description
Suggest	Shows the suggestion in the drop-down list user.

Append	Appends the first suggestion to the text.
SuggestAppend	Shows the suggestion in the drop-down list and appends the first suggestion to the text.
None	In None mode, the search algorithm starts even when the item is not available in the data source.

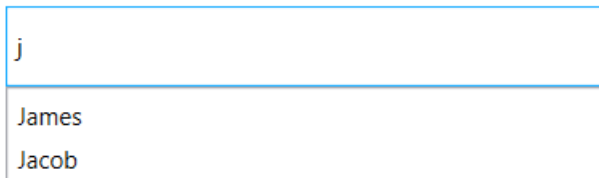
Note: The default value of [AutoCompleteMode](#) is None. So, running the control without specifying this property will not show any suggestions.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
SuggestionMode="StartsWith"
AutoCompleteSource="{Binding Employees}" />
```

C#

```
textBoxExt.AutoCompleteMode = AutoCompleteMode.Suggest;
```



Selection

By default single selection is enable in AutoComplete control. It can set the [MultiSelectMode](#) property to specify whether a single or multiple selection.

Index of the selected items can be retrieved using the [SuggestionIndex](#) property.

The selected items of the AutoComplete can be retrieved using the [SelectedItem](#) property for single selection.

In Multi-selection, [SelectedItems](#) property contains the items that are selected in the control.

The selected values of the AutoComplete can be retrieved using the [SelectedValue](#) property.

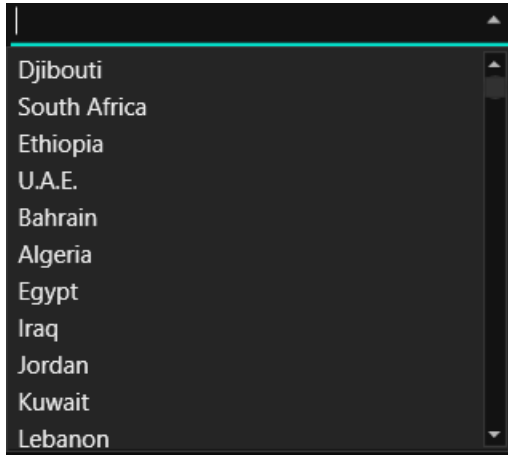
For further details, refer to the [Retrieving SelectedValue](#) and [Setting and retrieving SelectedItem](#).

Note: View [sample](#) in GitHub

Theme

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

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



Autocomplete and filtering in WPF Autocomplete (SfTextBoxExt)

The AutoComplete functionality provides several modes of suggestions while typing. The suggested text can be appended to the original text, or can be displayed in a drop-down list so that searched item can be chosen based on the filtering option set.

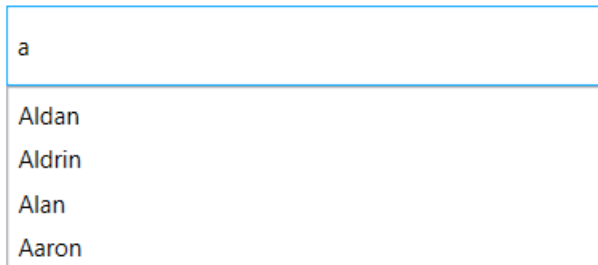
AutoComplete source

The [SfTextBoxExt](#) control can be populated with a predefined list of items bind to the [AutoCompleteSource](#) property. The data can be either a list of strings or a custom data.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}" />
```

For further details, refer to [Populating Autocomplete with Data](#).



Custom data

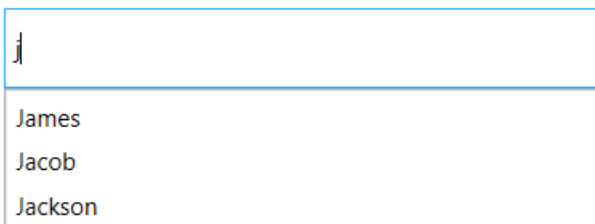
The [SearchItemPath](#) property specifies the property path, by which the filtering has to be done when a custom data is bound to the [AutoCompleteSource](#) property. This property defines the value to be displayed in the drop-down suggestion box.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}" />
```

C#

```
textBoxExt.SearchItemPath = "Name";
```

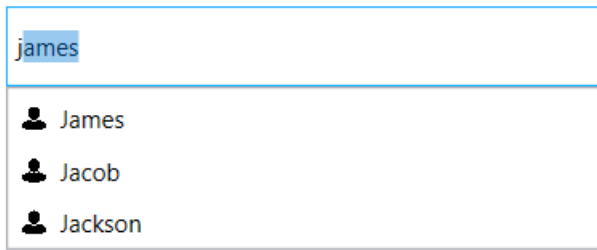


Customize using the ItemTemplate

By default the drop-down window lists the filtered items as a text based on the [SearchItemPath](#) property set for the data. The [AutoCompleteItemTemplate](#) property helps to decorate the filtered items with visual elements. The following code block explains how to add an image to the drop-down list items.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="400"
Height="40"
SearchItemPath="Name"
AutoCompleteMode="SuggestAppend"
AutoCompleteSource="{Binding Employees}" >
  <editors:SfTextBoxExt.AutoCompleteItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <Image Source="User.png" Margin="2" Stretch="Uniform" Width="12"/>
        <TextBlock Text="{Binding Name}" Margin="5 2"/>
      </StackPanel>
    </DataTemplate>
  </editors:SfTextBoxExt.AutoCompleteItemTemplate>
</editors:SfTextBoxExt>
```



Filtering options

The phenomenon of string comparison for filtering suggestions can be changed using the [SuggestionMode](#) property. The default filtering strategy is “StartsWith” and it is case-insensitive.

SuggestionMode	Description
None	The control returns the entire collection without filtering.
StartsWith	Displays all matches that begins with the typed characters in the text field. This strategy is case-insensitive.
StartsWithCaseSensitive	Displays all matches that begins with the typed characters in the text field. This strategy is case-sensitive.
StartsWithOrdinal	The control returns all possible matches that begins with the typed text based on the OrdinalIgnoreCase.
StartsWithOrdinalCaseSensitive	The control returns all possible matches that begins with the typed text based on the Ordinal, which is case-sensitive.
Contains	Displays all matches that contains typed characters in the text field. This strategy is case-insensitive.
ContainsCaseSensitive	The control returns all possible matches that contains the typed text, which is culture and case-sensitive.
ContainsOrdinal	The control returns all possible matches that contains the typed text based on the OrdinalIgnoreCase.
ContainsOrdinalCaseSensitive	The control returns all possible matches that contains the typed text based on the Ordinal, which is case-sensitive.
Equals	Displays all words that completely matches the typed characters in the text field. This strategy is case-insensitive.
EqualsCaseSensitive	Displays all words that completely matches the typed characters in the text field. This strategy is case-sensitive.
EqualsOrdinal	The control returns all possible matches that equals the typed text based on the OrdinalIgnoreCase.
EqualsOrdinalCaseSensitive	The control returns all possible matches that equals the typed text based on the Ordinal, which is case-sensitive.

Custom	The control returns all possible matches based on the Filter property. Filter property is of type SuggestionPredicate. In the MyFilter method, filtration is done by checking whether the collection contains the typed text.
EndsWith	Displays all matches that ends with the typed characters in the text field. This strategy is case-insensitive.
EndsWithCaseSensitive	Displays all matches that ends with the typed characters in the text field. This strategy is case-sensitive.
EndsWithOrdinal	The control returns all possible matches ending with the typed text based on the OrdinalIgnoreCase.
EndsWithOrdinalCaseSensitive	The control returns all possible matches ending with the typed text based on the Ordinal, which is case-sensitive.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="400"
Height="40"
SearchItemPath="Name"
SuggestionMode="Contains"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}"/>
```

C#

```
textBoxExt.SuggestionMode = SuggestionMode.Contains;
```

e

Eric

James

Custom

Filter items in the suggestion list based on the users' custom search. This will help you to apply our typo toleration functionality to the control.

era

Albania

Algeria

American Samoa

Andorra

Angola

Anguilla

XML

```

<Window x:Class="AutoCompleteWPF.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:AutoCompleteWPF"
xmlns:editors="http://schemas.syncfusion.com/wpf"
xmlns:ListCollection="http://schemas.microsoft.com/netfx/2009/xaml/presentation"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<StackPanel>
<editors:SfTextBoxExt x:Name="autoComplete"
Width="300"
Height="40"
HorizontalAlignment="Center"
VerticalAlignment="Center"
AutoCompleteMode="Suggest"
SuggestionMode="Custom">
<editors:SfTextBoxExt.AutoCompleteSource>
<x:Array Type="{x:Type sys:String}">
<sys:String>Albania</sys:String>
<sys:String>Algeria</sys:String>
<sys:String>American Samoa</sys:String>
<sys:String>Andorra</sys:String>
<sys:String>Angola</sys:String>
<sys:String>Anguilla</sys:String>
</x:Array>
</editors:SfTextBoxExt.AutoCompleteSource>
</editors:SfTextBoxExt>
</StackPanel>
</Window>

```

C#

```

using System;
using System.Windows;
namespace AutoCompleteWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            autoComplete.Filter = ContainingSpaceFilter;
        }
        public bool ContainingSpaceFilter(string search, object item)
        {
            string text = item.ToString().ToLower();
            if (item != null)
            {

```

```
try
{
    var split = search.Split(' ');
    foreach (var results in split)
    {
        if (!text.Contains(results.ToLower()))
        {
            return true;
        }
        else
            return false;
    }
    return true;
}
catch (Exception)
{
    return (text.Contains(search));
}
else
    return false;
}
```

Prefix characters constraint

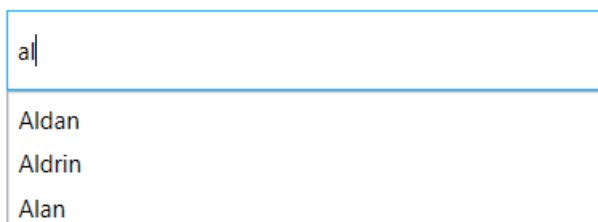
Instead of displaying suggestion list on every character entry, matches can be filtered and displayed after a few character entries. This can be done using the [MinimumPrefixCharacter](#) property. By default the constraint is set for each character entry.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
MinimumPrefixCharacters="2"
AutoCompleteSource="{Binding Employees}" />
```

C#

```
textBoxExt.MinimumPrefixCharacters = 2;
```



a
Aldan
Aldrin
Alan

Working with case sensitivity

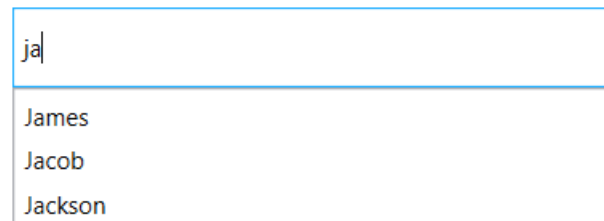
[IgnoreCase](#) option allows the control to filter the suggestions by ignoring the case. The default value is false.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="400"
Height="40"
SearchItemPath="Name"
IgnoreCase="True"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}"/>
```

C#

```
textBoxExt.IgnoreCase = true;
```



Showing image in token and drop-down

To display image in token use the [ImageMemberPath](#) property.

Note: This feature is applicable only for MultiSelectMode with Token mode.

To display image for each drop-down item a custom template can be assigned to [AutoCompleteItemTemplate](#) support.

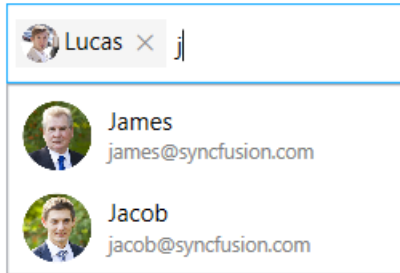
XML

```
<editors:SfTextBoxExt HorizontalAlignment="Left"
AutoCompleteMode="Suggest"
SearchItemPath="Name"
ImageMemberPath="Image"
MultiSelectMode="Token"
TokensWrapMode="None"
Height="40"
AutoCompleteSource="{Binding Employees}"
VerticalAlignment="Center"
Width="200">
  <editors:SfTextBoxExt.AutoCompleteItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal" Height="40">
        <Image Source="{Binding Image}" Margin="2" Height="35" Stretch="Uniform"
Width="35"/>
        <StackPanel Margin="2"
Orientation="Vertical">
```

```

<TextBlock Text="{Binding Name}" Margin="4,2,4,0" FontSize="12"
Foreground="Black"/>
<TextBlock Text="{Binding Email}" Margin="4,1,2,2" FontSize="10"
Foreground="Gray"/>
</StackPanel>
</StackPanel>
</DataTemplate>
</editors:SfTextBoxExt.AutoCompleteItemTemplate>
</editors:SfTextBoxExt>

```



Note: View [sample](#) in GitHub

Display a message when suggestions are empty

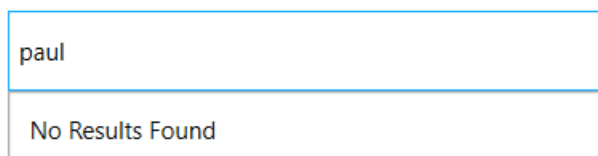
When the entered item is not in the suggestion list, AutoComplete displays a text indicating that there is no search results found. The text to be displayed for this can be customized using the [NoResultsFoundTemplate](#) property.

XML

```

<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Height="40"
Width="400"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}">
<editors:SfTextBoxExt.NoResultsFoundTemplate>
<DataTemplate>
<Label Content="No Results Found" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</DataTemplate>
</editors:SfTextBoxExt.NoResultsFoundTemplate>
</editors:SfTextBoxExt>

```



Restricting the maximum items filtered

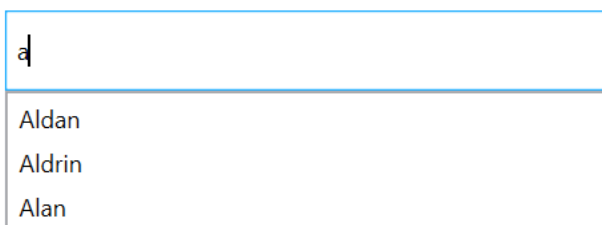
The [MaximumSuggestionsCount](#) property is used to restrict the number of suggestions displayed in the suggestion box.

XML

```
<editors:SfTextBoxExt
Width="300"
Height="40"
HorizontalAlignment="Center"
VerticalAlignment="Center"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}"
MaximumSuggestionsCount="3"
SearchItemPath="Name" />
```

C#

```
textBoxExt.MaximumSuggestionsCount = 3;
```



Note: View [sample](#) in GitHub

Single and multiple selection in WPF Autocomplete (SfTextBoxExt)

In AutoComplete selection can be performed using single selection or multi-selection. This can be handled by [MultiSelectMode](#) property. The default value of `MultiSelectMode` is `None` which performs single selection.

Single selection

The single selection can be performed by setting the [MultiSelectMode](#) property to `None`. In this mode we can set and retrieve the selected item using the [SelectedItem](#) property.

Multi selection

The multi-selection, there are two ways to display the selection in the control.

- Token
- Delimiter

Multiple selection using tokens

Each selected items can be displayed as a token representation having a close button for each token.

In token representation the control behavior of arranging the items can be done in two ways which is handled by the property [TokensWrapMode](#).

- **Wrap** - The selected items will be wrapped to the next line of the AutoComplete.
- **None** - The selected items will be arranged in horizontal layout in single line.

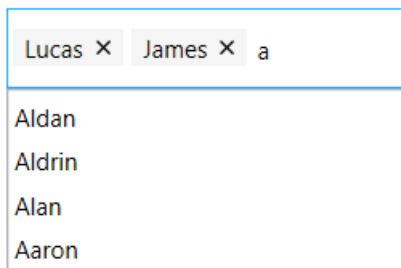
The [SelectedItems](#) property can be used to set and retrieve the items in **MultiSelectMode**.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Left"
AutoCompleteMode="Suggest"
SearchItemPath="Name"
MultiSelectMode="Token"
Height="40"
AutoCompleteSource="{Binding Employees}"
VerticalAlignment="Center"
Width="200"/>
```

C#

```
textBoxExt.MultiSelectMode = MultiSelectMode.Token;
```



Customization of Tokens

The token can be customized by overriding the default style targeting the **TokenItem** class.

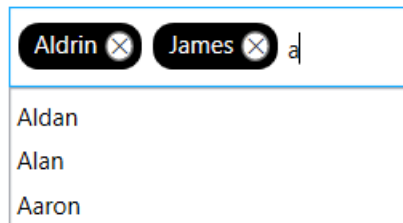
XML

```
<Window.Resources>
<Style TargetType="editors:TokenItem">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="editors:TokenItem">
<Border x:Name="TokenBorder" Margin="4,4,2,4" Background="Black"
CornerRadius="10" Height="{TemplateBinding Height}" >
<Grid Margin="2" x:Name="TokenGrid">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="auto"/>
<ColumnDefinition Width="*/>
<ColumnDefinition Width="auto"/>
</Grid.ColumnDefinitions>
<TextBlock x:Name="TokenTextBlock" Text="{TemplateBinding Text}"
Foreground="White" Grid.Column="1" Margin="2,0,2,3"
VerticalAlignment="Center" Padding="4,0,0,0" />
<Button Grid.Column="2" Margin="2" x:Name="TokenCloseButton"
IsTabStop="False" Background="White" CommandParameter="{Binding
RelativeSource={RelativeSource Mode=TemplatedParent}}" >
```

```

<Button.Content>
<TextBlock x:Name="TokenButtonContent" Text="&#xE711;"
VerticalAlignment="Center" FontSize="10" Foreground="Black"
FontFamily="Segoe MDL2 Assets" />
</Button.Content>
<Button.Resources>
<Style TargetType="{x:Type Border}">
<Setter Property="CornerRadius" Value="13"/>
</Style>
</Button.Resources>
</Button>
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
<editors:SfTextBoxExt HorizontalAlignment="Left"
AutoCompleteMode="Suggest"
SearchItemPath="Name"
MultiSelectMode="Token"
Height="40"
AutoCompleteSource="{Binding Employees}"
VerticalAlignment="Center"
Width="200"/>

```



Enable autosize in token mode

In token representation when we have set **Wrap** mode enabling the [EnableAutoSize](#) property re-renders the control height based on the number of lines the tokens are wrapped inside the control.

To use this feature, it is need to set the **MultiSelectMode** as **Token** and **TokensWrapMode** as **Wrap**. By default this feature is disabled.

XML

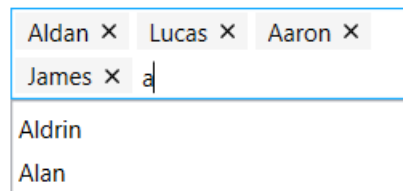
```

<editors:SfTextBoxExt HorizontalAlignment="Left"
AutoCompleteMode="Suggest"
SearchItemPath="Name"
MultiSelectMode="Token"
TokensWrapMode="Wrap"
EnableAutoSize="True"
AutoCompleteSource="{Binding Employees}"
VerticalAlignment="Center"
Width="200"/>

```

C#

```
textBoxExt.MultiSelectMode = MultiSelectMode.Token;  
textBoxExt.TokensWrapMode = TokensWrapMode.Wrap;  
textBoxExt.EnableAutoSize = true;
```



ShowClearButton:

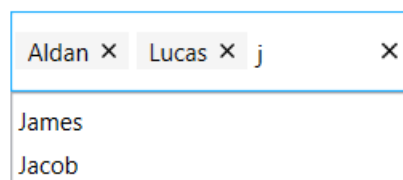
This feature allows to show or hide the clear button in Token mode for the AutoComplete control using the [ShowClearButton](#) property.

XML

```
<editors:SfTextBoxExt  
x:Name="textBoxExt"  
Width="200"  
HorizontalAlignment="Center"  
VerticalAlignment="Center"  
AutoCompleteMode="Suggest"  
AutoCompleteSource="{Binding Employees}"  
MultiSelectMode="Token"  
SearchItemPath="Name"  
ShowClearButton="True" />
```

C#

```
textBoxExt.ShowClearButton = true;
```



Note: The default `ShowClearButton` property value is false. It will be only applicable for `MultiSelectMode` is `Token`.

See also [Multiple selection using tokens](#) topic in AutoComplete.

Multiple selection using delimiter

In `Delimiter` mode, each item is separated by a character that is set to the [Delimiter](#) property. By default the items are separated by `,`(Comma).

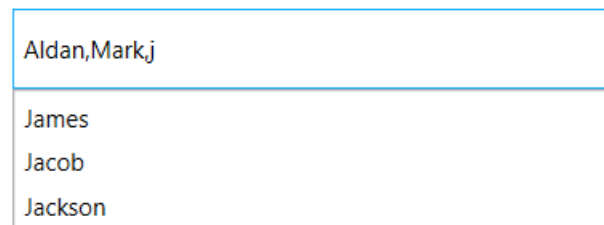
XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
```

```
VerticalAlignment="Center"  
Width="300"  
Height="40"  
MultiSelectMode="Delimiter"  
SearchItemPath="Name"  
AutoCompleteMode="Suggest"  
AutoCompleteSource="{Binding Employees}" />
```

C#

```
textBoxExt.MultiSelectMode = MultiSelectMode.Delimiter;
```



Setting and retrieving SelectedItem

The [SelectedItem](#) property is used to select a particular item from the suggestion list or to retrieve the item that is selected. The [SelectedItem](#) property can be used in single selection where the [MultiSelectMode](#) as [None](#). For multi-selection where the [MultiSelectMode](#) is set either as [Token](#) or [Delimiter](#) the selected items can be set or retrieved using the [SelectedItems](#) property.

The [SelectedItem](#) and [SelectedItems](#) contains the object of the custom data and using the [SearchItemPath](#) property the value displayed in the text field can be retrieved.

Model Class:

C#

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

In EmployeeViewModel class the [SelectedItem](#) updated initially from the Employees collection.

C#

```
public class EmployeeViewModel
{
    private List<Employee> employees;
    public List<Employee> Employees
    {
        get { return employees; }
        set { employees = value; }
    }
    private object selectedItem;
    public object SelectedItem
    {
        get { return selectedItem; }
        set { selectedItem = value; }
    }
    public EmployeeViewModel()
    {
        Employees = new List<Employee>();
        Employees.Add(new Employee() { Name = "Eric", Email = "Eric@syncfusion.com" });
        Employees.Add(new Employee() { Name = "James", Email = "James@syncfusion.com" });
        Employees.Add(new Employee() { Name = "Jacob", Email = "Jacob@syncfusion.com" });
        Employees.Add(new Employee() { Name = "Jackson", Email = "Jackson@syncfusion.com" });
        Employees.Add(new Employee() { Name = "Lucas", Email = "Lucas@syncfusion.com" });
        SelectedItem = Employees[0];
    }
}
```

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
x:Name="autoComplete"
MaxDropDownHeight="100"
MultiSelectMode="None"
AutoCompleteMode="Suggest"
SearchItemPath="Name"
SelectedItem="{Binding SelectedItem}"
AutoCompleteSource="{Binding Employees}"
VerticalAlignment="Center"
Height="40"
Width="200"/>
```



C#

```
private void TextBoxExt_SelectedItemChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    SfTextBoxExt textBoxExt = d as SfTextBoxExt;
```

```
string selectedItem = "";
if (textBoxExt.SelectedItem != null)
{
    selectedItem = ((textBoxExt.SelectedItem as Employee).Name).ToString();
}
MessageBox.Show(selectedItem, "SelectedItem", MessageBoxButton.OK,
    MessageBoxImage.None);
}
```

Model Class:

C#

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

In EmployeeViewModel class the [SelectedItems](#) updated initially from the Employees collection.

C#

```
public class EmployeeViewModel
{
    private List<Employee> employees;
    public List<Employee> Employees
    {
        get { return employees; }
        set { employees = value; }
    }
    private ObservableCollection<object> selectedItems;
    public ObservableCollection<object> SelectedItems
    {
        get { return selectedItems; }
        set { selectedItems = value; }
    }
    public EmployeeViewModel()
    {
        Employees = new List<Employee>();
        SelectedItems = new ObservableCollection<object>();
        Employees.Add(new Employee() { Name = "Eric", Email = "Eric@syncfusion.com" });
        Employees.Add(new Employee() { Name = "James", Email = "James@syncfusion.com" });
    }
}
```

```

Employees.Add(new Employee() { Name = "Jacob", Email =
"Jacob@syncfusion.com" });
Employees.Add(new Employee() { Name = "Jackson", Email =
"Jackson@syncfusion.com" });
Employees.Add(new Employee() { Name = "Lucas", Email =
"Lucas@syncfusion.com" });
SelectedItems.Add(Employees[2]);
SelectedItems.Add(Employees[0]);
SelectedItems.Add(Employees[1]);
}
}

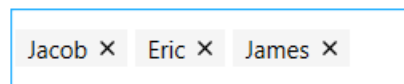
```

XML

```

<editors:SfTextBoxExt HorizontalAlignment="Center"
x:Name="autoComplete"
MaxDropDownHeight="100"
MultiSelectMode="Token"
AutoCompleteMode="Suggest"
SearchItemPath="Name"
SelectedItems="{Binding SelectedItems}"
AutoCompleteSource="{Binding Employees}"
VerticalAlignment="Center"
Height="40"
Width="200"/>

```



Retrieving SelectedValue

The [SelectedValue](#) property is used to retrieve the selected values from the suggestion list. We have to set the [ValueMemberPath](#) property when using custom data for the value we want to retrieve from [SelectedValue](#) based on the [SelectedItem](#) object.

XML

```

<editors:SfTextBoxExt HorizontalAlignment="Left"
x:Name="autoComplete"
MaxDropDownHeight="100"
MultiSelectMode="None"
AutoCompleteMode="Suggest"
SearchItemPath="Name"
ValueMemberPath="Email"
AutoCompleteSource="{Binding Employees}"
VerticalAlignment="Center"
SelectedItemChanged="AutoComplete_SelectedItemChanged"
Height="40"
Width="200"/>

```

C#

```

private void AutoComplete_SelectedItemChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)

```

```
{
    SfTextBoxExt textBoxExt = d as SfTextBoxExt;
    if (textBoxExt.SelectedValue != null)
    {
        MessageBox.Show(textBoxExt.SelectedValue.ToString(), "SelectedValue",
            MessageBoxButton.OK, MessageBoxImage.None);
    }
}
```

Retrieving SuggestionIndex

When an item is selected from suggestion list, their index can be retrieved using the [SuggestionIndex](#) property.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Left"
    x:Name="autoComplete"
    MaxDropDownHeight="100"
    MultiSelectMode="None"
    AutoCompleteMode="Suggest"
    SearchItemPath="Name"
    ValueMemberPath="Email"
    AutoCompleteSource="{Binding Employees}"
    VerticalAlignment="Center"
    SelectedItemChanged="AutoComplete_SelectedItemChanged"
    Height="40"
    Width="200"/>
```

C#

```
private void AutoComplete_SelectedItemChanged(DependencyObject d,
    DependencyPropertyChangedEventArgs e)
{
    string suggestionIndex = "";
    suggestionIndex = ((d as SfTextBoxExt).SuggestionIndex).ToString();
    MessageBoxResult messageBoxResult = MessageBox.Show(suggestionIndex,
        "SuggestionIndex");
}
```

Displaying images

This feature allows to provide the path for the image to be displayed in the text box control using the [ImageMemberPath](#) property.

Note: This feature is applicable only for MultiSelectMode with Token mode.

For further details, refer to [Showing image in token and dropdown](#).

Note: View [sample](#) in GitHub

Diacritic Sensitivity in WPF Autocomplete (SfTextBoxExt)

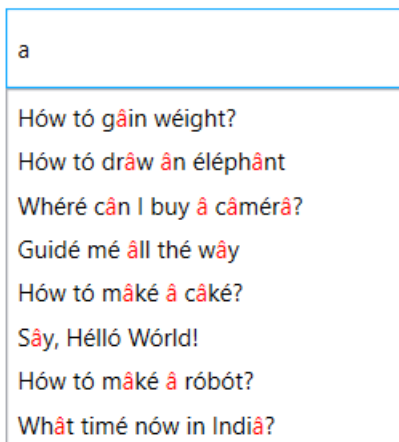
The control does not stick with one type of keyboard, so it can be populated with the items from a language with letters containing diacritics, and search for them with English characters from an en-US keyboard. Enable or disable the diacritic sensitivity using the [IgnoreDiacritic](#) property. The following code example demonstrates how to enable the diacritic sensitivity.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Height="40"
Width="200"
AutoCompleteMode="Suggest"
SuggestionMode="Contains"
IgnoreDiacritic="False"
HighlightedTextColor="Red"
TextHighlightMode="MultipleOccurrence"
SearchItemPath="Item"
AutoCompleteSource="{Binding DiacriticCollenction}"/>
```

C#

```
textBoxExt.IgnoreDiacritic = false;
```



Note: View [sample](#) in GitHub

Highlighting Matched Text in WPF Autocomplete (SfTextBoxExt)

By using the [TextHighlightMode](#) property, highlight matching and unmatched characters in a suggestion list to pick an item with more clarity. The default value is None. The matching text can be highlighted in the following two ways:

- First occurrence
- Multiple occurrence
- Unmatched

The text highlight can be indicated by customizing the color of the characters using [HighlightedTextColor](#) property and style of the characters using [HighlightedTextStyle](#) property.

Note: The [HighlightedTextStyle](#) property style target type is [Run](#).

First occurrence

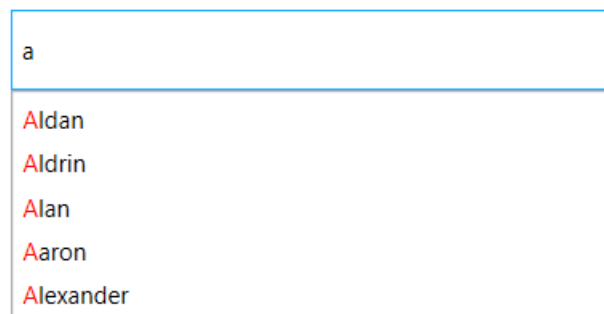
It highlights the first position of the matching characters in the suggestion list.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
HighlightedTextColor="Red"
TextHighlightMode="FirstOccurrence"
AutoCompleteSource="{Binding Employees}" />
```

C#

```
textBoxExt.TextHighlightMode = OccurrenceMode.FirstOccurrence;
```



Multiple occurrence

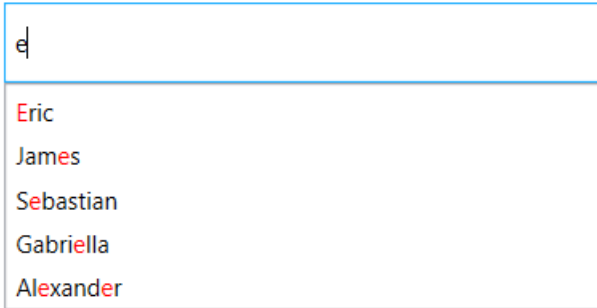
It highlights the matching character that presents everywhere in the suggestion list for "Contains" case in SuggestionMode.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
SuggestionMode="Contains"
HighlightedTextColor="Red"
TextHighlightMode="MultipleOccurrence"
AutoCompleteSource="{Binding Employees}" />
```

C#

```
textBoxExt.TextHighlightMode = OccurrenceMode.MultipleOccurrence;
```



Unmatched

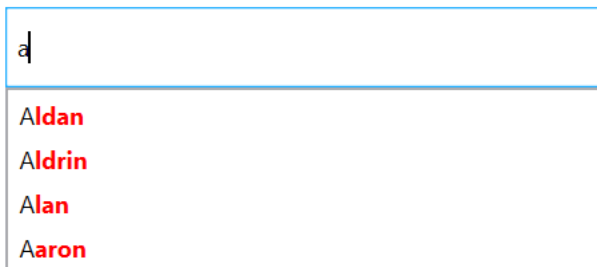
It highlights unmatched characters in the suggestion list.

XML

```
<Window.Resources>
<Style x:Key="highlightedTextStyle" TargetType="Run">
<Setter Property="FontWeight" Value="Bold" />
</Style>
</Window.Resources>
<editors:SfTextBoxExt
Width="300"
Height="40"
HorizontalAlignment="Center"
VerticalAlignment="Center"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}"
HighlightedTextColor="Red"
HighlightedTextStyle="{StaticResource highlightedTextStyle}"
SearchItemPath="Name"
SuggestionMode="StartsWith"
TextHighlightMode="Unmatched" />
```

C#

```
textBoxExt.TextHighlightMode = OccurrenceMode.Unmatched;
```



Note: View [sample](#) in GitHub

Textbox customization in WPF Autocomplete (SfTextBoxExt)

AutoComplete provides the user-friendly customizing options for text box. This section explains how to customize the entire AutoComplete control.

Water mark

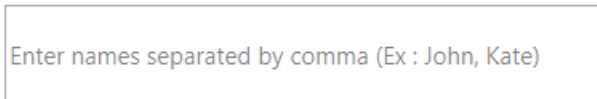
[Watermark](#) property, allows to specify some information when the text is empty. For illustration, let us create a simple textbox and indicate enter names separated by a comma.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="50"
Watermark="Enter names separated by comma (Ex : John, Kate)"/>
```

C#

```
textBoxExt.Watermark= "Enter names separated by comma (Ex : John, Kate)";
```



Note: The Watermark property is of the object type. So, any framework elements can be hosted as Watermark content. The following example shows how to host an image and text as Watermark content.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Height="50"
Width="300">
  <editors:SfTextBoxExt.Watermark>
    <StackPanel Orientation="Horizontal">
      <Image Source="Windows 8.png"
Margin="2"/>
      <TextBlock Text="Search Windows"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Opacity="0.5"/>
    </StackPanel>
  </editors:SfTextBoxExt.Watermark>
</editors:SfTextBoxExt>
```




Water mark template

Any business object can be bound to the Watermark property and that object can be decorated by applying the [WatermarkTemplate](#) property.

XML

```
<Window.Resources>
<DataTemplate x:Key="WatermarkTemplate">
<TextBlock Text="{Binding}"
FontStyle="Italic"
Opacity="1"/>
</DataTemplate>
</Window.Resources>
<Window.Content>
<Grid>
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="200"
Height="40"
Watermark="Type here"
WatermarkTemplate="{StaticResource WatermarkTemplate}">
</editors:SfTextBoxExt>
</Grid>
</Window.Content>
```



Customizing the TextBox

The [Text](#), [FontSize](#), [FontWeight](#), and [FontFamily](#) properties are used to customize the text in the AutoComplete control.

XML

```
<editors:SfTextBoxExt x:Name="textBoxExt"
Text="TextBox"
FontSize="20"
FontWeight="Bold"
FontFamily="Times New Roman"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="200">
</editors:SfTextBoxExt>
```

C#

```
textBoxExt.Text = "TextBox";
textBoxExt.FontSize = 20;
textBoxExt.FontWeight = FontWeights.Bold;
textBoxExt.FontFamily = new FontFamily("Times New Roman");
```



Setting the Dropdown Icon

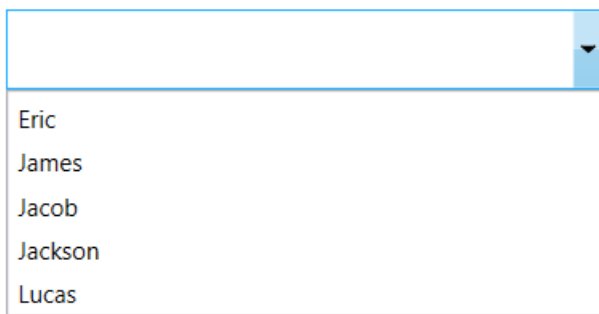
This feature allows to set the drop-down icon for the TextBox control using the [ShowDropDownButton](#).

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
ShowDropDownButton="True"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}" />
```

C#

```
textBoxExt.ShowDropDownButton = true;
```



Setting the ClearButton

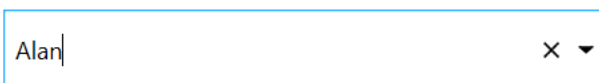
This feature allows to set the ClearButton for the [SfTextBoxExt](#) control using the [ShowClearButton](#).

XML

```
<editors:SfTextBoxExt
Width="300"
Height="40"
HorizontalAlignment="Center"
VerticalAlignment="Center"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}"
SearchItemPath="Name"
ShowClearButton="True"
ShowDropDownButton="True" />
```

C#

```
textBoxExt.ShowClearButton = true;
```



Note: View [sample](#) in GitHub

Dropdown customization in WPF Autocomplete (SfTextBoxExt)

Suggestion box is the drop-down list box, which displays the filtered suggestions inside a pop-up. This section explains the properties and customizations that deals with drop-down list in the **AutoComplete** control.

Customize the background

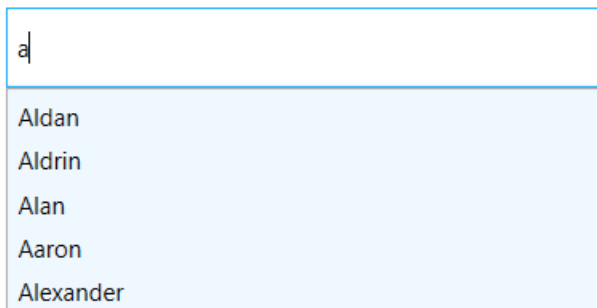
The [DropDownBackground](#) property is used to modify the background color of suggestion box. The following code example demonstrates how to change the background color of suggestion box.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
SearchItemPath="Name"
AutoCompleteMode="Suggest"
DropDownBackground="AliceBlue"
AutoCompleteSource="{Binding Employees}" />
```

C#

```
textBoxExt.DropDownBackground = new SolidColorBrush(Colors.AliceBlue);
```



Drop-down placement

The [SuggestionBoxPlacement](#) property, defines the position of pop-up relative to the control. It contains three built-in options:

1. Top
2. Bottom
3. None

The default value is bottom.

Top

The drop-down list will open at the top of the text field.

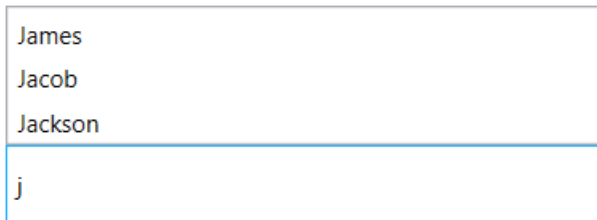
XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
```

```
Width="300"  
Height="40"  
SearchItemPath="Name"  
SuggestionBoxPlacement="Top"  
AutoCompleteMode="Suggest"  
SuggestionMode="StartsWith"  
AutoCompleteSource="{Binding Employees}"/>
```

C#

```
textBoxExt.SuggestionBoxPlacement = SuggestionBoxPlacement.Top;
```



Bottom

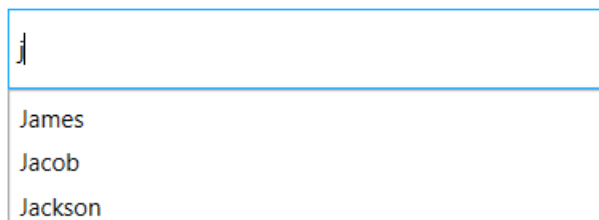
The drop-down list will open at the bottom of the text field.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"  
VerticalAlignment="Center"  
Width="300"  
Height="40"  
SearchItemPath="Name"  
SuggestionBoxPlacement="Bottom"  
AutoCompleteMode="Suggest"  
SuggestionMode="StartsWith"  
AutoCompleteSource="{Binding Employees}"/>
```

C#

```
textBoxExt.SuggestionBoxPlacement = SuggestionBoxPlacement.Bottom;
```



None

The drop-down list will show the filtered items.

XML


```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="300"
Height="40"
SearchItemPath="Name"
SuggestionBoxPlacement="None"
AutoCompleteMode="Suggest"
SuggestionMode="StartsWith"
AutoCompleteSource="{Binding Employees}"/>
```

C#

```
textBoxExt.SuggestionBoxPlacement = SuggestionBoxPlacement.None;
```



Setting the maximum height

The maximum height of the suggestion box in the AutoComplete control can be changed using the [MaximumDropDownHeight](#) property.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="400"
SearchItemPath="Name"
MaxDropDownHeight="500"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}"/>
```

C#

```
textBoxExt.MaxDropDownHeight = 500;
```



Open the drop-down on focus

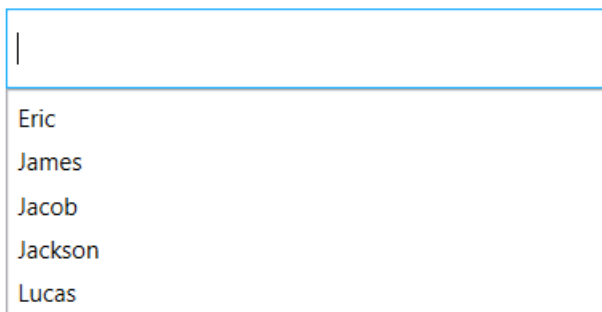
Suggestion box can be shown whenever the control receives focus using the [ShowSuggestionsOnFocus](#) property. At that time, suggestion list is the complete list of data source.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="400"
SearchItemPath="Name"
ShowSuggestionsOnFocus="True"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}"/>
```

C#

```
textBoxExt.ShowSuggestionsOnFocus = true;
```



Open drop-down with a delay

The [PopupDelay](#) specifies the delay after, which the suggestion pop-up should open.

XML

```
<editors:SfTextBoxExt HorizontalAlignment="Center"
VerticalAlignment="Center"
Width="400"
SearchItemPath="Name"
PopupDelay="00:00:02"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}"/>
```

C#

```
textBoxExt.PopupDelay = new TimeSpan(00, 00, 02);
```

Customizing the SelectedItem background

The [SelectionBackgroundColor](#) property is used to set the background color of the selected item in the suggestion box.

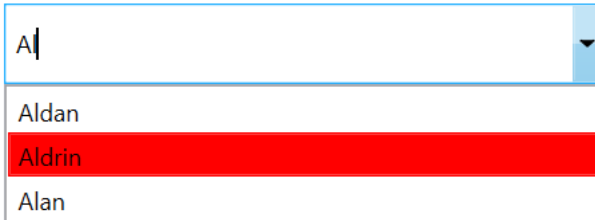
XML

```
<editors:SfTextBoxExt
Width="300"
Height="40"
HorizontalAlignment="Center"
VerticalAlignment="Center"
```

```
AutoCompleteMode="Suggest"  
AutoCompleteSource="{Binding Employees}"  
SearchItemPath="Name"  
SelectionBackgroundColor="Red"  
ShowDropDownButton="True" />
```

C#

```
textBoxExt.SelectionBackgroundColor = new SolidColorBrush(Colors.Red);
```



Note: View [sample](#) in GitHub

How to

Perform multi-path search in WPF Autocomplete

Multi-path search can be achieved using the custom search feature by setting the [SuggestionMode](#) property as **Custom**. Users can define a custom filter and can be assigned to the [Filter](#) property.

Model

C#

```
public class Model  
{  
    private int id;  
    public int ID  
    {  
        get { return id; }  
        set { id = value; }  
    }  
    private string name;  
    public string Name  
    {  
        get { return name; }  
        set { name = value; }  
    }  
}
```

View model

C#

```
public class Viewmodel  
{  
    public ICommand ControlLoaded  
    {  
        get;
```

```

set;
}
private ObservableCollection<Model> employees;
public ObservableCollection<Model> Employees
{
    get { return employees; }
    set { employees = value; }
}
public ViewModel()
{
    ControlLoaded = new DelegateCommand(Loaded);
    Employees = new ObservableCollection<Model>();
    Employees.Add(new Model() { ID = 1, Name = "Eric" });
    Employees.Add(new Model() { ID = 2, Name = "James" });
    Employees.Add(new Model() { ID = 3, Name = "Jacob" });
    Employees.Add(new Model() { ID = 4, Name = "Lucas" });
    Employees.Add(new Model() { ID = 5, Name = "Mark" });
    Employees.Add(new Model() { ID = 6, Name = "Aldan" });
    Employees.Add(new Model() { ID = 7, Name = "Aldrin" });
    Employees.Add(new Model() { ID = 8, Name = "Alan" });
    Employees.Add(new Model() { ID = 9, Name = "Aaron" });
}
private void Loaded(object obj)
{
    var autocomplete = obj as SfTextBoxExt;
    if (autocomplete != null)
    {
        autocomplete.Filter = Filtering;
    }
}
public bool Filtering(string search, object item)
{
    var model = item as Model;
    if (model != null)
    {
        if ((model.Name.ToLower().Contains(search.ToLower())) ||
            (model.ID.ToString().ToLower().Contains(search.ToLower())))
        {
            return true;
        }
    }
    return false;
}
}

```

XML

```

<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<StackPanel Margin="10" VerticalAlignment="Center">
<editors:SfTextBoxExt
x:Name="autocomplete"
AutoCompleteMode="Suggest"
AutoCompleteSource="{Binding Employees}"
SearchItemPath="Name"

```

```

SuggestionMode="Custom">
<editors:SfTextBoxExt.AutoCompleteItemTemplate>
<DataTemplate>
<Grid HorizontalAlignment="Stretch">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="20" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<TextBlock
Grid.Column="0"
Padding="2,8,0,8"
VerticalAlignment="Center"
FontFamily="SegoeUI"
FontSize="12"
FontWeight="Normal"
Text="{Binding ID}" />
<TextBlock
Grid.Column="1"
Padding="10,8,0,8"
VerticalAlignment="Center"
FontFamily="SegoeUI"
FontSize="12"
FontWeight="Normal"
Text="{Binding Name}" />
</Grid>
</DataTemplate>
</editors:SfTextBoxExt.AutoCompleteItemTemplate>
<interaction:Interaction.Triggers>
<interaction:EventTrigger EventName="Loaded">
<interaction:InvokeCommandAction Command="{Binding ControlLoaded}"
CommandParameter="{Binding ElementName=autocomplete}" />
</interaction:EventTrigger>
</interaction:Interaction.Triggers>
</editors:SfTextBoxExt>
</StackPanel>

```

C#

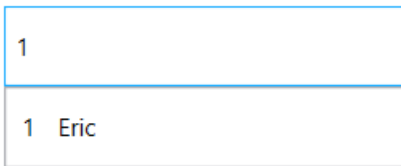
```

Viewmodel viewmodel = new Viewmodel();
this.DataContext = viewmodel;
StackPanel stackPanel = new StackPanel()
{
    Margin = new Thickness(10),
    VerticalAlignment = VerticalAlignment.Center
};
SfTextBoxExt autocomplete = new SfTextBoxExt()
{
    Width = 200,
    Height = 40,
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center,
    AutoCompleteMode = AutoCompleteMode.Suggest,
    AutoCompleteSource = viewmodel.Employees,
    Filter= viewmodel.Filtering,
    SearchItemPath = "Name",
    SuggestionMode = SuggestionMode.Custom
}

```

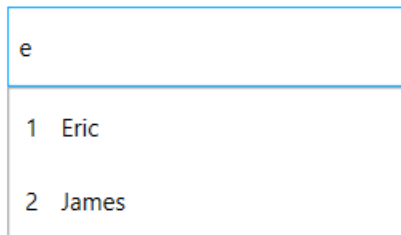
```
};  
FrameworkElementFactory grid = new FrameworkElementFactory(typeof(Grid));  
grid.SetValue(Grid.HorizontalAlignmentProperty,  
HorizontalAlignment.Stretch);  
FrameworkElementFactory firstColumn = new  
FrameworkElementFactory(typeof(ColumnDefinition));  
firstColumn.SetValue(ColumnDefinition.WidthProperty, new GridLength(20.0));  
FrameworkElementFactory secondColumn = new  
FrameworkElementFactory(typeof(ColumnDefinition));  
secondColumn.SetValue(ColumnDefinition.WidthProperty, new GridLength(0.0,  
GridUnitType.Auto));  
FrameworkElementFactory textBlockID = new  
FrameworkElementFactory(typeof(TextBlock));  
textBlockID.SetBinding(TextBlock.TextProperty, new Binding("ID"));  
FrameworkElementFactory textBlockName = new  
FrameworkElementFactory(typeof(TextBlock));  
textBlockName.SetValue(TextBlock.TextProperty, new Binding("Name"));  
grid.AppendChild(textBlockID);  
grid.AppendChild(textBlockName);  
DataTemplate template = new DataTemplate { VisualTree = grid };  
autoComplete.AutoCompleteItemTemplate = template;  
stackPanel.Children.Add(autoComplete);  
this.Content = stackPanel;
```

Search the items with ID:



The screenshot shows a search interface with a text input field at the top containing the number '1'. Below the input field is a list of results. The first result is '1 Eric', where '1' is the ID and 'Eric' is the name. The list is currently showing one item.

Search the items with Name:



The screenshot shows a search interface with a text input field at the top containing the letter 'e'. Below the input field is a list of results. The first result is '1 Eric' and the second result is '2 James', where the first number is the ID and the second is the name. The list is currently showing two items.

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

SfTextInputLayout

WPF TextInputLayout (SfTextInputLayout) Overview

The text input layout control for WPF adds decorative elements such as floating labels, icons, and assistive labels on the top of [TextBox](#) control.

Key features

- Displays floating labels when the input view is focused.
- Displays help labels.
- Displays error labels.
- Displays character count.
- Displays leading and trailing icons.
- Supports filled, outlined, and none container types.
- Provides options to customize the thickness of the base line.
- Provides option to customize the corner radius of the outlined border.
- Provides RTL support.

Getting Started with WPF TextInputLayout (SfTextInputLayout)

This section explains the steps needed to configure the control of the text input layout.

Adding TextInputLayout 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 TextInputLayout to your project.

Initialize TextInputLayout

Import the namespace of the text input layout as shown in the following code snippet.

XML

```
xmlns:inputLayout="clr-namespace:Syncfusion.UI.Xaml.TextInputLayout;assembly=Syncfusion.SfTextInputLayout.WPF"
```

C#

```
using Syncfusion.UI.Xaml.TextInputLayout;
```

You can either use the below schemas or the above mentioned namespace to refer the TextInputLayout control in xaml.

XML

```
xmlns:inputLayout="http://schemas.syncfusion.com/wpf"
```

Then, initialize the text input layout as demonstrated in the following code snippet.

XML

```
<inputLayout:SfTextInputLayout>  
<TextBox/>  
</inputLayout:SfTextInputLayout>
```

C#

```
SfTextInputLayout inputLayout = new SfTextInputLayout();  
inputLayout.InputView = new TextBox();
```

Adding hint

Floating label for the text input layout can be added by setting the `Hint` property. We can specify the display state of the hint label using the `HintVisibility` property, the type of which is [Visibility](#).

XML

```
<inputLayout:SfTextInputLayout
Hint="Name">
<TextBox />
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.InputView = new TextBox();
```

When focusing the input view, the hint label will be moved to the top position; it will be returned to the original position when proceeding further (on unfocused) without entering any value.

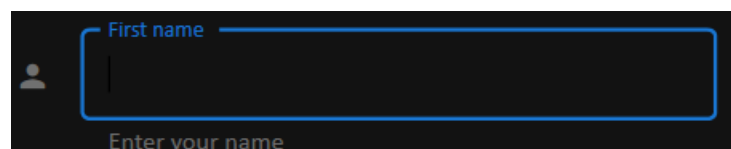
Run the project, and check if you get following output to make sure that the project has been configured properly to add the text input layout control.



Theme

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

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



Supported Input Views in WPF TextInputLayout (SfTextInputLayout)

Input views can be added to the text input layout control by setting the `InputView` property. To reduce the XAML syntax, the `InputView` property is applied with the `ContentPropertyAttribute`. The [SfTextInputLayout](#) has the following controls as the supported input views.

- `TextBox`
- `PasswordBox`
- `ComboBox`
- [ComboBoxAdv](#)

- [SfTextBoxExt](#)

TextBox

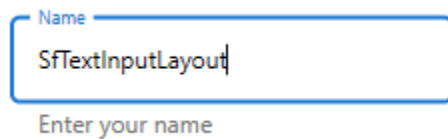
You can enter the text as an input by adding the [TextBox](#) in the [SfTextInputLayout](#).

XML

```
<inputLayout:SfTextInputLayout Hint="Name" HelperText="Enter your name">
  <TextBox/>
</inputLayout:SfTextInputLayout>
```

C#

```
SfTextInputLayout inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.HelperText = "Enter your name";
inputLayout.InputView = new TextBox();
this.Content = inputLayout;
```



PasswordBox

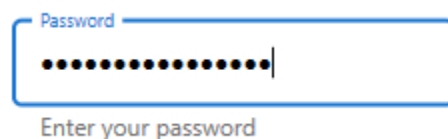
You can enter the password characters as an input by adding the [PasswordBox](#) in the [SfTextInputLayout](#).

XML

```
<inputLayout:SfTextInputLayout Hint="Password" HelperText="Enter your
password">
  <PasswordBox/>
</inputLayout:SfTextInputLayout>
```

C#

```
SfTextInputLayout inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Password";
inputLayout.HelperText = "Enter your password";
inputLayout.InputView = new PasswordBox();
this.Content = inputLayout;
```



ComboBox

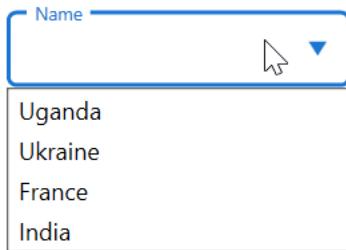
You can use the [ComboBox](#) control as an input in the [SfTextInputLayout](#).

XML

```
<inputLayout:SfTextInputLayout Hint="Name" VerticalAlignment="Center"
HorizontalAlignment="Center">
<ComboBox x:Name="comboBox" Width="150" Height="10" ItemsSource="{Binding
Countries}"/>
</inputLayout:SfTextInputLayout>
```

C#

```
SfTextInputLayout sfTextInputLayout = new SfTextInputLayout() { Hint =
"Name" };
sfTextInputLayout.HorizontalAlignment = HorizontalAlignment.Center;
sfTextInputLayout.VerticalAlignment = VerticalAlignment.Center;
ComboBox comboBox = new ComboBox();
comboBox.Width = 150;
comboBox.Height = 10;
comboBox.ItemsSource = viewModel.Countries;
sfTextInputLayout.InputView = comboBox;
this.Content = sfTextInputLayout;
```

**ComboBoxAdv**

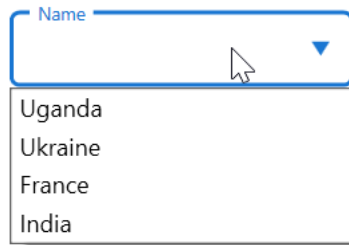
You can use the [ComboBoxAdv](#) control as an input in the [SfTextInputLayout](#).

XML

```
<inputLayout:SfTextInputLayout Hint="Name" VerticalAlignment="Center"
HorizontalAlignment="Center">
<inputLayout:ComboBoxAdv x:Name="comboBox" ItemsSource="{Binding Countries}"
Width="150" Height="10"/>
</inputLayout:SfTextInputLayout>
```

C#

```
SfTextInputLayout sfTextInputLayout = new SfTextInputLayout() { Hint =
"Name" };
sfTextInputLayout.HorizontalAlignment = HorizontalAlignment.Center;
sfTextInputLayout.VerticalAlignment = VerticalAlignment.Center;
ComboBoxAdv comboBox = new ComboBoxAdv();
comboBox.Width = 150;
comboBox.Height = 10;
comboBox.ItemsSource = viewModel.Countries;
sfTextInputLayout.InputView = comboBox;
this.Content = sfTextInputLayout;
```



Autocomplete (SfTextBoxExt)

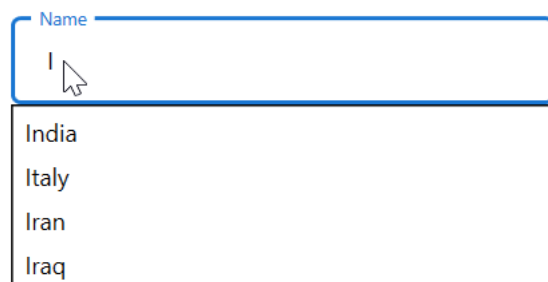
You can use the [SfTextBoxExt](#) control to enter the text as an input in the [SfTextInputLayout](#).

XML

```
<inputLayout:SfTextInputLayout Hint="Name" VerticalAlignment="Center"
HorizontalAlignment="Center">
<inputLayout:SfTextBoxExt AutoCompleteMode="Suggest" Width="250"
AutoCompleteSource="{Binding Countries}">
</inputLayout:SfTextBoxExt>
</inputLayout:SfTextInputLayout>
```

C#

```
SfTextInputLayout sfTextInputLayout = new SfTextInputLayout() { Hint =
"Name" };
sfTextInputLayout.HorizontalAlignment = HorizontalAlignment.Center;
sfTextInputLayout.VerticalAlignment = VerticalAlignment.Center;
SfTextBoxExt sfTextBoxExt = new SfTextBoxExt();
sfTextBoxExt.AutoCompleteMode = AutoCompleteMode.Suggest;
sfTextBoxExt.Width = 250;
sfTextBoxExt.AutoCompleteSource = viewModel.Countries;
sfTextInputLayout.InputView = sfTextBoxExt;
this.Content = sfTextInputLayout;
```



Container Type in WPF TextInputLayout (SfTextInputLayout)

Containers enhance the discoverability of the input view by creating a contrast between the input view and the assistive elements.

Note: The default value of the `ContainerType` is `Outlined`.

Outlined

The container will be covered with a rounded border.

XML

```
<inputLayout:SfTextInputLayout  
Hint="Name"  
ContainerType="Outlined">  
<TextBox Text="John" />  
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Name";  
inputLayout.ContainerType = ContainerType.Outlined;  
inputLayout.InputView = new TextBox() { Text = "John" };
```



Filled

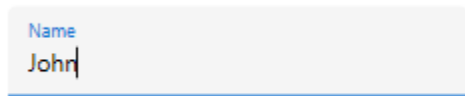
The background of the input view will be filled with container color, and the base line stroke and thickness will be changed based on the state of the input view.

XML

```
<inputLayout:SfTextInputLayout  
Hint="Name"  
ContainerType="Filled">  
<TextBox Text="John" />  
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Name";  
inputLayout.ContainerType = ContainerType.Filled;  
inputLayout.InputView = new TextBox() { Text = "John" };
```



None

The container will have an empty background and enough space.

XML

```
<inputLayout:SfTextInputLayout  
Hint="Name"  
ContainerType="None">  
<TextBox Text="John" />  
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.None;
inputLayout.InputView = new TextBox() { Text = "John" };
```

Name

John

Hint Position in WPF TextInputLayout (SfTextInputLayout)

We can decide how the floating label will display by setting the `HintFloatMode` property.

NOTE

The default value of the `HintFloatMode` is `Float`.

Float

The hint label will be float to the top of input view get focused.

XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
HintFloatMode="Float"
HelperText="Enter your name">
<TextBox />
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.HintFloatMode = HintFloatMode.Float;
inputLayout.HelperText = "Enter your name";
inputLayout.InputView = new TextBox();
```

Name

Enter your name

AlwaysFloat

The hint label will be positioned always at the top of input view.

XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
HintFloatMode="AlwaysFloat">
```

```
HelperText="Enter your name">
<TextBox />
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.HintFloatMode = HintFloatMode.AlwaysFloat;
inputLayout.HelperText= "Enter your name";
inputLayout.InputView = new TextBox();
```



None

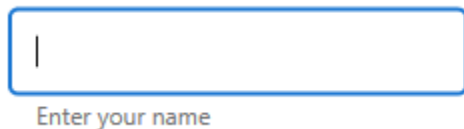
The hint label will be hidden when the input view is focused.

XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
HintFloatMode= "None"
HelperText="Enter your name">
<TextBox />
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.HintFloatMode = HintFloatMode.None;
inputLayout.HelperText= "Enter your name";
inputLayout.InputView = new TextBox();
```



Assistive Labels in WPF TextInputLayout (SfTextInputLayout)

Assistive labels provide additional information about the text entered in the control of the input view.

Helper text

Helper text provides additional guidance on the **Input** field, such as **How to use it?** and can be set using the **HelperText** property.

XML

```
<inputLayout:SfTextInputLayout  
Hint="Name"  
HelperText="Enter your name">  
<TextBox />  
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Name";  
inputLayout.HelperText = "Enter your name";  
inputLayout.InputView = new TextBox();
```



Helper text visibility

We can specify the display state of the helper text using the `HelperTextVisibility` property, the type of which is [Visibility](#).

Error message

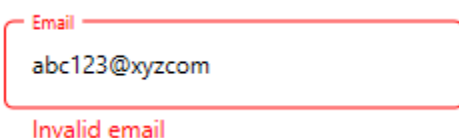
If the text `Input` is not acknowledged, the troubleshooting instructions will be shown in the error message. Error messages are shown below the input line until the correct text has been entered. It can be set using the `ErrorText` property, but it will only be shown when the `HasError` property is set to `true`.

XML

```
<inputLayout:SfTextInputLayout  
Hint="Email"  
HelperText="Enter your email address"  
ErrorText="Invalid email"  
HasError="true">  
<TextBox />  
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Email";  
inputLayout.HelperText = "Enter your email address";  
inputLayout.ErrorText = "Invalid email";  
inputLayout.HasError = true;  
inputLayout.InputView = new TextBox();
```



Note: Error validations should be done in the application level.

Character counter

Character counter is used when characters need to be limited. Use the `CharMaxLength` property to set the limit for characters. We can specify the display state of the character count using the `CharCountVisibility` property, the type of which is [Visibility](#).

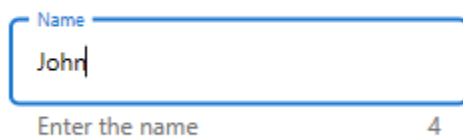
XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
CharCountVisibility="Visible"
CharMaxLength="7"
HelperText="Enter 5 to 7 characters">
<TextBox />
</inputLayout:SfTextInputLayout>
```

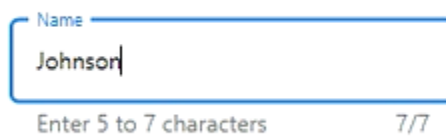
C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.CharMaxLength = 7;
inputLayout.CharCountVisibility = Visibility.Visible;
inputLayout.HelperText = "Enter 5 to 7 characters";
inputLayout.InputView = new TextBox();
```

Without CharMaxLength



With CharMaxLength



Note: When the number of characters to be entered in the input view exceeds the `CharMaxLength`, the `ErrorForeground` value will be applied to the hint label, base line, border and counter label.

Custom Icons in WPF TextInputLayout (SfTextInputLayout)

Any custom icons can be added to the leading edge or the trailing edge of the input view in the text input layout control. Events and commands linked to custom icons should be handled at the application level.

Unicode or font icons for labels can be shown as icons.

Leading view

A label can be added as a leading icon to the input view by setting the `LeadingView` property. It can be placed inside or outside the container by setting the `LeadingViewPosition` property. By default, it is placed `Outside`.

XML

```
<inputLayout:SfTextInputLayout
Hint="Birth date"
```



```

LeadingViewPosition="Inside" >
<TextBox />
<inputLayout:SfTextInputLayout.LleadingView>
<Label
FontFamily="/Assets/Sync FontIcons.ttf#Sync FontIcons"
Text="&#x1F5D3;">
</Label>
</inputLayout:SfTextInputLayout.LleadingView>
</inputLayout:SfTextInputLayout>

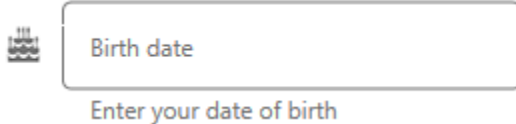
```

C#

```

var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Birth date";
inputLayout.LleadingViewPosition = ViewPosition.Inside;
inputLayout.LleadingView = new Label() { Text = "\U0001F5D3" };
inputLayout.InputView = new TextBox();

```

**Trailing view**

A label can be added as a trailing icon to the input view by setting the `TrailingView` property. It can be placed either inside or outside the input view container by setting the `TrailingViewPosition` property. By default, it is placed `Inside`.

XML

```

<inputLayout:SfTextInputLayout
Hint="Birth date"
TrailingViewPosition="Outside">
<TextBox />
<inputLayout:SfTextInputLayout.TrailingView>
<Label
FontFamily="/Assets/Sync FontIcons.ttf#Sync FontIcons"
Text="&#x1F5D3;">
</Label>
</inputLayout:SfTextInputLayout.TrailingView>
</inputLayout:SfTextInputLayout>

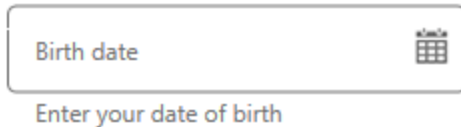
```

C#

```

var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Birth date";
inputLayout.TrailingViewPosition = ViewPosition.Outside;
inputLayout.TrailingView = new Label() { Text = "\U0001F5D3" };
inputLayout.InputView = new TextBox();

```



Customization in WPF TextInputLayout (SfTextInputLayout)

Based on the text input layout state, the colors will be applied to the hint label, base line, border and assistive labels.

Focused color

When the input view is focused, the value of the **FocusedForeground** property will be added to the hint label, base line and border.

Information: The cursor color of the input view is the same as the **Accent** color of the application.

XML

```
<inputLayout:SfTextInputLayout
Hint="User name"
FocusedForeground="Green"
HelperText="Enter your name"
<TextBox Text="John" />
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "User name";
inputLayout.FocusedForeground = Brushes.Green;
inputLayout.InputView = new TextBox() { Text = "John" };
```



Note: The current active color of the text input layout can be obtained from the **ActiveForeground** property.

Since error is not a state, the error color will not be set to **ActiveForeground** when **HasError** property is set to **true**.

Unfocused color

When the input view is unfocused, the **Foreground** property value will be applied to the hint label, base line and border.

XML

```
<inputLayout:SfTextInputLayout
Hint="User name"
Foreground="Gray"
HelperText="Enter your name"
<TextBox Text="John" />
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "User name";
inputLayout.Foreground = Brushes.Gray;
inputLayout.InputView = new TextBox() { Text = "John" };
```



Error color

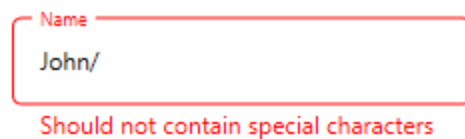
When the input layout is set to error state, the `ErrorForeground` property value will be added to the hint label, base line, border and error text.

XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
ErrorForeground="Red"
ErrorText="Should not contain special characters"
HasError="True">
<TextBox Text="John/" />
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ErrorForeground = Brushes.Red;
inputLayout.ErrorText = "Should not contain special characters";
inputLayout.HasError = true;
inputLayout.InputView = new TextBox() { Text = "John/" };
```



Container color

The color of the container can be customized by setting the `ContainerBackground` property. It is applicable when the `ContainerType` property is set to `Filled` and `Outlined`.

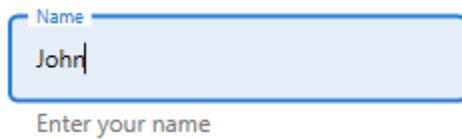
XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
FocusedForeground="Blue"
ContainerType="Outlined"
ContainerBackground="LightBlue">
```

```
<TextBox Text="John" />
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.FocusedColor = Brushes.Blue;
inputLayout.ContainerBackground = Brushes.LightBlue;
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.InputView = new Entry() { TextBox = "John" };
```



Note: Container color is not applicable for 'None' type.

Outline corner radius

The corner radius of the outlined border can be customized by setting `OutlineCornerRadius` property.

XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
ContainerType="Outlined"
OutlineCornerRadius="8">
<TextBox />
</inputLayout:SfTextInputLayout>
```

C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.OutlineCornerRadius = 8;
inputLayout.InputView = new TextBox();
```



NOTE

It is only applicable to the `Outlined` type of container.

Right to Left in WPF TextInputLayout (SfTextInputLayout)

The `TextInputLayout` supports to change the flow of text to the right-to-left direction by setting the `FlowDirection` to `RightToLeft`.

XML

```
<inputLayout:SfTextInputLayout
x:Name="textinputlayout"
FlowDirection="RightToLeft"
ContainerType="Outlined"
Hint="نام"
HelperText="اپنا نام درج کریں" >
<TextBox Text="جانسن" />
</inputLayout:SfTextInputLayout>
```

C#

```
textinputlayout.FlowDirection = FlowDirection.RightToLeft;
textinputlayout.ContainerType = ContainerType.Outlined;
textinputlayout.Hint = "نام";
textinputlayout.HelperText = "اپنا نام درج کریں";
textinputlayout.InputView = new TextBox() { Text = "جانسن"};
```

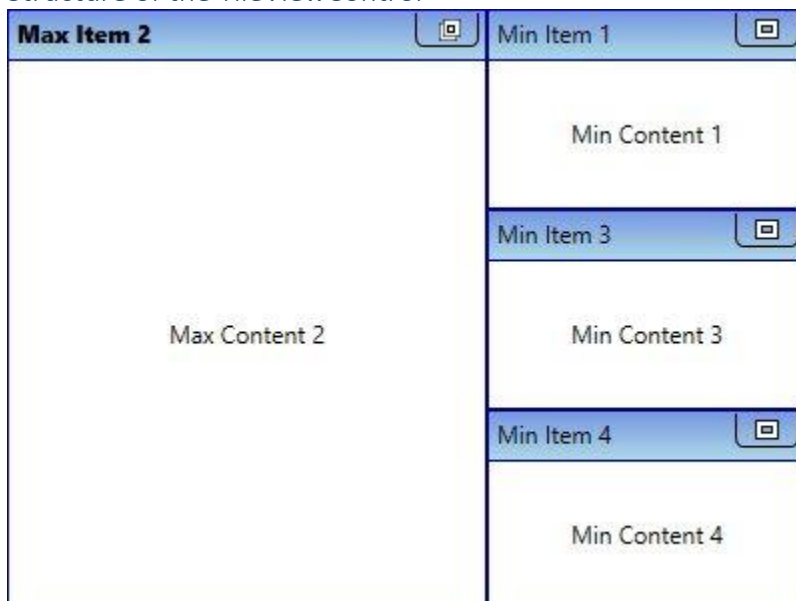


TileView

WPF Tile View Overview

The [TileViewControl](#) acts as a container that holds a set of [TileViewItem](#)s, in which you can host rich information. You can maximize, minimize and drag the items of the [TileViewControl](#) in a matrix position to achieve the best layout.

Structure of the TileViewControl



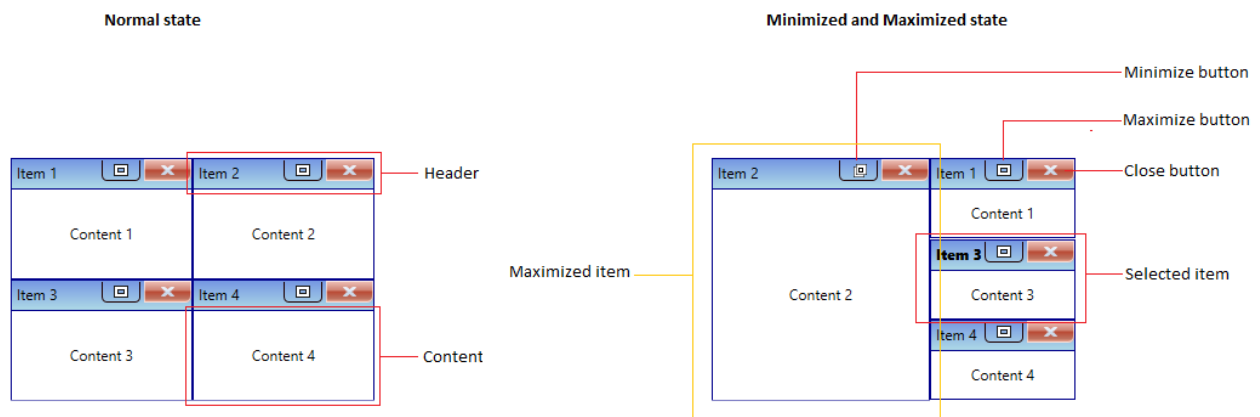
Key features

- Auto arrangements in the matrix order.
- Drag-drop support - To arrange the TileViewItem's.
- Maximizing and minimizing support.
- The minimized position of the TileViewItems horizontal and vertical positions.
- Separate custom UI for minimized and maximized TileViewItem
- Header and content UI customization
- Closing support
- Scroll bar support

Getting Started with WPF Tile View

This section describes how to create a [TileViewControl](#) control in a WPF application and overview of its basic functionalities.

Structure of TileViewControl



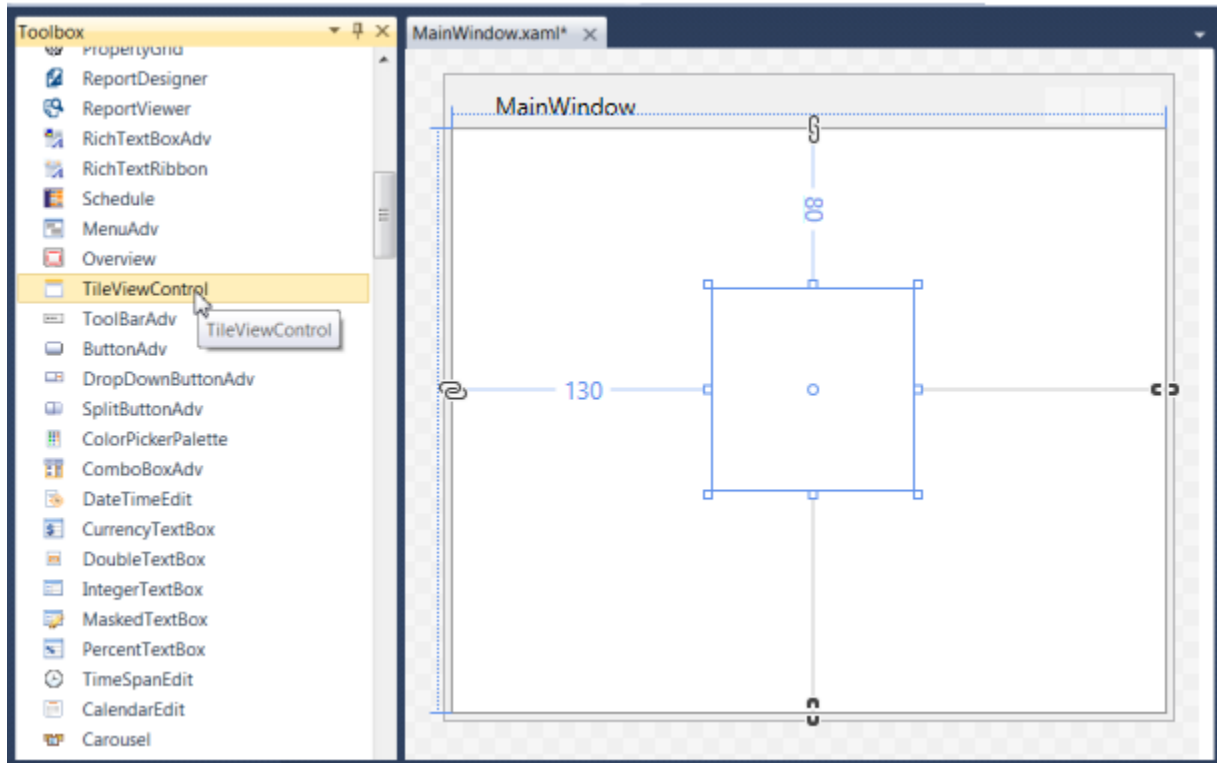
Assembly deployment

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 TileViewControl via designer

1) The **TileViewControl** can be added to an application by dragging it from the toolbox to a designer view. The following dependent assemblies will be added automatically: * Syncfusion.Shared.WPF



2) Set the properties for **TileViewControl** in design mode using the SmartTag feature.

Adding WPF TileViewControl via XAML

To add the **TileViewControl** manually in XAML, follow these steps:

1) Create a new WPF project in Visual Studio. 2) Add the following required assembly references to the project: * Syncfusion.Shared.WPF 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the **TileViewControl** in XAML page.

XML

```
<Window x:Class="TileViewControl_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:TileViewControl_sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid Name="grid">
<syncfusion:TileViewControl Name="tileViewControl"
Height="200"
Width="200"/>
</Grid>
</Window>
```

Adding WPF TileViewControl via C#

To add the **TileViewControl** 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 `TileViewControl`, and add it to the window.

C#

```
// Creating an instance of the TileViewControl
TileViewControl tileViewControl = new TileViewControl();
// Setting height and width to TileViewControl
tileViewControl.Height = 200;
tileViewControl.Width = 300;
```

Note: [View Sample in GitHub](#)

Populating items using `TileViewItem`

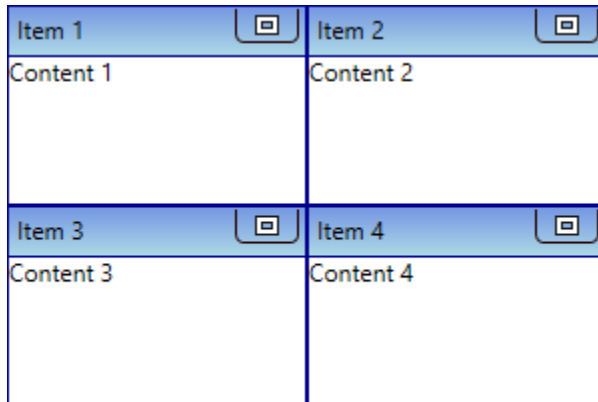
You can add the tileview items inside the control by adding the [TileViewItem](#) into the `TileViewControl.Items` collection property.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
<syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
<syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
<syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Content="Content 1",
Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 2",
Header = "Item 2" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 3",
Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem() { Content = "Content 4",
Header = "Item 4" });
```

Note: [View Sample in GitHub](#)

Populating items using collection binding

You can populate items to the `TileViewControl` by setting the collection value to the `ItemsSource` property.

C#

```
// Model.cs
public class TileItem {
    public string Header { get; set; }
    public string Content { get; set; }
    public TileItem() {
    }
}

//ViewModel.cs
public class ViewModel : NotificationObject {
    private ObservableCollection<TileItem> tileViewItems;
    public ObservableCollection<TileItem> TileViewItems {
        get { return tileViewItems; }
        set {
            tileViewItems = value;
            this.RaisePropertyChanged(nameof(TileViewItems));
        }
    }

    public ViewModel() {
        tileViewItems = new ObservableCollection<TileItem>();
        PopulateCollection();
    }

    public void PopulateCollection() {
        //Adding the tileview items into the collection
        TileViewItems.Add(new TileItem() { Header = "Item 1", Content = "Content 1"
    });
        TileViewItems.Add(new TileItem() { Header = "Item 2", Content = "Content 2"
    });
        TileViewItems.Add(new TileItem() { Header = "Item 3", Content = "Content 3"
    });
        TileViewItems.Add(new TileItem() { Header = "Item 4", Content = "Content 4"
    });
    }
}
```

XML

```
<syncfusion:TileViewControl ItemsSource="{Binding TileViewItems}"
Name="tileViewControl">
<syncfusion:TileViewControl.ItemContainerStyle>
<Style TargetType="{x:Type syncfusion:TileViewItem}">
<Setter Property="Header" Value="{Binding Header}" />
<Setter Property="Content" Value="{Binding Content}" />
</Style>
</syncfusion:TileViewControl.ItemContainerStyle>
<syncfusion:TileViewControl.DataContext>
<local:ViewModel/>
</syncfusion:TileViewControl.DataContext>
</syncfusion:TileViewControl>
```

Item 1	Item 2
Content 1	Content 2
Item 3	Item 4
Content 3	Content 4

Note: [View Sample in GitHub](#)

Select a TileViewItem

You can select any `TileViewItem` by mouse click on the specific `TileViewItem`. You can get the selected item by using the `SelectedItem` property. You can also get the selected value and its index by using the `SelectedValue` and `SelectedIndex` properties. The default value of `SelectedItem` property is `null`.

Note: You can select only one item at a time.

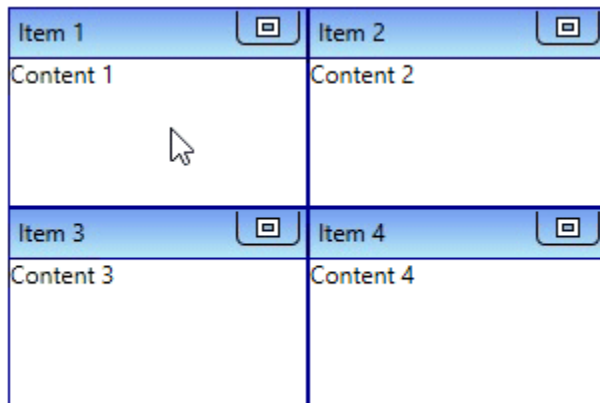
XML

```
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
<syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
<syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
<syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Content="Content 1",
Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 2",
Header = "Item 2" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 3",
Header = "Item 3" });
```

```
tileViewControl.Items.Add(new TileViewItem() { Content = "Content 4",
Header = "Item 4" });
```



Note: [View Sample in GitHub](#)

Select TileViewItem programmatically using property

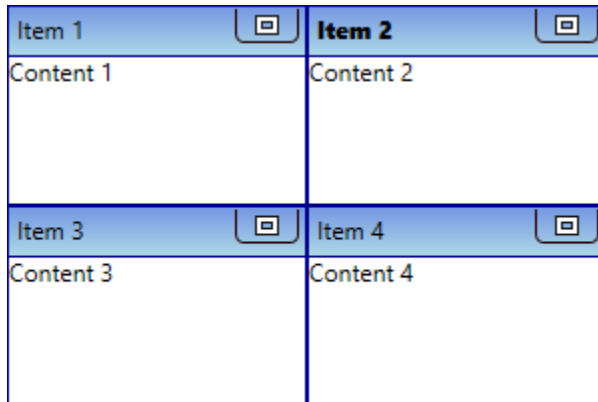
You can select a particular `TileViewItem` programmatically by using the [TileViewItem.IsSelected](#) property.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
<syncfusion:TileViewItem IsSelected="True"
Content="Content 2" Header="Item 2" />
<syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
<syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Content="Content 1",
Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 2",
Header = "Item 2", IsSelected = true });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 3",
Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem() { Content = "Content 4",
Header = "Item 4" });
```



Note: [View Sample in GitHub](#)

Selected item changed notification

The selected item changed in `TileViewControl` can be examined using `SelectionChanged` event. The `SelectionChanged` event contains the old and newly selected item in the `RemovedItems` and `AddedItems` properties.

XML

```
<syncfusion:TileViewControl
  SelectionChanged="TileViewControl_SelectionChanged"
  Name="tileViewControl">
  <syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
  <syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
  <syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
  <syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.SelectionChanged += TileViewControl_SelectionChanged;
```

You can handle the event as follows,

C#

```
private void TileViewControl_SelectionChanged(object sender,
  SelectionChangedEventArgs e) {
  //Get old and new selected TileView item
  var oldItem = e.RemovedItems;
  var newItem = e.AddedItems;
}
```

Arrange TileViewItem in rows and columns

You can change the number of tileview items displayed in view by setting the value to `RowCount` and `ColumnCount` properties. The default value of `RowCount` and `ColumnCount` properties is 0.

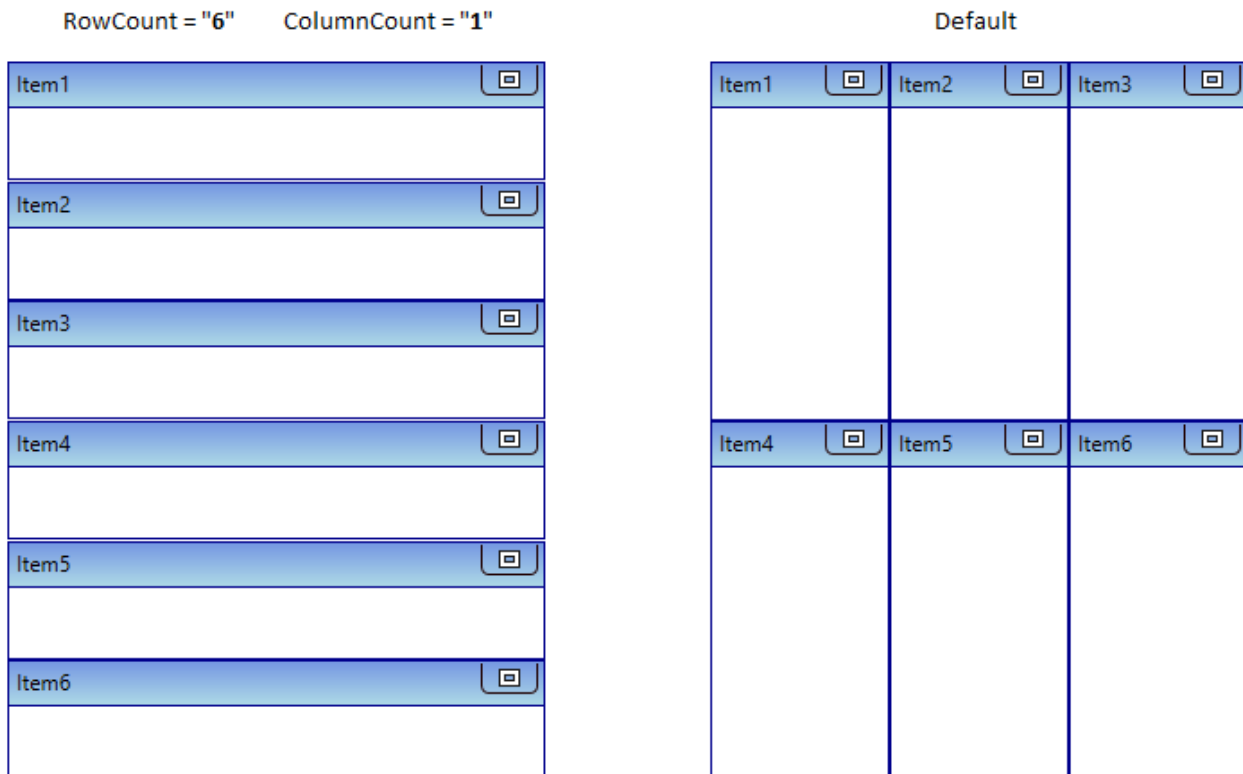
XML

```
<syncfusion:TileViewControl RowCount="6" ColumnCount="1"
  Name="tileViewControl" >
```

```
<syncfusion:TileViewItem Header="Item1"/>
<syncfusion:TileViewItem Header="Item2"/>
<syncfusion:TileViewItem Header="Item3"/>
<syncfusion:TileViewItem Header="Item4"/>
<syncfusion:TileViewItem Header="Item5"/>
<syncfusion:TileViewItem Header="Item6"/>
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.RowCount = 6;
tileViewControl.ColumnCount = 1;
```



Note: [View Sample in GitHub](#)

Minimize or maximize the TileViewItem

You can minimize or maximize the `TileViewItem` by clicking on the `MinMaxButton`. you can change the header and content for the minimized and maximized items separately by using the `TileViewItem.MinimizedHeader`, `TileViewItem.MinimizedItemContent` and `TileViewItem.MaximizedHeader`, `TileViewItem.MaximizedItemContent` properties.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Header="Item 1" Content="Content 1"
MinimizedItemContent="Min Content 1"
MaximizedItemContent="Max Content 1"
MinimizedHeader="Min Item 1"
MaximizedHeader="Max Item 1"/>
```

```

<syncfusion:TileViewItem Header="Item2" Content="Content 2"
MinimizedItemContent="Min Content 2"
MaximizedItemContent="Max Content 2"
MinimizedHeader="Min Item 2"
MaximizedHeader="Max Item 2"/>
<syncfusion:TileViewItem Header="Item3" Content="Content 3"
MinimizedItemContent="Min Content 3"
MaximizedItemContent="Max Content 3"
MinimizedHeader="Min Item 3"
MaximizedHeader="Max Item 3"/>
<syncfusion:TileViewItem Header="Item4" Content="Content 4"
MinimizedItemContent="Min Content 4"
MaximizedItemContent="Max Content 4"
MinimizedHeader="Min Item 4"
MaximizedHeader="Max Item 4"/>
</syncfusion:TileViewControl>

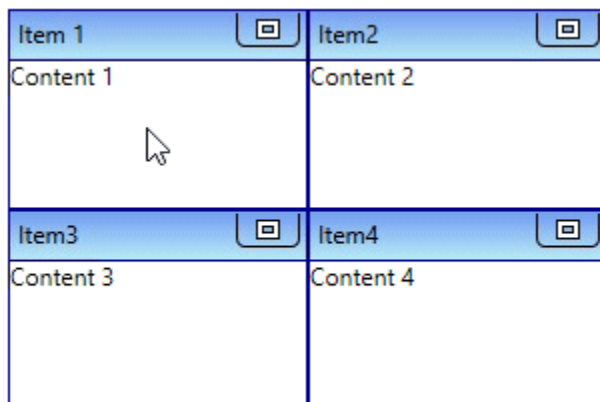
```

C#

```

TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
Content = "Content 1", MaximizedItemContent = "Max Content 1",
MinimizedItemContent = "Min Content 1",
MinimizedHeader="Min Item 1", MaximizedHeader="Max Item 1"});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
Content = "Content 2", MaximizedItemContent = "Max Content 2",
MinimizedItemContent = "Min Content 1",
MinimizedHeader="Min Item 2", MaximizedHeader="Max Item 2"});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
Content = "Content 3", MaximizedItemContent = "Max Content 3",
MinimizedItemContent = "Min Content 1",
MinimizedHeader="Min Item 3", MaximizedHeader="Max Item 3"});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
Content = "Content 4", MaximizedItemContent = "Max Content 4",
MinimizedItemContent = "Min Content 1",
MinimizedHeader="Min Item 4", MaximizedHeader="Max Item 4"});

```



Note: [View Sample in GitHub](#)

Closing TileViewItem

You can close the `TileViewItem` by clicking the close button which is placed top-right corner of the header panel. If you want to display the close button on `TileViewItem`, use the [TileViewItem.CloseButtonVisibility](#) property value as `Visible`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item 1"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 2"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 3"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 4"
    CloseButtonVisibility="Visible"/>
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
    CloseButtonVisibility= Visibility.Visible});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
    CloseButtonVisibility = Visibility.Visible });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
    CloseButtonVisibility = Visibility.Visible });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
    CloseButtonVisibility = Visibility.Visible });
```



Note: [View Sample in GitHub](#)

Custom UI of TileViewItem

You can customize the header and content of `TileViewItem` appearance by using the [HeaderTemplate](#) and [ItemTemplate](#) properties. The `DataContext` of the `HeaderTemplate` property is `TileViewItem.Header` and `dataContext` of the `ItemTemplate` property is `TileViewItem.Content`.

XML

```
<Window.Resources>
```

```

<DataTemplate x:Key="contentTemplate">
  <Grid>
    <TextBlock VerticalAlignment="Center"
      HorizontalAlignment="Center"
      Text="{Binding}"
      Foreground="Blue"/>
  </Grid>
</DataTemplate>
<DataTemplate x:Key="headerTemplate">
  <Grid>
    <TextBlock VerticalAlignment="Center"
      HorizontalAlignment="Center"
      Text="{Binding}"
      Foreground="Red"/>
  </Grid>
</DataTemplate>
</Window.Resources>
<Grid>
  <syncfusion:TileViewControl HeaderTemplate="{StaticResource headerTemplate}"
    ItemTemplate="{StaticResource contentTemplate}"
    Name="tileViewControl">
    <syncfusion:TileViewItem Header="Item 1" Content="Content 1"/>
    <syncfusion:TileViewItem Header="Item 2" Content="Content 2" />
    <syncfusion:TileViewItem Header="Item 3" Content="Content 3"/>
    <syncfusion:TileViewItem Header="Item 4" Content="Content 4"/>
  </syncfusion:TileViewControl>
</Grid>

```

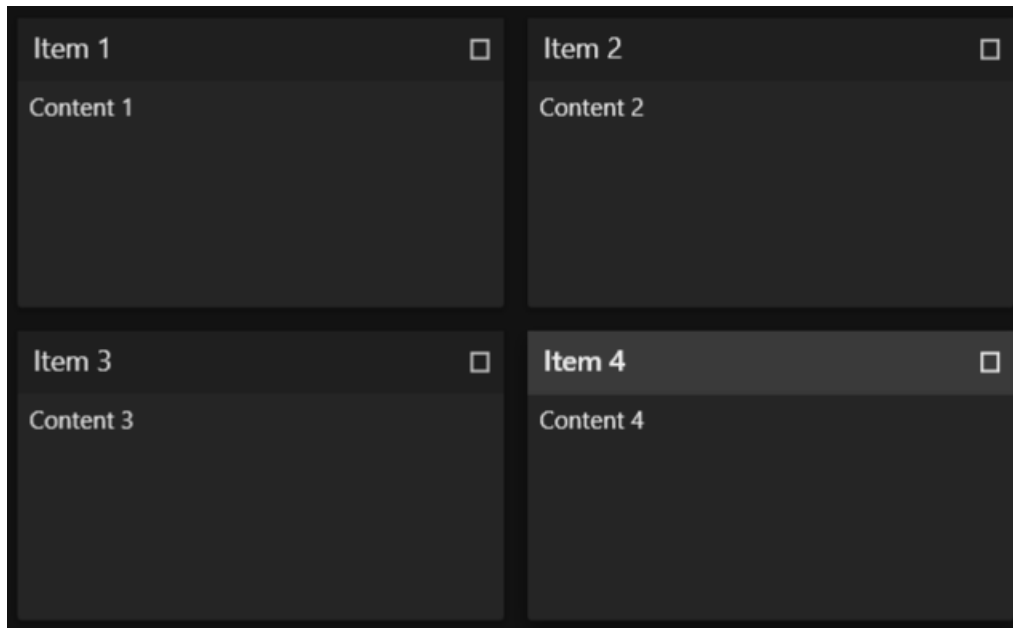
Item 1	Item 2
Content 1	Content 2
Item 3	Item 4
Content 3	Content 4

Note: [View Sample in GitHub](#)

Theme

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

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



Data Binding in WPF Tile View

You can add a [TileViewItem](#) using data binding in the WPF [TileViewControl](#).

Data binding to Objects

The `TileViewControl` can bound to an external source to auto create `TileViewItem` and display the data using `ItemsSource` property. When you are auto generating `TileViewItem` using `ItemsSource`, you need to set `Header` property or `HeaderTemplate` in `ItemContainerStyle` to define header and use `Content` property or `ContentTemplate` in `ItemContainerStyle` to display the content of the `TileViewItem` item.

Note: To bind `ItemsSource` to `TileViewControl`, you need to have collection with data object which holds header and content details.

Here, `TileItem` class defined with `Header` and `Content` properties and `ViewModel` class has `ItemsSource` property of type `ObservableCollection<TileItem>`.

C#

```
// Model.cs
public class TileItem {
    public string Header { get; set; }
    public string Content { get; set; }
    public TileItem() {
    }
}

//ViewModel.cs
public class ViewModel : NotificationObject {
    private ObservableCollection<TileItem> tileViewItems;
    public ObservableCollection<TileItem> TileViewItems {
        get { return tileViewItems; }
        set {
            tileViewItems = value;
            this.RaisePropertyChanged(nameof(TileViewItems));
        }
    }
}
```

```

}
public ViewModel() {
    tileViewItems = new ObservableCollection<TileItem>();
    PopulateCollection();
}
public void PopulateCollection() {
    //Adding the tileview items into the collection
    TileViewItems.Add(new TileItem() { Header = "Item 1", Content = "Content 1"
});
    TileViewItems.Add(new TileItem() { Header = "Item 2", Content = "Content 2"
});
    TileViewItems.Add(new TileItem() { Header = "Item 3", Content = "Content 3"
});
    TileViewItems.Add(new TileItem() { Header = "Item 4", Content = "Content 4"
});
}
}

```

XML

```

<syncfusion:TileViewControl ItemsSource="{Binding TileViewItems}"
    Name="tileViewControl">
    <syncfusion:TileViewControl.ItemContainerStyle>
    <Style TargetType="{x:Type syncfusion:TileViewItem}">
    <Setter Property="Header" Value="{Binding Header}" />
    <Setter Property="Content" Value="{Binding Content}" />
    </Style>
    </syncfusion:TileViewControl.ItemContainerStyle>
    <syncfusion:TileViewControl.DataContext>
    <local:ViewModel/>
    </syncfusion:TileViewControl.DataContext>
</syncfusion:TileViewControl>

```

Item 1	Item 2
Content 1	Content 2
Item 3	Item 4
Content 3	Content 4

Note: [View Sample in GitHub](#)

Data binding with XML

You can bind the XML file as `ItemsSource` for creating the `TileViewItem` in the `TileViewControl`. You can easily populate the items from the XML files using the `ItemTemplate` and `ContentTemplate` or `ItemContainerStyle` properties.

1. Create an XML file with the required details and name it as `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"/>
  <Book Name="Programming WPF"
    Description="A tutorial on XAML, the new HTML-like markup
    language for declaring Windows UI"/>
  <Book Name="Essential WPF"
    Description="Visuals and media, including 2D, 3D, video,
    and animation"/>
  <Book Name="WPF Unleashed"
    Description="Examines the WPF feature areas in incredible
    depth: controls, layout, resources, data binding,
    styling, graphics, animation, and more"/>
</Books>
```

2. Add `XmlDataProvider` for the above `Data.xml` document and bind the data to `ItemsSource` property for the `TileViewControl`.

XML

```
<Window.Resources>
  <XmlDataProvider Source="Data.xml" x:Key="xmlSource" XPath="Books"/>
</Window.Resources>
<Grid>
  <syncfusion:TileViewControl Name="tileViewControl"
    ItemsSource="{Binding Source={StaticResource xmlSource},
    XPath=Book}" />
  <syncfusion:TileViewControl.ItemContainerStyle>
    <Style TargetType="{x:Type syncfusion:TileViewItem}">
      <Setter Property="Header" Value="{Binding XPath=@Name}" />
      <Setter Property="ContentTemplate">
        <Setter.Value>
          <DataTemplate>
            <TextBlock Text="{Binding XPath=@Description}" />
          </DataTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </syncfusion:TileViewControl.ItemContainerStyle>
</syncfusion:TileViewControl>
</Grid>
```

Programming C# 4.0  Learn C# fundamentals, such as variables, flow control, loops, and methods	Programming WPF  A tutorial on XAML, the new HTML-like markup language for declaring Windows UI
Essential WPF  Visuals and media, including 2D, 3D, video, and animation	WPF Unleashed  Examines the WPF feature areas in incredible depth: controls, layout, resources, data binding, styling, graphics, animation, and more

Note: [View Sample in GitHub](#)

Virtualization support

You can enable the UI virtualization support in `TileViewControl`, which allows the users to load large sets of data without affecting loading or scrolling performance by setting the `IsVirtualizing` property value as `true`. This feature allows users to reduce the loading time of `TileView` items regardless of items count. The default value of `IsVirtualizing` property is `false`.

XML

```
<syncfusion:TileViewControl IsVirtualizing="True"
Name="tileViewControl"/>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.IsVirtualizing = true;
```

TileViewItem header

You can define the `TileViewItem` header using `HeaderTemplate` or `TileViewItem.ItemContainerStyle` property. Otherwise, `TileViewItem` header will display the data object class name which is associated with `TileViewItem`. The `DataContext` of the `HeaderTemplate` property is `TileViewItem.Header`.

C#

```
// Model.cs
public class TileItem {
    public string Header { get; set; }
    public string Content { get; set; }
    public TileItem() {
    }
}

// ViewModel.cs
public class ViewModel : NotificationObject {
    private ObservableCollection<TileItem> tileViewItems;
    public ObservableCollection<TileItem> TileViewItems {
```





```
get { return tileViewItems; }
set {
    tileViewItems = value;
    this.RaisePropertyChanged(nameof(TileViewItems));
}
}

public ViewModel() {
    tileViewItems = new ObservableCollection<TileItem>();
    PopulateCollection();
}

public void PopulateCollection() {
    //Adding the tileview items into the collection
    TileViewItems.Add(new TileItem() { Header = "Item 1", Content = "Content 1"
});
    TileViewItems.Add(new TileItem() { Header = "Item 2", Content = "Content 2"
});
    TileViewItems.Add(new TileItem() { Header = "Item 3", Content = "Content 3"
});
    TileViewItems.Add(new TileItem() { Header = "Item 4", Content = "Content 4"
});
}
}
```

XML

```
<syncfusion:TileViewControl Name="tileViewControl"
ItemsSource="{Binding TileViewItems}">
<syncfusion:TileViewControl.HeaderTemplate>
<DataTemplate>
<TextBlock Text="{Binding Header}"
HorizontalAlignment="Center"
VerticalAlignment="Center" />
</DataTemplate>
</syncfusion:TileViewControl.HeaderTemplate>
<syncfusion:TileViewControl.ItemContainerStyle>
<Style TargetType="syncfusion:TileViewItem">
<Setter Property="Content" Value="{Binding Content}"/>
</Style>
</syncfusion:TileViewControl.ItemContainerStyle>
<syncfusion:TileViewControl.DataContext>
<viewModel:ViewModel/>
</syncfusion:TileViewControl.DataContext>
</syncfusion:TileViewControl>
```

Item 1 	Item 2 
Content 1	Content 2
Item 3 	Item 4 
Content 3	Content 4





Note: [View Sample in GitHub](#)

TileViewItem content

You can define the `TileViewItem` content using `ItemTemplate` or `TileViewItem.ItemContainerStyle` property. Otherwise, `TileViewItem` content will display the data object class name which is associated with `TileViewItem`. The `DataContext` of the `ItemTemplate` property is `TileViewItem.Content`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl"
ItemsSource="{Binding TileViewItems}">
<syncfusion:TileViewControl.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Content}"
HorizontalAlignment="Center"
VerticalAlignment="Center" />
</DataTemplate>
</syncfusion:TileViewControl.ItemTemplate>
<syncfusion:TileViewControl.ItemContainerStyle>
<Style TargetType="syncfusion:TileViewItem">
<Setter Property="Header" Value="{Binding Header}" />
</Style>
</syncfusion:TileViewControl.ItemContainerStyle>
<syncfusion:TileViewControl.DataContext>
<viewModel:ViewModel/>
</syncfusion:TileViewControl.DataContext>
</syncfusion:TileViewControl>
```

Item 1 	Item 2 
Content 1	Content 2
Item 3 	Item 4 
Content 3	Content 4

Note: [View Sample in GitHub](#)

Different UI for TileViewItem content

You can change the various UI for the content of `TileViewItem` based on the provided logic by using the `ItemTemplateSelector`. You can also use the `ItemContainerStyleSelector` property to apply the different UI for the `TileViewItem` content. The `DataContext` of the `ItemTemplate` property is `TileViewItem.Content`.

C#

```
//ItemTemplateSelector class for select a DataTemplate
public class MyTemplateSelector : DataTemplateSelector {
    public DataTemplate Template1 { get; set; }
    public DataTemplate Template2 { get; set; }
    public DataTemplate Template3 { get; set; }
    public override DataTemplate SelectTemplate(object item, DependencyObject container) {
        if (item is TileItem && (item as TileItem).Header == "Item 1")
            return Template1;
        else if (item is TileItem && (item as TileItem).Header == "Item 4")
            return Template2;
        else
            return Template3;
    }
}
```

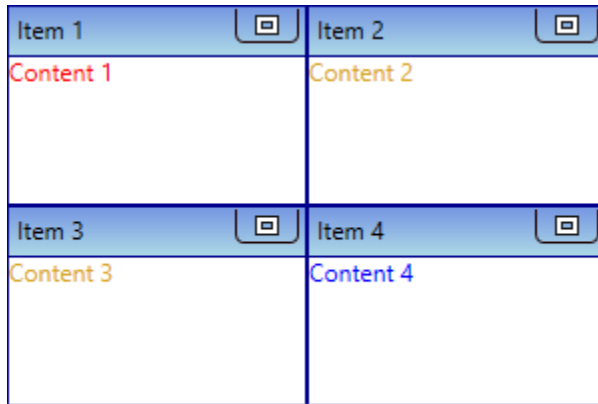
XML

```
<Window.Resources>
    <local:MyTemplateSelector x:Key="Mytemplate">
        <local:MyTemplateSelector.Template1>
            <DataTemplate>
                <TextBlock Text="{Binding Content}" Foreground="Red"/>
            </DataTemplate>
        </local:MyTemplateSelector.Template1>
        <local:MyTemplateSelector.Template2>
            <DataTemplate>
                <TextBlock Text="{Binding Content}" Foreground="Blue"/>
            </DataTemplate>
        </local:MyTemplateSelector.Template2>
        <local:MyTemplateSelector.Template3>
            <DataTemplate>
                <TextBlock Text="{Binding Content}" Foreground="Goldenrod"/>
            </DataTemplate>
        </local:MyTemplateSelector.Template3>
    </local:MyTemplateSelector>
</Window.Resources>
<Grid>
    <syncfusion:TileViewControl ItemTemplateSelector="{StaticResource Mytemplate}"
        ItemsSource="{Binding TileViewItems}"
        Name="tileViewControl">
        <syncfusion:TileViewControl.ItemContainerStyle>
            <Style TargetType="syncfusion:TileViewItem">
                <Setter Property="Header" Value="{Binding Header}"/>
            </Style>
        </syncfusion:TileViewControl.ItemContainerStyle>
    </syncfusion:TileViewControl>
</Grid>
```

```

</syncfusion:TileViewControl.ItemContainerStyle>
<syncfusion:TileViewControl.DataContext>
<viewModel:ViewModel/>
</syncfusion:TileViewControl.DataContext>
</syncfusion:TileViewControl>
</Grid>

```



Note: [View Sample in GitHub](#)

Working with TileView in WPF Tile View

This section explains different UI customization and common features available in [TileViewControl](#) control.

Populating items using TileViewItem

You can add the tileview items inside the control by adding the [TileViewItem](#) into the `TileViewControl.Items` collection property.

XML

```

<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
<syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
<syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
<syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>

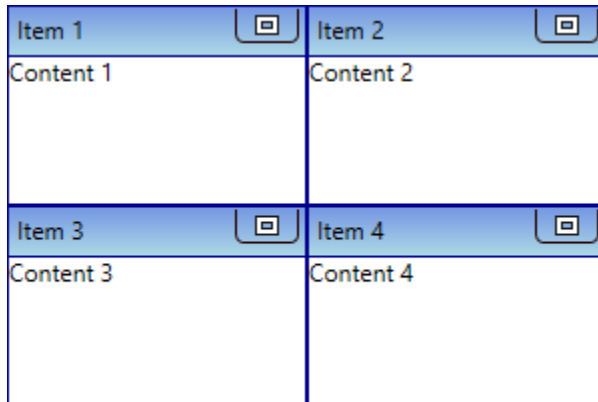
```

C#

```

TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Content="Content 1",
Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 2",
Header = "Item 2" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 3",
Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem() { Content = "Content 4",
Header = "Item 4" });

```

Note: [View Sample in GitHub](#)

Populating items using binding

You can populate items to the `TileView` control by setting the collection value to the `ItemsSource` property.

Note: Please refer [Data Binding](#) page to know more details about binding support available in the `TileViewControl`.

Select a `TileViewItem`

You can select any `TileViewItem` by mouse click on the specific `TileViewItem`. You can get the selected item by using the `SelectedItem` property. You can also get the selected value and its index by using the `SelectedValue` and `SelectedIndex` properties. The default value of `SelectedItem` property is `null`.

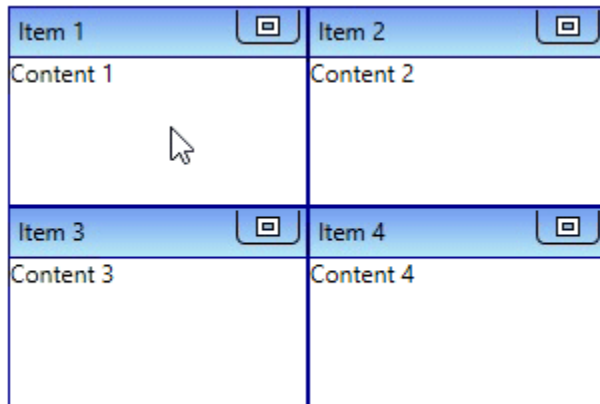
Note: You can select only one item at a time.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
  <syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
  <syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
  <syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Content="Content 1",
Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 2",
Header = "Item 2" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 3",
Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem() { Content = "Content 4",
Header = "Item 4" });
```



Note: [View Sample in GitHub](#)

Select TileViewItem programmatically using property

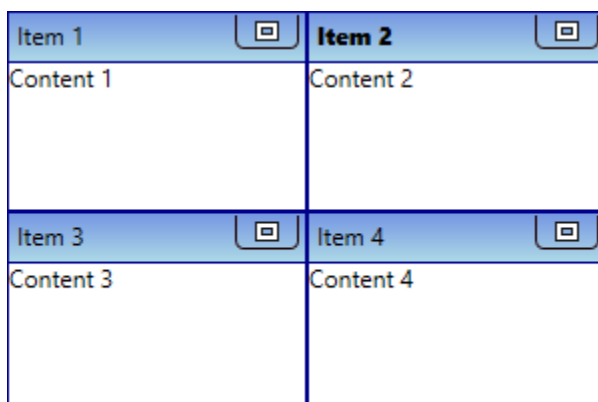
You can select a particular `TileViewItem` programmatically by using the `TileViewItem.IsSelected` property.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
  <syncfusion:TileViewItem IsSelected="True"
    Content="Content 2" Header="Item 2" />
  <syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
  <syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Content="Content 1",
Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 2",
Header = "Item 2", IsSelected = true });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 3",
Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem() { Content = "Content 4",
Header = "Item 4" });
```



Note: [View Sample in GitHub](#)

Selected item changed notification

The selected item changed in `TileViewControl` can be examined using `SelectionChanged` event. The `SelectionChanged` event contains the old and newly selected item in the `RemovedItems` and `AddedItems` properties.

XML

```
<syncfusion:TileViewControl
  SelectionChanged="TileViewControl_SelectionChanged"
  Name="tileViewControl">
  <syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
  <syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
  <syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
  <syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.SelectionChanged += TileViewControl_SelectionChanged;
```

You can handle the event as follows,

C#

```
private void TileViewControl_SelectionChanged(object sender,
  SelectionChangedEventArgs e) {
  //Get old and new selected TileView item
  var oldItem = e.RemovedItems;
  var newItem = e.AddedItems;
}
```

Display TileViewItem splitter

If you want to display the splitter between maximized and each minimized items, use the [SplitterVisibility](#) property value as `Visible`. You can also change the splitter color and thickness by using the [SplitterColor](#) and [SplitterThickness](#) properties. The default value of `SplitterVisibility` is `Collapsed`, `SplitterColor` value is `Gray` and `SplitterThickness` value is `0`.

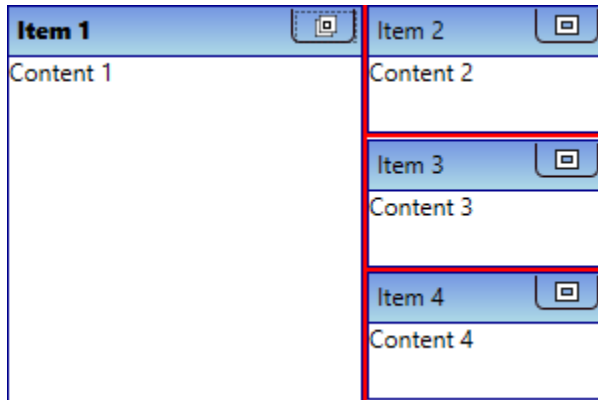
XML

```
<syncfusion:TileViewControl SplitterColor="Red"
  SplitterThickness="2"
  SplitterVisibility="Visible"
  Name="tileViewControl">
  <syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
  <syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
  <syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
  <syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.SplitterColor = Brushes.Red;
```

```
tileViewControl.SplitterThickness = 2;  
tileViewControl.SplitterVisibility = Visibility.Visible;
```



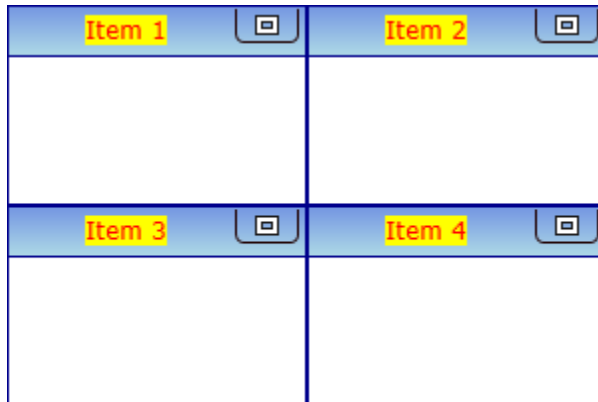
Note: [View Sample in GitHub](#)

Custom UI of TileViewItem header

You can customize the appearance of `TileViewItem` headers by using the [HeaderTemplate](#) property. The `DataContext` of the `HeaderTemplate` property is `TileViewItem.Header`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">  
  <syncfusion:TileViewItem Header="Item 1" />  
  <syncfusion:TileViewItem Header="Item 2" />  
  <syncfusion:TileViewItem Header="Item 3" />  
  <syncfusion:TileViewItem Header="Item 4" />  
  <syncfusion:TileViewControl.HeaderTemplate>  
    <DataTemplate x:Name="headerTemplate">  
      <Grid>  
        <TextBlock HorizontalAlignment="Center"  
          Text="{Binding}"  
          FontFamily="Verdana"  
          Background="Yellow"  
          Foreground="Red" />  
      </Grid>  
    </DataTemplate>  
  </syncfusion:TileViewControl.HeaderTemplate>  
</syncfusion:TileViewControl>
```



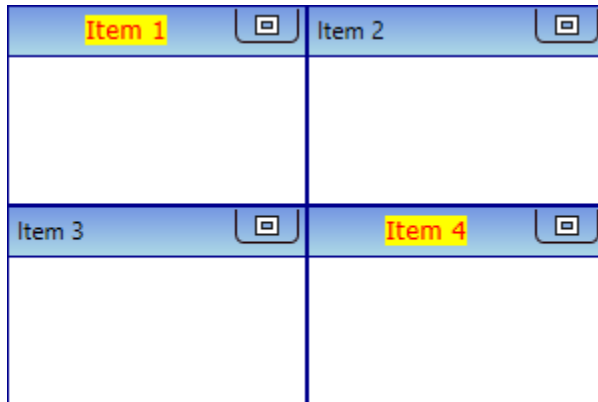
Note: [View Sample in GitHub](#)

Custom UI of specific TileViewItem header

You can customize the appearance of specific `TileViewItem` headers by using the `TileViewItem.HeaderTemplate` property. The `DataContext` of the `TileViewItem.HeaderTemplate` property is `TileViewItem.Header`.

XML

```
<Window.Resources>
<DataTemplate x:Key="headerTemplate">
<Grid>
<TextBlock HorizontalAlignment="Center"
Text="{Binding}"
FontFamily="Verdana"
Background="Yellow"
Foreground="Red"/>
</Grid>
</DataTemplate>
</Window.Resources>
<Grid>
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Header="Item 1"
HeaderTemplate="{StaticResource headerTemplate}" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4"
HeaderTemplate="{StaticResource headerTemplate}" />
</syncfusion:TileViewControl>
</Grid>
```



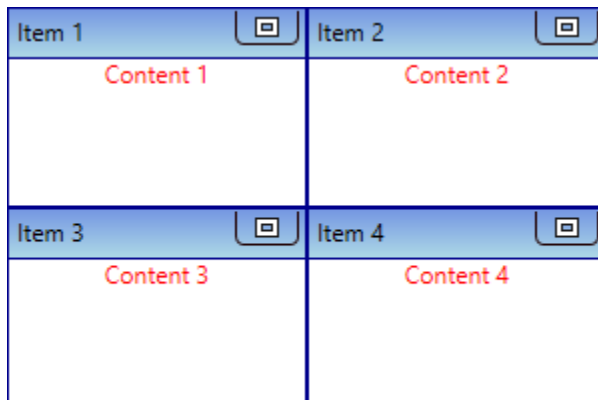
Note: [View Sample in GitHub](#)

Custom UI of TileViewItem content

You can customize the appearance of `TileViewItem` content by using the `ItemTemplate` property. The `DataContext` of the `ItemTemplate` property is `TileViewItem.Content`.

XML

```
<Window.Resources>
<DataTemplate x:Key="contentTemplate">
<Grid>
<TextBlock HorizontalAlignment="Center"
Text="{Binding}"
Foreground="Red"/>
</Grid>
</DataTemplate>
</Window.Resources>
<Grid>
<syncfusion:TileViewControl ItemTemplate="{StaticResource contentTemplate}"
Name="tileViewControl">
<syncfusion:TileViewItem Content="Content 1" Header="Item 1" />
<syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
<syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
<syncfusion:TileViewItem Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
</Grid>
```



Note: [View Sample in GitHub](#)

Custom UI of specific TileViewItem content

You can customize the appearance of specific `TileViewItem` content by using the `TileViewItem.ContentTemplate` property. The `DataContext` of the `TileViewItem.ContentTemplate` property is `TileViewItem.Content`.

XML

```
<Window.Resources>
<DataTemplate x:Key="contentTemplate">
<Grid>
<TextBlock HorizontalAlignment="Center"
Text="{Binding}"
Foreground="Red"/>
</Grid>
</DataTemplate>
</Window.Resources>
<Grid>
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem ContentTemplate="{StaticResource contentTemplate}"
Content="Content 1" Header="Item 1" />
<syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
<syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
<syncfusion:TileViewItem ContentTemplate="{StaticResource contentTemplate}"
Content="Content 4" Header="Item 4" />
</syncfusion:TileViewControl>
</Grid>
```

Item 1	Item 2
Content 1	Content 2
Item 3	Item 4
Content 3	Content 4

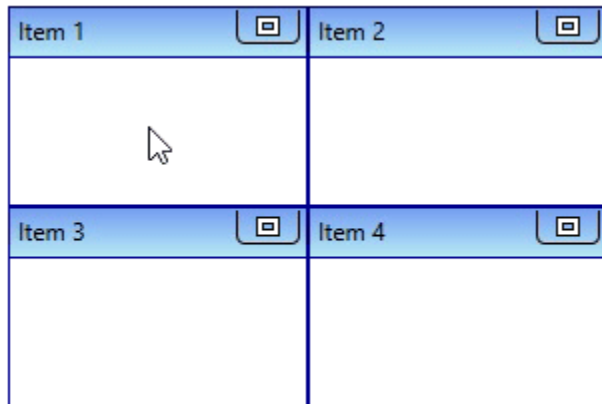
Note: [View Sample in GitHub](#)

Arrange TileViewItem in WPF Tile View

This section explains how to arrange and navigate to the `TileView` item and its alignment functionalities in the `TileViewControl`.

Rearrange position of TileViewItem

If you want to rearrange the `TileViewItem` position, drag that item and drop to anywhere you want to place it in the `TileViewControl`.



Note: [View Sample in GitHub](#)

Restrict rearranging of TileViewItem

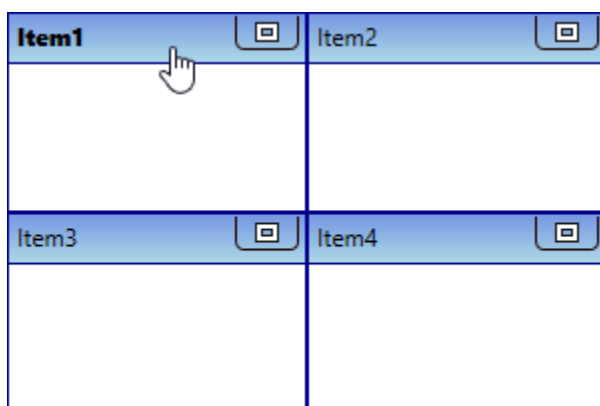
You can restrict the rearranging of `TileViewItem` by setting the `AllowItemRepositioning` property value as `false`. The default value of `AllowItemRepositioning` property is `true`.

XML

```
<syncfusion:TileViewControl AllowItemRepositioning="False"
Name="tileViewControl">
<syncfusion:TileViewItem Header="Item 1" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.AllowItemRepositioning = false;
```



Note: [View Sample in GitHub](#)

Arrange TileViewItem in rows and columns

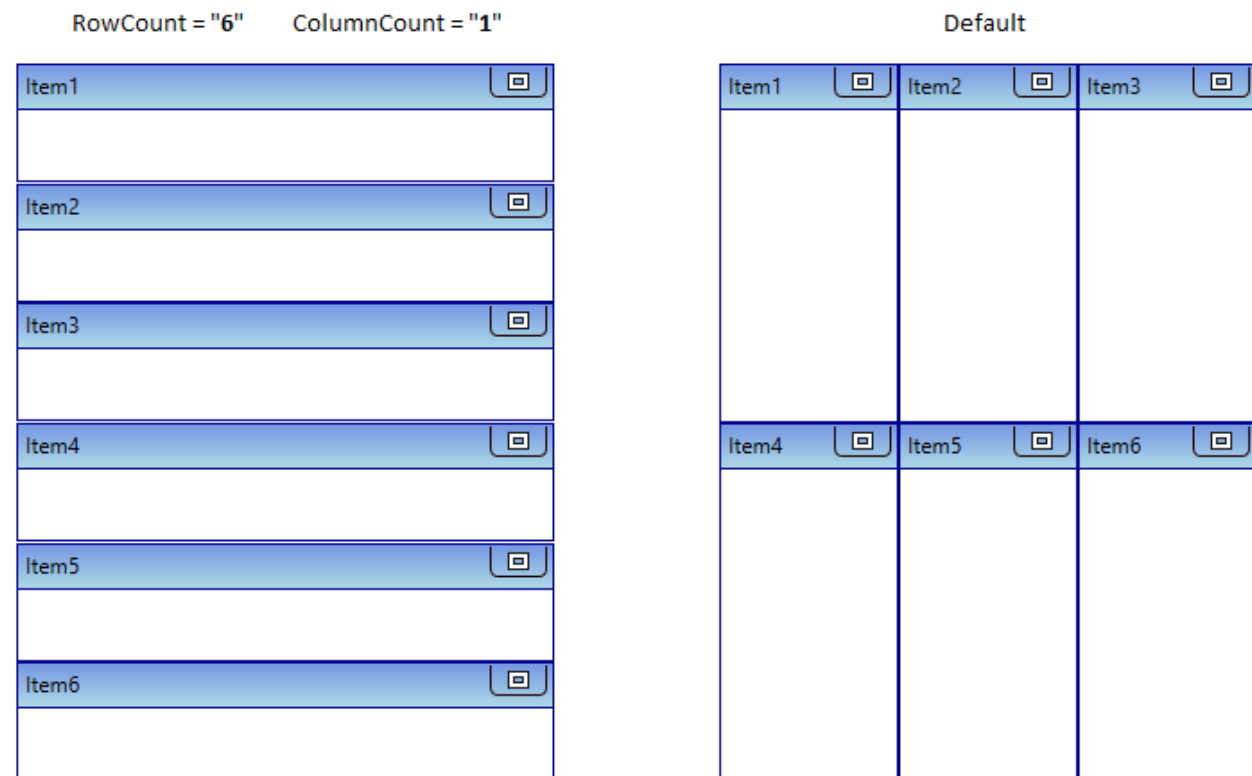
You can change the number of tileview items displayed in view by setting the value to `RowCount` and `ColumnCount` properties. The default value of `RowCount` and `ColumnCount` properties is `0`.

XML

```
<syncfusion:TileViewControl RowCount="6" ColumnCount="1"
Name="tileViewControl" >
<syncfusion:TileViewItem Header="Item1"/>
<syncfusion:TileViewItem Header="Item2"/>
<syncfusion:TileViewItem Header="Item3"/>
<syncfusion:TileViewItem Header="Item4"/>
<syncfusion:TileViewItem Header="Item5"/>
<syncfusion:TileViewItem Header="Item6"/>
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.RowCount = 6;
tileViewControl.ColumnCount = 1;
```



Note: [View Sample in GitHub](#)

Arrange TileViewItem in custom order

You can change the order in which the items are displayed in **TileView** control using the [CurrentItemsOrder](#) list property. By default, the value of **CurrentItemsOrder** property is **null**.

Note: **CurrentItemsOrder** property works only when Virtualization is disabled i.e., **IsVirtualizing** property value is **false**.

Note: **CurrentItemsOrder** list property should contain position of all the items in the **TileView** control i.e., the count of **CurrentItemsOrder** list should be same as the count of **TileView** items.

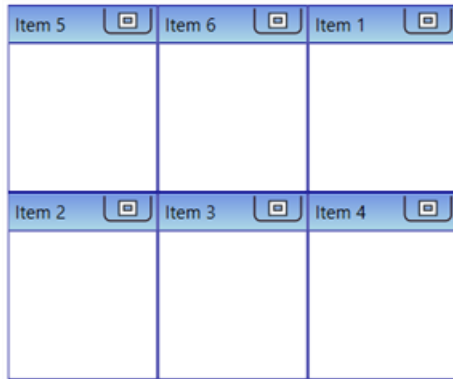
Note: The `CurrentItemsOrder` list property should not contain same position for more than one item.

XML

```
<Grid>
<Grid.DataContext>
<local:ViewModel />
</Grid.DataContext>
<syncfusion:TileViewControl Height="250" CurrentItemsOrder="{Binding
CustomItemOrders, Mode=TwoWay}"
Width="300">
<syncfusion:TileViewItem Header="Item 1"/>
<syncfusion:TileViewItem Header="Item 2"/>
<syncfusion:TileViewItem Header="Item 3"/>
<syncfusion:TileViewItem Header="Item 4"/>
<syncfusion:TileViewItem Header="Item 5"/>
<syncfusion:TileViewItem Header="Item 6"/>
</syncfusion:TileViewControl>
</Grid>
```

C#

```
public class ViewModel : NotificationObject
{
    private List<int> customItemOrders;
    public List<int> CustomItemOrders
    {
        get { return customItemOrders; }
        set
        {
            customItemOrders = value;
            this.RaisePropertyChanged(nameof(this.CustomItemOrders));
        }
    }
    public ViewModel()
    {
        CustomItemOrders = new List<int>();
        CustomItemOrders.Add(4);
        CustomItemOrders.Add(5);
        CustomItemOrders.Add(0);
        CustomItemOrders.Add(1);
        CustomItemOrders.Add(2);
        CustomItemOrders.Add(3);
    }
}
```



Change row and column size

By default, `TileViewItem` rows and columns size are allocated based on the control size and number of items in a control. You can change the row and column size of the `TileViewItem` by setting the value to `RowHeight` and `ColumnWidth` properties. The default value of `RowHeight` and `ColumnWidth` properties is `auto`.

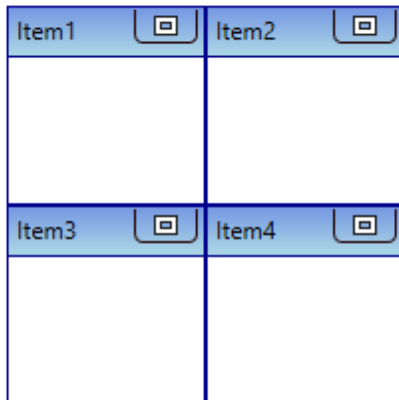
XML

```
<syncfusion:TileViewControl RowHeight="100"
ColumnWidth="100"
Name="tileViewControl">
<syncfusion:TileViewItem Header="Item1"/>
<syncfusion:TileViewItem Header="Item2"/>
<syncfusion:TileViewItem Header="Item3"/>
<syncfusion:TileViewItem Header="Item4"/>
</syncfusion:TileViewControl>
```

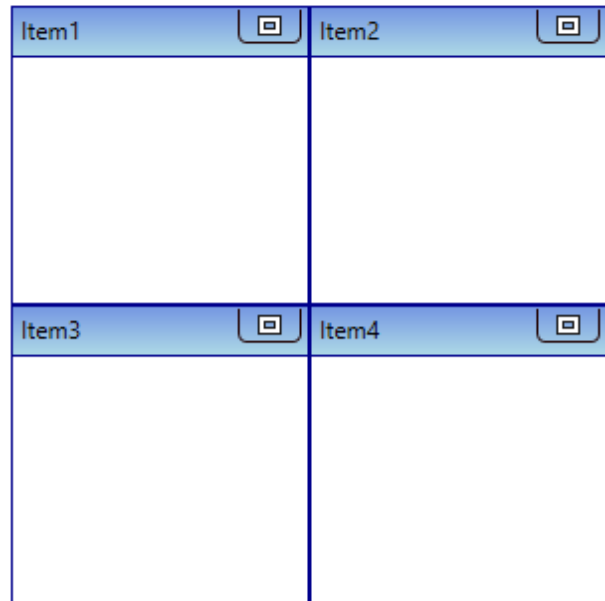
C#

```
tileViewControl.RowHeight = 100;
tileViewControl.ColumnWidth = 100;
```

RowHeight = "100" ColumnWidth = "100"



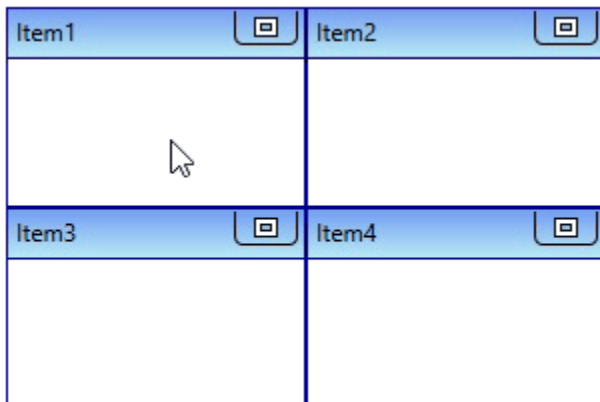
Default



Note: [View Sample in GitHub](#)

Navigate to TileViewItem

You can navigate from one `TileViewItem` to any other `TileViewItem` by using the mouse click on the specific `TileViewItem`.



Navigate to hidden items using scroll bar

If you add more items and set the row and column size to place the items that exceeds the control size, then some items are hidden from the view. You can easily navigate to the hidden items by using the scroll bar. You can enable the vertical and horizontal scroll bars by using the [HorizontalScrollBarVisibility](#) and [VerticalScrollBarVisibility](#) properties value as `auto` or `Visible`. The default value of `HorizontalScrollBarVisibility` and `VerticalScrollBarVisibility` properties is `Disabled`.

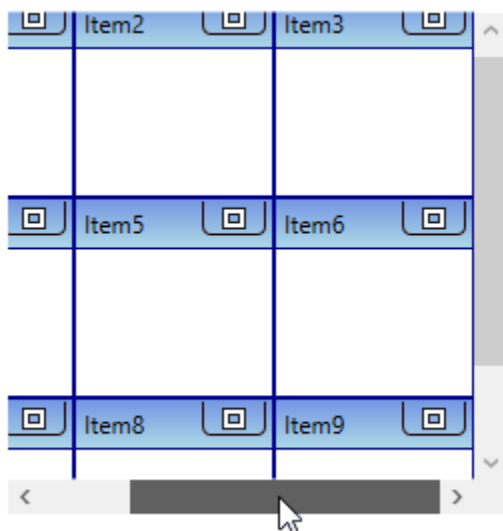
XML

```
<syncfusion:TileViewControl ColumnWidth="100"
RowHeight="100"
HorizontalScrollBarVisibility="Auto"
```

```
VerticalScrollBarVisibility="Auto"  
Name="tileViewControl">  
<syncfusion:TileViewItem Header="Item1"/>  
<syncfusion:TileViewItem Header="Item2"/>  
<syncfusion:TileViewItem Header="Item3" />  
<syncfusion:TileViewItem Header="Item4"/>  
<syncfusion:TileViewItem Header="Item5"/>  
<syncfusion:TileViewItem Header="Item6"/>  
<syncfusion:TileViewItem Header="Item7" />  
<syncfusion:TileViewItem Header="Item8"/>  
<syncfusion:TileViewItem Header="Item9"/>  
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.RowHeight = 100;  
tileViewControl.ColumnWidth = 100;  
tileViewControl.HorizontalScrollBarVisibility = Visibility.Auto;  
tileViewControl.VerticalScrollBarVisibility = Visibility.Auto;
```



Note: [View Sample in GitHub](#)

Change built-in animation duration

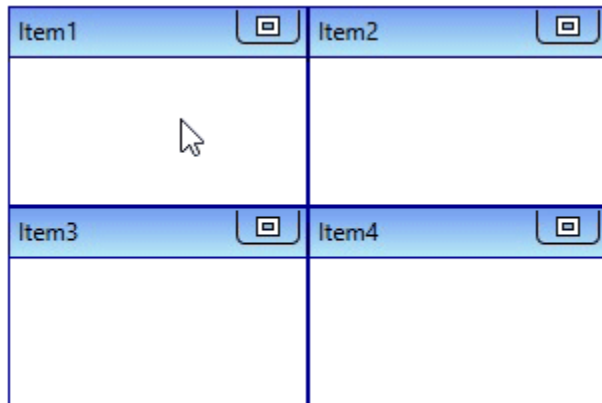
If you want to change the animation duration of navigation, use the [AnimationDuration](#) property. The default value of `AnimationDuration` property is `{00:00:00.7000000}`.

XML

```
<syncfusion:TileViewControl AnimationDuration="00:00:00.300"  
Name="tileViewControl">  
<syncfusion:TileViewItem Header="Item1"/>  
<syncfusion:TileViewItem Header="Item2"/>  
<syncfusion:TileViewItem Header="Item3" />  
<syncfusion:TileViewItem Header="Item4"/>  
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.AnimationDuration = new TimeSpan(0, 0, 0, 0, 300);
```



Note: [View Sample in GitHub](#)

Disable built-in navigation animation

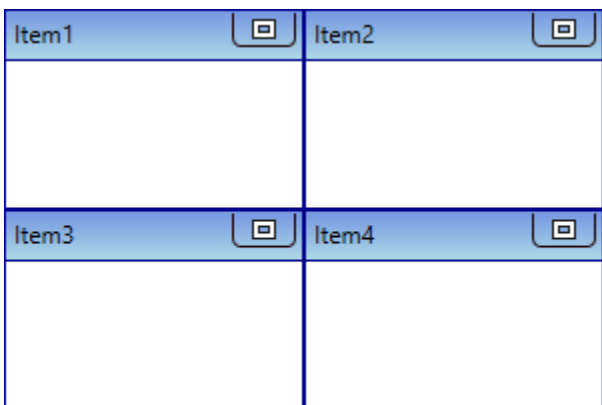
By default, navigation of `TileViewItem` animation is enabled. If you want to disable the animation while navigation of `TileViewItem`, use the [EnableAnimation](#) property value as `false`.

XML

```
<syncfusion:TileViewControl EnableAnimation="False"
Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item1"/>
  <syncfusion:TileViewItem Header="Item2"/>
  <syncfusion:TileViewItem Header="Item3" />
  <syncfusion:TileViewItem Header="Item4"/>
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.EnableAnimation = false;
```



Note: [View Sample in GitHub](#)

Close TileViewItem in WPF Tile View

This section explains how to closing the [TileViewItem](#) and its functionalities in the [TileViewControl](#).

Show close button

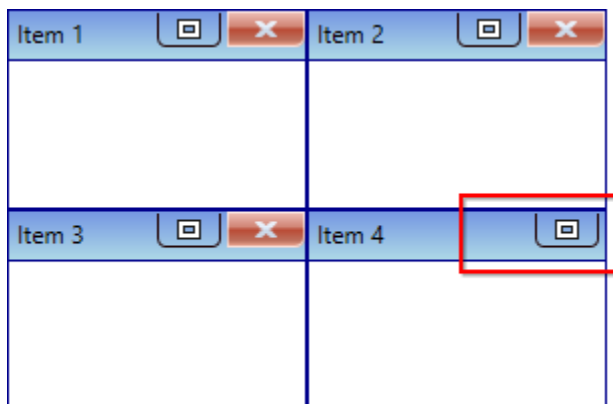
By default, the close button is not displayed in the [TileViewItem](#). If you want to display the close button on specific [TileViewItem](#), use the [TileViewItem.CloseButtonVisibility](#) property value as [Visible](#). The default value of [TileViewItem.CloseButtonVisibility](#) property is [Collapsed](#).

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item 1"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 2"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 3"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 4"
    CloseButtonVisibility="Collapsed"/>
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
    CloseButtonVisibility= Visibility.Visible});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
    CloseButtonVisibility = Visibility.Visible });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
    CloseButtonVisibility = Visibility.Visible });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
    CloseButtonVisibility = Visibility.Collapsed });
```



Note: [View Sample in GitHub](#)

Closing TileViewItem

You can close the [TileViewItem](#) by clicking the close button which is placed top-right corner of the header panel.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item 1"
  CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 2"
  CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 3"
  CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 4"
  CloseButtonVisibility="Visible"/>
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
CloseButtonVisibility= Visibility.Visible});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
CloseButtonVisibility = Visibility.Visible });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
CloseButtonVisibility = Visibility.Visible });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
CloseButtonVisibility = Visibility.Visible });
```



Note: [View Sample in GitHub](#)

Closing TileViewItem programmatically

If you want to close the `TileViewItem` programmatically, pass that items into the [CloseTileViewItem\(\)](#) method.

XML

```
<Button Content="Close Item"
Click="CloseItem_Click"/>
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item 1" />
  <syncfusion:TileViewItem Header="Item 2" />
  <syncfusion:TileViewItem Header="Item 3" />
  <syncfusion:TileViewItem />
</syncfusion:TileViewControl>
```

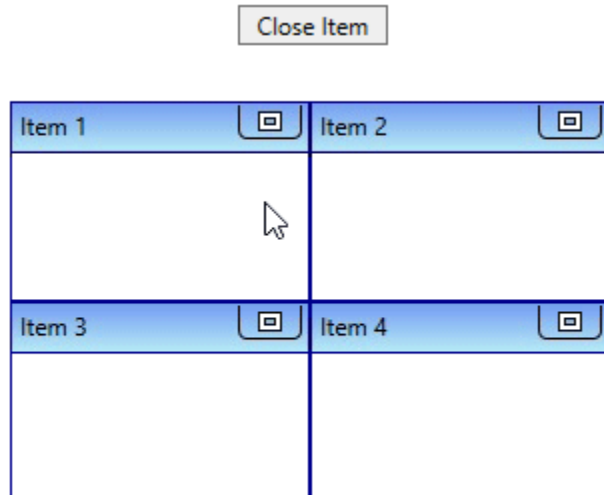

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem());
Button button= new Button();
button.Click += CloseItem_Click
```

You can handle the CloseItem_Click event as follows:

C#

```
private void CloseItem_Click(object sender, RoutedEventArgs e) {
    //First item in the Items collection will be removed
    tileViewControl.CloseTileViewItem(tileViewControl.Items[0] as TileViewItem);
}
```



Hide or delete TileViewItem when closing a item

You can decide whether the `TileViewItem` can be only hidden from the view or removed from the items collection of `TileViewControl` by using the `CloseMode` property while closing it. If you set `CloseMode` property as `Hide`, the `TileViewItem` will be hidden and the selection will be moved to previous index while hiding it. Also, if the property `CloseMode` is `Delete`, the `TileViewItem` will be removed from the items collection and the selection will be retained in the same index while removing it. The default value of the `CloseMode` property is `Hide`.

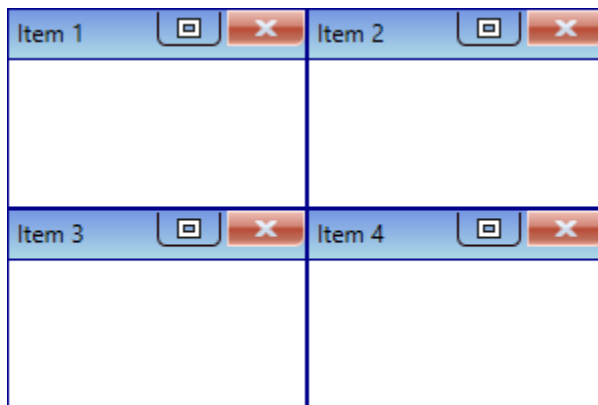
XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item 1" CloseMode="Delete"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 2" CloseMode="Delete"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 3" CloseMode="Delete"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Header="Item 4" CloseMode="Delete"
```

```
CloseButtonVisibility="Visible"/>
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
CloseButtonVisibility= Visibility.Visible, CloseMode= CloseMode.Delete});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
CloseButtonVisibility = Visibility.Visible, CloseMode= CloseMode.Delete});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
CloseButtonVisibility = Visibility.Visible, CloseMode= CloseMode.Delete});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
CloseButtonVisibility = Visibility.Visible, CloseMode= CloseMode.Delete});
```



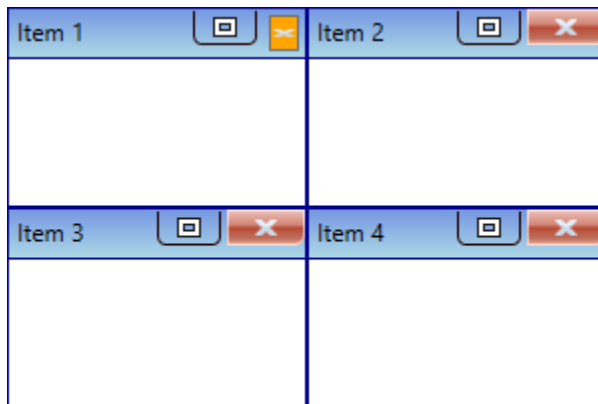
Note: [View Sample in GitHub](#)

Custom UI of close button

You can customize the appearance of particular `TileViewItem`'s close button by using the [CloseButtonStyle](#) property. You can also change the margin of the `TileViewItem` close button by using the [CloseButtonMargin](#) property. The `DataContext` of the `CloseButtonStyle` property is [TileViewItemCloseButton](#).

XML

```
<Window.Resources>
<Style x:Key="closeButtonStyle"
TargetType="syncfusion:TileViewItemCloseButton">
<Setter Property="Background" Value="Orange"/>
</Style>
</Window.Resources>
<Grid>
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem CloseButtonMargin="3"
CloseButtonStyle="{StaticResource closeButtonStyle}"
Header="Item 1" CloseButtonVisibility="Visible"/>
<syncfusion:TileViewItem Header="Item 2" CloseButtonVisibility="Visible"/>
<syncfusion:TileViewItem Header="Item 3" CloseButtonVisibility="Visible"/>
<syncfusion:TileViewItem Header="Item 4" CloseButtonVisibility="Visible"/>
</syncfusion:TileViewControl>
</Grid>
```



Note: [View Sample in GitHub](#)

TileViewItem closing notification

When the `TileViewItem` is closing, it will be notified by using the [TileViewItem.Closing](#) event. You can restrict the closing of `TileViewItem` by using the [CloseEventArgs.Cancel](#) property value as `true`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Closing="TileViewItem_Closing"
    Header="Item 1"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Closing="TileViewItem_Closing"
    Header="Item 2"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Closing="TileViewItem_Closing"
    Header="Item 3"
    CloseButtonVisibility="Visible"/>
  <syncfusion:TileViewItem Closing="TileViewItem_Closing"
    Header="Item 4"
    CloseButtonVisibility="Visible"/>
</syncfusion:TileViewControl>
```

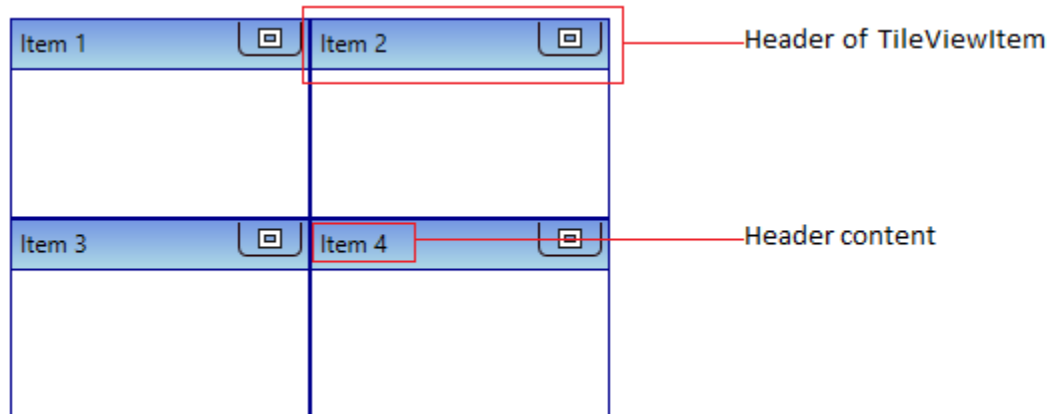
You can handle the event as follows:

C#

```
private void TileViewItem_Closing(object sender, TileViewItem.CloseEventArgs
args) {
    //Restrict closing
    args.Cancel = true;
}
```

TileViewItem Header in WPF Tile View

This section explains how to set header text and UI customization of the `TileViewItem` header in the [TileViewControl](#).



Setting TileViewItem header

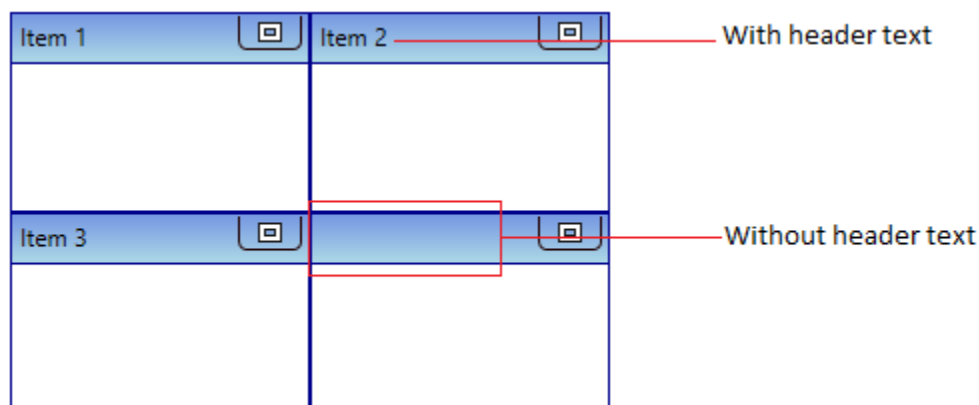
You can add a text for the each [TileViewItem](#) header by using the `TileViewItem.Header` property. The default value of `TileViewItem.Header` property is `null`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item 1" />
  <syncfusion:TileViewItem Header="Item 2" />
  <syncfusion:TileViewItem Header="Item 3" />
  <syncfusion:TileViewItem />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem());
```



Note: [View Sample in GitHub](#)

Change minimized and maximized header

By default, the `TileViewItem.Header` property value is displayed as `TileViewItem` header text. If you want to change the header text on minimized and maximized state, use the

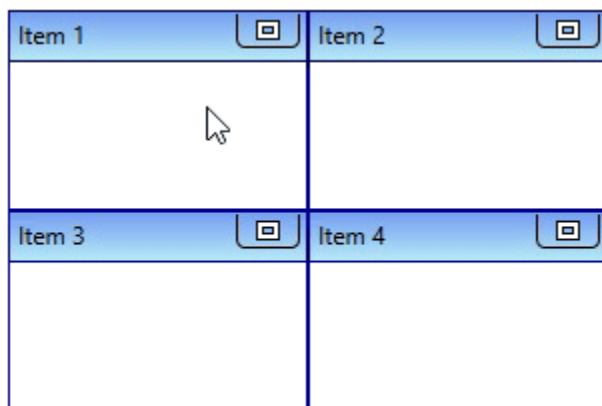
[TileViewItem.MinimizedHeader](#) and [TileViewItem.MaximizedHeader](#) properties. Based on the minimized and maximized state of the [TileViewItem](#), respective header text is displayed. The default value of [TileViewItem.MinimizedHeader](#) and [TileViewItem.MaximizedHeader](#) property is .

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item 1" MinimizedHeader="Min Item 1"
    MaximizedHeader="Max Item 1" />
  <syncfusion:TileViewItem Header="Item 2" MinimizedHeader="Min Item 2"
    MaximizedHeader="Max Item 2" />
  <syncfusion:TileViewItem Header="Item 3" MinimizedHeader="Min Item 3"
    MaximizedHeader="Max Item 3" />
  <syncfusion:TileViewItem Header="Item 4" MinimizedHeader="Min Item 4"
    MaximizedHeader="Max Item 4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
    MinimizedHeader = "Min Item 1", MaximizedHeader = "Max Item 1"});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
    MinimizedHeader = "Min Item 2", MaximizedHeader = "Max Item 2"});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
    MinimizedHeader = "Min Item 3", MaximizedHeader = "Max Item 3"});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
    MinimizedHeader = "Min Item 4", MaximizedHeader = "Max Item 4"});
```



Note: [View Sample in GitHub](#)

Hide the [TileViewItem](#) header

If you want to hide the specific [TileViewItem](#)'s header panel, use the [TileViewItem.HeaderVisibility](#) property value as [Collapsed](#). The default value of [TileViewItem.HeaderVisibility](#) property is [Visible](#).

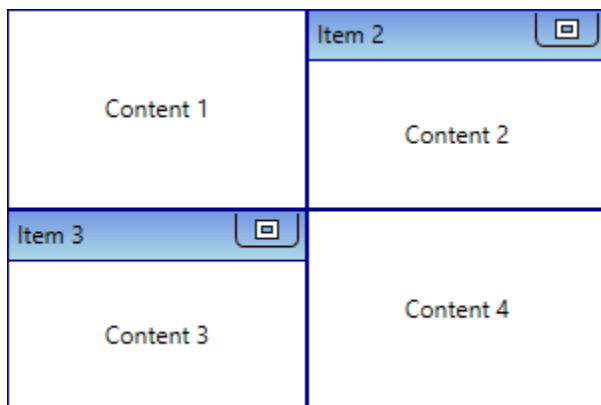
XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Content="Content 1" Header="Item 1"
    HeaderVisibility="Collapsed"/>
  <syncfusion:TileViewItem Content="Content 2" Header="Item 2" />
</syncfusion:TileViewControl>
```

```
<syncfusion:TileViewItem Content="Content 3" Header="Item 3" />
<syncfusion:TileViewItem Content="Content 4" Header="Item 4"
HeaderVisibility="Collapsed"/>
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Content="Content 1",
Header = "Item 1", HeaderVisibility= Visibility.Collapsed });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 2",
Header = "Item 2", });
tileViewControl.Items.Add(new TileViewItem() { Content="Content 3",
Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem() { Content = "Content 4",
Header = "Item 4", HeaderVisibility = Visibility.Collapsed });
```



Note: [View Sample in GitHub](#)

Change TileViewItem header height

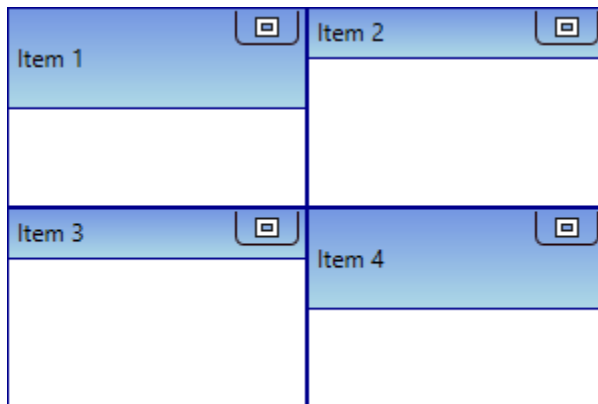
If you want to change height of the specific `TileViewItem`'s header panel, use the [TileViewItem.HeaderHeight](#) property. The default value of `TileViewItem.HeaderHeight` property is 25.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Header="Item 1" HeaderHeight="50" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4" HeaderHeight="50" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
HeaderHeight = 50 });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
HeaderHeight = 50 });
```



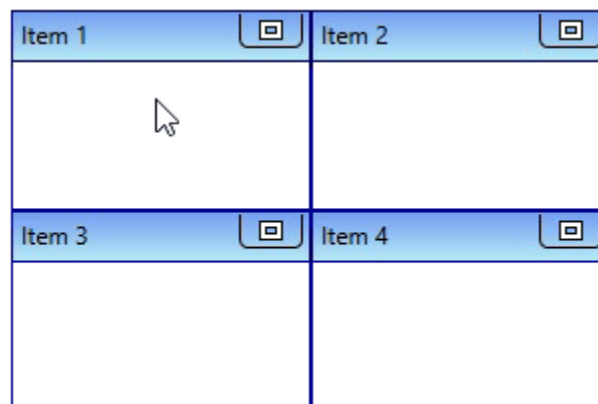
Note: [View Sample in GitHub](#)

Change TileViewItem header cursor

If you want to change specific `TileViewItem` header's mouse hover cursor, use the `TileViewItem.HeaderCursor` property. The default value of `TileViewItem.HeaderCursor` property is `Cursors.Hand`.

C#

```
TileViewControl tileViewControl = new TileViewControl();  
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",  
HeaderCursor = Cursors.UpArrow });  
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",  
HeaderCursor = Cursors.ArrowCD });  
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",  
HeaderCursor = Cursors.Cross });  
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",  
HeaderCursor = Cursors.Help });
```



Custom appearance of TileViewItem header

You can change the foreground, background and border appearance of the `TileViewItem` header.

Change foreground for TileViewItem header

You can change the foreground color of the each `TileViewItem` header separately by using the [TileViewItem.HeaderForeground](#) property. The default value of `TileViewItem.HeaderForeground` property is `Black`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem HeaderForeground="Red" Header="Item1" />
<syncfusion:TileViewItem HeaderForeground="Blue" Header="Item2" />
<syncfusion:TileViewItem HeaderForeground="Green" Header="Item3" />
<syncfusion:TileViewItem HeaderForeground="Yellow" Header="Item4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
HeaderForeground = Brushes.Red });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
HeaderForeground = Brushes.Blue});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
HeaderForeground = Brushes.Green });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
HeaderForeground = Brushes.Yellow });
```



Note: [View Sample in GitHub](#)

Change background for TileViewItem header

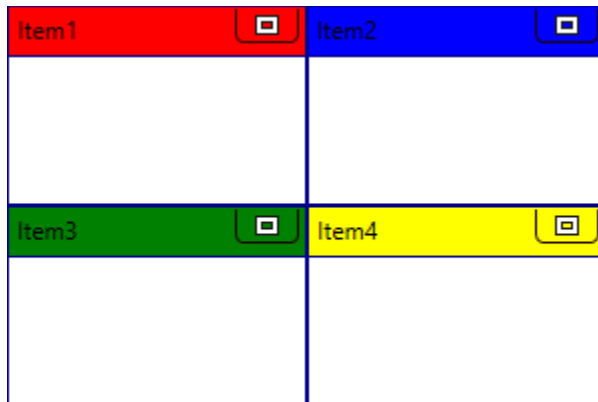
You can change the background color of the each `TileViewItem` header separately by using the [TileViewItem.HeaderBackground](#) property. The default value of `TileViewItem.HeaderBackground` property is `Cornflower Blue`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem HeaderBackground="Red" Header="Item1"/>
<syncfusion:TileViewItem HeaderBackground="Blue" Header="Item2"/>
<syncfusion:TileViewItem HeaderBackground="Green" Header="Item3"/>
<syncfusion:TileViewItem HeaderBackground="Yellow" Header="Item4"/>
</syncfusion:TileViewControl>
```


C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
HeaderBackground = Brushes.Red });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
HeaderBackground = Brushes.Blue});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
HeaderBackground = Brushes.Green });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
HeaderBackground = Brushes.Yellow });
```



Note: [View Sample in GitHub](#)

Change border for TileViewItem header

You can change the border color of the each `TileViewItem` header separately by using the `TileViewItem.HeaderBorderBrush` property. You can also change the header border thickness by using the `TileViewItem.HeaderBorderThickness` property. The default value of `TileViewItem.HeaderBorderBrush` property is Dark Blue and `TileViewItem.HeaderBorderThickness` property is `{0,0,0,1}`.

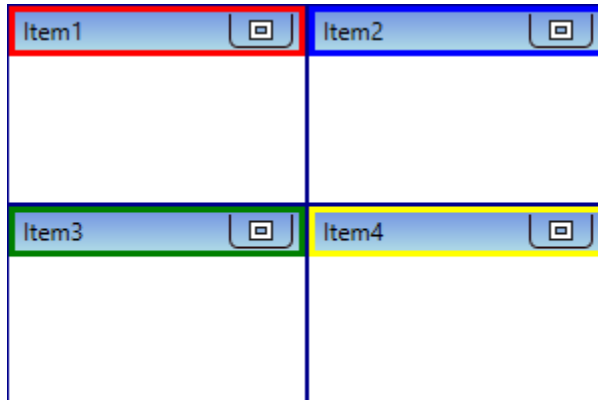
XML

```
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem HeaderBorderThickness="3" Header="Item1"
HeaderBorderBrush="Red"/>
<syncfusion:TileViewItem HeaderBorderThickness="3" Header="Item2"
HeaderBorderBrush="Blue"/>
<syncfusion:TileViewItem HeaderBorderThickness="3" Header="Item3"
HeaderBorderBrush="Green"/>
<syncfusion:TileViewItem HeaderBorderThickness="3" Header="Item4"
HeaderBorderBrush="Yellow"/>
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
HeaderBorderBrush = Brushes.Red, HeaderBorderThickness= new Thickness(3) });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
```

```
HeaderBorderBrush = Brushes.Blue, HeaderBorderThickness= new Thickness(3));
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
HeaderBorderBrush = Brushes.Green, HeaderBorderThickness= new
Thickness(3)});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
HeaderBorderBrush = Brushes.Yellow, HeaderBorderThickness= new
Thickness(3)});
```



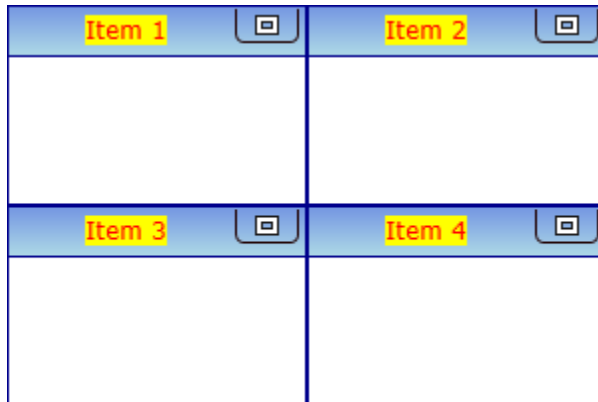
Note: [View Sample in GitHub](#)

Custom UI of TileViewItem header

You can customize the appearance of TileViewItem headers by using the [HeaderTemplate](#) property. The [DataContext](#) of the [HeaderTemplate](#) property is [TileViewItem.Header](#).

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item 1" />
  <syncfusion:TileViewItem Header="Item 2" />
  <syncfusion:TileViewItem Header="Item 3" />
  <syncfusion:TileViewItem Header="Item 4" />
  <syncfusion:TileViewControl.HeaderTemplate>
    <DataTemplate x:Name="headerTemplate">
      <Grid>
        <TextBlock HorizontalAlignment="Center"
          Text="{Binding}"
          FontFamily="Verdana"
          Background="Yellow"
          Foreground="Red" />
      </Grid>
    </DataTemplate>
  </syncfusion:TileViewControl.HeaderTemplate>
</syncfusion:TileViewControl>
```



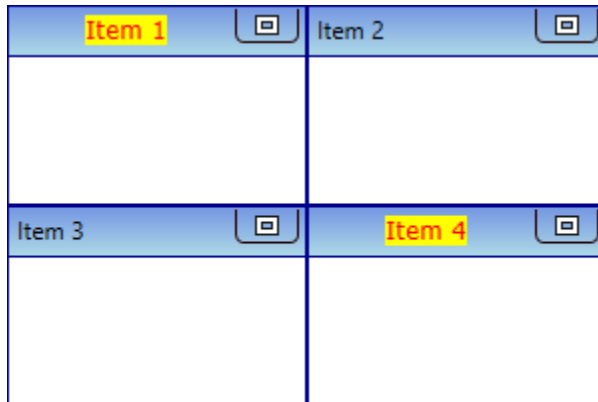
Note: [View Sample in GitHub](#)

Custom UI of specific TileViewItem header

You can customize the appearance of specific `TileViewItem` headers by using the `TileViewItem.HeaderTemplate` property. The `DataContext` of the `TileViewItem.HeaderTemplate` property is `TileViewItem.Header`.

XML

```
<Window.Resources>
<DataTemplate x:Key="headerTemplate">
<Grid>
<TextBlock HorizontalAlignment="Center"
Text="{Binding}"
FontFamily="Verdana"
Background="Yellow"
Foreground="Red" />
</Grid>
</DataTemplate>
</Window.Resources>
<Grid>
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Header="Item 1"
HeaderTemplate="{StaticResource headerTemplate}" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4"
HeaderTemplate="{StaticResource headerTemplate}" />
</syncfusion:TileViewControl>
</Grid>
```



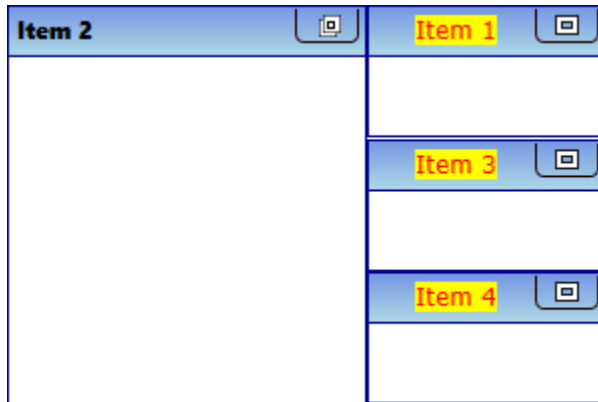
Note: [View Sample in GitHub](#)

Custom UI of minimized TileViewItem header

You can customize the appearance of minimized TileViewItem headers by using the [MinimizedHeaderTemplate](#) property. The `DataContext` of the `MinimizedHeaderTemplate` property is `TileViewItem.MinimizedHeader`.

XML

```
<Window.Resources>
<DataTemplate x:Key="minimizedHeaderTemplate">
<Grid>
<TextBlock HorizontalAlignment="Center"
Text="{Binding}"
FontFamily="Verdana"
Background="Yellow"
Foreground="Red"/>
</Grid>
</DataTemplate>
</Window.Resources>
<Grid>
<syncfusion:TileViewControl MinimizedHeaderTemplate="{StaticResource
minimizedHeaderTemplate}"
Name="tileViewControl">
<syncfusion:TileViewItem Header="Item 1" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
</Grid>
```



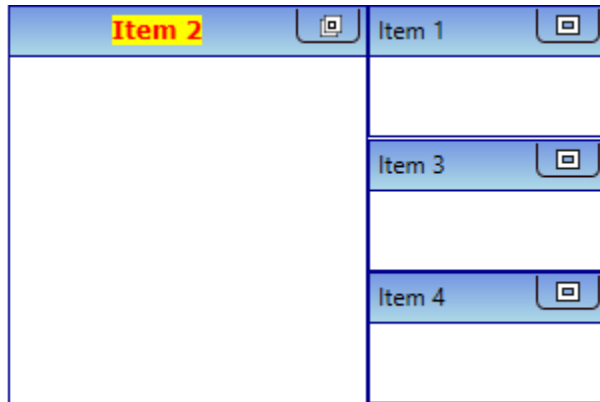
Note: [View Sample in GitHub](#)

Custom UI of maximized TileViewItem header

You can customize the appearance of maximized TileViewItem headers by using the [MaximizedHeaderTemplate](#) property. The `DataContext` of the `MaximizedHeaderTemplate` property is `TileViewItem.MaximizedHeader`.

XML

```
<Window.Resources>
<DataTemplate x:Key="maximizedHeaderTemplate">
<Grid>
<TextBlock HorizontalAlignment="Center"
Text="{Binding}"
FontFamily="Verdana"
Background="Yellow"
Foreground="Red"/>
</Grid>
</DataTemplate>
</Window.Resources>
<Grid>
<syncfusion:TileViewControl MaximizedHeaderTemplate="{StaticResource
maximizedHeaderTemplate}"
Name="tileViewControl">
<syncfusion:TileViewItem Header="Item 1" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
</Grid>
```



Note: [View Sample in GitHub](#)

Minimize TileViewItem in WPF Tile View

You can minimize the [TileViewItem](#) and change its appearance in the [TileViewControl](#).

Minimize the TileViewItem

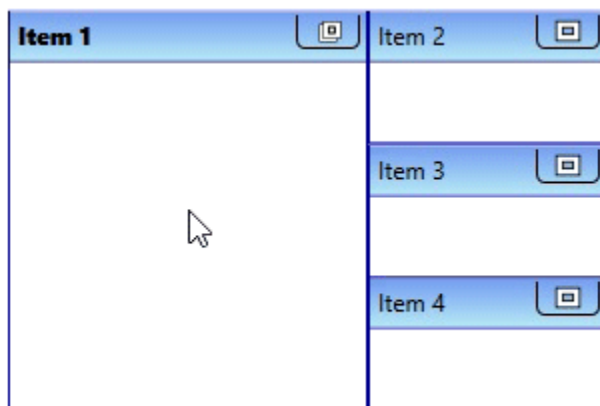
You can minimize the maximized [TileViewItem](#) by clicking on the minimize button.

XML

```
<syncfusion:TileViewControl Name="tileViewControl" >
  <syncfusion:TileViewItem Header="Item 1" />
  <syncfusion:TileViewItem Header="Item 2" />
  <syncfusion:TileViewItem Header="Item 3" />
  <syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4" });
```



Note: [View Sample in GitHub](#)

Direction for minimized items

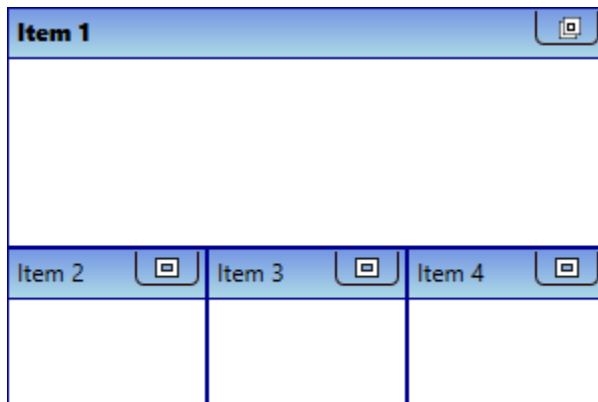
If you want to change direction of placing the minimized items, use the [MinimizedItemsOrientation](#) property. The default value of `MinimizedItemsOrientation` property is `Right`.

XML

```
<syncfusion:TileViewControl MinimizedItemsOrientation="Bottom"
Name="tileViewControl" >
<syncfusion:TileViewItem Header="Item 1" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.MinimizedItemsOrientation =
MinimizedItemsOrientation.Bottom;
```



Note: [View Sample in GitHub](#)

Allocate size for minimized TileViewItem

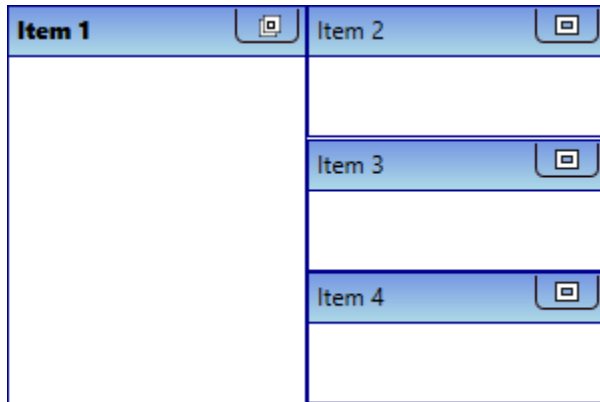
If you want to allocate a certain percentage of the total size to the minimized items, use the [MinimizedItemsPercentage](#) property. The default value of `MinimizedItemsPercentage` property is `20`.

XML

```
<syncfusion:TileViewControl MinimizedItemsPercentage="50"
Name="tileViewControl" >
<syncfusion:TileViewItem Header="Item 1" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.MinimizedItemsPercentage = 50;
```



Note: [View Sample in GitHub](#)

Change minimized TileViewItem content

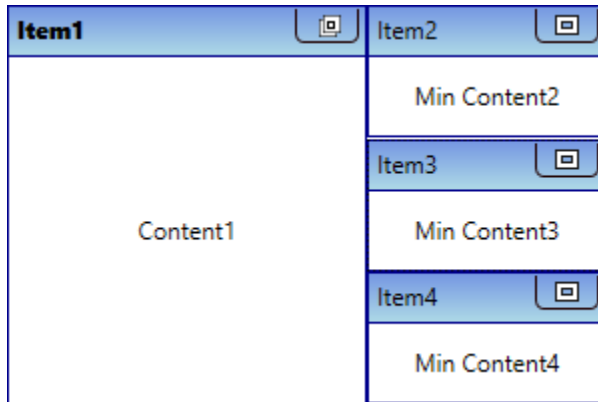
By default, `TileViewItem.Content` property values displayed as `TileViewItem` content on minimized state. If you want to change the content of the `TileViewItem` on minimized state, use the [MinimizedItemContent](#) property. The default value of `MinimizedItemContent` property is `null`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item1" Content="Content1"
    MinimizedItemContent="Min Content1" />
  <syncfusion:TileViewItem Header="Item2" Content="Content2"
    MinimizedItemContent="Min Content2" />
  <syncfusion:TileViewItem Header="Item3" Content="Content3"
    MinimizedItemContent="Min Content3" />
  <syncfusion:TileViewItem Header="Item4" Content="Content4"
    MinimizedItemContent="Min Content4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
  Content = "Content1", MinimizedItemContent = "Min Content1" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
  Content = "Content2", MinimizedItemContent = "Min Content2" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
  Content = "Content3", MinimizedItemContent = "Min Content3" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
  Content = "Content4", MinimizedItemContent = "Min Content4" });
```

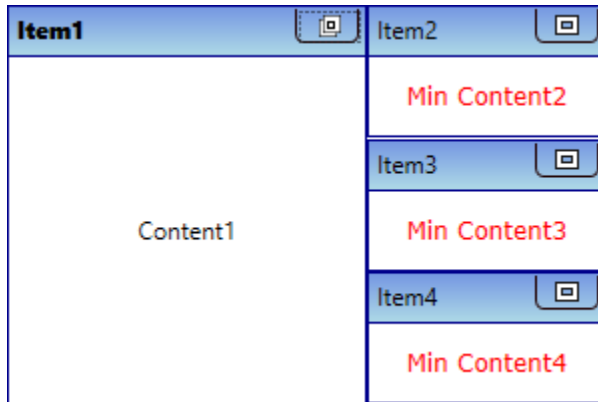
Note: [View Sample in GitHub](#)

Custom UI of minimized TileViewItem content

You can customize the appearance of minimized TileViewItem content by using the [MinimizedItemTemplate](#) property. The DataContext of the MinimizedItemTemplate property is TileViewItem.MinimizedItemContent.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item1" Content="Content1"
    MinimizedItemContent="Min Content1" />
  <syncfusion:TileViewItem Header="Item2" Content="Content2"
    MinimizedItemContent="Min Content2" />
  <syncfusion:TileViewItem Header="Item3" Content="Content3"
    MinimizedItemContent="Min Content3" />
  <syncfusion:TileViewItem Header="Item4" Content="Content4"
    MinimizedItemContent="Min Content4" />
  <syncfusion:TileViewControl.MinimizedItemTemplate>
    <DataTemplate x:Name="MinTemplate">
      <Grid>
        <TextBlock HorizontalAlignment="Center"
          Text="{Binding}"
          FontFamily="Verdana"
          Foreground="Red" />
      </Grid>
    </DataTemplate>
  </syncfusion:TileViewControl.MinimizedItemTemplate>
</syncfusion:TileViewControl>
```



Note: [View Sample in GitHub](#)

Change minimized TileViewItem header

If you want to change the header of the `TileViewItem` on minimized state, use the [MinimizedHeader](#) property. The default value of `MinimizedHeader` property is `null`.

Note: Please refer [Minimized TileViewItem header](#) topic to know more details about minimized `TileViewItem` header and its customization available in the `TileViewControl`.

Minimized state changed notification

The `TileViewControl` notifies that the minimized state changed in the `TileViewItem` by using [Minimized](#) event. You can get the minimized items by using the [Source](#) property. You can also use the [OldState](#) and [NewState](#) properties to get the old and new state of `TileViewItem`.

XML

```
<syncfusion:TileViewControl Minimized="TileViewControl_Minimized"
Name="tileViewControl" />
```

C#

```
tileViewControl.Minimized += TileViewControl_Minimized;
```

You can handle the event as follows:

C#

```
private void TileViewControl_Minimized(object sender, TileViewEventArgs
args) {
    var minimizedItem = args.Source;
    var oldState = args.OldState;
    var newState = args.NewState;
}
```

Maximize TileViewItem in WPF Tile View

You can maximize the [TileViewItem](#) and change its appearance in the [TileViewControl](#).

Maximize the TileViewItem

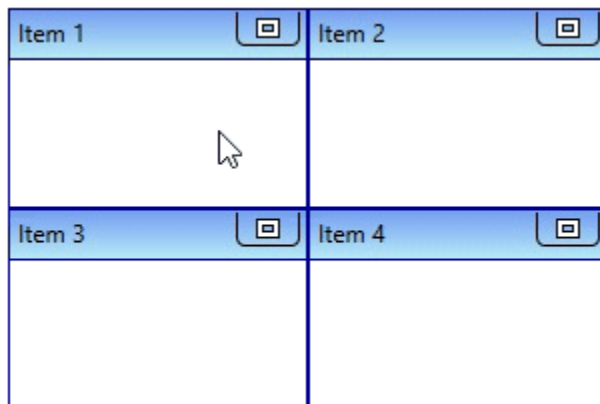
You can maximize the `TileViewItem` by clicking on the maximize button.

XML

```
<syncfusion:TileViewControl Name="tileViewControl" >
<syncfusion:TileViewItem Header="Item 1" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4" });
```



Note: [View Sample in GitHub](#)

Maximize on click the header

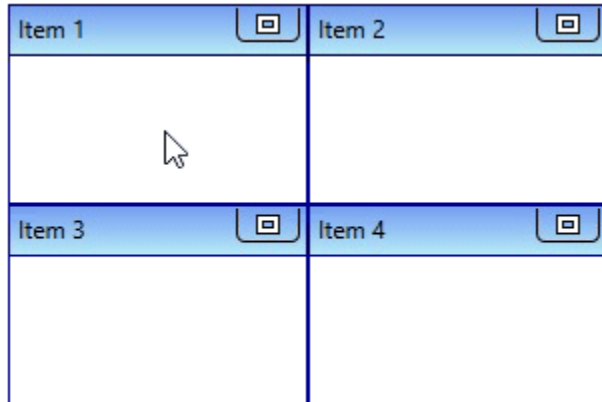
If you want to maximize the `TileViewItem` on clicking the header, use the [ClickHeaderToMaximize](#) property value as `true`. The default value of `ClickHeaderToMaximize` property is `false`.

XML

```
<syncfusion:TileViewControl ClickHeaderToMaximize="True"
Name="tileViewControl" >
<syncfusion:TileViewItem Header="Item 1" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.ClickHeaderToMaximize = true;
```



Note: [View Sample in GitHub](#)

Show maximize button only on mouse hover

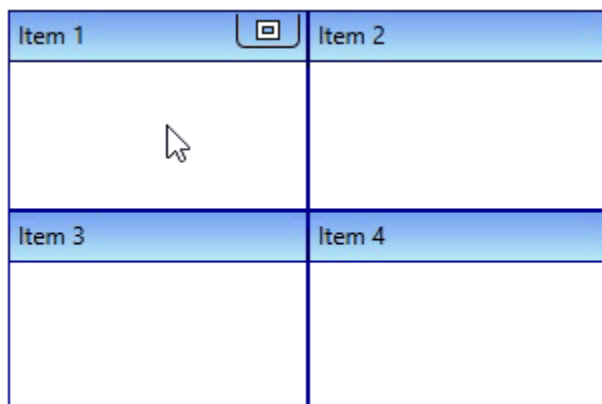
If you want to show the maximize button only by mouse hover on the particular `TileViewItem`, use the `IsMinMaxButtonOnMouseOverOnly` property value as `true`. The default value of `IsMinMaxButtonOnMouseOverOnly` property is `false`.

XML

```
<syncfusion:TileViewControl IsMinMaxButtonOnMouseOverOnly="True"
Name="tileViewControl" >
<syncfusion:TileViewItem Header="Item 1" />
<syncfusion:TileViewItem Header="Item 2" />
<syncfusion:TileViewItem Header="Item 3" />
<syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.IsMinMaxButtonOnMouseOverOnly = true;
```



Note: [View Sample in GitHub](#)

Hide maximize button

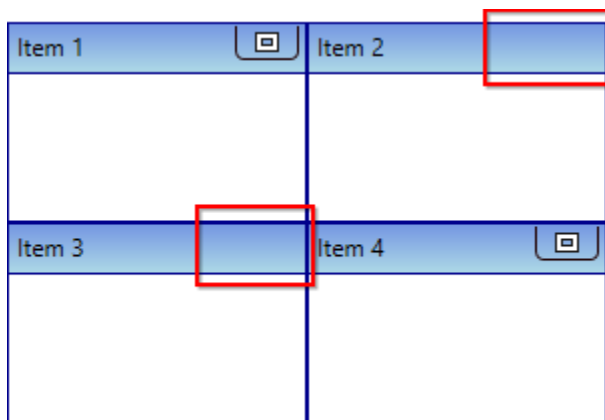
You can hide the maximize button for the specific `TileViewItem` by using the [TileViewItem.MinMaxButtonVisibility](#) property value as `Collapsed`. The default value of `TileViewItem.MinMaxButtonVisibility` property is `Visible`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl" >
  <syncfusion:TileViewItem Header="Item 1" />
  <syncfusion:TileViewItem Header="Item 2"
  MinMaxButtonVisibility="Collapsed" />
  <syncfusion:TileViewItem Header="Item 3"
  MinMaxButtonVisibility="Collapsed"/>
  <syncfusion:TileViewItem Header="Item 4" />
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
MinMaxButtonVisibility= Visibility.Collapsed });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
MinMaxButtonVisibility = Visibility.Collapsed });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4" });
```



Note: [View Sample in GitHub](#)

Custom UI of the maximize button

You can customize the appearance of particular `TileViewItem`'s maximize button by using the [MinMaxButtonStyle](#) property. You can also change the margin of the `TileViewItem` maximize button by using the [MinMaxButtonMargin](#) property. The `DataContext` of the `MinMaxButtonStyle` property is [TileViewItemMinMaxButton](#).

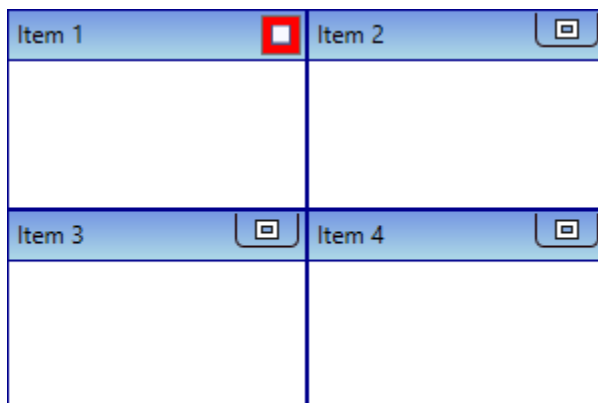
XML

```
<Window.Resources>
  <Style x:Key="tileViewItemMinMaxButton"
  TargetType="syncfusion:TileViewItemMinMaxButton">
    <Setter Property="Background" Value="Red"/>
  </Style>
</Window.Resources>
```

```

<Setter Property="Width" Value="20"/>
<Setter Property="Height" Value="20"/>
<Setter Property="Content">
<Setter.Value>
<Border Background="Red">
<Image Source="/maximize.png" />
</Border>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
<Grid>
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Header="Item 1"
MinMaxButtonMargin="2"
MinMaxButtonStyle="{StaticResource tileViewItemMinMaxButton}"/>
<syncfusion:TileViewItem Header="Item 2"/>
<syncfusion:TileViewItem Header="Item 3"/>
<syncfusion:TileViewItem Header="Item 4"/>
</syncfusion:TileViewControl>
</Grid>

```



Note: [View Sample in GitHub](#)

Change maximized TileViewItem content

By default, `TileViewItem.Content` property values displayed as `TileViewItem` content on maximized state. If you want to change the content of the `TileViewItem` on maximized state, use the [MaximizedItemContent](#) property. The default value of `MaximizedItemContent` property is `null`.

XML

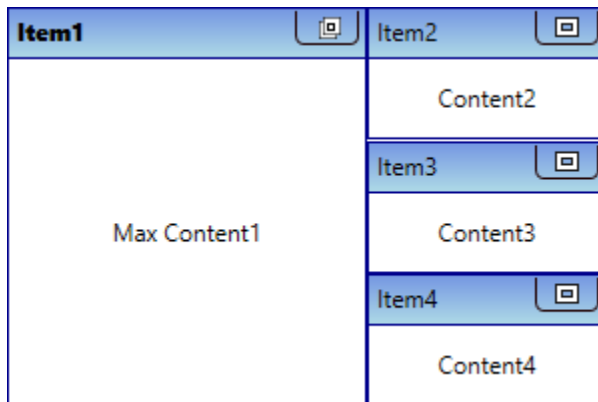
```

<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem Header="Item1" Content="Content1"
MaximizedItemContent="Max Content1" />
<syncfusion:TileViewItem Header="Item2" Content="Content2"
MaximizedItemContent="Max Content2" />
<syncfusion:TileViewItem Header="Item3" Content="Content3"
MaximizedItemContent="Max Content3" />
<syncfusion:TileViewItem Header="Item4" Content="Content4"
MaximizedItemContent="Max Content4" />
</syncfusion:TileViewControl>

```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
Content = "Content1", MaximizedItemContent = "Max Content1" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
Content = "Content2", MaximizedItemContent = "Max Content2" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
Content = "Content3", MaximizedItemContent = "Max Content3" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
Content = "Content4", MaximizedItemContent = "Max Content4" });
```



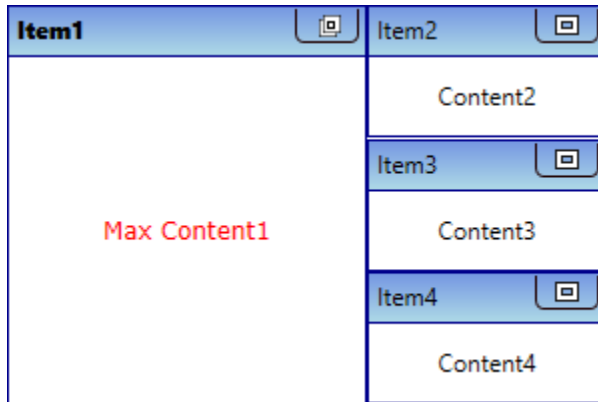
Note: [View Sample in GitHub](#)

Custom UI of maximized TileViewItem content

You can customize the appearance of maximized TileViewItem content by using the [MaximizedItemTemplate](#) property. The DataContext of the MaximizedItemTemplate property is TileViewItem.MaximizedItemContent.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Header="Item1" Content="Content1"
  MaximizedItemContent="Max Content1" />
  <syncfusion:TileViewItem Header="Item2" Content="Content2"
  MaximizedItemContent="Max Content2" />
  <syncfusion:TileViewItem Header="Item3" Content="Content3"
  MaximizedItemContent="Max Content3" />
  <syncfusion:TileViewItem Header="Item4" Content="Content4"
  MaximizedItemContent="Max Content4" />
  <syncfusion:TileViewControl.MaximizedItemTemplate>
    <DataTemplate x:Name="MaxTemplate">
      <Grid>
        <TextBlock HorizontalAlignment="Center"
        Text="{Binding}"
        FontFamily="Verdana"
        Foreground="Red"/>
      </Grid>
    </DataTemplate>
  </syncfusion:TileViewControl.MaximizedItemTemplate>
</syncfusion:TileViewControl>
```



Note: [View Sample in GitHub](#)

Change maximized TileViewItem header

If you want to change the header of the `TileViewItem` on maximized state, use the [MaximizedHeader](#) property. The default value of `MaximizedHeader` property is `null`.

Note: Please refer [Maximized TileViewItem header](#) topic to know more details about maximized `TileViewItem` header and its customization available in the `TileViewControl`.

Maximized state changed notification

The `TileViewControl` notifies that the maximized state changed in the `TileViewItem` by using [Maximized](#) event. You can get the maximized item by using the [Source](#) property. You can also use the [OldState](#) and [NewState](#) properties to get the old and new state of `TileViewItem`.

XML

```
<syncfusion:TileViewControl Maximized="TileViewControl_Maximized"
Name="tileViewControl" />
```

C#

```
tileViewControl.Maximized += TileViewControl_Maximized;
```

You can handle the event as follows:

C#

```
private void TileViewControl_Maximized(object sender, TileViewEventArgs
args) {
    var maximizedItem = args.Source;
    var oldState = args.OldState;
    var newState = args.NewState;
}
```

Appearance in WPF Tile View

This section explains different styling, theming options available in [TileViewControl](#) control.

Setting the foreground

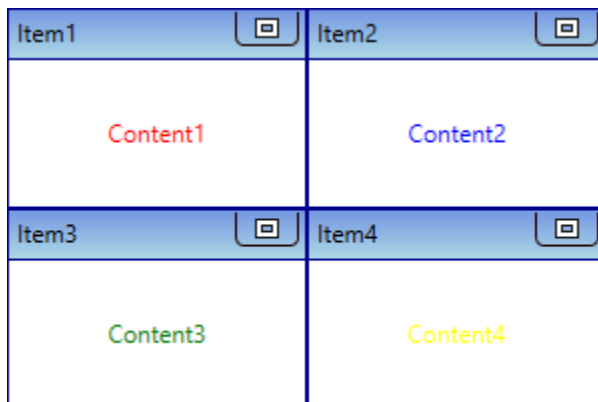
You can change the foreground color of the each `TileViewItem` separately by using the `TileViewItem.Foreground` property. The default value of `TileViewItem.Foreground` property is Black.

XML

```
<syncfusion:TileViewControl Name="tileViewControl">
  <syncfusion:TileViewItem Foreground="Red" Header="Item1"
  Content="Content1"/>
  <syncfusion:TileViewItem Foreground="Blue" Header="Item2"
  Content="Content2"/>
  <syncfusion:TileViewItem Foreground="Green" Header="Item3"
  Content="Content3"/>
  <syncfusion:TileViewItem Foreground="Yellow" Header="Item4"
  Content="Content4"/>
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
Content = "Content1", Foreground = Brushes.Red });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
Content = "Content2", Foreground = Brushes.Blue});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
Content = "Content3", Foreground = Brushes.Green });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
Content = "Content4", Foreground = Brushes.Yellow });
```



Note: [View Sample in GitHub](#)

Setting the background

You can change the background color of the each `TileViewItem` separately by using the `TileViewItem.Background` property. The default value of `TileViewItem.Background` property is White.

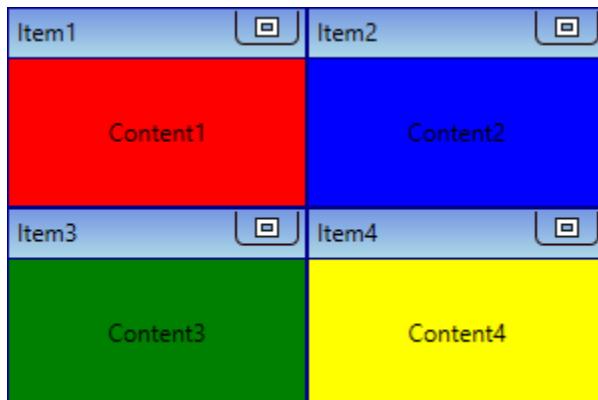
XML

```
<syncfusion:TileViewControl Name="tileViewControl">
```

```
<syncfusion:TileViewItem Background="Red" Header="Item1"
Content="Content1"/>
<syncfusion:TileViewItem Background="Blue" Header="Item2"
Content="Content2"/>
<syncfusion:TileViewItem Background="Green" Header="Item3"
Content="Content3"/>
<syncfusion:TileViewItem Background="Yellow" Header="Item4"
Content="Content4"/>
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
Content = "Content1", Background = Brushes.Red });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
Content = "Content2", Background = Brushes.Blue});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
Content = "Content3", Background = Brushes.Green });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
Content = "Content4", Background = Brushes.Yellow });
```



Note: [View Sample in GitHub](#)

Setting the border

You can change the border color of the each `TileViewItem` separately by using the `TileViewItem.BorderBrush` property. You can also change the border thickness by using the `TileViewItem.BorderThickness` property. The default value of `TileViewItem.BorderBrush` property is Dark Blue and `TileViewItem.BorderThickness` property is 1.

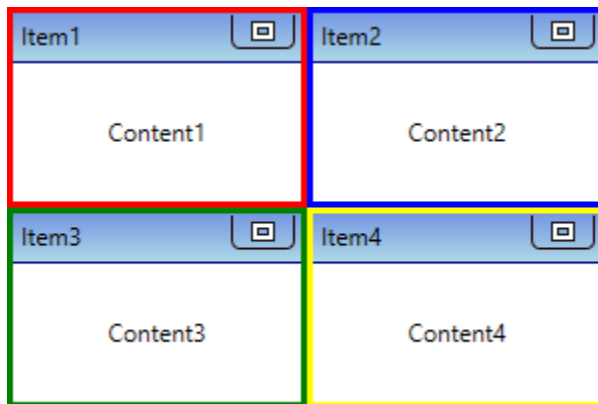
XML

```
<syncfusion:TileViewControl Name="tileViewControl">
<syncfusion:TileViewItem BorderThickness="3" BorderBrush="Red"
Header="Item1" Content="Content1"/>
<syncfusion:TileViewItem BorderThickness="3" BorderBrush="Blue"
Header="Item2" Content="Content2"/>
<syncfusion:TileViewItem BorderThickness="3" BorderBrush="Green"
Header="Item3" Content="Content3"/>
<syncfusion:TileViewItem BorderThickness="3" BorderBrush="Yellow"
Header="Item4" Content="Content4"/>
```

```
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1", Content =
"Content1",
BorderBrush = Brushes.Red, BorderThickness= new Thickness(3)});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2", Content =
"Content2",
BorderBrush = Brushes.Blue, BorderThickness= new Thickness(3)});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3", Content =
"Content3",
BorderBrush = Brushes.Green, BorderThickness= new Thickness(3)});
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4", Content =
"Content4",
BorderBrush = Brushes.Yellow, BorderThickness= new Thickness(3)});
```



Note: [View Sample in GitHub](#)

Change flow direction





You can change the flow direction of the `TileViewControl` layout from right to left by setting the `TileViewControl.FlowDirection` property value as `RightToLeft`. The default value of `TileViewControl.FlowDirection` property is `LeftToRight`.

XML

```
<syncfusion:TileViewControl FlowDirection="RightToLeft"
Name="tileViewControl" >
<syncfusion:TileViewItem Header="Item1" Content="Content1"/>
<syncfusion:TileViewItem Header="Item2" Content="Content2"/>
<syncfusion:TileViewItem Header="Item3" Content="Content3"/>
<syncfusion:TileViewItem Header="Item4" Content="Content4"/>
</syncfusion:TileViewControl>
```

C#

```
tileViewControl.FlowDirection = FlowDirection.RightToLeft;
```

 Item2	 Item1
Content2	Content1
 Item4	 Item3
Content4	Content3

Note: [View Sample in GitHub](#)

Change flow direction for specific TileViewItem

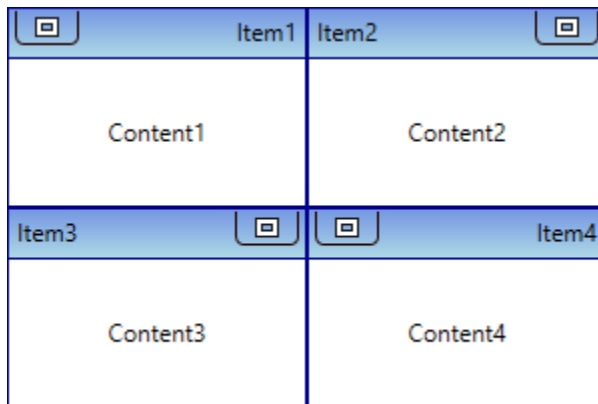
You can change the flow direction of the specific `TileViewItem` layout from right to left by setting the `TileViewItem.FlowDirection` property value as `RightToLeft`. The default value of `TileViewItem.FlowDirection` property is `LeftToRight`.

XML

```
<syncfusion:TileViewControl Name="tileViewControl" />
<syncfusion:TileViewItem FlowDirection="RightToLeft" Header="Item1"
Content="Content1"/>
<syncfusion:TileViewItem Header="Item2" Content="Content2"/>
<syncfusion:TileViewItem Header="Item3" Content="Content3"/>
<syncfusion:TileViewItem FlowDirection="RightToLeft" Header="Item4"
Content="Content4"/>
</syncfusion:TileViewControl>
```

C#

```
TileViewControl tileViewControl = new TileViewControl();
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 1",
Content = "Content1", FlowDirection = FlowDirection.RightToLeft });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 2",
Content = "Content2" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 3",
Content = "Content3" });
tileViewControl.Items.Add(new TileViewItem() { Header = "Item 4",
Content = "Content4", FlowDirection = FlowDirection.RightToLeft });
```

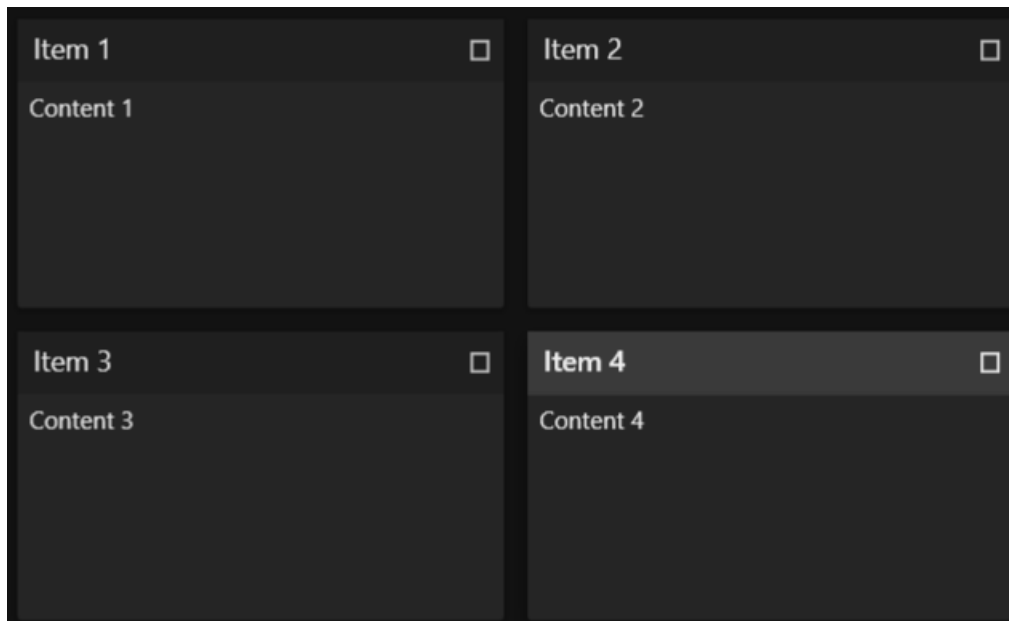


Note: [View Sample in GitHub](#)

Theme

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

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

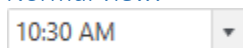


SfTimePicker

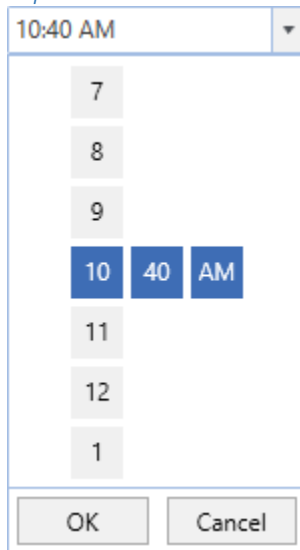
WPF TimePicker (SfTimePicker) Overview

The [SfTimePicker](#) control allows the user to select time values in a touch friendly manner.

Normal view:



Expanded view:



Key Features

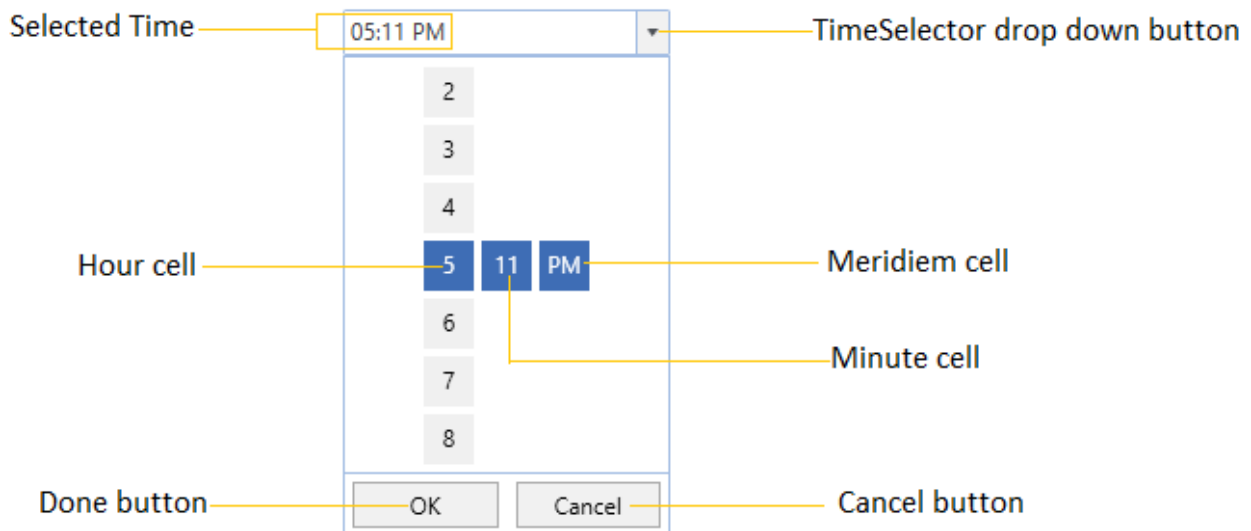
Formatting – The Control displays the selected time value in a various formats.

Time Selector – The drop-down portion used for selecting the time can be customized.

Getting Started with WPF TimePicker (SfTimePicker)

This section explains how to create a [WPF TimePicker](#) (SfTimePicker) and explains about its structure.

Structure of SfTimePicker



Assembly deployment

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

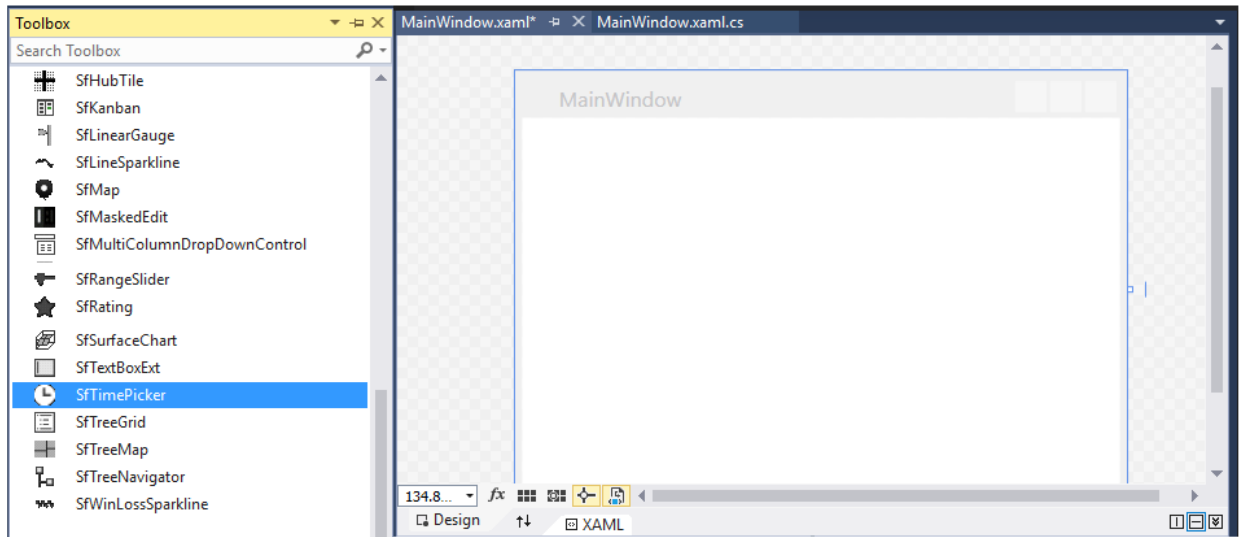
You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

Add control through designer

The [WPF TimePicker](#) (SfTimePicker) control can be added to an application by dragging it from the toolbox to a designer view. The following required assembly references will be added automatically:

- Syncfusion.SfInput.WPF
- Syncfusion.SfShared.WPF



Adding control manually in XAML

To add the control manually in XAML, follow the given steps:

1. Add the following required assembly references to the project: *Syncfusion.SfInput.WPF* and *Syncfusion.SfShared.WPF*
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the **SfTimePicker** 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="SfTimePickerSample.MainWindow"
Title="SfTimePicker Sample" Height="350" Width="525">
<Grid>
<!-- Adding SfTimePicker control -->
<syncfusion:SfTimePicker x:Name="sfTimePicker"
Width="200"/>
</Grid>
</Window>
```

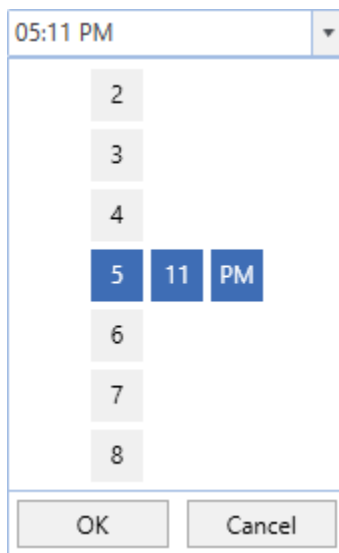
Add control manually in C\#

To add the control manually in C#, follow the given steps:

1. Add the following required assembly references to the project: *Syncfusion.SfInput.WPF* and *Syncfusion.SfShared.WPF*
2. Import the **SfTimePicker** namespace using **Syncfusion.Windows.Controls.Input**;
3. Create an **SfTimePicker** instance, and add it to the window.

C#

```
using Syncfusion.Windows.Controls.Input;
namespace SfTimePickerSample {
public partial class MainWindow : Window {
public MainWindow() {
InitializeComponent();
//Creating an instance of SfTimePicker control
SfTimePicker sfTimePicker = new SfTimePicker();
//Adding SfTimePicker as window content
this.Content = sfTimePicker;
}
}
}
```

**Setting the time**

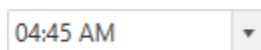
We can set or change the selected time by using [Value](#) property. If we not assign any value for the [Value](#) property, it will automatically assign the current system time as [Value](#) property value.

XML

```
<syncfusion:SfTimePicker Value="04:45:00"
Name="sfTimePicker" />
```

C#

```
SfTimePicker sfTimePicker= new SfTimePicker();
sfTimePicker.Value = new TimeSpan(04, 45, 00);
```

**Time changed notification**

When the selected time of [SfTimePicker](#) is changed, it will be notified by using the [ValueChanged](#) event. You can get the details about the checked item in [ItemCheckedEventArgs](#).

- **OldValue** : Gets a time which is previously selected.
- **newValue** : Gets a time which is currently selected.

XML

```
<syncfusion:SfTimePicker ValueChanged="SftimePicker_ValueChanged"
Name="sfTimePicker"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.ValueChanged += SftimePicker_ValueChanged;
```

You can handle the event as follows:

C#

```
private void SftimePicker_ValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
Console.WriteLine("The Old selected time: " + e.OldValue.ToString());
Console.WriteLine("The Newly selected time: " + e.NewValue.ToString());
}
```

Display the time using the `FormatString`

We can edit and display the selected time with various formatting like short time, long time, universal time and 24 hour time formats by using the `FormatString` property. The default value of `FormatString` property is "h:mm tt".

XML

```
<syncfusion:SfTimePicker x:Name="sfTimePicker"
FormatString="HH:mm:ss"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.FormatString = "HH:mm:ss";
```



Here, `SfTimePicker` with 24 hour time format

Specifying format for the `TimeSelector`

We can allow the user to select the pair of hour, minutes, seconds and meridiem selector or any single selector cell from the `SfTimeSelector` by using the `SelectorFormatString` property. The default value of `SelectorFormatString` property is "h:mm tt" and the hour, minutes and meridiem value selector is enabled in the `SfTimeSelector`.

XML

```
<syncfusion:SfTimePicker x:Name="sfTimePicker"
SelectorFormatString="h/t"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();  
sfTimePicker.SelectorFormatString = "h/t";
```



Here, the **SfTimeSelector** with only hour and meridiem value selector.

Click [here](#) to download the sample that showcases the edit, display time formatting and time selection formatting by the **SfTimePicker**.

Set selected value on lost focus

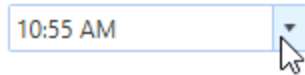
If we want to update the selected time of **SfTimeSelector** to the **SfTimeSelector.Value** property by moving the focus from **SfTimeSelector** to anywhere, use the [SetValueOnLostFocus](#) property value as **true**. By default, the selected time of **SfTimeSelector** can be sets to the **SfTimeSelector.Value** property only by clicking the **OK** button, otherwise the selected value not updated by the move focus.

XML

```
<syncfusion:SfTimePicker SetValueOnLostFocus="True"  
Name="sfTimePicker" />
```

C#

```
SfTimePicker sfTimePicker= new SfTimePicker();  
sfTimePicker.SetValueOnLostFocus = true;
```



Click [here](#) to download the sample that showcases the value setting support in the SfTimePicker.

Localization support

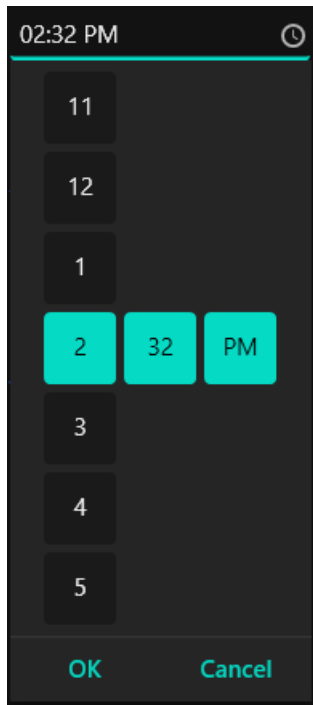
Localization is the process of translating the application resources into different language for the specific cultures. You can localize the **Ok** and **Cancel** button text in SfTimePicker control by adding resource file for each language.

Note: Refer [Localization](#) page to know more about how to provide a localization support for the SfTimePicker.

Theme

WPF TimePicker (SfTimePicker) supports various built-in themes. Refer to the below links to apply themes for the SfTimePicker,

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



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

Setting Time in WPF TimePicker (SfTimePicker)

We can change the value of [SfTimePicker](#) by using the [SfTimeSelector](#) and keyboard interaction.

Setting Time using property

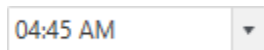
We can set or change the selected time by using [Value](#) property. If we not assign any value for the [Value](#) property, it will automatically assign the current system time as [Value](#) property value.

XML

```
<syncfusion:SfTimePicker Value="04:45:00"
Name="sfTimePicker" />
```

C#

```
SfTimePicker sfTimePicker= new SfTimePicker();
sfTimePicker.Value = new TimeSpan(04, 45, 00);
```



Setting Null Value

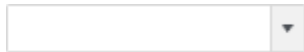
If we want to set null value for the [SfTimePicker](#), set the [AllowNull](#) property as [true](#) and [Value](#) property as [null](#). If [AllowNull](#) property is [false](#), then the current system time is updated in [Value](#) property and displayed instead of [null](#).

XML

```
<syncfusion:SfTimePicker AllowNull="True"
Value="{x:Null}"
Name="sfTimePicker" />
```

C#

```
SfTimePicker sfTimePicker= new SfTimePicker();
sfTimePicker.AllowNull = true;
sfTimePicker.Value = null;
```

**Setting WaterMark text**

We can prompt the user with some information by using the [Watermark](#) property. This will apply on when the SfTimePicker contains the [Value](#) property as `null` and [AllowNull](#) property as `true`. If [AllowNull](#) property is `false`, then the current system time is updated in [Value](#) property and displayed instead of [Watermark](#) text.

XML

```
<syncfusion:SfTimePicker Watermark="Select the Time"
AllowNull="True"
Value="{x:Null}"
Name="sfTimePicker" >
</syncfusion:SfTimePicker>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.Watermark = "Select the Time";
sfTimePicker.AllowNull = true;
sfTimePicker.Value = null;
```

**Setting WaterMark Template**

We can change the template of the [Watermark](#) by using the [WatermarkTemplate](#) property.

XML

```
<syncfusion:SfTimePicker Name="sfTimePicker"
AllowNull="True"
Value="{x:Null}"
Watermark="Select the Time" >
<syncfusion:SfTimePicker.WatermarkTemplate >
<DataTemplate>
<Border Background="Yellow">
<TextBlock Foreground="Blue"
FontWeight="Bold"
Text="{Binding}"
TextAlignment="Center"/>
</Border>
</DataTemplate>
</syncfusion:SfTimePicker.WatermarkTemplate>
</syncfusion:SfTimePicker>
```



Set selected value on lost focus

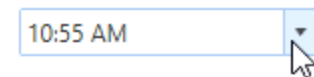
If we want to update the selected time of `SfTimeSelector` to the `SfTimeSelector.Value` property by moving the focus from `SfTimeSelector` to anywhere, use the [SetValueOnLostFocus](#) property value as `true`. By default, the selected time of `SfTimeSelector` can be sets to the `SfTimeSelector.Value` property only by clicking the `OK` button, otherwise the selected value not updated by the move focus.

XML

```
<syncfusion:SfTimePicker SetValueOnLostFocus="True"
Name="sfTimePicker" />
```

C#

```
SfTimePicker sfTimePicker= new SfTimePicker();
sfTimePicker.SetValueOnLostFocus = true;
```



Setting the time using editing

If we want to perform the validation after the user completely entering their time inputs, use the [AllowInlineEditing](#) property value as `true`. Then the entered time value is validated with the [FormatString](#) property value by pressing the `Enter` key or lost focus. If entered value is not suit with `FormatString` property, the selected time will be set as default format value.

By default, the user entering each input numbers are automatically validated with the `FormatString` formats and assigned the proper value for it, then it will move to next input part of the time format.

XML

```
<syncfusion:SfTimePicker Name="sfTimePicker"
AllowInlineEditing="True" />
```

C#

```
SfTimePicker sfTimePicker= new SfDatePicker();
sfTimePicker.AllowInlineEditing = true;
```

Selected Time : 2:16:04 PM

02:16 PM ▾

Setting the Input Scope for the On-Screen Keyboard

We can change the input type of the on-screen keyboard by using the [InputScope](#) property. When the [InputScope](#) property set to [Number](#), only the numeric keypad will be visible in the on-screen keyboard.

Note: The [AllowInlineEditing](#) property must be set to [True](#) for this property to take effect.

XML

```
<syncfusion:SfTimePicker Name="sfTimePicker"
AllowInlineEditing="True"
InputScope="Time"/>
```

C#

```
SfTimePicker sfTimePicker= new SfTimePicker();
sfTimePicker.AllowInlineEditing = true;
sfTimePicker.InputScope = InputScopeNameValue.Time;
```

07:24 PM ▾

Restrict selecting time limit

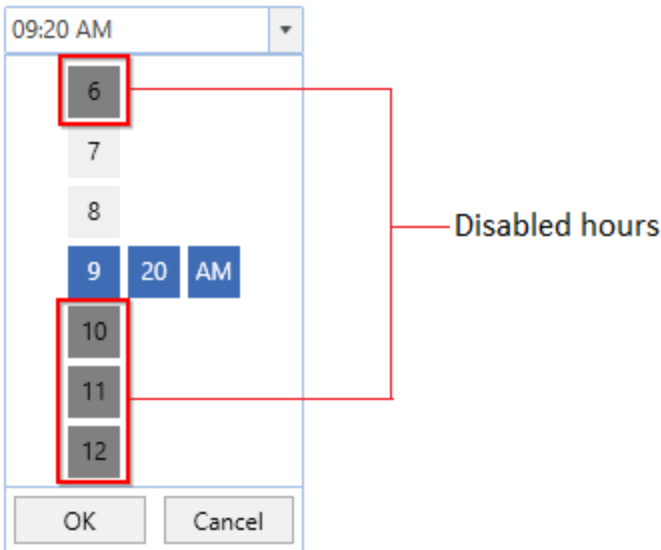
we can restrict the user to select a time in the specific time limit by setting the value for the [MinTime](#) and [MaxTime](#) properties. If we assign the value for the [Value](#) property lower than [MinTime](#), then [MinTime](#) will be the selected time. If we assign the value for the [Value](#) property higher than [MaxTime](#), then [MaxTime](#) will be the selected time.

XML

```
<syncfusion:SfTimePicker MinTime="07:00:00"
MaxTime="09:00:00"
Name="sfTimePicker"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.MinTime = new TimeSpan(07, 00, 00);
sfTimePicker.MaxTime = new TimeSpan(09, 00, 00);
```



Here, the users can select the hour from 7 to 9 only.

Time changed notification

When the selected time of `SfTimePicker` is changed, it will be notified by using the [ValueChanged](#) event. You can get the details about the checked item in [ItemCheckedEventArgs](#).

- **OldValue** : Gets a time which is previously selected.
- **NewValue** : Gets a time which is currently selected.

XML

```
<syncfusion:SfTimePicker ValueChanged="SftimePicker_ValueChanged"
Name="sfTimePicker"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.ValueChanged += SftimePicker_ValueChanged;
```

You can handle the event as follows:

C#

```
private void SftimePicker_ValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    Console.WriteLine("The Old selected time: " + e.OldValue.ToString());
    Console.WriteLine("The Newly selected time: " + e.NewValue.ToString());
}
```

Click [here](#) to download the sample that showcases the input types and selected time with its notification supports.

Time Formatting in WPF TimePicker (SfTimePicker)

The [SfTimePicker](#) control allows the user to select and display the time in various formats.

Display the time using the `FormatString`

We can edit and display the selected time with various formatting like short time, long time, universal time and 24 hour time formats by using the `FormatString` property. The default value of `FormatString` property is "h:mm tt".

XML

```
<syncfusion:SfTimePicker x:Name="sfTimePicker"
FormatString="HH:mm:ss"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.FormatString = "HH:mm:ss";
```



Here, `SfTimePicker` with 24 hour time format

Specifying format for the `TimeSelector`

We can allow the user to select the pair of hour, minutes, seconds and meridiem selector or any single selector cell from the `SfTimeSelector` by using the `SelectorFormatString` property. The default value of `SelectorFormatString` property is "h:mm tt" and the hour, minutes and meridiem value selector is enabled in the `SfTimeSelector`.

XML

```
<syncfusion:SfTimePicker x:Name="sfTimePicker"
SelectorFormatString="M"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.SelectorFormatString = "M";
```



Here, we can only able to select the hour and meridiem value from the **SfTimeSelector**

Click [here](#) to download the sample that showcases the edit, display time formatting and time selection formatting by the **SfTimePicker**.

Note: A detailed explanation of standard time formatting is available [here](#). The result string produced by these format specifiers are influenced by the settings in the Regional Options control panel. Computers with different cultures or different time and time settings will generate different result strings.

Customizing DropDown in WPF TimePicker (SfTimePicker)

We can customize the **SfTimeSelector** visibility, drop down button visibility and height of the **SfTimeSelector**.

Change DropDown height

The height of drop down can be changed using **DropDownHeight** property.

XML

```
<syncfusion:SfTimePicker DropDownHeight="300"
Name="sfTimePicker"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.DropDownHeight = 300;
```



Show or hide DropDown button

If we want to restrict the user to selecting a time from a drop down time selector, we can hide the drop down button by using the **ShowDropDownButton** property value as **false**. The default value of **ShowDropDownButton** property is **true**.

XML

```
<syncfusion:SfTimePicker ShowDropDownButton="False"
Name="sfTimePicker"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.ShowDropDownButton = false;
```

ShowDropDownButton = "false"

04:08 PM

ShowDropDownButton = "true"

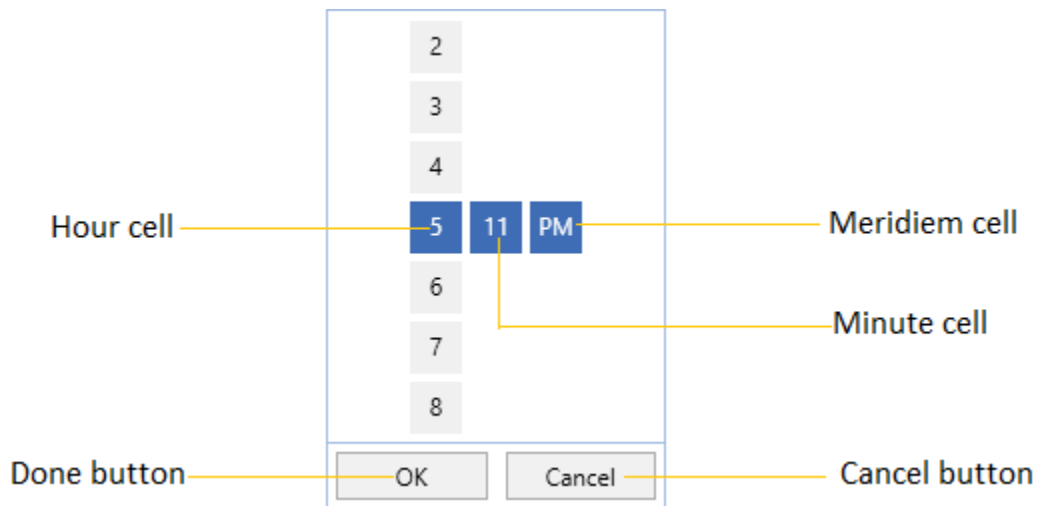
04:05 PM



Click [here](#) to download the sample that showcases the DropDown customization support.

Time Selector in WPF TimePicker (SfTimePicker)

The [SfTimeSelector](#) is a sub-control of [SfTimePicker](#) which opens inside the drop-down popup and used to select the time for the [SfTimePicker](#). It contains the hour, minutes and meridiem selection cells for select the time. The selected time of the [SfTimeSelector](#) is assigned to the [SfTimePicker.Value](#) property.



The visual elements of the time selector can be customized using the [SelectorStyle](#) property.

Change the Cell templates

We can change the template for the each hour, minute or meridiem selector by using the [HourCellTemplate](#), [MinuteCellTemplate](#) or [MeridiemCellTemplate](#) which are available in the [SfTimeSelector](#).

Note: The DataContext of Hour, Minute, Meridiem Selection cell is [DateTimeWrapper](#).

Change the HourCell Template

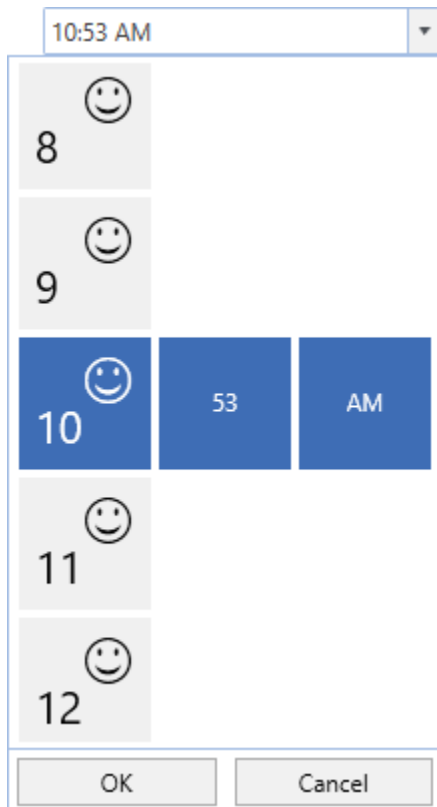
We can change the hour selector template by using the [HourCellTemplate](#) property. In that, we can add like image, icon or text with the hour values.

XML

```

<syncfusion:SfTimePicker DropDownHeight="380"
SelectorItemHeight="70"
SelectorItemWidth="70"
Width="200"
Name="sfTimePicker">
  <syncfusion:SfTimePicker.SelectorStyle>
    <Style TargetType="syncfusion:SfTimeSelector">
      <Setter Property="HourCellTemplate">
        <Setter.Value>
          <DataTemplate>
            <Grid>
              <TextBlock VerticalAlignment="Top"
HorizontalAlignment="Right"
Margin="5"
FontSize="22"
FontFamily="Segoe UI Symbol"
Text="&#xE170;" />
              <TextBlock Text="{Binding HourNumber}"
VerticalAlignment="Bottom"
Margin="5"
FontSize="22" />
            </Grid>
          </DataTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </syncfusion:SfTimePicker.SelectorStyle>
</syncfusion:SfTimePicker>

```

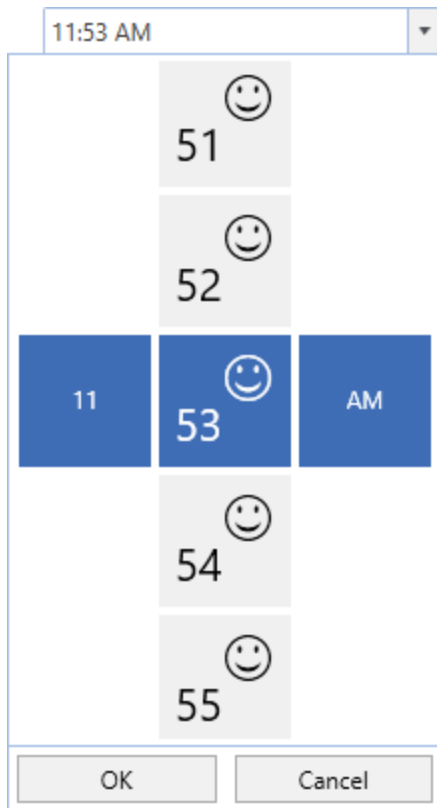


Change the MinuteCell Template

We can change the minute selector template by using the [MinuteCellTemplate](#) property. In that, we can add like image, icon or text with the minute values.

XML

```
<syncfusion:SfTimePicker DropDownHeight="380"
SelectorItemHeight="70"
SelectorItemWidth="70"
Width="200"
Name="sfTimePicker">
  <syncfusion:SfTimePicker.SelectorStyle>
    <Style TargetType="syncfusion:SfTimeSelector">
      <Setter Property="MinuteCellTemplate">
        <Setter.Value>
          <DataTemplate>
            <Grid>
              <TextBlock VerticalAlignment="Top"
HorizontalAlignment="Right"
Margin="5"
FontSize="22"
FontFamily="Segoe UI Symbol"
Text="&#xE170;" />
              <TextBlock Text="{Binding MinuteNumber}"
VerticalAlignment="Bottom"
Margin="5"
FontSize="22" />
            </Grid>
          </DataTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </syncfusion:SfTimePicker.SelectorStyle>
</syncfusion:SfTimePicker>
```



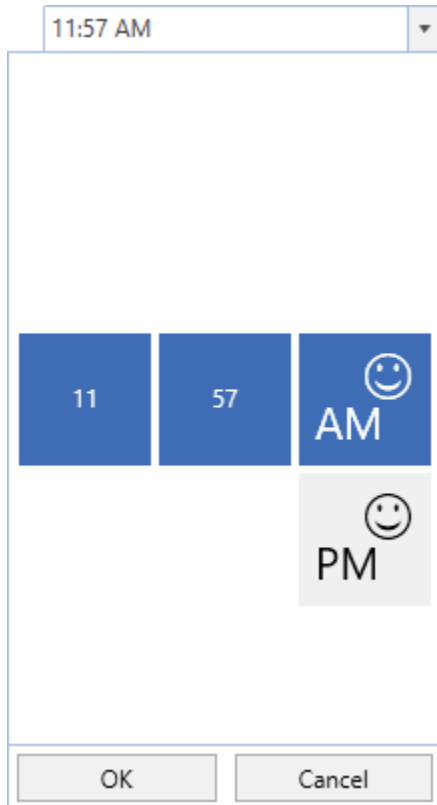
Change the MeridiemCell Template

We can change the meridiem selector template by using the [MeridiemCellTemplate](#) property. In that, we can add like image, icon or text with the meridiem values.

XML

```
<syncfusion:SfTimePicker DropDownHeight="380"
SelectorItemHeight="70"
SelectorItemWidth="70"
Width="200"
Name="sfTimePicker">
  <syncfusion:SfTimePicker.SelectorStyle>
    <Style TargetType="syncfusion:SfTimeSelector">
      <Setter Property="MeridiemCellTemplate">
        <Setter.Value>
          <DataTemplate>
            <Grid>
              <TextBlock VerticalAlignment="Top"
HorizontalAlignment="Right"
Margin="5"
FontSize="22"
FontFamily="Segoe UI Symbol"
Text="&#xE170;" />
              <TextBlock Text="{Binding AmPmString}"
VerticalAlignment="Bottom"
Margin="5"
FontSize="22" />
            </Grid>
          </DataTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </syncfusion:SfTimePicker.SelectorStyle>
</syncfusion:SfTimePicker>
```

```
</Setter.Value>
</Setter>
</Style>
</syncfusion:SfTimePicker.SelectorStyle>
</syncfusion:SfTimePicker>
```



Change size of cells

We can change the cell size in the `SfTimeSelector` control by setting the [SelectorItemWidth](#) and [SelectorItemHeight](#) properties. The default value of the `SelectorItemWidth` and `SelectorItemHeight` properties is 30 and 30.

XML

```
<syncfusion:SfTimePicker SelectorItemWidth="60"
SelectorItemHeight="60"
x:Name="sfTimePicker"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.SelectorItemWidth = 60;
sfTimePicker.SelectorItemHeight = 60;
```



TimeSelector item spacing

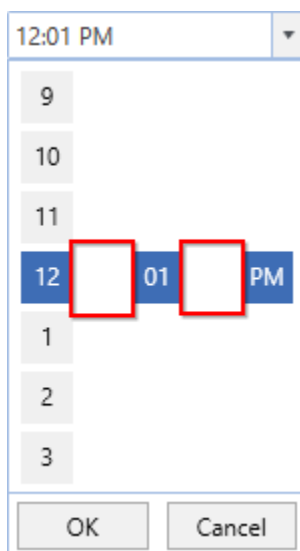
We can change the space between **SfTimeSelector** hour, minutes and meridiem items by using the [SelectorItemSpacing](#) property. The default value of the **SelectorItemSpacing** property is 4.

XML

```
<syncfusion:SfTimePicker SelectorItemSpacing="50"
x:Name="sfTimePicker"/>
```

C#

```
SfTimePicker sfTimePicker = new SfTimePicker();
sfTimePicker.SelectorItemSpacing = 50;
```



Click [here](#) to download the sample that showcases the **SfTimeSelector** template customization.

Appearance in WPF TimePicker (SfTimePicker)

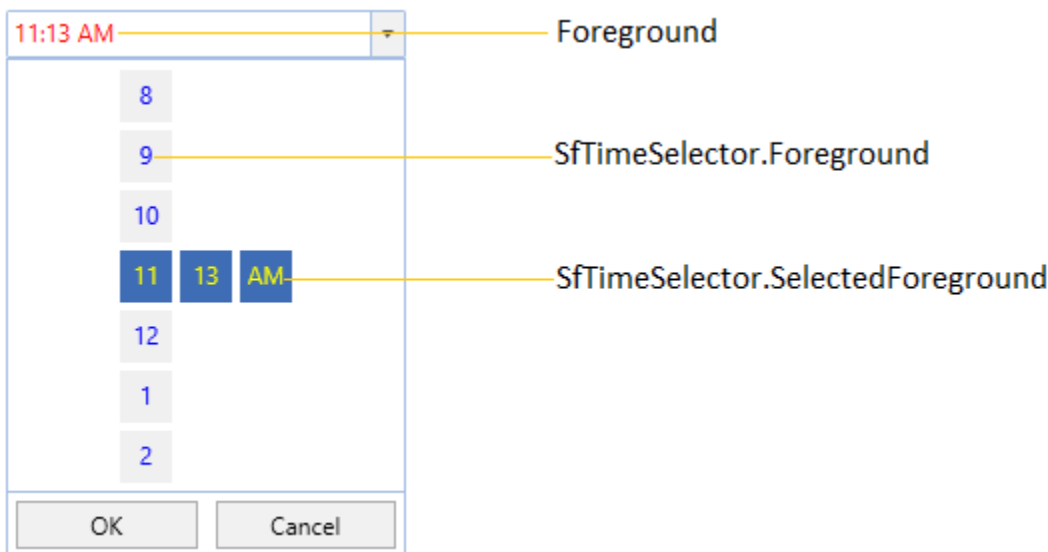
This section explains different UI customization, styling, theming options available in [SfTimePicker](#) control.

Setting the Foreground

We can change a foreground of the [SfTimePicker](#) by using the [Foreground](#) property and also we can change the [SfTimeSelector](#) items and selected time item foreground by using the [Foreground](#) and [SfTimeSelector.SelectedForeground](#) properties of [SfTimeSelector](#).

XML

```
<syncfusion:SfTimePicker Name="sfTimePicker"
Foreground="Red"
Width="200">
<syncfusion:SfTimePicker.SelectorStyle>
<Style TargetType="syncfusion:SfTimeSelector">
<Setter Property="Foreground" Value="Blue"/>
<Setter Property="SelectedForeground" Value="Yellow"/>
</Style>
</syncfusion:SfTimePicker.SelectorStyle>
</syncfusion:SfTimePicker>
```



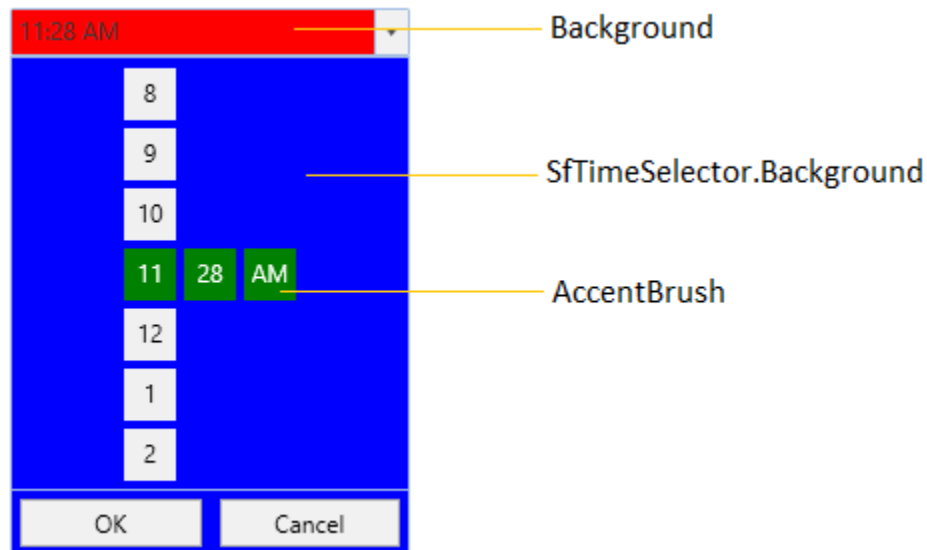
Setting the Background

We can change a background of the [SfTimePicker](#) by using the [background](#) property and also we can change the [SfTimeSelector](#) items and selected time item background by using the [Background](#) and [SfTimeSelector.AccentBrush](#) properties of [SfTimeSelector](#).

XML

```
<syncfusion:SfTimePicker Name="sfTimePicker"
Background="Red"
AccentBrush="Green"
Width="200">
<syncfusion:SfTimePicker.SelectorStyle>
<Style TargetType="syncfusion:SfTimeSelector">
```

```
<Setter Property="Background" Value="Blue"/>
</Style>
</syncfusion:SfTimePicker.SelectorStyle>
</syncfusion:SfTimePicker>
```



Change flow direction

We can change the flow direction of the SfTimePicker control from right to left by setting the FlowDirection property value as RightToLeft. The Default value of FlowDirection property is LeftToRight.

XML

```
<syncfusion:SfTimePicker FlowDirection="RightToLeft" Name="sfTimePicker"/>
```

C#

```
SfTimePicker sfTimePicker= new SfTimePicker();
sfTimePicker.FlowDirection = FlowDirection.RightToLeft;
```

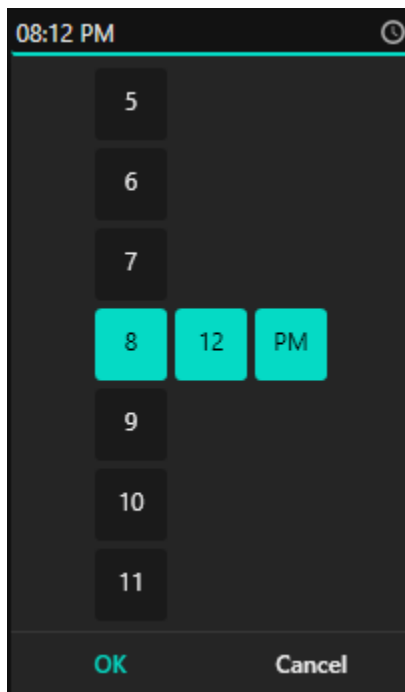


Click [here](#) to download the sample that showcases the different UI customization and styling supports.

Theme

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

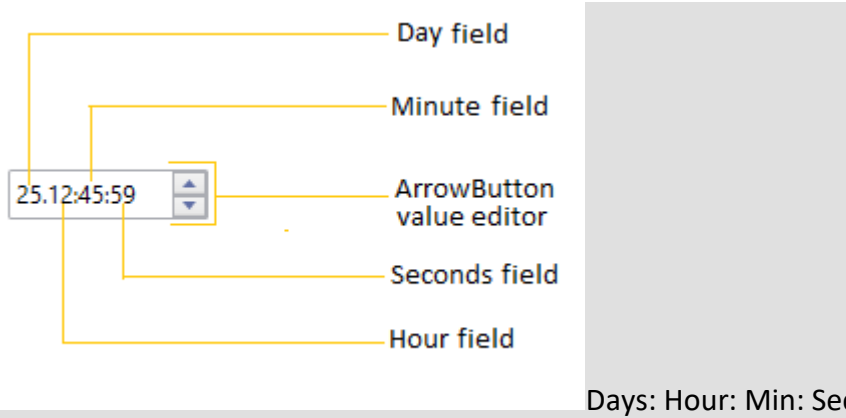
- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



TimeSpanEdit

WPF TimeSpan Editor (TimeSpanEdit) Overview

[TimeSpanEdit](#) is a control which set or display the time as



Days: Hour: Min: Sec format. The fields can be incremented or decremented using the up/down keys.

![[TimeSpanEdit control structure]](Getting-Startedimages/Control/Structure.png)

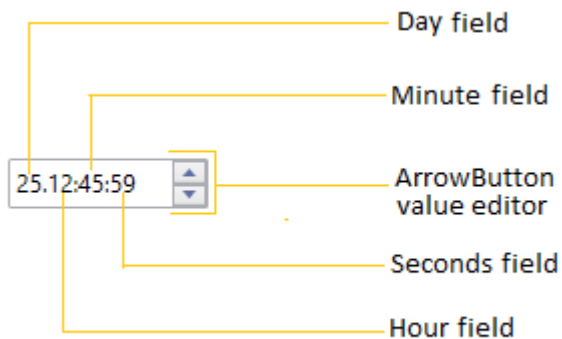
Key features

- **Custom Format String** - Supports to display the custom string for denote the numerals.
- **Keyboard Navigation** - Supports to move from one field to another using Keyboard.
- **SpinArrowButtons** - Supports to increase or decrease the values in any field.

Getting Started with WPF TimeSpan Editor (TimeSpanEdit)

This section explains how to create a [WPF TimeSpan Editor](#) (TimeSpanEdit) 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.

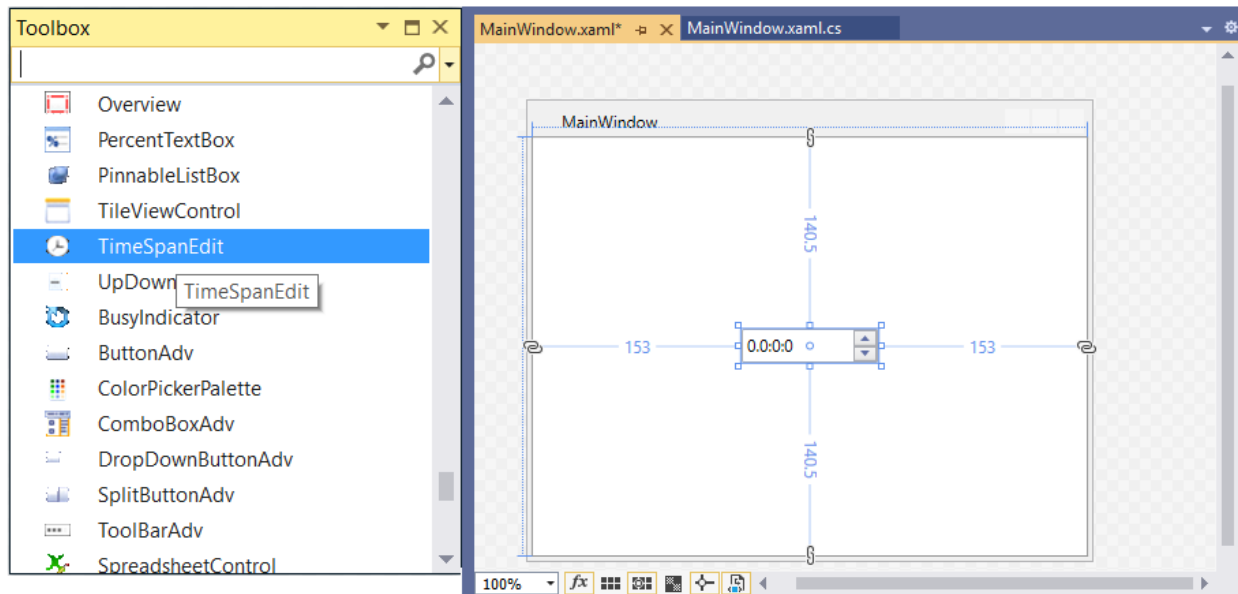
You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

Adding WPF TimeSpanEdit via designer

You can add the **WPF TimeSpan Editor (TimeSpanEdit)** 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 TimeSpanEdit via XAML

To add the **TimeSpanEdit** 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.Shared.WPF
3. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> and declare the **TimeSpanEdit** control in XAML page.
4. Declare the **TimeSpanEdit** 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="TimeSpanEditSample.MainWindow"
Title="TimeSpanEdit Sample" Height="350" Width="525">
<Grid>
<!--Adding TimeSpanEdit control -->
<syncfusion:TimeSpanEdit x:Name="timeSpanEdit"
Width="100"
Height="25" />
</Grid>
</Window>
```

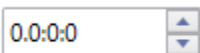
Adding WPF TimeSpanEdit via C\#

To add the **TimeSpanEdit** control manually in C#, follow these steps:

1. Create a new WPF application via Visual Studio.
2. Add the following assembly references to the project,
 - Syncfusion.Shared.WPF
3. Include the required namespace and create an instance of `TimeSpanEdit` and add it to the window.
4. Declare the `TimeSpanEdit` control using C#.

C#

```
using Syncfusion.Windows.Tools.Controls;
public partial class MainWindow : Window {
    public MainWindow() {
        InitializeComponent();
        //Creating an instance of TimeSpanEdit control
        TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
        timeSpanEdit.Width = 100;
        timeSpanEdit.Height = 25;
        //Adding TimeSpanEdit as window content
        this.Content = timeSpanEdit;
    }
}
```



Set or change time span value

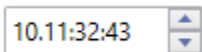
You can set or change the selected time of the `TimeSpanEdit` programmatically by setting the value to the `Value` property. You can also change the selected time at runtime using the mouse and keyboard interaction. Please refer the [Change time span value](#) page to know more about the mouse and keyboard interaction to change the value. The default value of `Value` property is `0.0:0:0`.

XML

```
<syncfusion:TimeSpanEdit Value="10.11:32:43"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.Value = new TimeSpan(10, 11, 32, 43);
```



Note: View [Sample](#) in GitHub

Change display format of time span

You can format the each fields to show what the numerals denotes i.e. hours, minutes or days by using the `Format` property. The default value of `Format` is `d.h:m:s`. You can show only the days, hours or minutes values by using any one the following respective fields to the `Format` property.

- d - It displays the days value.
- h - It displays the hours value.

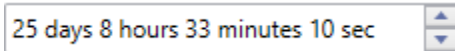
- m - It displays the minutes value.
- s - It displays the seconds value.

XML

```
<syncfusion:TimeSpanEdit Format="d 'days' h 'hours' m 'minutes' s 'sec'"
Value="25.08:33:10"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.Format = "d 'days' h 'hours' m 'minutes' s 'sec'";
timeSpanEdit.Value = new TimeSpan(25, 08, 33, 10);
```



Note: View [Sample](#) in GitHub

Value Changed Notification

The selected time span changed in `TimeSpanEdit` can be examined using [ValueChanged](#) event. The `ValueChanged` event contains the old and newly selected time span values in the `OldValue` and `NewValue` properties.

XML

```
<syncfusion:TimeSpanEdit ValueChanged="TimeSpanEdit_ValueChanged"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.ValueChanged += TimeSpanEdit_ValueChanged;
```

You can handle this event as follows,

C#

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

Restrict the time within minimum and maximum time span

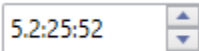
The selecting time in `TimeSpanEdit` can be restricted within the maximum and minimum time span limits. Once the selected time has reached the minimum or maximum time span limits, the selected time does not exceed the limit. You can change the minimum and maximum time span limits by using the [MinValue](#) and [MaxValue](#) properties.

XML

```
<syncfusion:TimeSpanEdit MinValue="2.0:0:0"  
MaxValue="10.0:0:0"  
Value="5.2:25:52"  
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();  
timeSpanEdit.MinValue = new TimeSpan(2, 0, 0, 0);  
timeSpanEdit.MaxValue = new TimeSpan(10, 0, 0, 0);  
timeSpanEdit.Value = new TimeSpan(5, 2, 25, 52);
```



Note: View [Sample](#) in GitHub

Localization support

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the `TimeSpanEdit` control by adding resource file for each language.

Refer the following links to know more about how provide a localization support for the `TimeSpanEdit`,

- <https://help.syncfusion.com/wpf/localization>
- <https://github.com/syncfusion/wpf-controls-localization-resx-files>

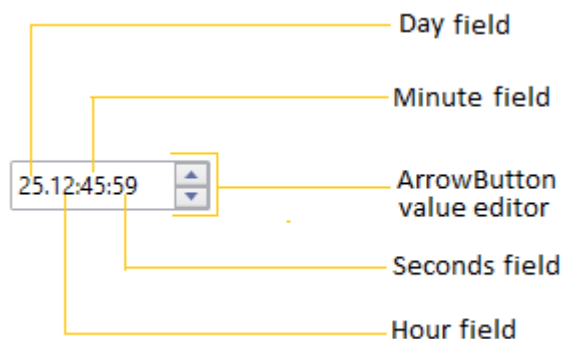
Theme

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

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

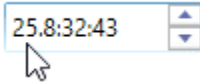
Working with TimeSpanEdit in WPF TimeSpan Editor (TimeSpanEdit)

This section explains how to change the time value and time formats in the WPF [TimeSpanEdit](#).



Date, hour and minute field navigation

By default, the focus field will be navigated automatically after the value has been validated to the corresponding field. If you want to manually change the day, hour or minute values, before that you navigate to the respective field by using the mouse or move the **Left-Right** keys in the keyboard.



Increase or decrease the time fields with specific interval

If you want to increase or decrease the time span field values with specific interval, use the [StepInterval](#) property. Selected time span field will be increased or decreased based on **StepInterval** field value by pressing the **Up-Down** arrow keys, **UpDown** button in **TimeSpanEdit** or using mouse wheel. The default value of **StepInterval** property is **{1.01:01:01}**.

For example, if value is **1.1:1:10**, seconds will increase or decrease in **10** seconds interval. other fields will increase or decrease **1** minute, hour, day interval.

XML

```
<syncfusion:TimeSpanEdit StepInterval="1.1:1:10"
Value="25.08:33:10"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.StepInterval = new TimeSpan(2, 0, 1, 10);
timeSpanEdit.Value = new TimeSpan(25, 08, 33, 10);
```



Note: View [Sample](#) in GitHub

Change the time value

You can change the time value of **TimeSpanEdit** by programmatically and using mouse or key interactions.

Change time programmatically

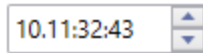
You can set or change the selected time of the **TimeSpanEdit** programmatically by setting the value to the **Value** property.

XML

```
<syncfusion:TimeSpanEdit Value="10.11:32:43"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.Value = new TimeSpan(10, 11, 32, 43);
```



Note: View [Sample](#) in GitHub

Change time using updown buttons

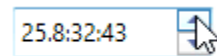
You can increase or decrease the selected time span field value based on the `StepInterval` value by pressing the Up or Down arrow buttons in the `TimeSpanEdit`. If you want to restrict the user to change time by clicking the UpDown buttons, use the `ShowArrowButtons` property value as `false`. It will hide the Arrow buttons. The default value of `ShowArrowButtons` property is `true`.

XML

```
<syncfusion:TimeSpanEdit ShowArrowButtons="True"
Value="25.09:32:43"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.ShowArrowButtons = true;
timeSpanEdit.Value = new TimeSpan(25, 09, 32, 43);
```



Note: View [Sample](#) in GitHub

Change time on mouse wheel

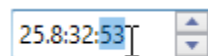
You can increase or decrease the selected time span field value based on the `StepInterval` value by mouse scrolling over the `TimeSpanEdit`. If you want to restrict the user to change time by using mouse scrolling, use the `IncrementOnScrolling` property value as `false`. The default value of `IncrementOnScrolling` property is `true`.

XML

```
<syncfusion:TimeSpanEdit IncrementOnScrolling="True"
Value="25.08:32:43"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.IncrementOnScrolling = true;
timeSpanEdit.Value = new TimeSpan(25, 08, 32, 43);
```



Note: View [Sample](#) in GitHub

Change time on click and drag

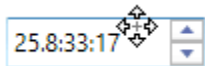
You can increase or decrease the selected time span field value based on the `StepInterval` value by clicking and dragging the mouse on up or down, use the [EnableExtendedScrolling](#) property value as `true`. This is effective only on when control is in unfocused state. The default value of `EnableExtendedScrolling` property is `false`.

XML

```
<syncfusion:TimeSpanEdit EnableExtendedScrolling="True"
Value="25.08:33:10"
Name="timeSpanEdit" />
```

C#

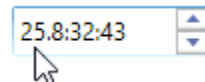
```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.EnableExtendedScrolling = true;
timeSpanEdit.Value = new TimeSpan(25, 08, 33, 10);
```



Note: View [Sample](#) in GitHub

Change time using keyboard interaction

You can increase or decrease the selected time span field value based on the `StepInterval` value by pressing `Up-Arrow` and `Down-Arrow` keys in keyboard.



Setting null value

If you want to set null value for the `TimeSpanEdit`, use the [AllowNull](#) property value as `true` and `Value` property as `null`. If `AllowNull` property is `false`, then the default time is displayed.

XML

```
<syncfusion:TimeSpanEdit AllowNull="True"
Value="{x:Null}"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.AllowNull = true;
timeSpanEdit.Value = null;
```



Note: View [Sample](#) in GitHub

Show watermark when value is null

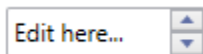
If you want to display any watermark text instead of null value, use the [NullString](#) property to setting the watermark text. You can enable it by setting the [AllowNull](#) property as `true` and [Value](#) property as `null`. The default value of [NullString](#) property is `string.Empty`.

XML

```
<syncfusion:TimeSpanEdit NullString="Edit here..."  
AllowNull="True"  
Value="{x:Null}"  
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();  
timeSpanEdit.NullString = "Edit here...";  
timeSpanEdit.AllowNull = true;  
timeSpanEdit.Value = null;
```



Note: View [Sample](#) in GitHub

Change display format of time span

You can format the each fields to show what the numerals denotes i.e. hours, minutes or days by using the [Format](#) property. The default value of [Format](#) is `d.h:m:s`. You can show only the days, hours or minutes values by using any one the following respective fields to the [Format](#) property.

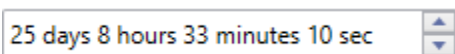
- `d` - It displays the days value.
- `h` - It displays the hours value.
- `m` - It displays the minutes value.
- `s` - It displays the seconds value.

XML

```
<syncfusion:TimeSpanEdit Format="d 'days' h 'hours' m 'minutes' s 'sec'"  
Value="25.08:33:10"  
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();  
timeSpanEdit.Format = "d 'days' h 'hours' m 'minutes' s 'sec'";  
timeSpanEdit.Value = new TimeSpan(25, 08, 33, 10);
```



Note: View [Sample](#) in GitHub

Display milliseconds

If you want to show the milliseconds in the time span, use the character **z** in the format string of the **Format** property.

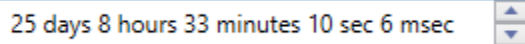
- **z** - It displays the milliseconds value.

XML

```
<syncfusion:TimeSpanEdit Format=" d 'days' h 'hours' m 'minutes' s 'sec' z 'msec' "
Value="25.08:33:10.6"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.Format = @" d 'days' h 'hours' m 'minutes' s 'sec' z 'msec'";
timeSpanEdit.Value = new TimeSpan(25, 08, 33, 10, 6);
```



Note: View [Sample](#) in GitHub

Value Changed Notification

The selected time span changed in **TimeSpanEdit** can be examined using [ValueChanged](#) event. The **ValueChanged** event contains the old and newly selected time span values in the **OldValue** and **NewValue** properties.

XML

```
<syncfusion:TimeSpanEdit ValueChanged="TimeSpanEdit_ValueChanged"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.ValueChanged += TimeSpanEdit_ValueChanged;
```

You can handle this event as follows,

C#

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

ReadOnly support

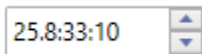
If you want to restrict the inputs from the user, use the `IsReadOnly` property value as `true`. However, value can be changed programmatically in readonly mode and the user can still select text. The default value of `IsReadOnly` property is `false`.

XML

```
<syncfusion:TimeSpanEdit IsReadOnly="True"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.IsReadOnly = true;
```



Note: View [Sample](#) in GitHub

Restrict the time within minimum and maximum time span

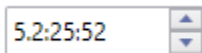
The selecting time in `TimeSpanEdit` can be restricted within the maximum and minimum time span limits. Once the selected time has reached the minimum or maximum time span limits, the selected time does not exceed the limit. You can change the minimum and maximum time span limits by using the [MinValue](#) property and [MaxValue](#) properties.

XML

```
<syncfusion:TimeSpanEdit MinValue="2.0:0:0"
MaxValue="10.0:0:0"
Value="5.2:25:52"
Name="timeSpanEdit" />
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.MinValue = new TimeSpan(2, 0, 0, 0);
timeSpanEdit.MaxValue = new TimeSpan(10, 0, 0, 0);
timeSpanEdit.Value = new TimeSpan(5, 2, 25, 52);
```



Note: View [Sample](#) in GitHub

Appearance in WPF TimeSpan Editor (TimeSpanEdit)

This section explains different UI customization and theming options available in [TimeSpanEdit](#).

Setting the background

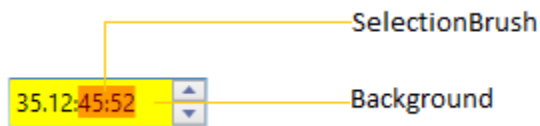
You can change the background color and selection color of `TimeSpanEdit` by using the `Background` and `SelectionBrush` property. The default value of `Background` property is `White` and `SelectionBrush` property is `Royal Blue`.

XML

```
<syncfusion:TimeSpanEdit Background="Yellow"
SelectionBrush="Red"
Value="35.12:45:52"
Name="timeSpanEdit"/>
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.Background = Brushes.Yellow;
timeSpanEdit.SelectionBrush = Brushes.Red;
timeSpanEdit.Value = new TimeSpan(35, 12, 45, 52);
```



Note: View [Sample](#) in GitHub

Setting the foreground

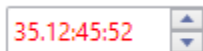
You can change the foreground color by using the `Foreground` property. The default value of `Foreground` property is `Black`.

XML

```
<syncfusion:TimeSpanEdit Foreground="Red"
Value="35.12:45:52"
Name="timeSpanEdit"/>
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();
timeSpanEdit.Foreground = Brushes.Red;
timeSpanEdit.Value = new TimeSpan(35, 12, 45, 52);
```



Note: View [Sample](#) in GitHub

Change flow direction

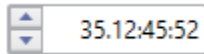
You can change the flow direction of the `TimeSpanEdit` layout from right to left by setting the `FlowDirection` property value as `RightToLeft`. The default value of `FlowDirection` property is `LeftToRight`.

XML

```
<syncfusion:TimeSpanEdit FlowDirection="RightToLeft"
Value="35.12:45:52"
Name="timeSpanEdit"/>
```

C#

```
TimeSpanEdit timeSpanEdit = new TimeSpanEdit();  
timeSpanEdit.FlowDirection = FlowDirection.RightToLeft;  
timeSpanEdit.Value = new TimeSpan(35, 12, 45, 52);
```

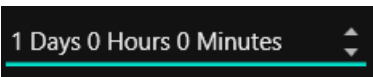


Note: View [Sample](#) in GitHub

Theme

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

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



ToolBarAdv

WPF ToolBar (ToolBarAdv) Overview

ToolBarAdv control is a container for a group of commands or control, which have relative functions. Usually it consist of buttons to invoke the commands.

Features

- Specifying the position of ToolBarAdv's in a ToolBarTrayAdv
- ToolBarAdv with Overflow items
- Show or hide the Gripper
- Orientation of ToolBarTrayAdv
- Add or Remove Buttons
- ToolBarAdv state
- Specifying location for floating ToolBarAdv
- ToolBarManager

Getting Started with WPF ToolBar (ToolBarAdv)

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.

This section explains how to add the [WPF ToolBar](#) (ToolBarAdv) control to an application and its structure.

Adding ToolBarAdv to a WPF Application

ToolBarAdv can be added to an application in a following way.

Create the ToolBarAdv Control to an application by using XAML:

The following ways explains how to add ToolBarAdv control using XAML code:

- Create a WPF project in Visual Studio and refer "Syncfusion.Shared.Wpf" assembly to the project.
- Include an XML namespace for the above assemblies to the Main window.

XML

```
<Window x:Class="Application_New.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525">
```

- Now add the WPF ToolBar (ToolBarAdv) control with a required optimal name using the namespace

XML

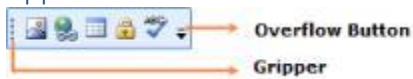
```
<syncfusion:ToolBarAdv Height="100" HorizontalAlignment="Left"
Margin="92, 90, 0, 0" Name="toolBarAdv1" VerticalAlignment="Top" Width="200" />
```

Create the ToolBarAdv control to an application by C#:

C#

```
ToolBarAdv tool = new ToolBarAdv();
tool.Name = "toolBarAdv1";
tool.Width = 100;
tool.Height = 50;
```

Appearance and Structure of the Control



- The Overflow button is a toggle button that displays on clicking the Overflow panel.
- The Gripper is used to drag the ToolBarAdv to change its Band. The state of the ToolBarAdv can be changed to float and dock by clicking the gripper and dragging the ToolBarAdv, when the ToolBarAdv is hosted in ToolBarManager.

Setting icon template

The [IconTemplate](#) supports adding any type of images such as path data, font icons, etc. as template in ToolBar. The icon will automatically resize the template content according to its size provided in the data template.

XML

```
<syncfusion:ToolBarManager x:Name="toolBarManager" Grid.Row="1"
CanDockAtLeft="{Binding IsChecked, ElementName=canDockAtLeft}"
CanDockAtTop="{Binding IsChecked, ElementName=canDockAtTop}"
CanDockAtRight="{Binding IsChecked, ElementName=canDockAtRight}"
CanDockAtBottom="{Binding IsChecked, ElementName=canDockAtBottom}">
<syncfusion:ToolBarManager.Resources>
```

```

<DataTemplate x:Key="NewDataTemplate">
<Path x:Name="New_Folder" 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="{Binding Foreground, ElementName=button1}"
HorizontalAlignment="Center" Stretch="Fill" Width="14" Height="14"
VerticalAlignment="Center" />
</DataTemplate>
<DataTemplate x:Key="OpenDataTemplate">
<Path x:Name="Open" 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="{Binding Foreground, ElementName=button1}"
HorizontalAlignment="Center" Stretch="Fill" Width="14" Height="14"
VerticalAlignment="Center"/>
</DataTemplate>
<DataTemplate x:Key="SaveDataTemplate">
<Path x:Name="Print" Data="M5.0000019,11 L5.0000019,15 11.000002,15
11.000002,11 z M4.0000019,1 L4.0000019,6 12.000002,6 12.000002,1 z M1,1
L1,13.174 2.7160001,15 4.0000019,15 4.0000019,10 12.000002,10 12.000002,15
15,15 15,1 13.000002,1 13.000002,7 3.0000019,7 3.0000019,1 z M0,0
L3.0000019,0 13.000002,0 16,0 16,16 12.000002,16 4.0000019,16 2.2840004,16
0,13.57 z"
Fill="{Binding Foreground, ElementName=button1}"
HorizontalAlignment="Center" Stretch="Fill" Width="16" Height="14"
VerticalAlignment="Center"/>
</DataTemplate>
<DataTemplate x:Key="SaveAllDataTemplate">
<Path Data="M5.0000019,11 L5.0000019,15 11.000002,15 11.000002,11 z
M4.0000019,1 L4.0000019,6 12.000002,6 12.000002,1 z M1,1 L1,13.174
2.7160001,15 4.0000019,15 4.0000019,10 12.000002,10 12.000002,15 15,15 15,1
13.000002,1 13.000002,7 3.0000019,7 3.0000019,1 z M0,0 L3.0000019,0
13.000002,0 16,0 16,16 12.000002,16 4.0000019,16 2.2840004,16 0,13.57 z"
Fill="{Binding Foreground, ElementName=button1}"
HorizontalAlignment="Center" Stretch="Fill" Width="10" Height="10"
VerticalAlignment="Center"/>
</DataTemplate>
<DataTemplate x:Key="PrintDataTemplate">
<Path Data="M5,11 L5,15 8.6761963,15 8.63936,14.971036 C7.6388242,14.144945
7,12.895562 7,11.5 7,11.344937 7.0078866,11.19168 7.0232801,11.040607
L7.0279369,11 z M13.145977,10.145986 L13.853984,10.853022
10.999985,13.706986 9.1459842,11.85303 9.8539908,11.145993
10.999985,12.293035 z M11.5,8 C9.5699999,8 7.9999999,9.5699999 8,11.5
7.9999999,13.43 9.5699999,15 11.5,15 13.43,15 15,13.43 15,11.5 15,9.5699999
13.43,8 11.5,8 z M2,7 L3,7 3,8 2,8 z M1,6 L1,11 4,11 4,10 7.2574663,10
7.2735391,9.954464 C7.9054199,8.2322998 9.5617187,7 11.5,7 12.895562,7
14.144945,7.6388242 14.971036,8.6393602 L15,8.6761966 15,6 12,6 4,6 z M5,1
L5,5 11,5 11,1 z M4,0 L12,0 12,5 16,5 16,11.5 C16,13.981 13.981,16 11.5,16
L4,16 4,12 0,12 0,5 4,5 z"
Fill="{Binding Foreground, ElementName=button1}"
HorizontalAlignment="Center" Stretch="Fill" Width="10" Height="10"
VerticalAlignment="Center"/>
</DataTemplate>
</syncfusion:ToolBarManager.Resources>
<syncfusion:ToolBarManager.TopToolBarTray>

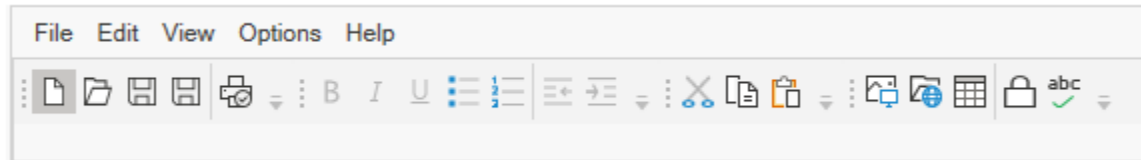
```

```
<syncfusion:ToolBarTrayAdv VerticalAlignment="Top" IsLocked="{Binding
IsChecked, ElementName=lockTop}">
<syncfusion:ToolBarAdv x:Name="toolBar" ToolBarName="Standard"
EnableAddRemoveButton="True">
<Button
x:Name="newFile"
Width="22"
Height="22"
Command="{Binding ButtonCommand}"
CommandParameter="{Binding ElementName=newFile}"
ToolTip="New"
Content="{StaticResource NewFile}">
</Button>
<Button
x:Name="openFile"
Width="22"
Height="22"
Command="{Binding ButtonCommand}"
CommandParameter="{Binding ElementName=openFile}"
ToolTip="Open"
Content="{StaticResource OpenFile}">
</Button>
<Button
x:Name="saveFile"
Width="22"
Height="22"
Command="{Binding ButtonCommand}"
CommandParameter="{Binding ElementName=saveFile}"
ToolTip="Save"
Content="{StaticResource SaveFile}">
</Button>
<Button
x:Name="saveAllFile"
Width="22"
Height="22"
Command="{Binding ButtonCommand}"
CommandParameter="{Binding ElementName=saveAllFile}"
ToolTip="SaveAll"
Content="{StaticResource SaveAll}">
</Button>
<syncfusion:ToolBarItemSeparator />
<Button
x:Name="printFile"
Width="22"
Height="22"
Command="{Binding ButtonCommand}"
CommandParameter="{Binding ElementName=printFile}"
ToolTip="Print"
Content="{StaticResource PrintFile}">
</Button>
</syncfusion:ToolBarAdv>
</syncfusion:ToolBarTrayAdv>
</syncfusion:ToolBarManager.TopToolBarTray>
<syncfusion:ToolBarManager.RightToolBarTray>
<syncfusion:ToolBarTrayAdv IsLocked="{Binding IsChecked,
ElementName=lockRight}">
</syncfusion:ToolBarTrayAdv>
```

```

</syncfusion:ToolBarManager.RightToolBarTray>
<syncfusion:ToolBarManager.BottomToolBarTray>
<syncfusion:ToolBarTrayAdv IsLocked="{Binding IsChecked,
ElementName=lockBottom}"/>
</syncfusion:ToolBarManager.BottomToolBarTray>
</syncfusion:ToolBarManager>

```



Properties

Properties of ToolBarAdv

Property	Description	Type	Data Type
Band	Gets or sets a value indicating where the ToolBarAdv should be placed in the ToolBarTrayAdv.	Dependency Property	Int
BandIndex	Gets or sets the band index number indicating the position of the ToolBarAdv on the band.	Dependency Property	Int
ToolBarName	Gets or sets the name of the ToolBarAdv.	Dependency Property	String
GripperVisibility	Gets or sets a value indicating whether gripper can be visible.	Dependency Property	Bool
FloatingBarLocation	Gets or sets the location for the floating ToolBarAdv.	Dependency Property	Point
ControlsResourceDictionary	Gets or sets resource dictionary in which ToolBarAdv will look up for Framework element's styles.	Dependency Property	Resource Dictionary
IsOverflowOpen	Gets or sets a value indicating whether overflow popup is open.	Dependency Property	Bool

ToolBarItemInfoCollection	Gets or sets the items to be displayed in the Add or Remove Buttons popup.	Dependency Property	ObservableCollection
IsOverflowItem	Gets or sets a value indicating whether an item can be displayed in overflow panel.	Attached Property	Bool
OverflowMode	Gets or sets an overflow mode for a specified item.	Attached Property	OverflowMode
Icon	Gets or sets an icon for specified item to be displayed in the Add or Remove Buttons menu.	Attached Property	ImageSource
Label	Gets or sets a label for specified item to be displayed in the Add or Remove Buttons menu.	Attached Property	String
IsAvailable	Gets or sets a value indicating whether a specified item should be hidden.	Attached Property	Boolean

Properties of ToolBarManager

Property	Description	Type	Data Type
IsLocked	Gets or Sets a value indicating whether ToolBarTrayAdv is locked.	Dependency property	bool
Orientation	Gets or Sets the orientation of the ToolBarAdv.	Dependency property	Orientation
ToolBars	Gets or sets toolbars.	Dependency property	ObservableCollection
TopToolBarTray	Gets or sets ToolBarTrayAdv which has to be	Dependency Property	ToolBarTrayAdv

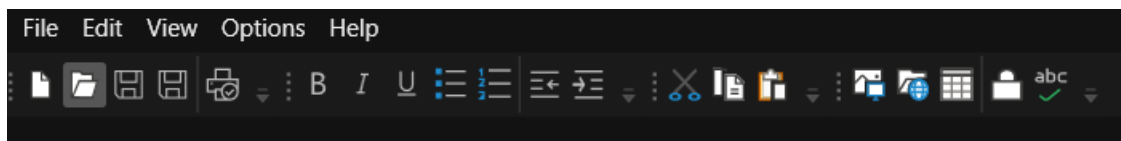
	displayed at the Top of ToolBarManager.			
BottomToolBarTray	Gets or sets ToolBarTrayAdv which has to be displayed at the bottom of ToolBarManager.	Dependency Property	ToolBarTrayAdv	
LeftToolBarTray	Gets or sets ToolBarTrayAdv which has to be displayed at the left of ToolBarManager.	Dependency Property	ToolBarTrayAdv	
RightToolBarTray	Gets or sets ToolBarTrayAdv which has to be displayed at the right of ToolBarManager.	Dependency Property	ToolBarTrayAdv	
CanDockAtTop	Gets or sets a value indicating whether toolbar can be docked at the top.	Dependency Property	bool	
CanDockAtBottom	Gets or sets a value indicating whether toolbar can be docked at the bottom.	Dependency Property	bool	
CanDockAtLeft	Gets or sets a value indicating whether toolbar can be docked at the left.	Dependency Property	bool	
CanDockAtRight	Gets or sets a value indicating whether toolbar can be docked at the right.	Dependency Property	bool	

Content	Gets or sets the content of the ToolBarManager.	Dependency Property	UIElement	
FloatingToolBarStyle	Gets or sets a style of floating tool bar	Dependency Property	Style	
ToolBarState	Gets or sets the state of the toolbar.	Attached Property	ToolBarState	

Theme

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

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



Dealing with ToolBarAdv control in WPF ToolBar (ToolBarAdv)

Specifying the Position of ToolBarAdv's in a ToolBarTrayAdv

The position of the ToolBarAdv in the ToolBarTrayAdv can be specified using the **Band** and the **BandIndex** properties. Band indicates the band in ToolBarTrayAdv, where ToolBarAdv has to be placed. BandIndex indicates the order in which the ToolBarAdv has to be placed within the band.

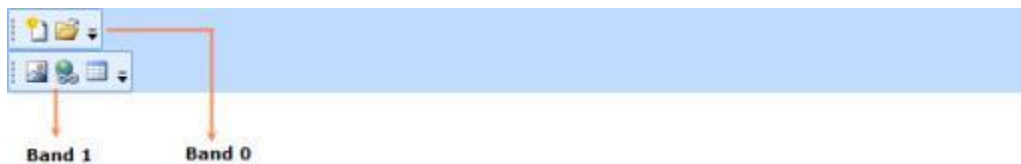
XML

```
<syncfusion:ToolBarTrayAdv >
  <syncfusion:ToolBarAdv ToolBarName="Standard">
    <Button syncfusion:ToolBarAdv.Icon="Images\NewDocumentHS.png">
      <Image Source="Images\NewDocumentHS.png" Width="16" Height="16"/>
    </Button>
    <Button>
      <Image Source="Images\openHS.png" Width="16" Height="16"/>
    </Button>
  </syncfusion:ToolBarAdv>
  <syncfusion:ToolBarAdv Band="1" ToolBarName="Extras">
    <Button>
      <Image Source="Images\InsertPictureHS.png" Width="16" Height="16"/>
    </Button>
    <Button>
      <Image Source="Images\InsertHyperlinkHS.png" Width="16" Height="16"/>
    </Button>
    <Button>
      <Image Source="Images\TableHS.png" Width="16" Height="16"/>
    </Button>
  </syncfusion:ToolBarAdv>
</syncfusion:ToolBarTrayAdv>
```

```
</syncfusion:ToolBarTrayAdv>
```

C#

```
ToolBarTrayAdv tray = new ToolBarTrayAdv();
ToolBarAdv toolBar = new ToolBarAdv();
Button button = new Button();
button.Width = 16;
button.Height = 16;
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/NewDocumentHS.png", UriKind.RelativeOrAbsolute) } };
toolBar.Items.Add(button);
button = new Button();
button.Width = 16;
button.Height = 16;
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/openHS.png", UriKind.RelativeOrAbsolute) } };
toolBar.Items.Add(button);
tray.ToolBars.Add(toolBar);
toolBar = new ToolBarAdv();
toolBar.Band = 1;
button = new Button();
button.Width = 16;
button.Height = 16;
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/InsertPictureHS.png", UriKind.RelativeOrAbsolute) } };
toolBar.Items.Add(button);
button = new Button();
button.Width = 16;
button.Height = 16;
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/InsertHyperlinkHS.png", UriKind.RelativeOrAbsolute) } };
toolBar.Items.Add(button);
button = new Button();
button.Width = 16;
button.Height = 16;
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/TableHS.png", UriKind.RelativeOrAbsolute) } };
toolBar.Items.Add(button);
tray.ToolBars.Add(toolBar);
Grid1.Children.Add(tray);
```

**ToolBarAdv with Overflow items**

When ToolBarAdv control contains more items, it cannot be displayed within the toolbar's size. So they are listed in the Overflow popup. On clicking the Overflow button, the items will be list out.

ToolBarAdv enables to set the overflow mode for each item.

- OverflowMode.Always – Specified item will always be listed in the Overflow popup.
- OverflowMode.Never - Specified item will never be moved to the Overflow popup.
- OverflowMode.AsNeeded - Specified item will be listed in the Overflow popup if required.



Show or hide Gripper

The gripper can show and hide in ToolBarAdv using the **GripperVisibility** property. To collapse the gripper in the ToolBarAdv, set the GripperVisibility as Collapsed. By default its value is visibility.

Following code illustrates how to hide the gripper:

XML

```
<syncfusion:ToolBarAdv GripperVisibility="Collapsed"/>
```

C#

```
ToolBarAdv toolBar = new ToolBarAdv();
toolBar.GripperVisibility = Visibility.Collapsed;
```



Orientation of ToolBarTrayAdv

ToolBarAdv provide two different orientation support such as Horizontal and Vertical. The desired orientation for ToolBarAdv can be changed using the **Orientation** property of ToolBarTrayAdv.

XML

```
<syncfusion:ToolBarTrayAdv Orientation="Vertical" >
  <syncfusion:ToolBarAdv ToolBarName="Standard">
    <Button syncfusion:ToolBarAdv.Icon="Images/NewDocumentHS.png">
      <Image Source="Images/NewDocumentHS.png" Width="16" Height="16"/>
    </Button>
    <Button >
      <Image Source="Images/openHS.png" Width="16" Height="16"/>
    </Button>
  </syncfusion:ToolBarAdv>
  <syncfusion:ToolBarAdv Band="1" ToolBarName="Extras">
    <Button >
      <Image Source="Images/InsertPictureHS.png" Width="16" Height="16"/>
    </Button>
    <Button >
      <Image Source="Images/InsertHyperlinkHS.png" Width="16" Height="16"/>
    </Button>
    <Button >
      <Image Source="Images/TableHS.png" Width="16" Height="16"/>
    </Button>
  </syncfusion:ToolBarAdv>
</syncfusion:ToolBarTrayAdv>
```

C#

```

ToolBarTrayAdv tray = new ToolBarTrayAdv();
tray.Orientation = Orientation.Vertical;
ToolBarAdv toolBar = new ToolBarAdv();
Button button = new Button();
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/InsertPictureHS.png", UriKind.RelativeOrAbsolute) } };
toolBar.Items.Add(button);
button = new Button();
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/InsertHyperlinkHS.png", UriKind.RelativeOrAbsolute) } };
toolBar.Items.Add(button);
button = new Button();
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/TableHS.png", UriKind.RelativeOrAbsolute) } };
toolBar.Items.Add(button);
tray.ToolBars.Add(toolBar);
Grid1.Children.Add(tray);

```

**Add or Remove buttons**

ToolBarAdv provides option to show or hide an item using the Add or Remove button. To enable the Add or Remove Button, `EnableAddRemoveButton` property of ToolBarAdv need to set as True. By default its value is false. Unselect the respective checkboxes of the items to hide.

Adding an Item to Add or Remove Button

An item can be added to Add or Remove Buttons by setting the ToolBarAdv.Icon and ToolBarAdv.Label properties.

Following code illustrates how to add an item in Add or Remove Buttons:

XML

```

<syncfusion:ToolBarTrayAdv>
  <syncfusion:ToolBarAdv EnableAddRemoveButton="True" >
    <Button syncfusion:ToolBarAdv.Label="New Document"
      syncfusion:ToolBarAdv.Icon="Images/NewDocumentHS.png">
      <Image Source="Images/NewDocumentHS.png" Width="16" Height="16"/>
    </Button>
    <Button syncfusion:ToolBarAdv.Label="Open Document"
      syncfusion:ToolBarAdv.Icon="Images/openHS.png">
      <Image Source="Images/openHS.png" Width="16" Height="16"/>
    </Button>
  </syncfusion:ToolBarAdv>
</syncfusion:ToolBarTrayAdv>

```

C#

```

ToolBarTrayAdv tray = new ToolBarTrayAdv();
ToolBarAdv toolBar = new ToolBarAdv();
toolBar.EnableAddRemoveButton = true;

```

```

Button button = new Button();
button.Width = 16;
button.Height = 16;
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/NewDocumentHS.png", UriKind.RelativeOrAbsolute) } };
ToolBarAdv.SetLabel(button, "New Document");
ToolBarAdv.SetIcon(button, new BitmapImage() { UriSource = new
Uri("Images/NewDocumentHS.png", UriKind.RelativeOrAbsolute) });
toolBar.Items.Add(button);
button = new Button();
button.Width = 16;
button.Height = 16;
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/openHS.png", UriKind.RelativeOrAbsolute) } };
ToolBarAdv.SetLabel(button, "Open Document");
ToolBarAdv.SetIcon(button, new BitmapImage() { UriSource = new
Uri("Images/openHS.png", UriKind.RelativeOrAbsolute) });
toolBar.Items.Add(button);
tray.ToolBars.Add(toolBar);
Grid1.Children.Add(tray);

```



Hiding the ToolBarItem

To hide the particular ToolBarItem, the `IsAvailable` an attached property of the ToolBarAdv need to set as False. By default its value is True.

XML

```

<syncfusion:ToolBarTrayAdv >
  <syncfusion:ToolBarAdv x:Name="Tooladv" ToolBarName="Standard" >
    <Button syncfusion:ToolBarAdv.Icon="Images\NewDocumentHS.png" >
      <Image Source="Images\NewDocumentHS.png" Width="16" Height="16"/>
    </Button>
    <Button>
      <Image Source="Images\openHS.png" Width="16" Height="16"/>
    </Button>
    <Button>
      <Image Source="Images\InsertPictureHS.png" Width="16"
      Height="16" syncfusion:ToolBarAdv.IsAvailable="False"/>
    </Button>
    <Button>
      <Image Source="Images\InsertHyperlinkHS.png" Width="16" Height="16"/>
    </Button>
    <Button>
      <Image Source="Images\TableHS.png" Width="16" Height="16"/>
    </Button>
  </syncfusion:ToolBarAdv>
</syncfusion:ToolBarTrayAdv>

```



ToolBarAdv State in WPF ToolBar (ToolBarAdv)

ToolBarAdv provides different states such as Docking, Floating or Hidden. It can be change using the property **ToolBarState** of the ToolBarManager.

XML

```
<syncfusion:ToolBarManager x:Name="toolBarManager" >
<syncfusion:ToolBarManager.TopToolBarTray>
<syncfusion:ToolBarTrayAdv >
<syncfusion:ToolBarAdv ToolBarName="Standard" Band="0">
<Button syncfusion:ToolBarAdv.Label="New Document"
syncfusion:ToolBarAdv.Icon="Images/NewDocumentHS.png">
<Image Source="Images/NewDocumentHS.png" Width="16" Height="16"/>
</Button>
<Button syncfusion:ToolBarAdv.Label="Open Document"
syncfusion:ToolBarAdv.Icon="Images/openHS.png">
<Image Source="Images/openHS.png" Width="16" Height="16"/>
</Button>
<syncfusion:ToolBarAdv Band="1" ToolBarName="Extras"
syncfusion:ToolBarManager.ToolBarState="Floating"
FloatingBarLocation="500,300">
<Button syncfusion:ToolBarAdv.Label="Insert Picture"
syncfusion:ToolBarAdv.Icon="Images/InsertPictureHS.png">
<Image Source="Images/InsertPictureHS.png" Width="16" Height="16"/>
</Button>
<Button syncfusion:ToolBarAdv.Label="Insert Hyperlink"
syncfusion:ToolBarAdv.Icon="Images/InsertHyperlinkHS.png">
<Image Source="Images/InsertHyperlinkHS.png" Width="16" Height="16"/>
</Button>
<Button syncfusion:ToolBarAdv.Label="Insert Table"
syncfusion:ToolBarAdv.Icon="Images/TableHS.png">
<Image Source="Images/TableHS.png" Width="16" Height="16"/>
</Button>
</syncfusion:ToolBarAdv>
</syncfusion:ToolBarAdv>
</syncfusion:ToolBarTrayAdv>
</syncfusion:ToolBarManager.TopToolBarTray>
<Grid >
<Grid.RowDefinitions>
<RowDefinition Height="*/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<ScrollViewer >
<Grid Margin="20">
<Border CornerRadius="2" Background="Black"
Opacity="0.3" Width="600" Height="700">
<Border.Effect>
<BlurEffect Radius="15"/>
</Border.Effect>
</Border>
<RichTextBox Width="600" Height="700" Padding="20"></RichTextBox>
</Grid>
</ScrollViewer>
</Grid>
</syncfusion:ToolBarManager>
```

C#

```
ToolBarAdv toolBar = new ToolBarAdv();  
toolBar.FloatingBarLocation = new Point(500, 300);  
ToolBarManager.SetToolBarState(toolBar, ToolBarState.Floating);  
Grid1.Children.Add(toolBar);
```



ToolBarAdv can be floated only when it is hosted in ToolBarManager.

Specifying location for floating ToolBarAdv.

The location of the floating ToolBarAdv can be changed using the `FloatingBarLocation` property. The following code illustrates this

XML

```
<syncfusion:ToolBarAdv FloatingBarLocation="50,50"/>
```

C#

```
ToolBarAdv toolBar = new ToolBarAdv();  
toolBar.FloatingBarLocation = new Point(50, 50);
```

{% endtabs%}

Restrict Docking of ToolBarAdv for a specific position

By default, the ToolBarAdv can be docked to any position. To restrict docking of ToolBarAdv to particular position, the following properties can be used. Each will restrict docking at corresponding positions in ToolBarManager.

- CanDockAtLeft—restricts docking at the left.
- CanDockAtTop—restricts docking at the top.
- CanDockAtRight—restricts docking at the right.
- CanDockAtBottom—restricts docking at the bottom.

Following code restricts docking at the top:

XML

```
<syncfusion:ToolBarManager CanDockAtTop="False"/>
```

C#

```
ToolBarManager toolBarManager = new ToolBarManager();
```

```
toolBarManager.CanDockAtTop = false;
```

```
{% endtabs%}
```

ToolBarManager in WPF ToolBar (ToolBarAdv)

ToolBarManager is a container in which the ToolBarTrayAdv can place in top, bottom, left or right provided with the following properties.

- TopToolBarTray
- BottomToolBarTray
- LeftToolBarTray
- RightToolBarTray

And the content of the ToolBarManager will be displayed in the remaining space.

The following code illustrates how to place the ToolBarAdv at the top:

XML

```
<syncfusion:ToolBarManager x:Name="toolBarManager" >
  <syncfusion:ToolBarManager.Resources>
    <Style TargetType="Button">
      <Setter Property="Height" Value="20" />
      <Setter Property="Width" Value="20"/>
    </Style>
    <Style TargetType="ToggleButton">
      <Setter Property="Height" Value="20"/>
      <Setter Property="Width" Value="20"/>
    </Style>
  </syncfusion:ToolBarManager.Resources>
  <syncfusion:ToolBarManager.TopToolBarTray>
    <syncfusion:ToolBarTrayAdv >
      <syncfusion:ToolBarAdv ToolBarName="Standard">
        <Button>
          <Image Source="Images/NewDocumentHS.png" Width="16" Height="16"/>
        </Button>
        <Button >
          <Image Source="Images/openHS.png" Width="16" Height="16"/>
        </Button>
      </syncfusion:ToolBarAdv>
    </syncfusion:ToolBarTrayAdv>
  </syncfusion:ToolBarManager.TopToolBarTray>
</syncfusion:ToolBarManager>
```

C#

```
ToolBarAdv toolBar = new ToolBarAdv();
Button button = new Button();
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/NewDocumentHS.png", UriKind.RelativeOrAbsolute) } };
toolBar.Items.Add(button);
button = new Button();
button.Content = new Image() { Source = new BitmapImage() { UriSource = new
Uri("Images/openHS.png", UriKind.RelativeOrAbsolute) } };
```

```

toolBar.Items.Add(button);
ToolBarTrayAdv tray = new ToolBarTrayAdv();
tray.ToolBars.Add(toolBar);
ToolBarManager manager = new ToolBarManager();
manager.TopToolBarTray = tray;
Grid1.Children.Add(manager);

```

Customization in WPF ToolBar (ToolBarAdv)

Customizing Floating ToolBarAdv

The floating ToolBarAdv can be customized using the FloatingToolBarAdvStyle property of ToolBarManager.

XML

```

<syncfusion:ToolBarManager FloatingToolBarStyle="{StaticResource toolstyle}"
>
<syncfusion:ToolBarTrayAdv >
<syncfusion:ToolBarAdv x:Name="Tooladv" ToolBarName="Standard" >
<Button syncfusion:ToolBarAdv.Icon="Images\NewDocumentHS.png" >
<Image Source="Images\NewDocumentHS.png" Width="16" Height="16"/>
</Button>
<Button>
<Image Source="Images\openHS.png" Width="16" Height="16" />
</Button>
<Button>
<Image Source="Images\InsertPictureHS.png" Width="16" Height="16"
syncfusion:ToolBarAdv.IsAvailable="False"/>
</Button>
<Button>
<Image Source="Images\InsertHyperlinkHS.png" Width="16" Height="16"/>
</Button>
<Button>
<Image Source="Images\TableHS.png" Width="16" Height="16"/>
</Button>
</syncfusion:ToolBarAdv>
</syncfusion:ToolBarTrayAdv>
</syncfusion:ToolBarManager>

```

Customize FrameworkElement's Style

In ToolBarAdv, style for FrameworkElement will be picked from a ResourceDictionary assigned in the ControlsResourceDictionary property of ToolBarAdv.

C#

```

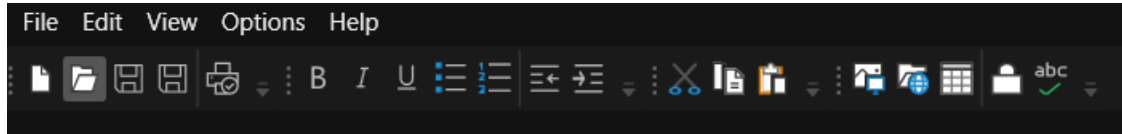
ToolBarAdv toolBar = new ToolBarAdv();
toolBar.ControlsResourceDictionary = new ResourceDictionary()
{
    Source = new Uri("ControlsResouce.xaml", UriKind.RelativeOrAbsolute)
};
}

```

Theme

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

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



SfTreeMap

WPF TreeMap (SfTreeMap) Overview

TreeMaps are ideal for visualizing large amounts of data. The visualization space is split up into rectangles that are sized and colored based on the quantitative variables. The levels in the hierarchy of the tree map are visualized as rectangles containing other rectangles.

Use Cases

Tree maps are used to represent large or complex data sets in various applications such as,

1. Stock market analysis in which the weight of each stock in the index is represented by the size of the rectangle and its range of loss or gain is represented by the color of the rectangle.
2. It is used in the visualization of Internet usage in certain categories - such as retail, social networks, and search/portal.
3. TreeMap can categorize the news aggregated by Google News in which the colors can represent different sections such as business or politics and the size of the boxes can represent how many similar stories also appear in Google News.
4. TreeMap is used to indicate weather report analysis around the world. The opacity of each rectangle can differ based on its humidity.

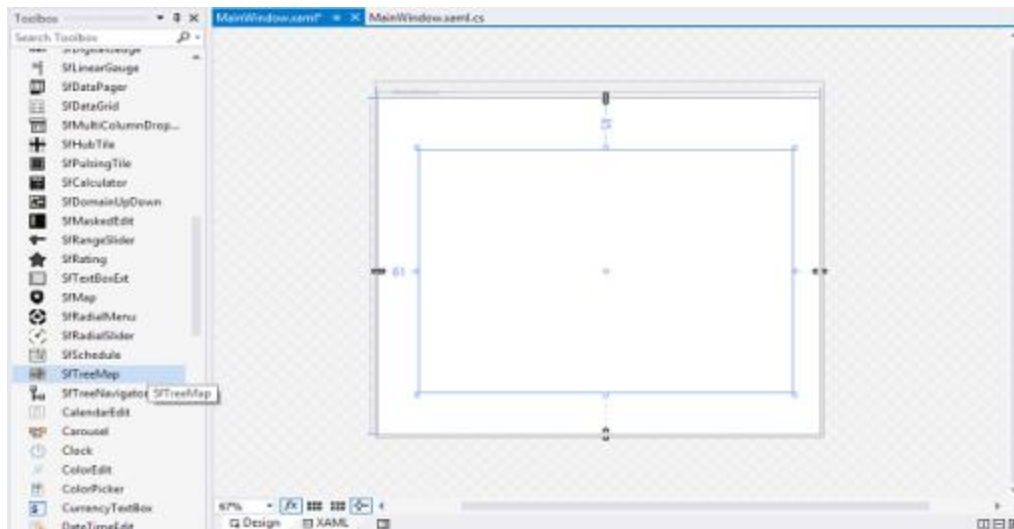
Getting Started with WPF TreeMap (SfTreeMap)

TreeMaps are a growing trend in data visualization. It displays hierarchical information in a series of clustered rectangles, which together represent a whole dataset. The size of each box represents a quantity. TreeMaps also use color to represent any number of values, but it is often used to categorize the various boxes within the tree map.

Configuring the SfTreeMap Control

Through Visual Studio

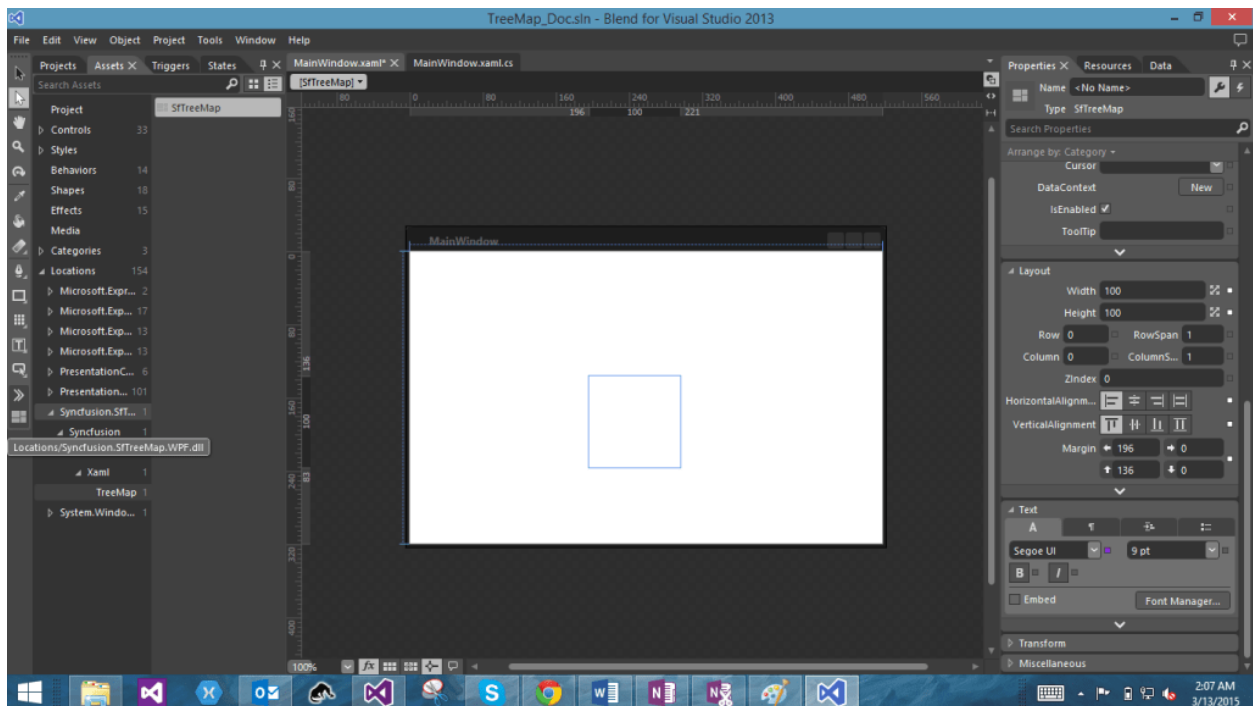
To create the SfTreeMap control through VisualStudio, drag SfTreeMap from Toolbox and drop it to the designer. It will generate the following the SfTreeMap control.



Through Expression Blend

The SfTreeMap control can also be created and configured using Expression Blend. Follow these steps to do so.

1. Create a WPF project in Expression Blend and reference the following assemblies.
 - i. Syncfusion.SfTreeMap.WPF
2. Search for SfTreeMap in the Toolbox.
3. Drag SfTreeMap to the designer. It will generate the SfTreeMap control with one child element.



Through XAML and C#

You can create the SfTreeMap control programmatically through XAML and C#.

In order to create a TreeMap you can refer to the following assembly and namespace.

Assembly: Syncfusion.SfTreeMap.WPF

Namespace: Syncfusion.UI.Xaml.TreeMap

In the following code example, the SfTreeMap control is created.

XML

```
<Window x:Class="TreeMapDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="clr-namespace:Syncfusion.UI.Xaml.TreeMap;assembly=Syncfusion.SfTreeMap.WPF"
Title="TreeMap Demo" WindowState="Maximized">
<Grid>
<syncfusion:SfTreeMap>
</syncfusion:SfTreeMap>
</Grid>
</Window>
```

C#

```
SfTreeMap treemap = new SfTreeMap()
{
    Height = 300,
    Width = 300,
};
```

Customizing the TreeMap Control

Populate ItemsSource

The ItemsSource property accepts the collection values as input. For example, you can provide the list of objects as input.

C#

```
public class PopulationViewModel
{
    public PopulationViewModel()
    {
        this.PopulationDetails = new ObservableCollection<PopulationDetail>();
        PopulationDetails.Add(new PopulationDetail() { Continent = "Asia", Country = "Indonesia", Growth = 3, Population = 237641326 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "Asia", Country = "Russia", Growth = 2, Population = 152518015 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "Asia", Country = "Malaysia", Growth = 1, Population = 29672000 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "North America", Country = "United States", Growth = 4, Population = 315645000 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "North America", Country = "Mexico", Growth = 2, Population = 112336538 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "North America", Country = "Canada", Growth = 1, Population = 35056064 });
    }
}
```

```

PopulationDetails.Add(new PopulationDetail() { Continent = "South America",
Country = "Colombia", Growth = 1, Population = 47000000 });
PopulationDetails.Add(new PopulationDetail() { Continent = "South America",
Country = "Brazil", Growth = 3, Population = 193946886 });
PopulationDetails.Add(new PopulationDetail() { Continent = "Africa", Country
= "Nigeria", Growth = 2, Population = 170901000 });
PopulationDetails.Add(new PopulationDetail() { Continent = "Africa", Country
= "Egypt", Growth = 1, Population = 83661000 });
PopulationDetails.Add(new PopulationDetail() { Continent = "Europe", Country
= "Germany", Growth = 1, Population = 81993000 });
PopulationDetails.Add(new PopulationDetail() { Continent = "Europe", Country
= "France", Growth = 1, Population = 65605000 });
PopulationDetails.Add(new PopulationDetail() { Continent = "Europe", Country
= "UK", Growth = 1, Population = 63181775 });
}
public ObservableCollection<PopulationDetail> PopulationDetails
{
    get;
    set;
}
public class PopulationDetail
{
    public string Continent { get; set; }
    public string Country { get; set; }
    public double Growth { get; set; }
    public double Population { get; set; }
}

```

After defining the ItemsSource, set the DataContext of the TreeMap as shown here:

XML

```

<Grid>
<Grid.DataContext>
<local:PopulationViewModel/>
</Grid.DataContext>
<syncfusion:SfTreeMap Name="TreeMap" ItemsSource="{Binding
PopulationDetails}" ItemsLayoutMode="Squarified"
WeightValuePath="Population"/>
<syncfusion:SfTreeMap.Levels>
<syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="5"
HeaderHeight="30">
<syncfusion:TreeMapFlatLevel.HeaderTemplate>
<DataTemplate>
<TextBlock Text="{Binding Header}" Foreground="#D6D6D6" FontSize="18"
FontWeight="Light" HorizontalAlignment="Left" VerticalAlignment="Center"/>
</DataTemplate>
</syncfusion:TreeMapFlatLevel.HeaderTemplate>
</syncfusion:TreeMapFlatLevel>
</syncfusion:SfTreeMap.Levels>
</Grid>

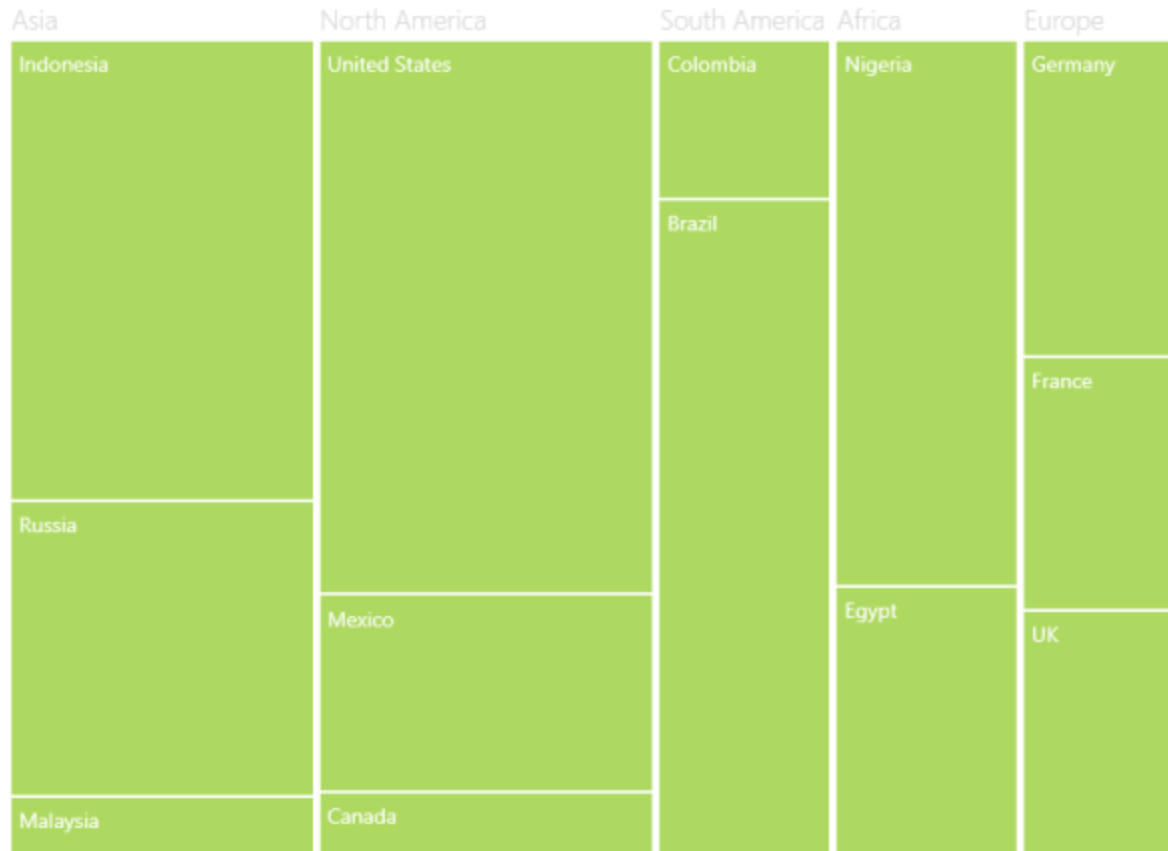
```

C#

```

this.TreeMap.DataContext = new PopulationViewModel();

```



Theme

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

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



see also

[How to apply gradient color mapping to SfTreeMap](#)

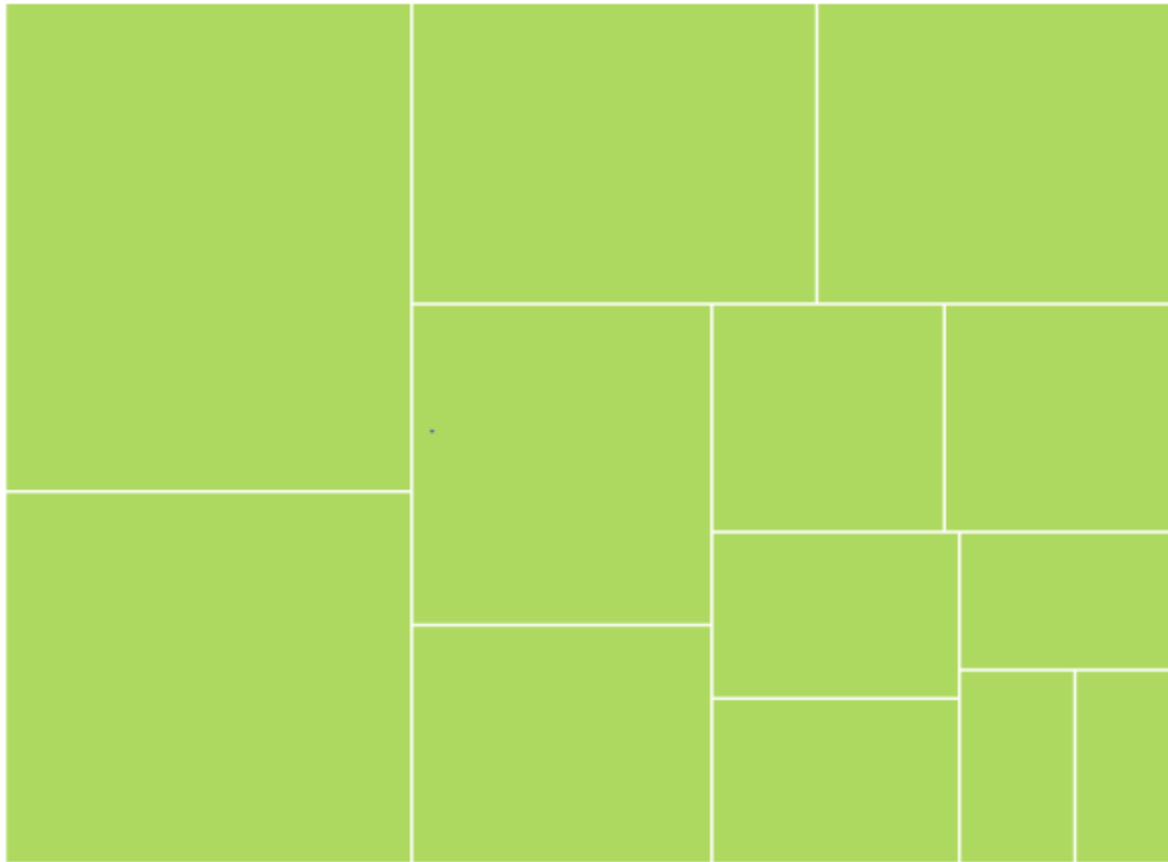
WeightValuePath in WPF TreeMap (SfTreeMap)

The WeightValuePath of SfTreeMap is a path to a field on the source object, which serve as the "weight" of the object.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    WeightValuePath="Population"/>
</Grid>
```

Note: The specified field must be available in each and every sub class (object) defined in hierarchical (nested) data collection.



ColorValuePath in WPF TreeMap (SfTreeMap)

The ColorValuePath of SfTreeMap is a path to a field on the source object, which serves as the "color" of the object.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    ColorValuePath="Growth"/>
</Grid>
```

Note: The specified field must be available in each and every sub class (object) defined in hierarchical (nested) data collection.

LeafItemSettings in WPF TreeMap (SfTreeMap)

LeafItemSettings of SfTreeMap is a setting by which we can settings the template for the leafNode.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
```

```
<syncfusion:SfTreeMap Name="TreeMap" ItemsSource="{Binding
PopulationDetails}" WeightValuePath="Population" ColorValuePath="Growth"
ItemsLayoutMode="Squarified" Margin="10">
<syncfusion:SfTreeMap.LeafItemSettings>
<syncfusion:LeafItemSettings/>
</syncfusion:SfTreeMap.LeafItemSettings>
</syncfusion:SfTreeMap>
</Grid>
```

Note: The specified field must be available in each and every sub class (object) defined in hierarchical (nested) data collection.

LabelPath

LabelPath of the leaves is WeightValuePath by default and you can change the LabelPath as desired based on the data provided.

XML

```
<Grid Background="Black">
<Grid.DataContext>
<local:PopulationViewModel/>
</Grid.DataContext>
<syncfusion:SfTreeMap Name="TreeMap" ItemsSource="{Binding
PopulationDetails}" WeightValuePath="Population" ColorValuePath="Growth"
ItemsLayoutMode="Squarified" Margin="10">
<syncfusion:SfTreeMap.LeafItemSettings>
<syncfusion:LeafItemSettings LabelPath="Country"/>
</syncfusion:SfTreeMap.LeafItemSettings>
</syncfusion:SfTreeMap>
</Grid>
```

LabelTemplate

LabelTemplate of LeafItemSettings class provides the template for the labels of the leafNodes.

XML

```
<Grid Background="Black">
<Grid.DataContext>
<local:PopulationViewModel/>
</Grid.DataContext>
<syncfusion:SfTreeMap Name="TreeMap" ItemsSource="{Binding
PopulationDetails}" WeightValuePath="Population" ColorValuePath="Growth"
ItemsLayoutMode="Squarified" Margin="10">
<syncfusion:SfTreeMap.LeafItemSettings>
<syncfusion:LeafItemSettings>
<syncfusion:LeafItemSettings.LabelTemplate>
<DataTemplate>
<TextBlock Text="{Binding Data.Country}" Foreground="White" FontSize="16"
FontWeight="Normal" HorizontalAlignment="Left" VerticalAlignment="Top"
Margin="5,5,0,0"/>
</DataTemplate>
</syncfusion:LeafItemSettings.LabelTemplate>
</syncfusion:LeafItemSettings>
</syncfusion:SfTreeMap.LeafItemSettings>
</syncfusion:SfTreeMap>
```

```
</Grid>
```

Gap

Gap provides the gap between the leaves at Leaf Level.

XML

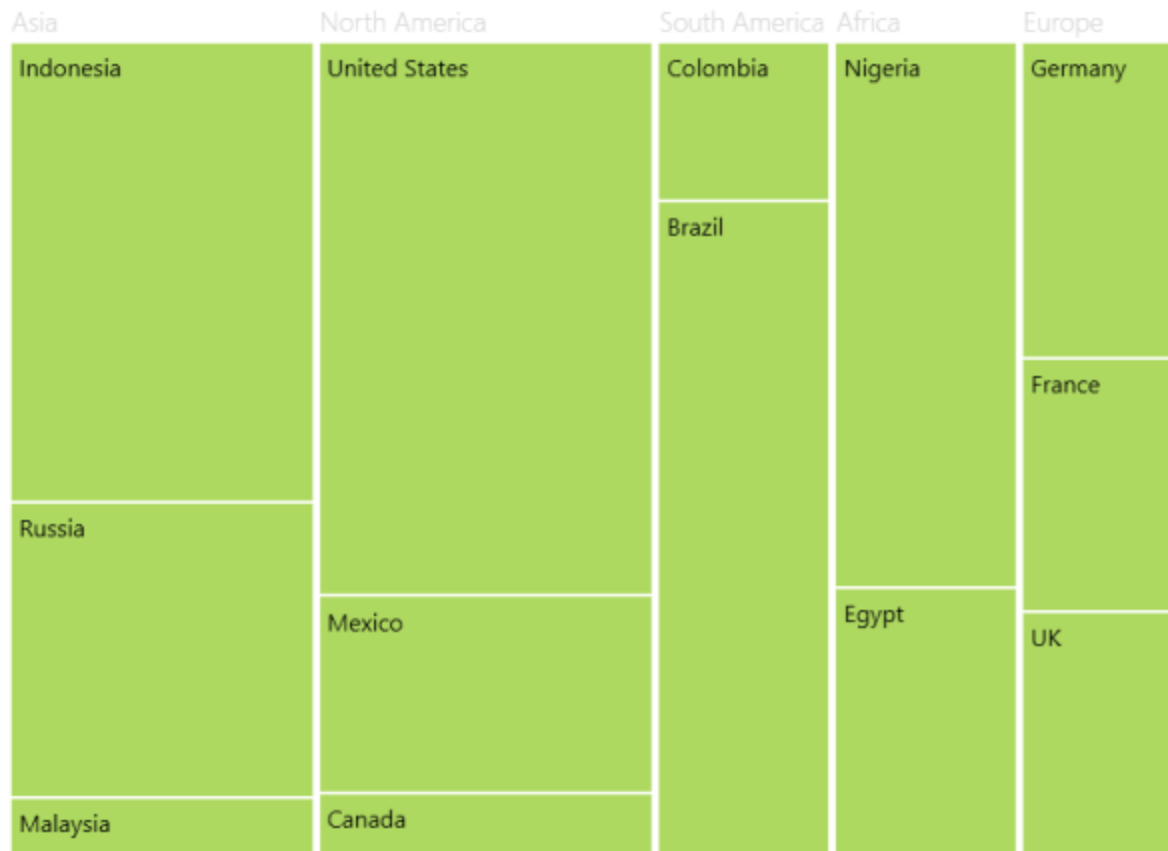
```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap Name="TreeMap" ItemsSource="{Binding
PopulationDetails}" WeightValuePath="Population" ColorValuePath="Growth"
ItemsLayoutMode="Squarified" Margin="10">
    <syncfusion:SfTreeMap.LeafItemSettings>
      <syncfusion:LeafItemSettings Gap="5">
      </syncfusion:LeafItemSettings>
    </syncfusion:SfTreeMap.LeafItemSettings>
  </syncfusion:SfTreeMap>
</Grid>
```

BorderBrush

BorderBrush provides the border color for the leafNodes and BorderThickness provides the thickness of the BorderBrush.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap Name="TreeMap" ItemsSource="{Binding
PopulationDetails}" WeightValuePath="Population" ColorValuePath="Growth"
ItemsLayoutMode="Squarified" Margin="10">
    <syncfusion:SfTreeMap.LeafItemSettings>
      <syncfusion:LeafItemSettings BorderBrush="Red" BorderThickness="3"/>
    </syncfusion:SfTreeMap.LeafItemSettings>
  </syncfusion:SfTreeMap>
</Grid>
```

TreeMap Levels in WPF TreeMap (SfTreeMap)

The levels of TreeMap can be categorized into two types such as,

- TreeMapFlatLevel
- TreeMapHierarchicalLevel

TreeMapFlatLevel

TreeMapFlatLevel is used to define levels for flat data collection.

ItemsSource:

The ItemsSource set for SfTreeMap must be a flat collection of data. The following code shows how to bind a flat collection as ItemsSource to a TreeMap.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    WeightValuePath="Population"
    ColorValuePath="Growth"/>
</Grid>
```

C#

```

public class PopulationViewModel
{
    public PopulationViewModel()
    {
        this.PopulationDetails = new
        ObservableCollection<PopulationDetail>();
        PopulationDetails.Add(new PopulationDetail() { Continent = "Asia", Country =
        "Indonesia", Growth = 3, Population = 237641326 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "Asia", Country =
        "Russia", Growth = 2, Population = 152518015 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "North America",
        Country = "United States", Growth = 4, Population = 315645000 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "North America",
        Country = "Mexico", Growth = 2, Population = 112336538 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "Africa", Country
        = "Nigeria", Growth = 2, Population = 170901000 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "Africa", Country
        = "Egypt", Growth = 1, Population = 83661000 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "Europe", Country
        = "Germany", Growth = 1, Population = 81993000 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "Europe", Country
        = "France", Growth = 1, Population = 65605000 });
        PopulationDetails.Add(new PopulationDetail() { Continent = "Europe", Country
        = "UK", Growth = 1, Population = 63181775 });
    }
    public ObservableCollection<PopulationDetail> PopulationDetails
    {
        get;
        set;
    }
    public class PopulationDetail
    {
        public string Continent { get; set; }
        public string Country { get; set; }
        public double Growth { get; set; }
        public double Population { get; set; }
    }
}

```

GroupPath:

You must specify the GroupPath for each and every flat level of TreeMap. It is a path to a field on the source object, which serves as the “Group” for the levels specified. Based upon the GroupPath, the data is grouped in the TreeMap. If GroupPath is not specified, then the items are not grouped, and it is shown in the order, in which they are specified in the ItemsSource.

XML

```

<Grid Background="Black">
    <Grid.DataContext>
        <local:PopulationViewModel/>
    </Grid.DataContext>
    <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    WeightValuePath="Population"
    ColorValuePath="Growth">
        <syncfusion:SfTreeMap.Levels>
            <syncfusion:TreeMapFlatLevel GroupPath="Continent" HeaderHeight="30">

```

```

<syncfusion:TreeMapFlatLevel.HeaderTemplate>
  <DataTemplate>
    <TextBlock Text="{Binding Header}" Foreground="#D6D6D6" FontSize="18"
      FontWeight="Light" HorizontalAlignment="Left" VerticalAlignment="Center"/>
  </DataTemplate>
</syncfusion:TreeMapFlatLevel.HeaderTemplate>
</syncfusion:TreeMapFlatLevel>
</syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
</Grid>

```

GroupGap:

You can specify GroupGap for separating the items of every flat level and it is used to differentiate the levels mentioned for TreeMap.

XML

```

<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    WeightValuePath="Population"
    ColorValuePath="Growth">
    <syncfusion:SfTreeMap.Levels>
      <syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="5"
        HeaderHeight="30">
      </syncfusion:TreeMapFlatLevel>
    </syncfusion:SfTreeMap.Levels>
  </syncfusion:SfTreeMap>
</Grid>

```

GroupBorderBrush

GroupBorderBrush provides the border color for the group levels.

XML

```

<Grid Background="Black">
  <Grid.Resources>
    <local:CountrySalesCollection x:Key="countrySalesCollection"/>
  </Grid.Resources>
  <syncfusion:SfTreeMap ItemsSource="{Binding Source={StaticResource
    countrySalesCollection}}">
    WeightValuePath="Sales"
    ColorValuePath="Expense">
    <syncfusion:SfTreeMap.Levels>
      <syncfusion:TreeMapFlatLevel GroupBorderBrush="Red" GroupPath="Continent"
        GroupBorderThickness="3">
      </syncfusion:TreeMapFlatLevel>
    </syncfusion:SfTreeMap.Levels>
  </syncfusion:SfTreeMap>
</Grid>

```

GroupBackground

GroupBackground specifies the background brush for the levels. The GroupBackground is effectively seen when we provide GroupPadding also while defining the Levels.

XML

```
<Grid Background="Black">
  <Grid.Resources>
    <local:CountrySalesCollection x:Key="countrySalesCollection"/>
  </Grid.Resources>
  <syncfusion:SfTreeMap ItemsSource="{Binding Source={StaticResource
countrySalesCollection}}"
WeightValuePath="Sales"
ColorValuePath="Expense">
    <syncfusion:SfTreeMap.Levels>
      <syncfusion:TreeMapFlatLevel GroupPadding="5" GroupPath="Continent"
GroupBackground="Red">
      </syncfusion:TreeMapFlatLevel>
    </syncfusion:SfTreeMap.Levels>
  </syncfusion:SfTreeMap>
</Grid>
```

TreeMapHierarchicalLevel:

TreeMapHierarchicalLevel is used to define levels for hierarchical data collection which contains tree-structured data.

ItemsSource:

The ItemsSource set for TreeMap must be a nested data collection. The following code shows how to bind a hierarchical data collection as ItemsSource for TreeMap.

XML

```
<Grid Background="Black">
  <Grid.Resources>
    <local:CountrySalesCollection x:Key="countrySalesCollection"/>
  </Grid.Resources>
  <syncfusion:SfTreeMap ItemsSource="{Binding Source={StaticResource
countrySalesCollection}}"
WeightValuePath="Sales"
ColorValuePath="Expense">
    <syncfusion:SfTreeMap.Levels>
      <syncfusion:TreeMapHierarchicalLevel/>
    </syncfusion:SfTreeMap.Levels>
  </syncfusion:SfTreeMap>
</Grid>
```

C#

```
public class CountrySalesCollection : ObservableCollection<CountrySale>
{
  public CountrySalesCollection()
  {
    this.Add(new CountrySale() { Name = "United States", Sales = 98456, Expense
= 87000 });
  }
}
```

```

this.Add(new CountrySale() { Name = "Canada", Sales = 43523, Expense = 40000
});
this.Add(new CountrySale() { Name = "Mexico", Sales = 45634, Expense = 46000
});
this[0].RegionalSalesCollection.Add(new RegionSale() { Country = "United
States", Name = "New York", Sales = 2353, Expense = 2000 });
this[0].RegionalSalesCollection.Add(new RegionSale() { Country = "United
States", Name = "Los Angeles", Sales = 3453, Expense = 3000 });
this[0].RegionalSalesCollection.Add(new RegionSale() { Country = "United
States", Name = "San Francisco", Sales = 8456, Expense = 8000 });
this[0].RegionalSalesCollection.Add(new RegionSale() { Country = "United
States", Name = "Chicago", Sales = 6785, Expense = 7000 });
this[0].RegionalSalesCollection.Add(new RegionSale() { Country = "United
States", Name = "Miami", Sales = 7045, Expense = 6000 });
this[1].RegionalSalesCollection.Add(new RegionSale() { Country = "Canada",
Name = "Toronto", Sales = 7045, Expense = 7000 });
this[1].RegionalSalesCollection.Add(new RegionSale() { Country = "Canada",
Name = "Vancouver", Sales = 4352, Expense = 4000 });
this[1].RegionalSalesCollection.Add(new RegionSale() { Country = "Canada",
Name = "Winnipeg", Sales = 7843, Expense = 7500 });
this[2].RegionalSalesCollection.Add(new RegionSale() { Country = "Mexico",
Name = "Mexico City", Sales = 7843, Expense = 6500 });
this[2].RegionalSalesCollection.Add(new RegionSale() { Country = "Mexico",
Name = "Cancun", Sales = 6683, Expense = 6000 });
}
public class CountrySale : INotifyPropertyChanged
{
    public string Name { get; set; }
    private double _sales = 0;
    public double Sales
    {
        get { return _sales; }
        set
        {
            if (_sales != value)
            {
                _sales = value;
                this.OnPropertyChanged(new
                PropertyChangedEventArgs("Sales"));
            }
        }
    }
    private double _expense = 0;
    public double Expense
    {
        get { return _expense; }
        set
        {
            if (_expense != value)
            {
                _expense = value;
                this.OnPropertyChanged(new
                PropertyChangedEventArgs("Expense"));
            }
        }
    }
    public ObservableCollection<RegionSale> RegionalSalesCollection

```

```
{ get; set; }
public CountrySale()
{
    this.RegionalSalesCollection = new ObservableCollection<RegionSale>();
}
#region INotifyPropertyChanged Members
public event PropertyChangedEventHandler PropertyChanged;
protected void OnPropertyChanged(PropertyChangedEventArgs e)
{
    if (this.PropertyChanged != null)
        this.PropertyChanged.Invoke(this, e);
}
#endregion
public class RegionSale : INotifyPropertyChanged
{
    public string Name { get; set; }
    public string Country { get; set; }
    private double _sales = 0;
    public double Sales
    {
        get { return _sales; }
        set
        {
            if (_sales != value)
            {
                _sales = value;
                this.OnPropertyChanged(new
                PropertyChangedEventArgs("Sales"));
            }
        }
    }
    private double _expense = 0;
    public double Expense
    {
        get { return _expense; }
        set
        {
            if (_expense != value)
            {
                _expense = value;
                this.OnPropertyChanged(new
                PropertyChangedEventArgs("Expense"));
            }
        }
    }
    #region INotifyPropertyChanged Members
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged(PropertyChangedEventArgs e)
    {
        if (this.PropertyChanged != null)
            this.PropertyChanged.Invoke(this, e);
    }
    #endregion
}
```

ChildPath:

You must specify ChildPath for each and every hierarchical level of TreeMap. It is a path to a field on the source object, which serves as the “Child” for the level specified. Based upon the ChildPath, the treemap contains child items.

XML

```
<Grid Background="Black">
<Grid.Resources>
<local:CountrySalesCollection x:Key="countrySalesCollection"/>
</Grid.Resources>
<syncfusion:SfTreeMap ItemsSource="{Binding Source={StaticResource
countrySalesCollection}}"
WeightValuePath="Sales"
ColorValuePath="Expense">
<syncfusion:SfTreeMap.Levels>
<syncfusion:TreeMapHierarchicalLevel
ChildPath="RegionalSalesCollection">
</syncfusion:TreeMapHierarchicalLevel>
</syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
</Grid>
```

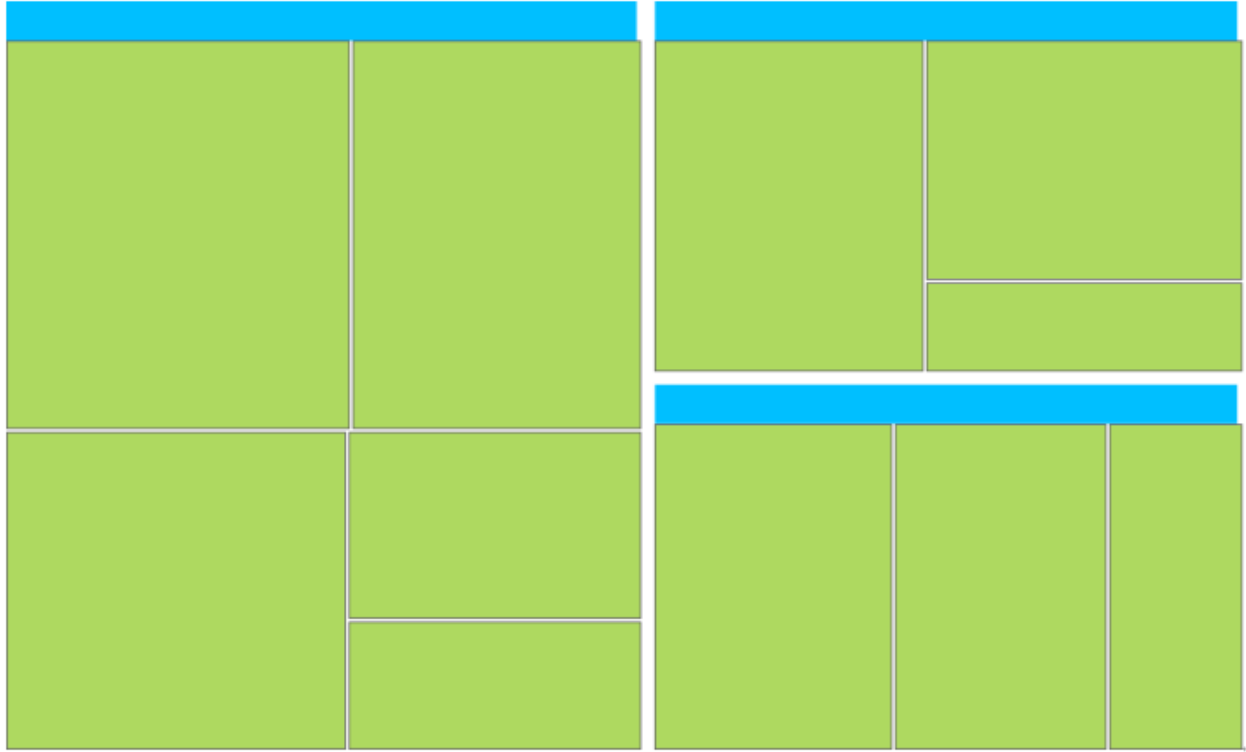
Note: The specified field must be a collection of sub class (object) specified in the nested data collection.

ChildGap:

You can specify ChildGap for separating the child items of every level and it is used to differentiate the levels mentioned for TreeMap.

XML

```
<Grid Background="Black">
<Grid.Resources>
<local:CountrySalesCollection x:Key="countrySalesCollection"/>
</Grid.Resources>
<syncfusion:SfTreeMap ItemsSource="{Binding Source={StaticResource
countrySalesCollection}}"
WeightValuePath="Sales"
ColorValuePath="Expense">
<syncfusion:SfTreeMap.Levels>
<syncfusion:TreeMapHierarchicalLevel ChildGap="10"
ChildPath="RegionalSalesCollection">
</syncfusion:TreeMapHierarchicalLevel>
</syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
</Grid>
```



TreeMap Layout in WPF TreeMap (SfTreeMap)

The `ItemsLayoutMode` for `SfTreeMap` specifies the layout mode of the tree map items. This layout is applied for all the tree map levels. There are four different TreeMap layouts such as,

Squarified Layout

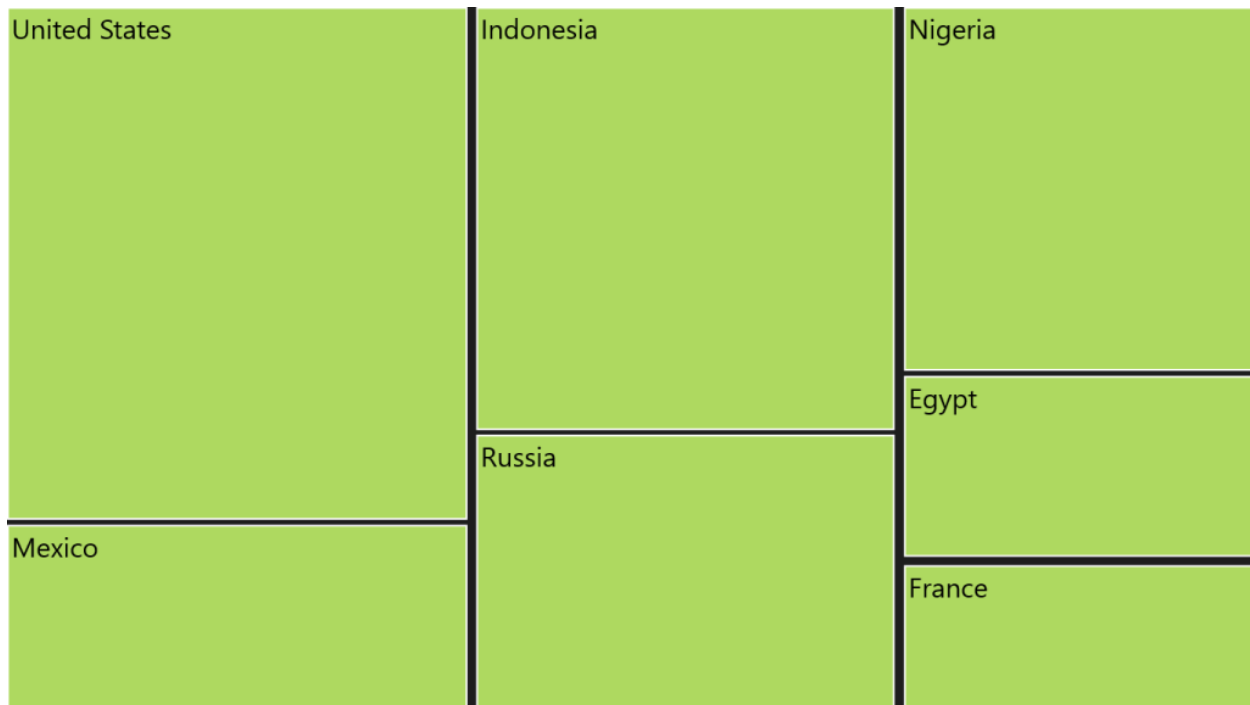
In this layout the data is visualized in the form of square-like rectangles with best aspect ratio.

The following code illustrates how to set a squarified layout in Treemap.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    ItemsLayoutMode="Squarified"
    WeightValuePath="Population"
    ColorValuePath="Growth">
  </syncfusion:SfTreeMap>
</Grid>
```

The following screenshot illustrates a squarified layout.



SliceAndDiceAuto Layout:

In this layout the data is visualized in the form of long-thin rectangles with high aspect ratio, which can be displayed either vertically or horizontally.

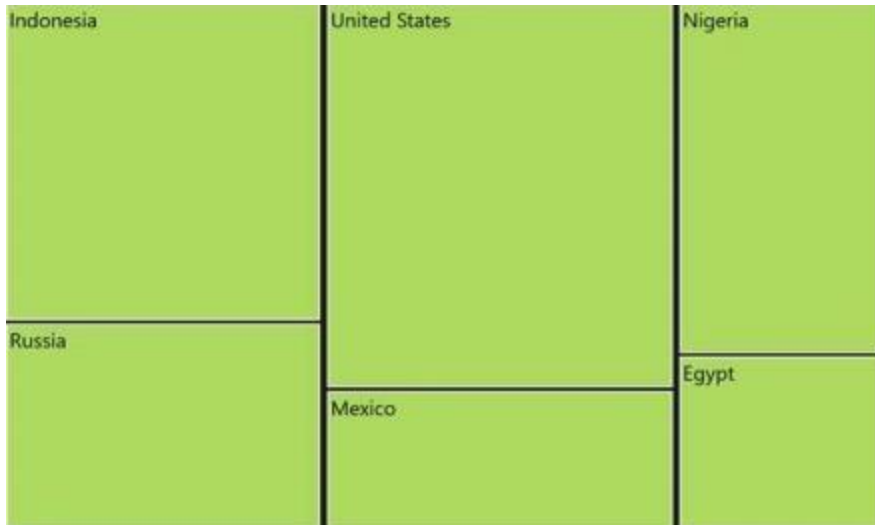
The following code illustrates how to set a slice and dice layout in Treemap.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    ItemsLayoutMode=" SliceAndDiceAuto"
    WeightValuePath="Population"
    ColorValuePath="Growth">
  </syncfusion:SfTreeMap>
</Grid>
```

The following screenshot illustrates a slice-and-dice layout.

Slice-and-dice layout



SliceAndDiceHorizontal Layout:

The following code illustrates how to set a slice and dice layout horizontally in TreeMap.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    ItemsLayoutMode=" SliceAndDiceHorizontal "
    WeightValuePath="Population"
    ColorValuePath="Growth">
  </syncfusion:SfTreeMap>
</Grid>
```

The following screenshot shows a Slice-and-dice TreeMap in horizontal layout.



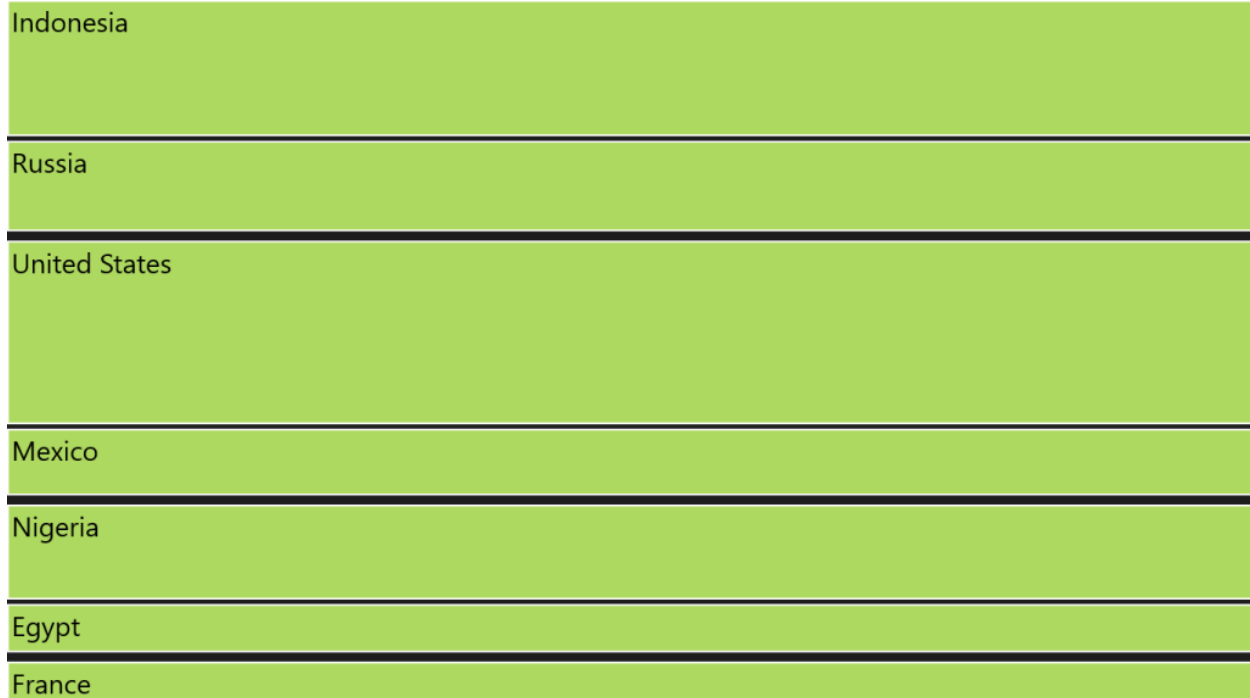
SliceAndDiceVertical Layout:

The following code illustrates how to set a slice and dice layout vertically in TreeMap.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    ItemsLayoutMode=" SliceAndDiceVertical"
    WeightValuePath="Population"
    ColorValuePath="Growth">
  </syncfusion:SfTreeMap>
</Grid>
```

The following screenshot shows a Slice-and-dice TreeMap in vertical layout.



ColorMapping in WPF TreeMap (SfTreeMap)

ColorMapping is categorized into four different types such as,

- UniColorMapping
- RangeBrushColorMapping
- DesaturationColorMapping
- PaletteColorMapping
- GroupColorMapping

TreeMap ColorMapping:

The leaf nodes of TreeMap can be colored by setting LeafColorMapping of TreeMap.

XML

```
<syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
WeightValuePath="Population"
ColorValuePath="Growth">
  <syncfusion:SfTreeMap.LeafColorMapping>
    <syncfusion:UniColorMapping Color="Crimson"/>
  </syncfusion:SfTreeMap.LeafColorMapping>
  <syncfusion:SfTreeMap.Levels>
    <syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="10"/>
  </syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
```

TreeMapLevel ColorMapping:

The headers of TreeMap level can also be colored using ColorMapping property of TreeMapLevel.

XML

```
<syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
```

```

WeightValuePath="Population"
ColorValuePath="Growth">
<syncfusion:SfTreeMap.LeafColorMapping>
<syncfusion:UniColorMapping Color="Orange"/>
</syncfusion:SfTreeMap.LeafColorMapping>
<syncfusion:SfTreeMap.Levels>
<syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="10"
HeaderHeight="20">
<syncfusion:TreeMapFlatLevel.ColorMapping>
<syncfusion:UniColorMapping Color="YellowGreen"/>
</syncfusion:TreeMapFlatLevel.ColorMapping>
</syncfusion:TreeMapFlatLevel>
<syncfusion:TreeMapFlatLevel GroupPath="Country" GroupGap="5"
HeaderHeight="15">
<syncfusion:TreeMapFlatLevel.ColorMapping>
<syncfusion:UniColorMapping Color="Crimson"/>
</syncfusion:TreeMapFlatLevel.ColorMapping>
</syncfusion:TreeMapFlatLevel>
</syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>

```

UniColorMapping

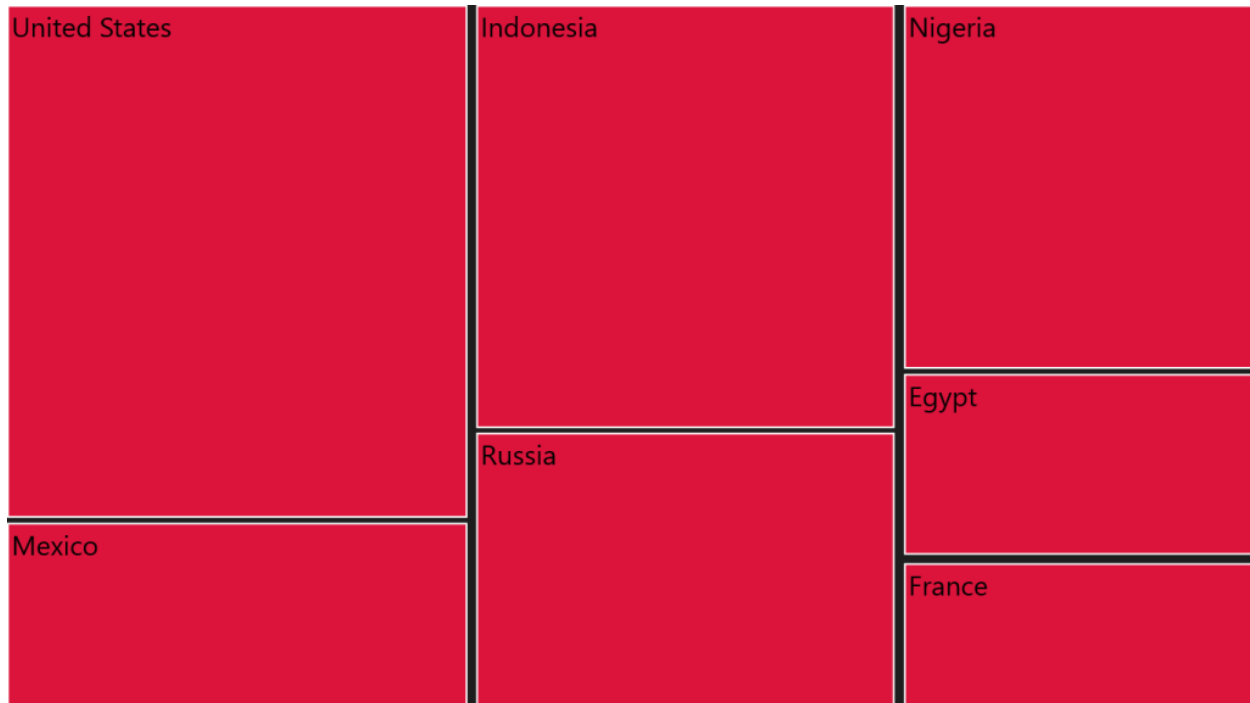
TreeMap leaf nodes can be colored with the help of Color property specified using UniColorMapping.

XML

```

<Grid Background="Black">
<Grid.DataContext>
<local:PopulationViewModel/>
</Grid.DataContext>
<syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
WeightValuePath="Population"
ColorValuePath="Growth">
<syncfusion:SfTreeMap.LeafColorMapping>
<syncfusion:UniColorMapping Color="Crimson"/>
</syncfusion:SfTreeMap.LeafColorMapping>
<syncfusion:SfTreeMap.Levels>
<syncfusion:TreeMapFlatLevel GroupPath="Continent"
GroupGap="10"/>
<syncfusion:TreeMapFlatLevel GroupPath="Country"
GroupGap="5"
ShowLabels="True"/>
</syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
</Grid>

```

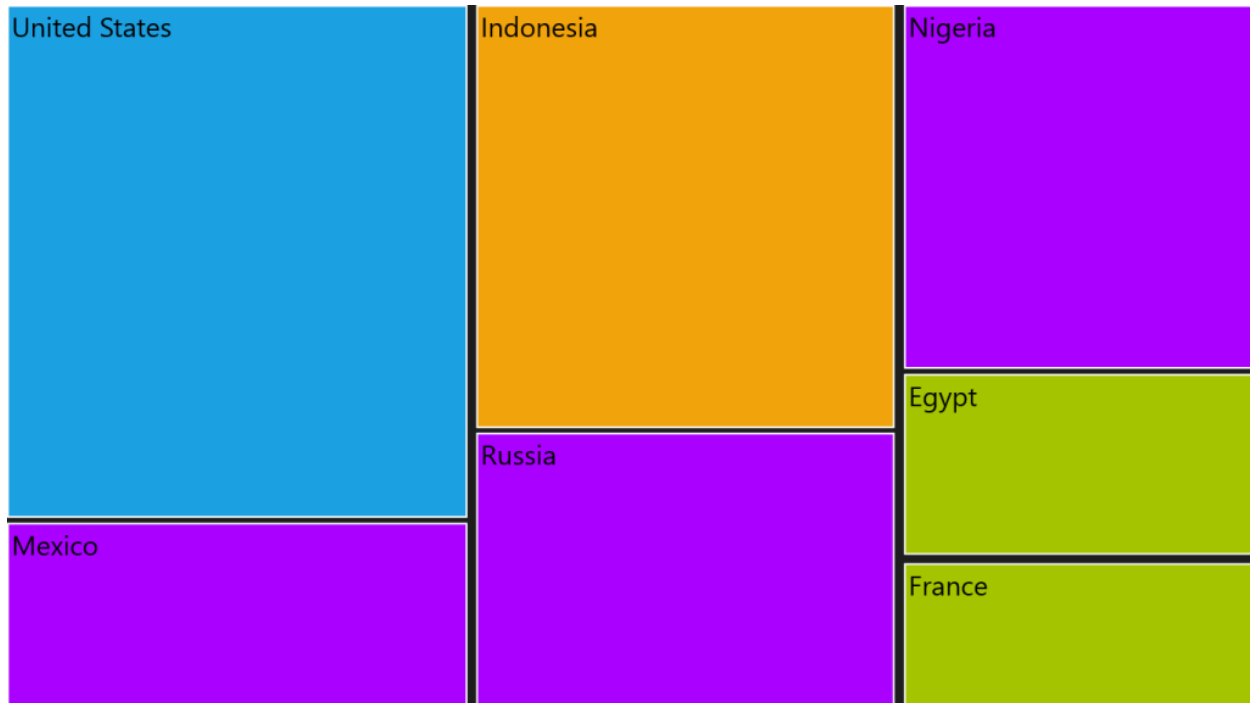


RangeBrushColorMapping

The leaf nodes of TreeMap can be colored based upon the range, such as From and To, and Brush specified using RangeBrush collection of RangeBrushColorMapping.

XML

```
<syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth">
  <syncfusion:SfTreeMap.LeafColorMapping>
    <syncfusion:RangeBrushColorMapping>
      <syncfusion:RangeBrushColorMapping.Brushes>
        <syncfusion:RangeBrush From="0" To="1" Color="#A4C400"/>
        <syncfusion:RangeBrush From="1" To="2" Color="#AA00FF"/>
        <syncfusion:RangeBrush From="2" To="3" Color="#F0A30A"/>
        <syncfusion:RangeBrush From="3" To="4" Color="#1BA1E2"/>
      </syncfusion:RangeBrushColorMapping.Brushes>
    </syncfusion:RangeBrushColorMapping>
  </syncfusion:SfTreeMap.LeafColorMapping>
  <syncfusion:SfTreeMap.Levels>
    <syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="10"/>
    <syncfusion:TreeMapFlatLevel GroupPath="Country" GroupGap="5"
ShowLabels="True"/>
  </syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
```

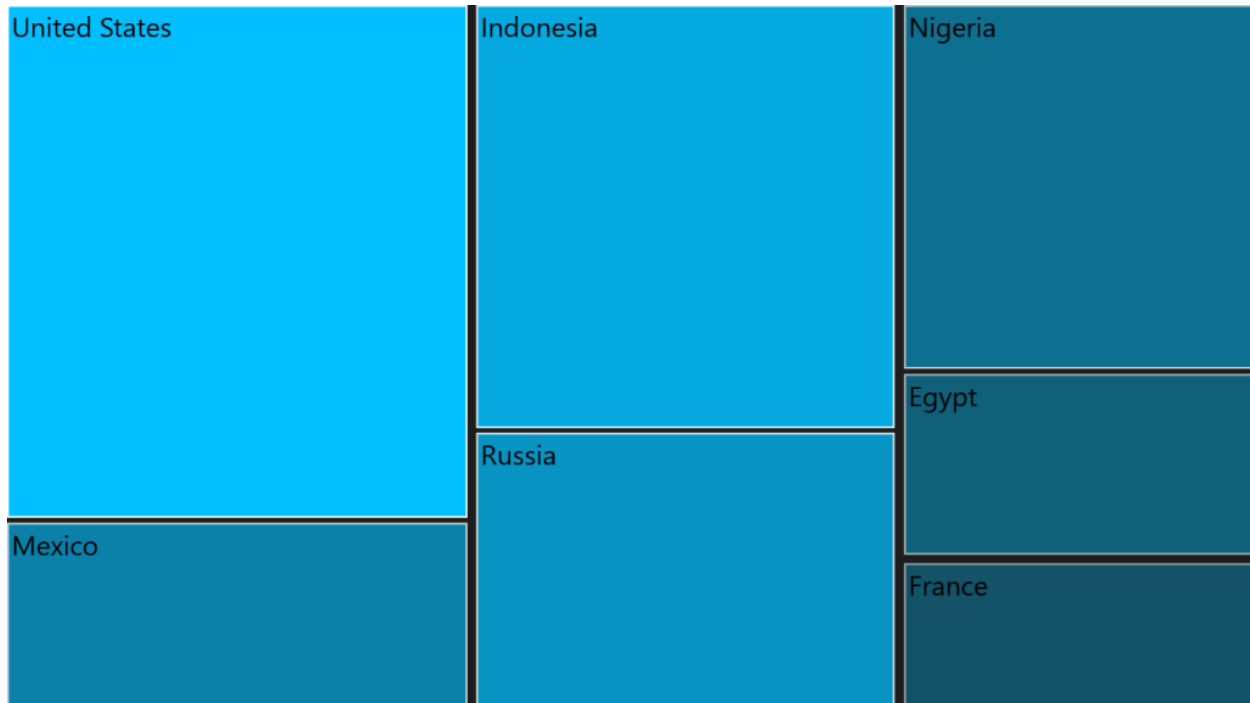


DesaturationColorMapping

The leaf nodes of TreeMap can be colored based upon the Color specified using DesaturationColorMapping. The RangeMinimum and RangeMaximum must be specified to determine the opacity for each leaf node. The opacity of leaf nodes are in the range of From and To mentioned in DesaturationColorMapping.

XML

```
<syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth">
  <syncfusion:SfTreeMap.LeafColorMapping>
    <syncfusion:DesaturationColorMapping From="1" To="0.5"
RangeMinimum="0" RangeMaximum="4" Color="DeepSkyBlue">
    </syncfusion:DesaturationColorMapping>
  </syncfusion:SfTreeMap.LeafColorMapping>
  <syncfusion:SfTreeMap.Levels>
    <syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="10"/>
    <syncfusion:TreeMapFlatLevel GroupPath="Country" GroupGap="5"
ShowLabels="True"/>
  </syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
```

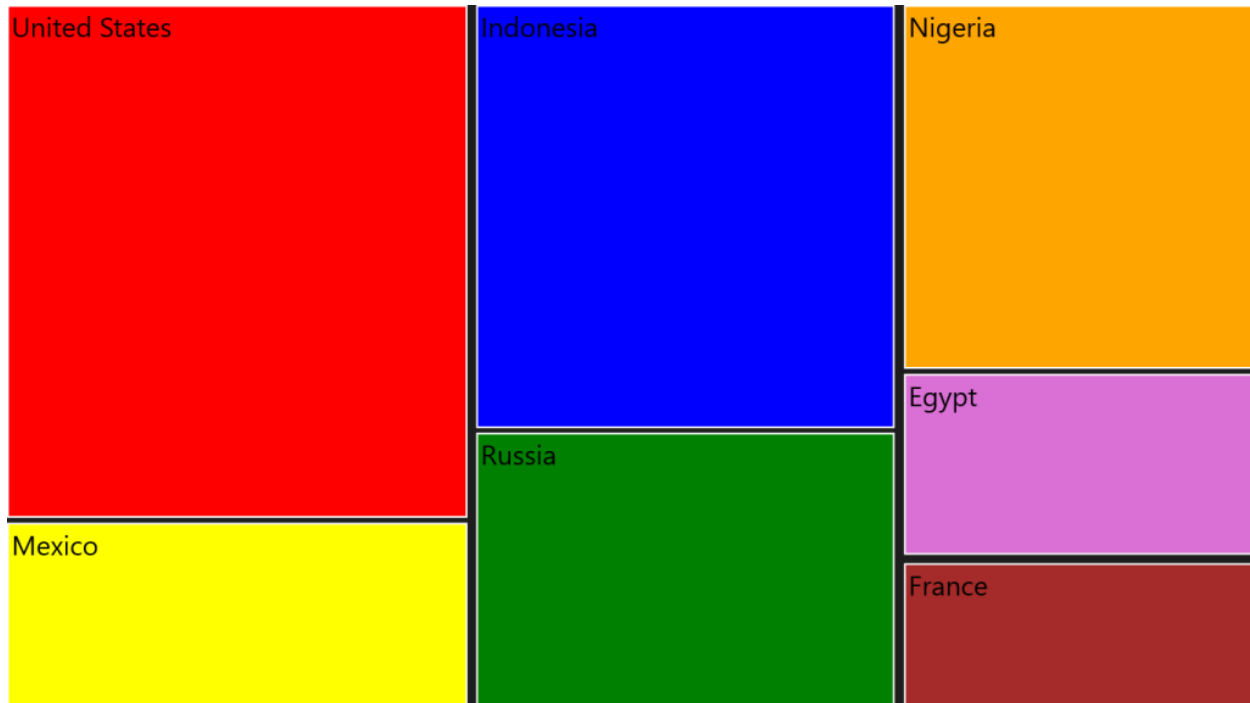


PaletteColorMapping

The leaf nodes are colored by using the brushes mentioned in Colors collection of PaletteColorMapping.

XML

```
<syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth">
  <syncfusion:SfTreeMap.LeafColorMapping>
    <syncfusion:PaletteColorMapping>
      <syncfusion:PaletteColorMapping.Colors>
        <SolidColorBrush Color="Red"/><SolidColorBrush Color="Blue"/>
        <SolidColorBrush Color="Green"/><SolidColorBrush
Color="Yellow"/><SolidColorBrush Color="Orange"/><SolidColorBrush
Color="Orchid"/><SolidColorBrush Color="Brown"/><SolidColorBrush
Color="BlueViolet"/><SolidColorBrush Color="OrangeRed"/>
        <SolidColorBrush Color="Magenta"/>
        <SolidColorBrush Color="Olive"/>
        <SolidColorBrush Color="Crimson"/>
        <SolidColorBrush Color="DeepSkyBlue"/>
      </syncfusion:PaletteColorMapping.Colors>
    </syncfusion:PaletteColorMapping>
  </syncfusion:SfTreeMap.LeafColorMapping>
  <syncfusion:SfTreeMap.Levels>
    <syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="10"/>
    <syncfusion:TreeMapFlatLevel GroupPath="Country" GroupGap="5"
ShowLabels="True"/>
  </syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
```

GroupColorMapping

The leaf nodes are colored by using different ColorMappings available in the TreeMap control. Each group can also be colored with different ColorMappings of TreeMapGroupColorMapping. GroupColorMapping is done based on the GroupID property.

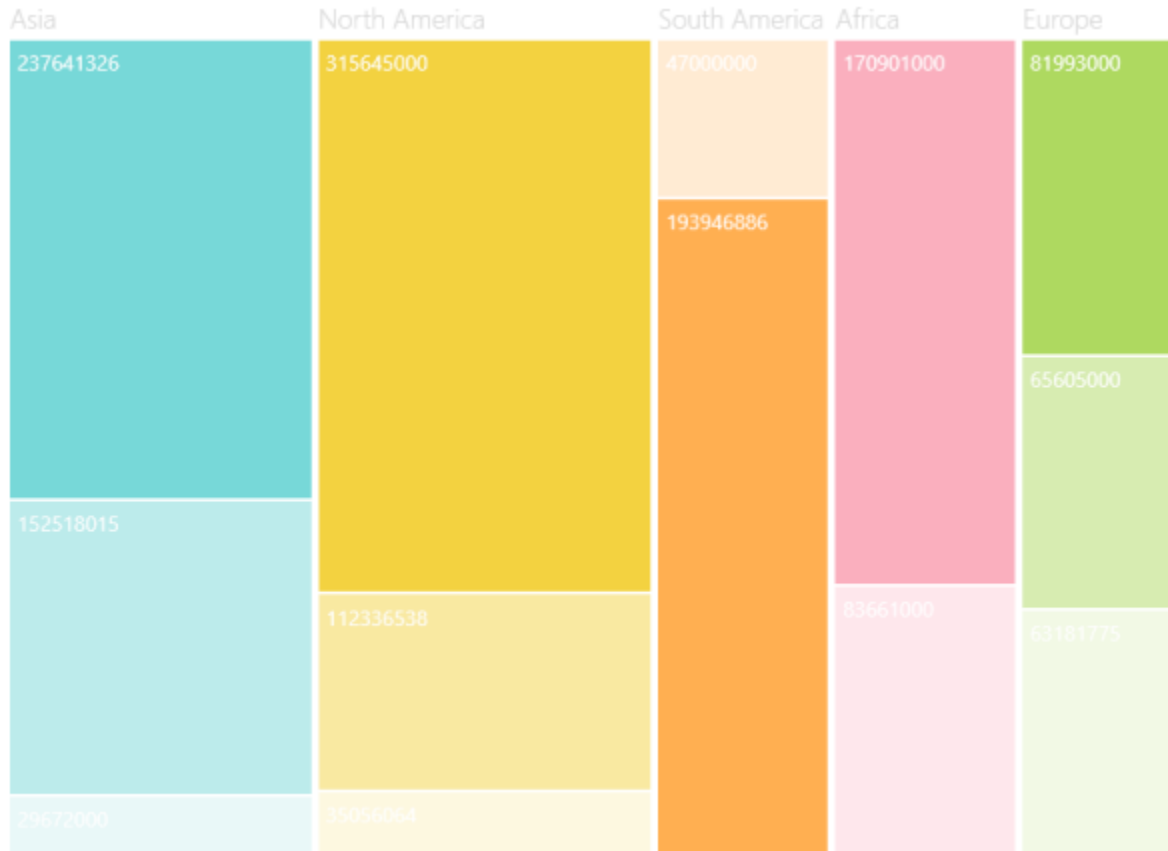
XML

```
<syncfusion:SfTreeMap Name="TreeMap" ItemsSource="{Binding
PopulationDetails}" WeightValuePath="Growth" ColorValuePath="Growth"
ItemsLayoutMode="Squarified" Margin="10">
  <syncfusion:SfTreeMap.GroupColorMappings>
    <syncfusion:GroupColorMapping GroupID="North America">
      <syncfusion:GroupColorMapping.TreeMapColorMapping>
        <syncfusion:DesaturationColorMapping From="1" To="0.1" Color="#F3D240"
RangeMinimum="0" RangeMaximum="6"/>
      </syncfusion:GroupColorMapping.TreeMapColorMapping>
    </syncfusion:GroupColorMapping>
    <syncfusion:GroupColorMapping GroupID="Asia">
      <syncfusion:GroupColorMapping.TreeMapColorMapping>
        <syncfusion:DesaturationColorMapping From="1" To="0.1" Color="#77D8D8"
RangeMinimum="0" RangeMaximum="7"/>
      </syncfusion:GroupColorMapping.TreeMapColorMapping>
    </syncfusion:GroupColorMapping>
    <syncfusion:GroupColorMapping GroupID="Africa">
      <syncfusion:GroupColorMapping.TreeMapColorMapping>
        <syncfusion:DesaturationColorMapping From="1" To="0.1" Color="#faafbe"
RangeMinimum="0" RangeMaximum="6"/>
      </syncfusion:GroupColorMapping.TreeMapColorMapping>
    </syncfusion:GroupColorMapping>
    <syncfusion:GroupColorMapping GroupID="Europe">
      <syncfusion:GroupColorMapping.TreeMapColorMapping>
        <syncfusion:DesaturationColorMapping From="1" To="0.1" Color="#AED960"
RangeMinimum="0" RangeMaximum="6"/>
      </syncfusion:GroupColorMapping.TreeMapColorMapping>
    </syncfusion:GroupColorMapping>
  </syncfusion:SfTreeMap.GroupColorMappings>
</syncfusion:SfTreeMap>
```

```

</syncfusion:GroupColorMapping.TreeMapColorMapping>
</syncfusion:GroupColorMapping>
<syncfusion:GroupColorMapping GroupID="South America">
<syncfusion:GroupColorMapping.TreeMapColorMapping>
<syncfusion:DesaturationColorMapping From="1" To="0" Color="#FFAF51"
RangeMinimum="0" RangeMaximum="6"/>
</syncfusion:GroupColorMapping.TreeMapColorMapping>
</syncfusion:GroupColorMapping>
</syncfusion:SfTreeMap.GroupColorMappings>
</syncfusion:SfTreeMap>

```



see also

[How to apply gradient color mapping to SfTreeMap](#)

TreeMap Legend in WPF TreeMap (SfTreeMap)

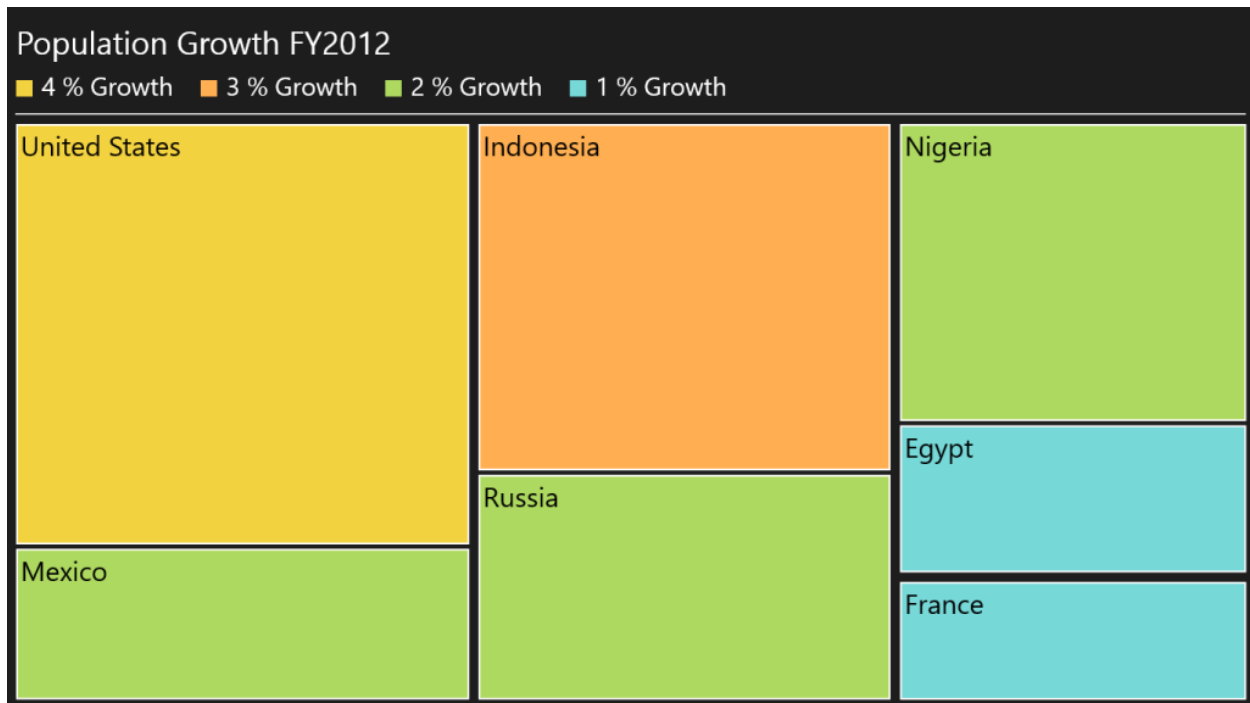
TreeMap legend is used to easily demonstrate about the color value of leaf nodes. But this legend could be appropriate only for the treemap having leaf nodes colored by using RangeBrushColorMapping. The labels of the legend item can be customized by specifying LegendLabel of RangeBrush mentioned in the Brushes of RangeBrushColorMapping.

The icon of legend item can be set by LegendIconStyle of TreeMapLegend. Custom legend icon can be set by assigning DataTemplate to LegendIconTemplate with LegendIconStyle as "Custom". The width and height of the legend icon can be modified by setting LegendIconWidth and LegendIconHeight of TreeMapLegend.

The legend can be positioned to Left, Right, Top or Bottom of TreeMap with the help of LegendPosition property.

XML

```
<syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth"
Margin="10" Name="TreeMap">
  <syncfusion:SfTreeMap.Legend>
    <syncfusion:TreeMapLegend Margin="0 5" LegendItemMargin="0 0 20 10"
LegendItemElementMargin="0 0 10 0" BorderThickness="0,0,0,2"
BorderBrush="#CCCCCC" HorizontalAlignment="Left" LegendIconStyle="Rectangle"
LegendIconHeight="18" LegendIconWidth="18" FontSize="25" Width="{Binding
ActualWidth, ElementName=TreeMap}">
      <syncfusion:TreeMapLegend.Header>
        <TextBlock Text="Population Growth FY2012" FontSize="35"
Margin="0,0,0,10"/>
      </syncfusion:TreeMapLegend.Header>
    </syncfusion:TreeMapLegend>
  </syncfusion:SfTreeMap.Legend>
  <syncfusion:SfTreeMap.LeafColorMapping>
    <syncfusion:RangeBrushColorMapping>
      <syncfusion:RangeBrushColorMapping.Brushes>
        <syncfusion:RangeBrush Color="#77D8D8" From="0" To="1" LegendLabel="1 %
Growth"/>
        <syncfusion:RangeBrush Color="#AED960" From="1" To="2" LegendLabel="2 %
Growth"/>
        <syncfusion:RangeBrush Color="#FFAF51" From="2" To="3" LegendLabel="3 %
Growth"/>
        <syncfusion:RangeBrush Color="#F3D240" From="3" To="4" LegendLabel="4 %
Growth"/>
      </syncfusion:RangeBrushColorMapping.Brushes>
    </syncfusion:RangeBrushColorMapping>
  </syncfusion:SfTreeMap.LeafColorMapping>
  <syncfusion:SfTreeMap.Levels>
    <syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="10"/>
    <syncfusion:TreeMapFlatLevel GroupPath="Country" GroupGap="5"
ShowLabels="True"/>
  </syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
```



Headers and Labels in WPF TreeMap (SfTreeMap)

Headers

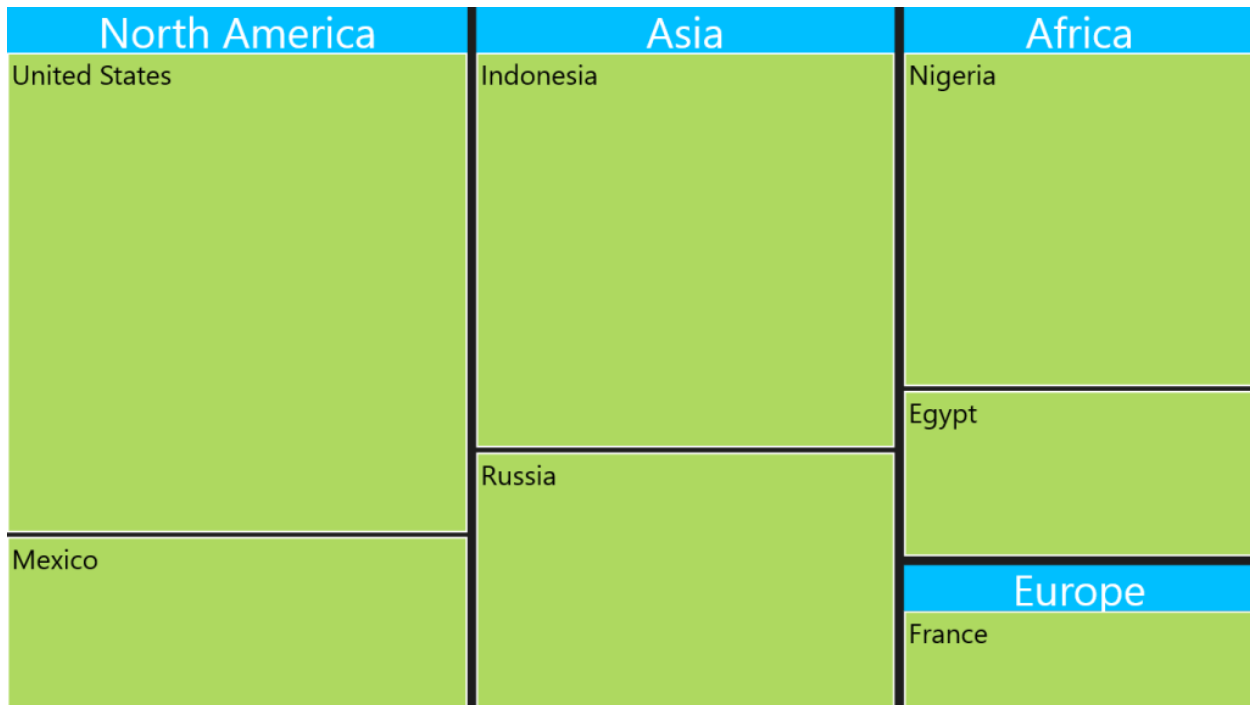
To show headers in TreeMap, you can set the `HeaderHeight` property of `TreeMapLevel`. For customizing default Header appearance, you can specify the `HeaderTemplate`.

TreeMap with Flat Collection:

If `HeaderTemplate` is specified for `TreeMapLevel`, then the header can be bound by referring Header object to the data template.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:PopulationViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
    WeightValuePath="Population" ColorValuePath="Growth">
    <syncfusion:SfTreeMap.Levels>
      <syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="10"
        HeaderHeight="50"/>
      <syncfusion:TreeMapFlatLevel GroupPath="Country" GroupGap="5"/>
    </syncfusion:SfTreeMap.Levels>
  </syncfusion:SfTreeMap>
</Grid>
```



TreeMap with Hierarchical Collection:

For TreeMap with Hierarchical Collection, HeaderPath must be specified. The header can be bound by referring Data.<FieldName> to the data template where FieldName refers to the field of object specified in the particular treemap level.

XML

```
<Grid Background="Black">
  <Grid.Resources>
    <local:CountrySalesCollection x:Key="countrySalesCollection"/>
  </Grid.Resources>
  <syncfusion:SfTreeMap ItemsSource="{Binding Source={StaticResource
countrySalesCollection}}" WeightValuePath="Sales" ColorValuePath="Expense">
    <syncfusion:SfTreeMap.Levels>
      <syncfusion:TreeMapHierarchicalLevel ChildPath="RegionalSalesCollection"
ChildGap="10" HeaderHeight="25" HeaderPath="Name">
      </syncfusion:TreeMapHierarchicalLevel>
    </syncfusion:SfTreeMap.Levels>
  </syncfusion:SfTreeMap>
</Grid>
```

Labels

To show labels in TreeMap, ShowLabels of TreeMapLevel should be enabled to True. For customizing default label appearance, you can specify LabelTemplate.

TreeMap with Flat Collection:

If LabelTemplate is specified for TreeMapLevel, then the label can be bound by referring Label object to the data template.

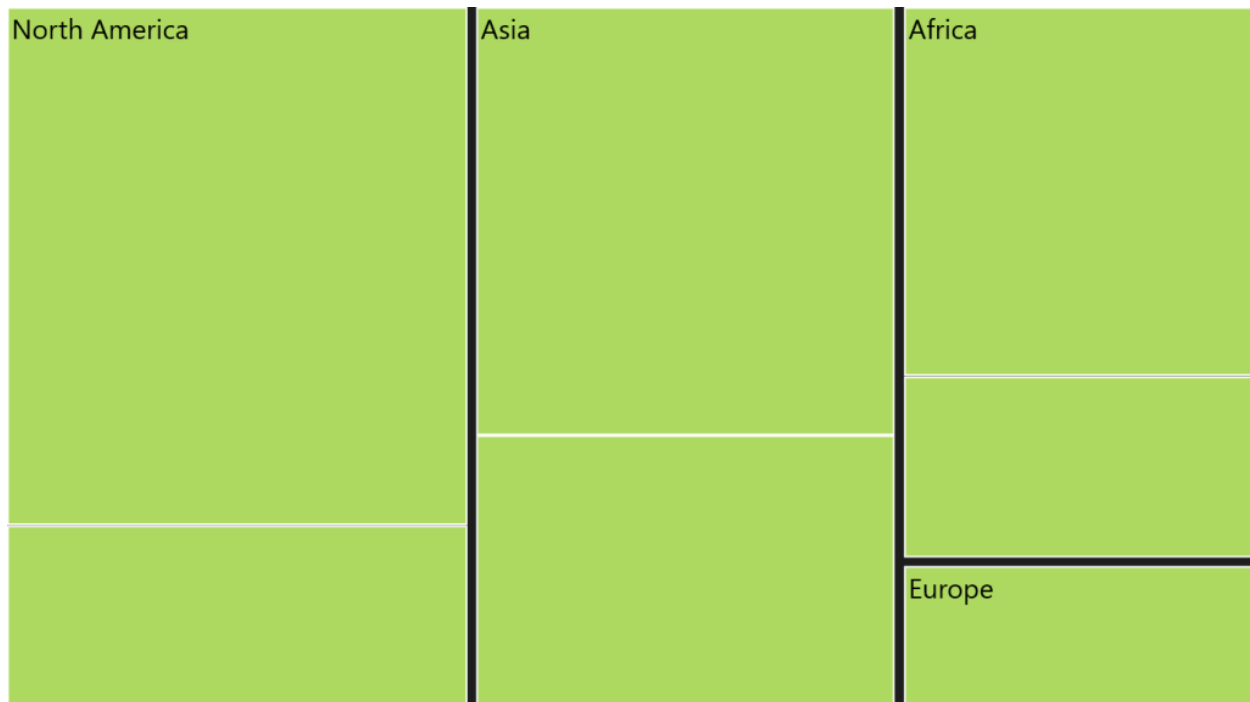
XML

```
<Grid Background="Black">
```

```

<Grid.DataContext>
<local:PopulationViewModel/>
</Grid.DataContext>
<syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
WeightValuePath="Population"
ColorValuePath="Growth">
<syncfusion:SfTreeMap.Levels>
<syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="10"
ShowLabels="True"/>
</syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
</Grid>

```



TreeMap with Hierarchical Collection:

For TreeMap with Hierarchical Collection, LabelPath must be specified. The label can be bound by referring Data. <FieldName> to the data template where FieldName refers to the field of object specified in the particular treemap level.

XML

```

<Grid Background="Black">
<Grid.Resources>
<local:CountrySalesCollection x:Key="countrySalesCollection"/>
</Grid.Resources>
<syncfusion:SfTreeMap ItemsSource="{Binding Source={StaticResource
countrySalesCollection}}">
WeightValuePath="Sales" ColorValuePath="Expense">
<syncfusion:SfTreeMap.Levels>
<syncfusion:TreeMapHierarchicalLevel ChildPath="RegionalSalesCollection"
ChildGap="10" ShowLabels="True" LabelPath="Name">
</syncfusion:TreeMapHierarchicalLevel>
</syncfusion:SfTreeMap.Levels>

```

```
</syncfusion:SfTreeMap>
</Grid>
```

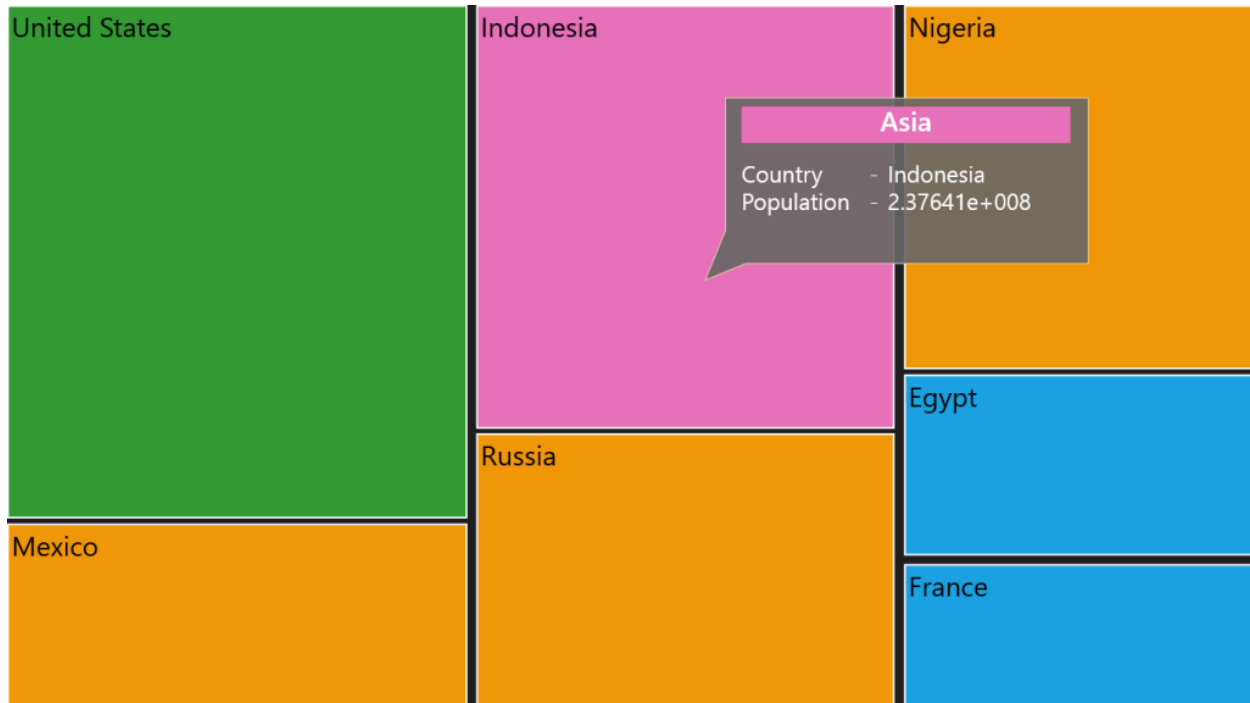
ToolTip Support in WPF TreeMap (SfTreeMap)

You can enable tooltip for TreeMap by setting [ShowToolTip](#) to "True". For modifying default appearance of tooltip, [ToolTipTemplate](#) can be specified and also you can set the [ToolTipShowDuration](#) property to define the animation speed.

XML

```
<syncfusion:SfTreeMap ItemsSource="{Binding PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth"
ShowToolTip="True">
  <syncfusion:SfTreeMap.ToolTipTemplate>
    <DataTemplate>
      <Grid Height="200" Width="420" Margin="0,-420,0,0">
        <Path Data="M0,0 L360,0 L360,200 L20,200 L-20,220 L0,160 L0,0 z"
          Fill="#666666" Stroke="Wheat" Opacity="0.9" Stretch="Fill"/>
        <StackPanel>
          <Grid Background="{Binding MappedColor}" Margin="40 10 20 0" Height="40">
            <TextBlock Text="{Binding Data.Continent}" Foreground="White"
              FontWeight="SemiBold" FontSize="30" TextAlignment="Center"/>
          </Grid>
          <StackPanel Margin="40 20">
            <StackPanel Orientation="Horizontal">
              <TextBlock Text="Country" Width="140" FontSize="25"/>
              <TextBlock Text="-" Width="20" FontSize="25"/>
              <TextBlock Text="{Binding Data.Country}" FontSize="25"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal">
              <TextBlock Text="Population" Width="140" FontSize="25"/>
              <TextBlock Text="-" Width="20" FontSize="25"/>
              <TextBlock Text="{Binding Data.Population}" FontSize="25"/>
            </StackPanel>
          </StackPanel>
        </Grid>
      </DataTemplate>
    </syncfusion:SfTreeMap.ToolTipTemplate>
    <syncfusion:SfTreeMap.LeafColorMapping>
      <syncfusion:RangeBrushColorMapping>
        <syncfusion:RangeBrushColorMapping.Brushes>
          <syncfusion:RangeBrush From="0" To="1" Color="#1BA1E2"/>
          <syncfusion:RangeBrush From="1" To="2" Color="#F09609"/>
          <syncfusion:RangeBrush From="2" To="3" Color="#E671B8"/>
          <syncfusion:RangeBrush From="3" To="4" Color="#339933"/>
        </syncfusion:RangeBrushColorMapping.Brushes>
      </syncfusion:RangeBrushColorMapping>
    </syncfusion:SfTreeMap.LeafColorMapping>
    <syncfusion:SfTreeMap.Levels>
      <syncfusion:TreeMapFlatLevel GroupPath="Continent" GroupGap="10"/>
      <syncfusion:TreeMapFlatLevel GroupPath="Country" GroupGap="5"
        ShowLabels="True"/>
    </syncfusion:SfTreeMap.Levels>
  </syncfusion:SfTreeMap>
```

The following screenshot shows a tree map with a tool tip.



Selection Support in WPF TreeMap (SfTreeMap)

While selecting a leaf node, you can highlight it by setting `HighlightOnSelection` property of `SfTreeMap` to "True". The border of highlight on selection can be customized by `HighlightBorderBrush` and `HighlightBorderThickness` properties of `SfTreeMap`. `SelectionMode` can also be set to either "Default" or "Multiple". "Multiple" selection of leaf nodes is made possible by pressing the control key continuously while Mouse Click happens.

XML

```
<syncfusion:SfTreeMap Name="TreeMap" ItemsSource="{Binding
PopulationDetails}"
WeightValuePath="Population"
ColorValuePath="Growth"
HighlightOnSelection="True"
HighlightBorderBrush="Red"
HighlightBorderThickness="4"
HighlightHoverBrush="Yellow"
SelectionMode="Multiple">
</syncfusion:SfTreeMap>
```




GroupSelection support is also provided under selection support where the whole group can be selected. While selecting a leaf node, you can highlight it by setting HighlightGroupOnSelection property of SfTreeMap to "True". The helper properties, HighlightBorderBrush, HighlightBorderThickness, and SelectionModes are shared for both HighlightOnSelection and HighlightGroupOnSelection.

XML

```
<syncfusion:SfTreeMap Name="TreeMap" ItemsSource="{Binding
PopulationDetails}"
WeightValuePath="Population"
ColorValuePath="Growth"
HighlightGroupOnSelection="True"
HighlightBorderBrush="Red"
HighlightBorderThickness="4"
SelectionMode="Multiple" >
</syncfusion:SfTreeMap>
```



see also

[How to highlight group selection](#)

Customizing Leaf Nodes in WPF TreeMap (SfTreeMap)

You can customize leaf nodes by assigning data template to LeafTemplate of SfTreeMap.

XML

```
<Grid Background="Black">
  <Grid.DataContext>
    <local:OlympicMedalsViewModel/>
  </Grid.DataContext>
  <syncfusion:SfTreeMap ItemsSource="{Binding OlympicMedalsDetails}"
    Margin="50" WeightValuePath="TotalMedals" ColorValuePath="GoldMedals">
    <syncfusion:SfTreeMap.LeafTemplate>
      <DataTemplate>
        <Border BorderBrush="Transparent" BorderThickness="3" Background="#D73028">
          <Image Source="{Binding Data.GameImgSource}" HorizontalAlignment="Center"
            VerticalAlignment="Center" Margin="0,25,0,0" Stretch="None"/>
        </Border>
      </DataTemplate>
    </syncfusion:SfTreeMap.LeafTemplate>
    <syncfusion:SfTreeMap.Levels>
      <syncfusion:TreeMapFlatLevel GroupPath="GameName" ShowLabels="True">
        <syncfusion:TreeMapFlatLevel.LabelTemplate>
          <DataTemplate>
```

```

<TextBlock Padding="10 5 0 0" Text="{Binding Label}" FontSize="20"
HorizontalAlignment="Left" VerticalAlignment="Top"/>
</DataTemplate>
</syncfusion:TreeMapFlatLevel.LabelTemplate>
</syncfusion:TreeMapFlatLevel>
</syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>
</Grid>

```

C#

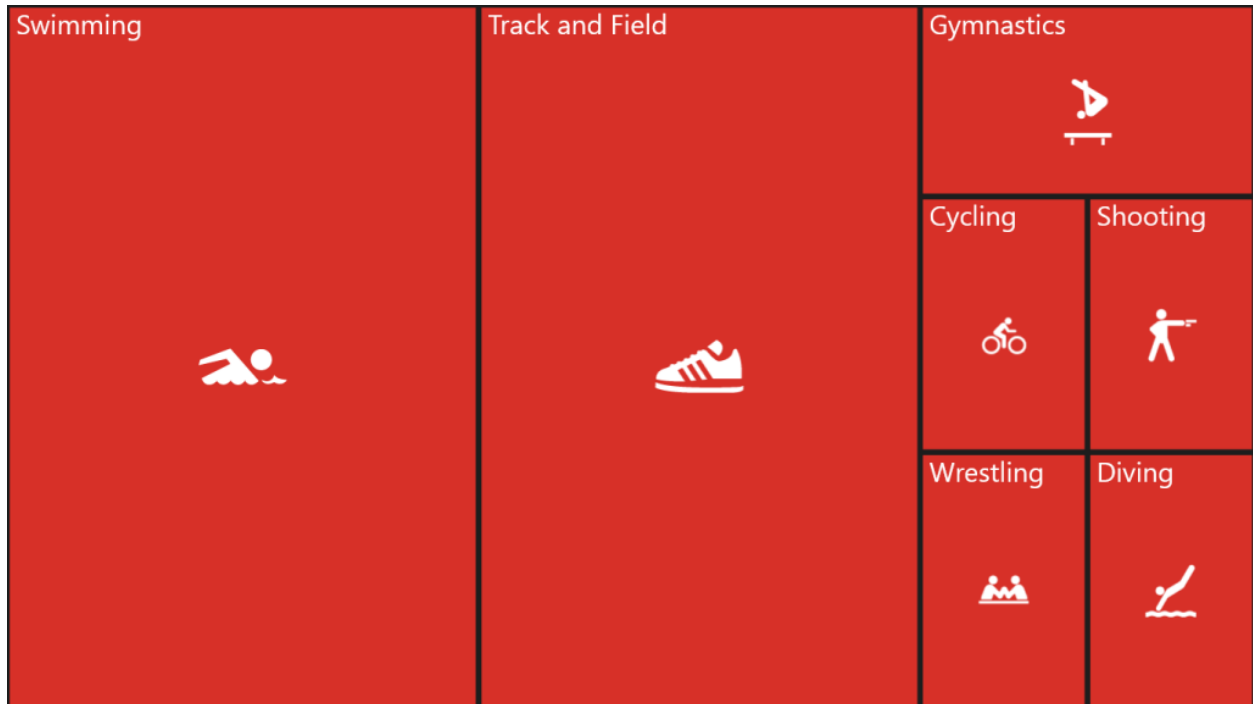
```

public class OlympicMedalsViewModel
{
    public ObservableCollection<OlympicMedals> OlympicMedalsDetails
    { get; set; }
    public OlympicMedalsViewModel()
    {
        this.OlympicMedalsDetails = new ObservableCollection<OlympicMedals>();
        this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Swimming", GoldMedals = 16, SilverMedals = 9, BronzeMedals = 6, TotalMedals
= 31, GameImgSource = new BitmapImage(new Uri("ms-
appx:/Assets/Swimming.png")) });
        this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Track and Field", GoldMedals = 9, SilverMedals = 13, BronzeMedals = 7,
TotalMedals = 29, GameImgSource = new BitmapImage(new Uri("ms-
appx:/Assets/TrackAndField.png")) });
        this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Gymnastics", GoldMedals = 3, SilverMedals = 1, BronzeMedals = 2,
TotalMedals = 6, GameImgSource = new BitmapImage(new Uri("ms-
appx:/Assets/Gymnastics.png")) });
        this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Cycling", GoldMedals = 1, SilverMedals = 2, BronzeMedals = 1, TotalMedals =
4, GameImgSource = new BitmapImage(new Uri("ms-appx:/Assets/Cycling.png"))
});
        this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Shooting", GoldMedals = 3, SilverMedals = 0, BronzeMedals = 1, TotalMedals
= 4, GameImgSource = new BitmapImage(new Uri("ms-
appx:/Assets/Shooting.png")) });
        this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Wrestling", GoldMedals = 2, SilverMedals = 0, BronzeMedals = 2, TotalMedals
= 4, GameImgSource = new BitmapImage(new Uri("ms-
appx:/Assets/Wrestling.png")) });
        this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Diving", GoldMedals = 1, SilverMedals = 1, BronzeMedals = 2, TotalMedals =
4, GameImgSource = new BitmapImage(new Uri("ms-appx:/Assets/Diving.png"))
});
    }
}

public class OlympicMedals
{
    public string Country { get; set; }
    public string GameName { get; set; }
    public double GoldMedals { get; set; }
    public double SilverMedals { get; set; }
    public double BronzeMedals { get; set; }
    public double TotalMedals { get; set; }
}

```

```
public ImageSource GameImgSource { get; set; }
```



Drill Down Support in WPF TreeMap (SfTreeMap)

TreeMap enables drill down to expose the hierarchy by clicking on a treemap node and allows drill up by clicking on drill down header. At a time, only one level of the hierarchy can be seen in the treemap.

Enabling Drill Down

Treemap items can be drilled down by enabling the property `EnableDrillDown` to `'true'`. The hierarchy of treemap levels can be shown by clicking on treemap items. The previous level can be drilled up by clicking on drill down header. DrillDown header can be customized with the help of `DrillDownHeaderTemplate` property.

Drill Down Properties

Property	Type	Description
<code>EnableDrillDown</code>	Bool	Gets or sets a value to indicate whether the drill down support should be enabled.
<code>DrillDownHeaderHeight</code>	Double	Gets or sets a value for specifying the height for drill down header.
<code>DrillDownHeaderTemplate</code>	DataTemplate	Gets or sets a template to customize drill down header.
<code>DrillDownSelectionStroke</code>	Brush	Gets or sets a color for highlighting tree map item while drill down.

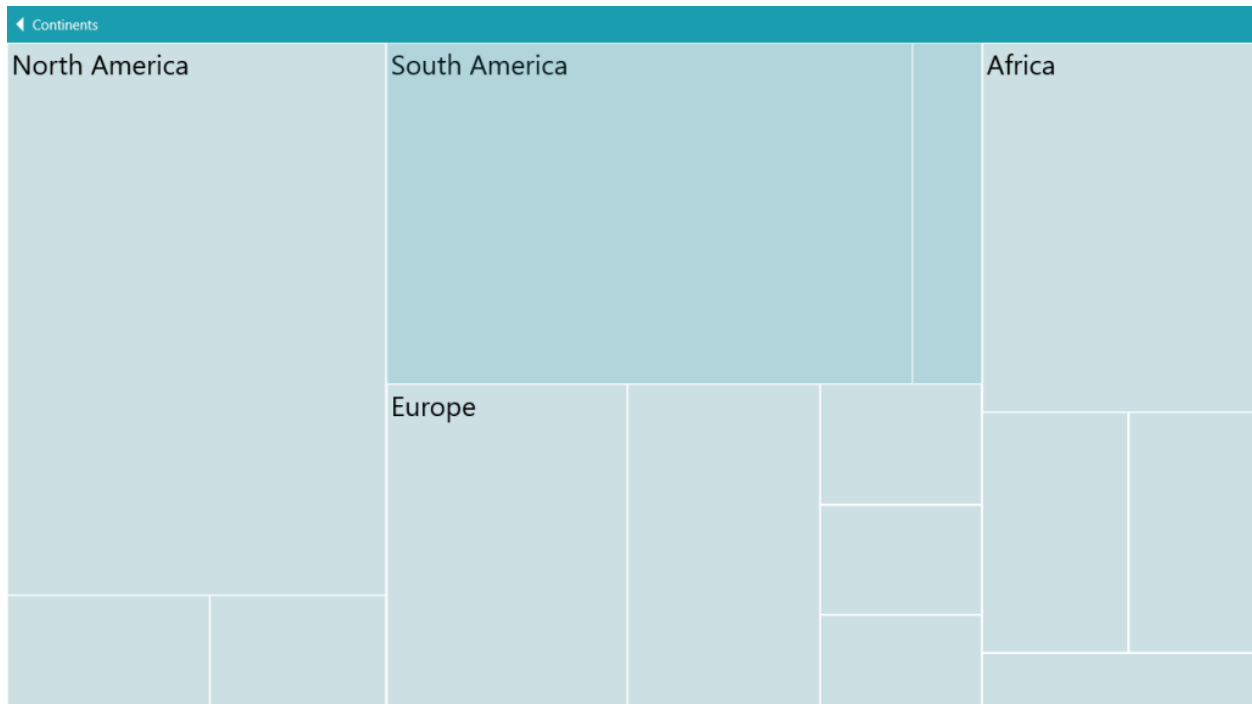
XML

```

<syncfusion:SfTreeMap x:Name="TreeMap" ItemsSource="{Binding}"
EnableDrillDown="True"
DrillDownSelectionStroke="#1A9DAF"
WeightValuePath="Population" ColorValuePath="Area"
LeafLabelPath="Name" BorderThickness="1">
<syncfusion:SfTreeMap.DrillDownHeaderTemplate>
<DataTemplate>
<Border Background="#1A9DAF">
<StackPanel Orientation="Horizontal" VerticalAlignment="Center" Margin="10
0">
<Path x:Name="path" Data="M197,153.5 L197,138 186.75,145.5 z" Height="16"
Width="8" Fill="White" Stretch="Fill" />
<TextBlock Text="{Binding}" Margin="10 0" FontSize="15"
FontWeight="Normal" FontFamily="Segoe UI" Foreground="White"/>
</StackPanel>
</Border>
</DataTemplate>
</syncfusion:SfTreeMap.DrillDownHeaderTemplate>
<syncfusion:SfTreeMap.LeafColorMapping>
<syncfusion:UniColorMapping Color="#CCDFE3"/>
</syncfusion:SfTreeMap.LeafColorMapping>
<syncfusion:SfTreeMap.Levels>
<syncfusion:TreeMapFlatLevel GroupPath="Continent" ShowLabels="True"/>
<syncfusion:TreeMapFlatLevel GroupPath="Country" ShowLabels="True"/>
</syncfusion:SfTreeMap.Levels>
</syncfusion:SfTreeMap>

```

The following screenshot illustrates a TreeMap with drill down support.



TreeMap with drill down support

see also

[How to customize leaf level SfTreeMap](#)

[How to apply colors based on the ColorValuePath to a leaf template in SfTreeMap](#)

SfTreeGrid

Getting Started with WPF TreeGrid (SfTreeGrid)

The [WPF TreeGrid](#) (SfTreeGrid) is a data oriented control that displays the self-relational and hierarchical data in tree structure with columns. The data can be loaded on-demand also.

Assembly Deployment

The following list of assemblies needs to be added as reference to use SfTreeGrid control in any application,

Required assemblies	Description
Syncfusion.Data.WPF	Syncfusion.Data.WPF assembly is dependent assembly for Syncfusion.SfGrid.WPF
Syncfusion.SfGrid.WPF	Syncfusion.SfGrid.WPF assembly contains classes that handles all UI operations of SfTreeGrid.SfTreeGrid control present in Syncfusion.UI.Xaml.TreeGrid namespace.This namespace also added in http://schemas.syncfusion.com/wpf Syncfusion WPF schema.
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF contains various editor controls (such as IntegerTextBox, DoubleTextBox and etc) which are used in SfTreeGrid.

In order to use export to excel and export to PDF functionalities of SfTreeGrid control, add the reference to following assemblies,

Optional Assemblies	Description
Syncfusion.SfGridConverter.WPF	Syncfusion.SfGridConverter.WPF contains static extension classes for exporting SfTreeGrid to excel and PDF in Syncfusion.UI.Xaml.TreeGrid.Converter namespace.
Syncfusion.XlsIO.Base	Syncfusion.XlsIO.Base contains fundamental and base classes for creating and manipulating excel files.
Syncfusion.Pdf.Base	Syncfusion.Pdf.Base contains fundamental and base classes for creating PDF.

Creating simple application with SfTreeGrid

In this walk through, you will create WPF application with SfTreeGrid control.

1. [Creating project](#)
2. [Adding control via Designer](#)
3. [Adding control manually in XAML](#)
4. [Adding control manually in C#](#)

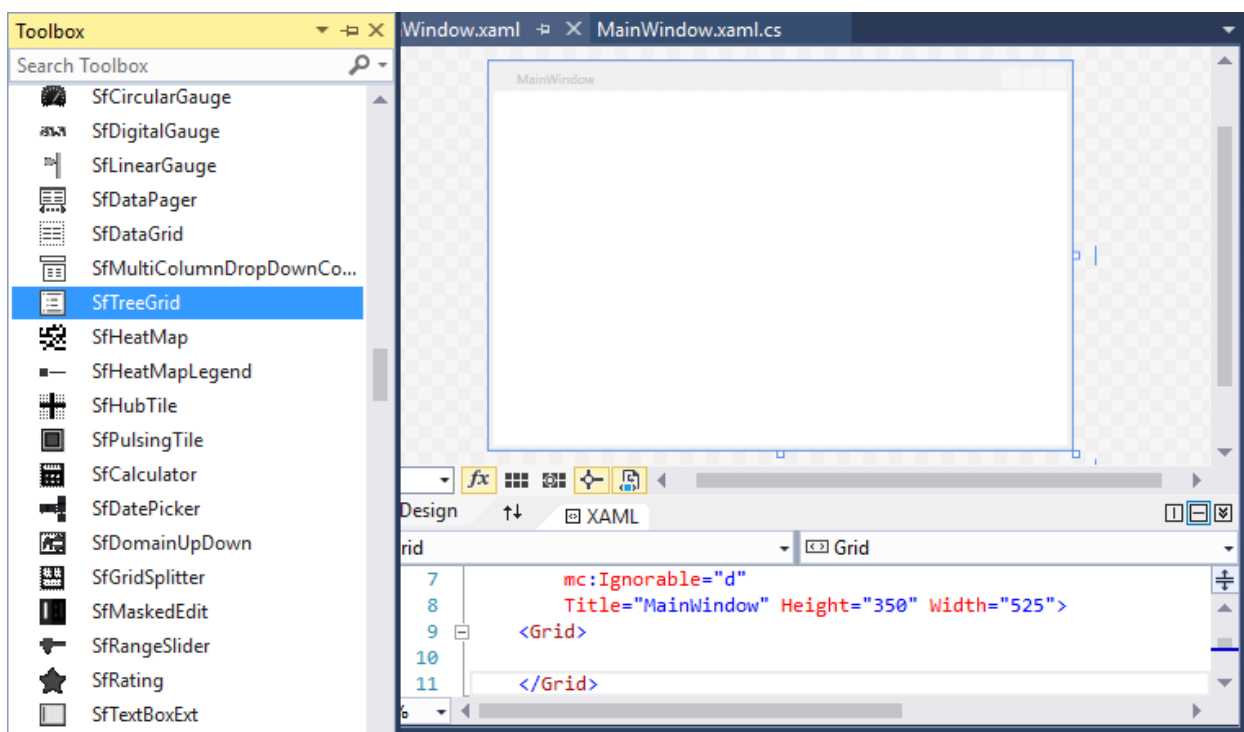
5. [Binding self-relational data in SfTreeGrid](#)
6. [Binding nested collection with SfTreeGrid](#)
7. [Populate SfTreeGrid in on-demand](#)
8. [Defining Columns](#)
9. [Selection](#)
10. [Sorting](#)
11. [Editing](#)

Creating the project

Create new WPF project in Visual Studio to display SfTreeGrid with data objects.

Adding control via Designer

WPF TreeGrid (SfTreeGrid) control can be added to the application by dragging it from Toolbox and dropping it in Designer view. The required assembly references will be added automatically.



Adding control manually in XAML

In order to add control manually in XAML, do the below steps,

1. Add the below required assembly references to the project as shown in the below image,
 - o Syncfusion.Data.WPF
 - o Syncfusion.SfGrid.WPF
 - o Syncfusion.Shared.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> or SfTreeGrid control namespace **Syncfusion.UI.Xaml.TreeGrid** in XAML page.
3. Declare SfTreeGrid control in XAML page.

XAML

```
<Window x:Class="WpfApplication1.MainWindow"
```

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow"
Width="525"
Height="350">
<Grid>
<syncfusion:SfTreeGrid x:Name="treeGrid" />
</Grid>
</Window>
```

Adding control manually in C\#

To add control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.Data.WPF
 - Syncfusion.SfGrid.WPF
 - Syncfusion.Shared.WPF
2. Import SfTreeGrid namespace **Syncfusion.UI.Xaml.TreeGrid**.
3. Create SfTreeGrid control instance and add it to the Page.

C#

```
using Syncfusion.UI.Xaml.TreeGrid;
namespace WpfApplication1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfTreeGrid treeGrid = new SfTreeGrid();
            Root_Grid.Children.Add(treeGrid);
        }
    }
}
```

Binding self-relational data in SfTreeGrid

WPF TreeGrid (SfTreeGrid) supports to bind self-relational data by setting [SfTreeGrid.ParentPropertyName](#) and [SfTreeGrid.ChildPropertyName](#) properties where tree structure is formed based on these two properties.

SfTreeGrid.ParentPropertyName – Denotes the property in data object which is used to identify the root nodes.

SfTreeGrid.ChildPropertyName - Denotes the property in data object which is used identify its parent by matching the property value with **ParentPropertyName** property value of other data objects.

The data objects which has unique property value in **SfTreeGrid.ParentPropertyName** or the data objects which has the property value as in [SfTreeGrid.SelfRelationRootValue](#) are root nodes.

Creating Data Model for self-relational collection

SfTreeGrid is a data-bound control. So, before binding to the control, you must create data model for application.

1. Creating data object class named **EmployeeInfo** and declare properties as shown below,

C#

```
public class EmployeeInfo
{
    int _id;
    string _firstName;
    string _lastName;
    private string _title;
    double? _salary;
    int _reportsTo;
    public string FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }
    public string LastName
    {
        get { return _lastName; }
        set { _lastName = value; }
    }
    public int ID
    {
        get { return _id; }
        set { _id = value; }
    }
    public string Title
    {
        get { return _title; }
        set { _title = value; }
    }
    public double? Salary
    {
        get { return _salary; }
        set { _salary = value; }
    }
    public int ReportsTo
    {
        get { return _reportsTo; }
        set { _reportsTo = value; }
    }
}
```

Note: If you want your data object (EmployeeInfo class) to automatically reflect property changes, then the object must implement [INotifyPropertyChanged](#) interface.

2. Create a **ViewModel** class with Employees property and Employees property is initialized with several data objects in constructor.

C#

```
public class ViewModel
{
    public ViewModel()
    {
        this.Employees = this.GetEmployees();
    }
    private ObservableCollection<EmployeeInfo> _employees;
    public ObservableCollection<EmployeeInfo> Employees
    {
        get { return _employees; }
        set { _employees = value; }
    }
    private ObservableCollection<EmployeeInfo> GetEmployees()
    {
        ObservableCollection<EmployeeInfo> employeeDetails = new
        ObservableCollection<EmployeeInfo>();
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Ferando", LastName =
        "Joseph", Title = "Management", Salary = 2000000, ReportsTo = -1, ID = 2 });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "John", LastName =
        "Adams", Title = "Accounts", Salary = 2000000, ReportsTo = -1, ID = 3 });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Thomas", LastName =
        "Jefferson", Title = "Sales", Salary = 300000, ReportsTo = -1, ID = 4 });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Andrew", LastName =
        "Madison", Title = "Marketing", Salary = 4000000, ReportsTo = -1, ID = 5 });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Ulysses", LastName =
        "Pierce", Title = "HumanResource", Salary = 1500000, ReportsTo = -1, ID = 6
        });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Jimmy", LastName =
        "Harrison", Title = "Purchasing", Salary = 200000, ReportsTo = -1, ID = 7
        });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Ronald", LastName =
        "Fillmore", Title = "Production", Salary = 2800000, ReportsTo = -1, ID = 8
        });
        //Management
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Andrew", LastName =
        "Fuller", ID = 9, Salary = 1200000, ReportsTo = 2, Title = "Vice President"
        });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Janet", LastName =
        "Leverling", ID = 10, Salary = 1000000, ReportsTo = 2, Title = "GM" });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Steven", LastName =
        "Buchanan", ID = 11, Salary = 900000, ReportsTo = 2, Title = "Manager" });
        //Accounts
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Nancy", LastName =
        "Davolio", ID = 12, Salary = 850000, ReportsTo = 3, Title = "Accounts
        Manager" });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Margaret", LastName =
        "Peacock", ID = 13, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Michael", LastName =
        "Suyama", ID = 14, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Robert", LastName =
        "King", ID = 15, Salary = 650000, ReportsTo = 3, Title = "Accountant" });
        //Sales
        employeeDetails.Add(new EmployeeInfo() { FirstName = "Laura", LastName =
        "Callahan", ID = 16, Salary = 900000, ReportsTo = 4, Title = "Sales Manager"
        });
    }
}
```

```

employeeDetails.Add(new EmployeeInfo() { FirstName = "Anne", LastName =
"Dodsworth", ID = 17, Salary = 800000, ReportsTo = 4, Title = "Sales
Representative" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Albert", LastName =
"Hellstern", ID = 18, Salary = 750000, ReportsTo = 4, Title = "Sales
Representative" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Tim", LastName =
"Smith", ID = 19, Salary = 700000, ReportsTo = 4, Title = "Sales
Representative" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Justin", LastName =
"Brid", ID = 20, Salary = 700000, ReportsTo = 4, Title = "Sales
Representative" });
//Back Office
employeeDetails.Add(new EmployeeInfo() { FirstName = "Caroline", LastName =
"Patterson", ID = 21, Salary = 800000, ReportsTo = 5, Title = "Receptionist"
});
employeeDetails.Add(new EmployeeInfo() { FirstName = "Xavier", LastName =
"Martin", ID = 22, Salary = 700000, ReportsTo = 5, Title = "Mail Clerk" });
//HR
employeeDetails.Add(new EmployeeInfo() { FirstName = "Laurent", LastName =
"Pereira", ID = 23, Salary = 900000, ReportsTo = 6, Title = "HR Manager" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Syed", LastName =
"Abbas", ID = 24, Salary = 650000, ReportsTo = 6, Title = "HR Assistant" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Amy", LastName =
"Alberts", ID = 25, Salary = 650000, ReportsTo = 6, Title = "HR Assistant"
});
//Purchasing
employeeDetails.Add(new EmployeeInfo() { FirstName = "Pamela", LastName =
"Ansman-Wolfe", ID = 26, Salary = 600000, ReportsTo = 7, Title = "Purchase
Manager" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Michael", LastName =
"Blythe", ID = 27, Salary = 550000, ReportsTo = 7, Title = "Store Keeper"
});
employeeDetails.Add(new EmployeeInfo() { FirstName = "David", LastName =
"Campbell", ID = 28, Salary = 450000, ReportsTo = 7, Title = "Store Keeper"
});
//Production
employeeDetails.Add(new EmployeeInfo() { FirstName = "Jillian", LastName =
"Carson", ID = 29, Salary = 600000, ReportsTo = 8, Title = "Production
Manager" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Shu", LastName =
"Ito", ID = 30, Salary = 550000, ReportsTo = 8, Title = "Production
Engineer" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Stephen", LastName =
"Jiang", ID = 31, Salary = 450000, ReportsTo = 8, Title = "Production
Engineer" });
return employeeDetails;
}
}

```

Binding to Data

To bind the SfTreeGrid to data, set [SfTreeGrid.ItemsSource](#) property to an IEnumerable of implementation and to form tree structure from self-relational data, set [SfTreeGrid.ParentPropertyName](#) and [SfTreeGrid.ChildPropertyName](#) properties. Each row is SfTreeGrid is bound to an object in ItemsSource and each column is bound to a property in data object.

Bind the self-relations collection created in the previous step to `SfTreeGrid.ItemsSource` property and set `ParentPropertyName` as `ID` and `ChildPropertyName` as `ReportsTo` to form the tree structure as shown below,

XML

```
<Window x:Class="GettingStarted.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow"
Width="525"
Height="350">
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid x:Name="Root_Grid">
<syncfusion:SfTreeGrid Name="treeGrid"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID"
SelfRelationRootValue="-1" />
</Grid>
</Window>
```

C#

```
using Syncfusion.UI.Xaml.TreeGrid;
namespace GettingStarted
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfTreeGrid treeGrid = new SfTreeGrid();
            ViewModel viewModel = new ViewModel();
            treeGrid.ParentPropertyName = "ID";
            treeGrid.ChildPropertyName = "ReportsTo";
            treeGrid.SelfRelationRootValue = -1;
            treeGrid.AutoExpandMode = AutoExpandMode.RootNodesExpanded;
            treeGrid.ItemsSource = viewModel.Employees;
            Root_Grid.Children.Add(treeGrid);
        }
    }
}
```

FirstName	LastName	ID	
⊕ Ferando	Joseph	2	M
⊖ John	Adams	3	A
Nancy	Davolio	12	A
Margaret	Peacock	13	A
Michael	Suyama	14	A
Robert	King	15	A
⊕ Thomas	Jefferson	4	S
⊕ Andrew	Madison	5	M
⊕ Ulysses	Pierce	6	H
⊖ Jimmy	Harrison	7	P
Pamela	Ansman-Wolfe	26	P

Binding Nested collection with SfTreeGrid

WPF TreeGrid (SfTreeGrid) supports to bind nested or hierarchical collection (where each data object has hierarchy within) by setting the property name to [SfTreeGrid.ChildPropertyName](#) which holds the child collection.

Creating Data Model for nested collection

1. Create data object class named **PersonInfo** and declare properties as shown below,

C#

```
public class PersonInfo
{
    private string _firstName;
    private string _lastName;
    private bool _available;
    private double _salary;
    private ObservableCollection<PersonInfo> _children;
    public string FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }
    public string LastName
    {
        get { return _lastName; }
        set { _lastName = value; }
    }
    public bool Availability
    {
        get { return _available; }
        set { _available = value; }
    }
}
```

```

    }
    public double Salary
    {
        get { return _salary; }
        set { _salary = value; }
    }
    public ObservableCollection<PersonInfo> Children
    {
        get { return _children; }
        set { _children = value; }
    }
}

```

2. Create a **ViewModel** class with PersonDetails property and PersonDetails property is initialized with several data objects in constructor

C#

```

public class ViewModel
{
    public ViewModel()
    {
        this.PersonDetails = this.CreatePersonData();
    }
    private ObservableCollection<PersonInfo> _personDetails;
    public ObservableCollection<PersonInfo> PersonDetails
    {
        get { return _personDetails; }
        set { _personDetails = value; }
    }
    private ObservableCollection<PersonInfo> CreatePersonData()
    {
        var personList = new ObservableCollection<PersonInfo>();
        ObservableCollection<PersonInfo> childCollection1 = new
        ObservableCollection<PersonInfo>();
        childCollection1.Add(new PersonInfo() { FirstName = "Andrew", LastName =
        "Fuller",Availability=true, Salary = 1200000 });
        childCollection1.Add(new PersonInfo() { FirstName = "Theodore", LastName =
        "Hoover",Availability=true, Salary = 1200000 });
        childCollection1.Add(new PersonInfo() { FirstName = "Harry", LastName =
        "Nixon",Availability=false, Salary = 1200000 });
        ObservableCollection<PersonInfo> childCollection2 = new
        ObservableCollection<PersonInfo>();
        childCollection2.Add(new PersonInfo { FirstName = "Ronald", LastName =
        "Fillmore", Availability = false, Salary = 23000 });
        childCollection2.Add(new PersonInfo() { FirstName = "Steven", LastName =
        "Buchanan", Availability = true, Salary = 340000 });
        childCollection2.Add(new PersonInfo() { FirstName = "Robert", LastName =
        "King", Availability = true, Salary = 32000 });
        personList.Add(new PersonInfo() { FirstName = "Obama", LastName =
        "bosh",Availability=false, Salary = 2000000, Children = childCollection1 });
        personList.Add(new PersonInfo() { FirstName = "John", LastName =
        "Adams",Availability=true, Salary = 2000000, Children = childCollection2 });
    }
}

```

```

personList.Add(new PersonInfo() { FirstName = "Thomas", LastName =
"Jefferson",Availability=true, Salary = 300000, Children = childCollection1
});
personList.Add(new PersonInfo() { FirstName = "Andrew", LastName =
"Madison",Availability=false, Salary = 400000, Children = childCollection2
});
personList.Add(new PersonInfo() { FirstName = "Ulysses", LastName =
"Pierce",Availability=true, Salary = 150000, Children = childCollection1
});
personList.Add(new PersonInfo() { FirstName = "Jimmy", LastName =
"Harrison",Availability=false, Salary = 200000, Children = childCollection2
});
personList.Add(new PersonInfo() { FirstName = "Ronald", LastName =
"Fillmore", Availability=false,Salary = 280000, Children = childCollection1
});
return personList;
}
}

```

Binding to Data

To bind the SfTreeGrid to data, set ItemsSource property to an IEnumerable of implementation and to form tree structure from nested collection data, set [SfTreeGrid.ChildPropertyName](#). Each row in SfTreeGrid is bound to an object in ItemsSource and each column is bound to a property in data object.

Bind the nested collection created in the previous step to [SfTreeGrid.ItemsSource](#) property and set ChildPropertyName as Children to form the tree structure as shown below,

XML

```

<Window x:Class="NestedCollectionDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:NestedCollectionDemo"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow"
Width="525"
Height="350">
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid x:Name="Root_Grid">
<syncfusion:SfTreeGrid Name="treeGrid"
ChildPropertyName="Children"
ItemsSource="{Binding PersonDetails}" />
</Grid>
</Window>

```

C#

```

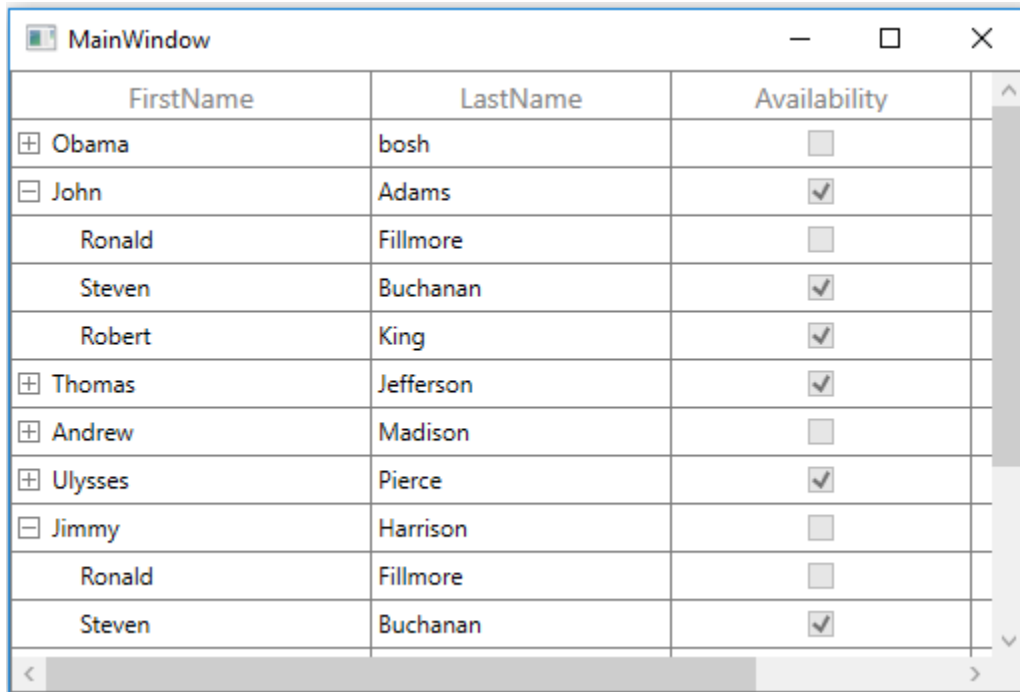
using Syncfusion.UI.Xaml.TreeGrid;
namespace NestedCollectionDemo
{
public partial class MainWindow : Window
{
public MainWindow()

```

```

{
InitializeComponent();
SfTreeGrid treeGrid = new SfTreeGrid();
ViewModel viewModel = new ViewModel();
treeGrid.ItemsSource = viewModel.PersonDetails;
treeGrid.ChildPropertyName = "Children";
Root_Grid.Children.Add(treeGrid);
}
}
}

```



Defining Columns

By default, the WPF TreeGrid (SfTreeGrid) control generates the columns automatically when value assigned to [SfTreeGrid.ItemsSource](#) property. The type of the column generated depends on the type of data in the column and the attribute of the property the column bound with.

The following table lists the column types and its constraints for auto column generation.

Generated Column Type	Data Type / Attribute
TreeGridTextColumn	Property of type String and any other type apart from below specified cases.
TreeGridNumericColumn	Property of type Int or Double
TreeGridDateTimeColumn	Property of type DateTime
TreeGridCheckBoxColumn	Property of type Bool
TreeGridCurrencyColumn	Property with Currency DataType attribute.[DataType(DataType.Currency)].

TreeGridMaskColumn	Property with PhoneNumber DataType attribute. [DataType(DataType.PhoneNumber)].
TreeGridHyperLinkColumn	Property of type Uri

When columns are auto-generated, you can handle the [SfTreeGrid.AutoGeneratingColumn](#) event to customize or cancel the columns before they are added to the SfTreeGrid.

You can prevent the automatic column generation by setting [SfTreeGrid.AutoGenerateColumns](#) property to false. When SfTreeGrid.AutoGenerateColumns property is false, you should define the columns to be displayed as below,

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="AllNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ItemsSource="{Binding PersonDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName" />
<syncfusion:TreeGridTextColumn HeaderText="Last Name" MappingName="LastName"
/>
<syncfusion:TreeGridTextColumn HeaderText="ID"
MappingName="Id"
TextAlignment="Left" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

Below is the list of column types provided in SfTreeGrid.

Column Type	Comments
TreeGridTextColumn	Represents SfTreeGrid column that hosts textual content in its cells.
TreeGridNumericColumn	Represents SfTreeGrid column that hosts DoubleTextBox control in its cells which is used to format and display Numeric values.
TreeGridDateTimeColumn	Represents SfTreeGrid column that hosts DateTimeEdit control in its cells which is used to display and format DateTime values.
TreeGridComboBoxColumn	Represents SfTreeGrid column that hosts ComboBox control in its cells.
TreeGridCheckBoxColumn	Represents SfTreeGrid column that hosts CheckBox control in its cells.

TreeGridMaskColumn	Represents SfTreeGrid column that hosts a <code>MaskedTextBox</code> control in its cells which is used to display textual content by applying Mask.
TreeGridCurrencyColumn	Represents SfTreeGrid column that hosts CurrencyTextBox control in its cells which is used to display numeric values with currency format.
TreeGridPercentColumn	Represents SfTreeGrid column that hosts PercentTextBox control in its cells which is used to display numeric values with percent format.
TreeGridHyperlinkColumn	Represents SfTreeGrid column that hosts <code>HyperLink</code> control in its cells.
TreeGridTemplateColumn	Represents SfTreeGrid column that hosts template-specified content in its cells

Selection

By default, the entire row is selected when a user clicks a cell in a SfTreeGrid. You can set the [SfTreeGrid.SelectionMode](#) property to specify whether a user can select single row or cell, or multiple rows or cells.

You can handle the selection operations with the help of [SfTreeGrid.SelectionChanging](#) and [SfTreeGrid.SelectionChanged](#) events.

Sorting

By default, you can sort columns in a SfTreeGrid by clicking the column header. You can configure the sorting by setting [SfTreeGrid.SortColumnDescriptions](#) property.

You can customize sorting by handling the [SfTreeGrid.SortColumnChanging](#) and [SfTreeGrid.SortColumnChanged](#) events. To cancel the default sort, set the `Cancel` property to true in `SfTreeGrid.SortColumnChanging` event.

Editing

Editing can be enabled by setting [SfTreeGrid.AllowEditing](#) property to True. You can customize the editing operations by handling [SfTreeGrid.CurrentCellBeginEdit](#) and [SfTreeGrid.CurrentCellEndEdit](#) events.

Filtering

Filtering can be enabled by setting the [SfTreeGrid.AllowFiltering](#) property to true, where advanced filter UI can be opened by clicking the filter icon in column header to filter the nodes in SfTreeGrid. The filtering operations can be customized by handling the [SfTreeGrid.FilterChanging](#) and [SfTreeGrid.FilterChanged](#) events.

First Name	Last Name	Employee ID	Date of Joining	Contact Number	City
Ches			11/15/2010	999116	Perth
Al			1/20/2011	999111	Barcelona
Ul			11/6/2011	999118	San Francisco
Za			1/10/2008	999116	Birmingham
Th			5/17/2008	999118	Madrid
Ul			1/11/2008	999115	San Francisco
Rc			8/14/2009	999118	Liverpool
Grov			1/10/2009	999117	San Francisco
Ge			9/18/2008	999112	Zurich
Za			11/25/2011	999113	London
W			7/1/2008	999113	Sydney
Ge			5/16/2010	999117	San Francisco
M			4/16/2011	999114	NewYork
M			1/27/2008	999112	NewYork
Mart			9/11/2010	999115	Adelaide
Fr			10/5/2010	999113	Canberra
Th			7/21/2010	999118	Cardiff
W			4/24/2008	999112	London
W			3/13/2008	999115	Manchester
Andrew	Clinton	115	3/18/2010	999112	NewYork

Theme

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

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

FirstName	LastName	ID	Title	Salary	ReportsTo
▼ Ferando	Joseph	2	Management	2000000.00	-1
Andrew	Fuller	9	Vice President	1200000.00	2
Janet	Leverling	10	GM	1000000.00	2
Steven	Buchanan	11	Manager	900000.00	2
▼ John	Adams	3	Accounts	2000000.00	-1
Nancy	Davolio	12	Accounts Manager	850000.00	3
Margaret	Peacock	13	Accountant	700000.00	3
Michael	Suyama	14	Accountant	700000.00	3
Robert	King	15	Accountant	650000.00	3
▼ Thomas	Jefferson	4	Sales	300000.00	-1
Laura	Callahan	16	Sales Manager	900000.00	4
Anne	Dodsworth	17	Sales Representative	800000.00	4
Albert	Hellstern	18	Sales Representative	750000.00	4
Tim	Smith	19	Sales Representative	700000.00	4
Justin	Brid	20	Sales Representative	700000.00	4
▼ Andrew	Madison	5	Marketing	4000000.00	-1
Caroline	Patterson	21	Receptionist	800000.00	5
Xavier	Martin	22	Mail Clerk	700000.00	5
▼ Ulysses	Pierce	6	HumanResource	1500000.00	-1
Laurent	Pereira	23	HR Manager	900000.00	6
Syed	Abbas	24	HR Assistant	650000.00	6
Amy	Alberts	25	HR Assistant	650000.00	6
▼ Jimmy	Harrison	7	Purchasing	200000.00	-1
Pamela	Ansman-Wolfe	26	Purchase Manager	600000.00	7
Michael	Blythe	27	Store Keeper	550000.00	7
David	Campbell	28	Store Keeper	450000.00	7
▼ Ronald	Fillmore	8	Production	2800000.00	-1
Jillian	Carson	29	Production Manager	600000.00	8
Shu	Ito	30	Production Engineer	550000.00	8
Stephen	Jiang	31	Production Engineer	450000.00	8

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Data Binding in WPF TreeGrid (SfTreeGrid)

SfTreeGrid is designed to display the self-relational and hierarchical data in tree structure with columns. The data binding can be achieved by assigning the data source to [SfTreeGrid.ItemsSource](#) property directly through self-relational binding or nested collection or retrieving the parent and child nodes items dynamically using [RequestTreeItems](#) or [LoadOnDemandCommand](#).

If the data source implements [INotifyCollectionChanged](#) interface, then SfTreeGrid control will automatically refresh the UI when item is added, removed or while list cleared. When you add, remove item in [ObservableCollection](#), SfTreeGrid automatically refresh the UI as [ObservableCollection](#) implements [INotifyCollectionChanged](#). But when you do the same in [List](#), SfTreeGrid will not refresh the UI automatically.

Below are the ways to bind the data source to SfTreeGrid.

- [Populate data using self-relational binding](#)
- [Populate data using nested collection](#)

- [Populate data with RequestTreeItems event](#)
- [Populate data with LoadOnDemandCommand](#)

Binding with IEnumerable

SfTreeGrid control supports to bind any collection that implements the [IEnumerable](#) interface. Data operations such as sorting is supported when you are binding collection derived from [IEnumerable](#).

Binding with dynamic data object

SfTreeGrid control supports to bind [dynamic data object](#). Below are the limitations when you are binding dynamic data object,

1. In UWP, UI won't get refreshed when you are changing the property value. This is limitation in UWP platform.
2. SfTreeGrid doesn't support [LiveNodeUpdateMode](#) - [AllowDataShaping](#).

Binding Complex properties

SfTreeGrid control provides support to bind complex property to its columns. To bind the complex property to [TreeGridColumn](#), set the complex property path to [MappingName](#).

XML

```
<syncfusion:SfTreeGrid x:Name="treeGrid"
ChildPropertyName="Children"
ItemsSource="{Binding PersonDetails}"
ShowRowHeader="True">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridColumn>
<syncfusion:TreeGridTextColumn MappingName="FirstName" />
<syncfusion:TreeGridTextColumn MappingName="Address.City" />
<syncfusion:TreeGridTextColumn MappingName="Address.Country" />
</syncfusion:TreeGridColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

Limitations when binding complex property:

SfTreeGrid doesn't support [LiveNodeUpdateMode](#) - [AllowDataShaping](#).

Binding Indexer properties

SfTreeGrid control provides support to bind an indexer property to its columns. To bind an indexer property to [TreeGridColumn](#), set the indexer property path to [MappingName](#).

XML

```
<syncfusion:SfTreeGrid x:Name="treeGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Students}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn MappingName="RollNo" />
<syncfusion:TreeGridTextColumn MappingName="Name" />
<syncfusion:TreeGridTextColumn MappingName="Marks[0]" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
this.treeGrid.Columns.Add(new TreeGridTextColumn()
{MappingName="Marks[0]"});
```

Limitations when binding complex property:

SfTreeGrid doesn't support [LiveNodeUpdateMode](#) - [AllowDataShaping](#).

AutoExpandMode

By setting [SfTreeGrid.AutoExpandMode](#) property, you can let the SfTreeGrid to expand the nodes while loading.

Mode	Description
AllNodesExpanded	All the nodes will be expanded while at the time of loading.
None	None defines the node is not expanded when loading. By default, root nodes only will be displayed.
RootNodeExpanded	Root nodes only will be expanded at the time of loading.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="AllNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID"/>
```

C#

```
treeGrid.AutoExpandMode = AutoExpandMode.AllNodesExpanded;
```

Expanding a tree node

You can expand a node based on its level or row index. Here is a list of ways to expand a node,

Method	Description
ExpandAllNodes()	Expands all the nodes including its inner leaf nodes
ExpandAllNodes(int level)	Expand all the nodes up to the specified level
ExpandAllNodes(TreeNode treeNode)	Expand the specific node and its child nodes
ExpandNode(int rowIndex)	Expand a node at the specific row Index
ExpandNode(TreeNode treeNode)	Expands the specific node.

Expand all the nodes

You can expand all the nodes programmatically at runtime by using [SfTreeGrid.ExpandAllNodes](#) method.

C#

```
treeGrid.ExpandAllNodes();
```

Expand the nodes based on its level

You can expand the nodes based on the level by using [SfTreeGrid.ExpandAllNodes](#) method by passing level as argument.

C#

```
treeGrid.ExpandAllNodes(1);
```

Expand the specific node

You can expand the specific node by using [SfTreeGrid.ExpandNode](#) method.

C#

```
var node = treeGrid.View.Nodes[0];  
treeGrid.ExpandNode(node);
```

You can expand the node at specific index also by using [SfTreeGrid.ExpandNode](#) method.

C#

```
treeGrid.ExpandNode(2);
```

Expand the specific node based on business object

You can expand the node corresponding to specific data object by resolving node and calling [SfTreeGrid.ExpandNode](#) method.

C#

```
var data = (this.DataContext as ViewModel).PersonDetails[0];  
var node = this.treeGrid.View.Nodes.GetNode(data);  
treeGrid.ExpandNode(node);
```

Expand all the nodes

You can expand the specific node and all its child nodes by using [SfTreeGrid.ExpandAllNodes](#) methods.

C#

```
var node = treeGrid.View.Nodes[0];  
treeGrid.ExpandAllNodes(node);
```

Cancel the node Expanding using NodeExpanding event

You can cancel the node expanding through [SfTreeGrid.NodeExpanding](#) event.

C#

```
treeGrid.NodeExpanding += TreeGrid_NodeExpanding;  
private void TreeGrid_NodeExpanding(object sender, NodeExpandingEventArgs e)
```

```
{
    e.Cancel = true;
}
```

You can cancel the specific node being expanded by using `SfTreeGrid.NodeExpanding` event and `Node` property,

C#

```
treeGrid.NodeExpanding += TreeGrid_NodeExpanding;
private void TreeGrid_NodeExpanding(object sender, NodeExpandingEventArgs e)
{
    if ((e.Node.Item as EmployeeInfo).ID == 2)
        e.Cancel = true;
}
```

NodeExpanded Event

You can get the notification once a node is expanded from [TreeGrid.NodeExpanded](#) event and here you can get the expanded node and its child nodes.

C#

```
private void TreeGrid_NodeExpanded(object sender, NodeExpandedEventArgs e)
{
    var node = e.Node;
    var childNodes = e.Node.ChildNodes;
}
```

Collapsing a tree node

You can collapse all the tree nodes or specific node and its child nodes. Here is a list of ways to collapse a node,

Method	Description
CollapseAllNodes()	Collapse all the nodes in SfTreeGrid
CollapseAllNodes(TreeNode treeNode)	Collapse the specific node and its child nodes
CollapseNode(int rowIndex)	Collapse the node at specific row index
CollapseNode(TreeNode treeNode)	Collapse the specific node in SfTreeGrid

Collapse all the nodes

You can collapse all the nodes programmatically at runtime by using [SfTreeGrid.CollapseAllNodes](#) methods

C#

```
treeGrid.CollapseAllNodes();
```

Collapse the specific node

You can collapse the specific node by using [SfTreeGrid.CollapseNode](#) method.

C#

```
var node = treeGrid.View.Nodes[0];  
treeGrid.CollapseNode(node);
```

You can collapse the node at specific index also by using [SfTreeGrid.CollapseNode](#) method.

C#

```
treeGrid.CollapseNode(2);
```

Collapse the specific node based on business object

You can collapse the node corresponding to specific data object by resolving node and calling [SfTreeGrid.CollapseNode](#) method.

C#

```
var data = (this.DataContext as ViewModel).PersonDetails[0];  
var node = this.treeGrid.View.Nodes.GetNode(data);  
treeGrid.CollapseNode(node);
```

Collapse all the nodes

You can collapse the specific node and all its child nodes by using [SfTreeGrid.CollapseAllNodes](#) methods.

C#

```
var node = treeGrid.View.Nodes[0];  
treeGrid.CollapseAllNodes(node);
```

Cancel the node collapsing using NodeCollapsing Event

You can cancel the node collapsing operation through [TreeGrid.NodeCollapsing](#) event.

C#

```
treeGrid.NodeCollapsing += TreeGrid_NodeCollapsing;  
private void TreeGrid_NodeCollapsing(object sender, NodeCollapsingEventArgs e)  
{  
    e.Cancel = true;  
}
```

You can cancel the specific node being collapsed by using [TreeGrid.NodeCollapsing](#) event.

C#

```
treeGrid.NodeCollapsing += TreeGrid_NodeCollapsing;  
private void TreeGrid_NodeCollapsing(object sender, NodeCollapsingEventArgs e)  
{  
    if ((e.Node.Item as EmployeeInfo).ID == 2)  
        e.Cancel = true;  
}
```

NodeCollapsed Event

You can get the notification once a node is collapsed from [TreeGrid.NodeCollapsed](#) event and here you can get the collapsed node.

C#

```
treeGrid.NodeCollapsed += TreeGrid_NodeCollapsed;
private void TreeGrid_NodeCollapsed(object sender, NodeCollapsedEventArgs e)
{
    var node = e.Node;
}
```

Expand/Collapse a node based on mapping property

SfTreeGrid supports to expand/collapse the nodes based on the value of a boolean mapping property in the underlying data object by using [SfTreeGrid.ExpandStateMappingName](#) property. TreeGrid expand/collapse the node when the specified property value in underlying data object gets changed.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
ExpandStateMappingName="IsExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding PersonDetails}"
ShowCheckBox="True"/>
```

C#

```
treeGrid.ExpandStateMappingName = "IsExpanded";
```

LiveNodeUpdateMode

SfTreeGrid listens and responds to the manipulation such as add, delete and data update (property change) at runtime. SfTreeGrid refreshes the sorting based on [SfTreeGrid.LiveNodeUpdateMode](#) property. If you set [LiveNodeUpdateMode](#) as [AllowDataShaping](#), sorting will be refreshed on data manipulation and property change.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowEditing="True"
AutoExpandMode="RootNodesExpanded"
LiveNodeUpdateMode="AllowDataShaping"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID"/>
```

C#

```
treeGrid.LiveNodeUpdateMode = LiveNodeUpdateMode.AllowDataShaping;
```

Events

ItemsSourceChanged

[SfTreeGrid.ItemsSourceChanged](#) event occurs when the data source is changed by using [ItemsSource](#) property.

This event receives two arguments namely sender that handles SfTreeGrid and [GridItemsSourceChangedEventArgs](#) as objects.

The [GridItemsSourceChangedEventArgs](#) object contains the following properties:

- [OldItemsSource](#) - Gets the value of old data source
- [NewItemsSource](#) - Get the value of new data source

View

SfTreeGrid has the [View](#) property of type [TreeGridView](#). [View](#) is responsible for maintaining and manipulating the nodes and other advanced operations like Sorting. When you bind [Collection](#) to [ItemsSource](#) property of SfTreeGrid or wire [RequestTreeItems](#) event, then [View](#) will be created and maintains the operations on nodes.

SfTreeGrid creates different types of views derived from [TreeGridView](#) based on data population methods.

[TreeGridSelfRelationalView](#) - While populating data using self-relational binding

[TreeGridNestedView](#) - While populating data using nested collection

[TreeGridUnboundView](#) - While populating data with [RequestTreeItems](#) event or [LoadOnDemandCommand](#) command

Events

The following events are associated with [View](#).

RecordPropertyChanged

This event is raised when the [DataModel](#) property value is changed, if the [DataModel](#) implements the [INotifyPropertyChanged](#) interface. The event receives with two arguments namely [sender](#) that handles the [DataModel](#) and [PropertyChangedEventArgs](#) as object.

[PropertyChangedEventArgs](#) has below property,

[PropertyName](#) – It denotes the [PropertyName](#) of the changed value.

NodeCollectionChanged

This event is raised whenever there is some change in nodes collection. The event receives two arguments namely sender that handles [View](#) object and [NotifyCollectionChangedEventArgs](#) as object.

[NotifyCollectionChangedEventArgs](#) has below properties,

- [Action](#) - It contains the current action. (i.e.) Add, Remove, Move, Replace, Reset.
- [NewItems](#) - It contains the list of new items involved in the change.
- [OldItems](#) - It contains the list of old items affected by the Action.
- [NewStartingIndex](#) - It contains the index at which the change occurred.
- [OldStartingIndex](#) - It contains the index at which the Action occurred.

SourceCollectionChanged

This event is raised when you make changes in **SourceCollection** for example add or remove the collection. The event receives two arguments namely sender that handles that handles View object and **NotifyCollectionChangedEventArgs** as object.

NotifyCollectionChangedEventArgs has below properties,

- **Action** - It contains the current action. (i.e) Add, Remove, Move, Replace, Reset.
- **NewItems** - It contains the list of new items involved in the change.
- **OldItems** - It contains the list of old items affected by the Action.
- **NewStartingIndex** - It contains the index at which the change occurred.
- **OldStartingIndex** - It contains the index at which the Action occurred.

Methods

The following are the methods that are associated with **View** which can be used to defer refresh the view.

DeferRefresh

Enter the defer cycle so that you can perform all data operations in view and update once. You can refresh the nodes or completely recreates the nodes by using [TreeViewRefreshMode](#) parameter.

C#

```
using (treeGrid.View.DeferRefresh(TreeViewRefreshMode.NodeRefresh))
{
    treeGrid.SortColumnDescriptions.Add(new SortColumnDescription() { ColumnName = "FirstName", SortDirection = ListSortDirection.Descending });
    treeGrid.SortColumnDescriptions.Add(new SortColumnDescription() { ColumnName = "Id", SortDirection = ListSortDirection.Descending });
}
```

BeginInit and EndInit

When `BeginInit` method is called, it suspends all the updates until `EndInit` method is called. You can perform all the update with in these methods and update the view at once. You can refresh the nodes or completely recreates the nodes by using [TreeViewRefreshMode](#) parameter.

C#

```
treeGrid.View.BeginInit(TreeViewRefreshMode.DeferRefresh);
treeGrid.SortColumnDescriptions.Add(new SortColumnDescription() { ColumnName = "FirstName", SortDirection = ListSortDirection.Descending });
treeGrid.SortColumnDescriptions.Add(new SortColumnDescription() { ColumnName = "Id", SortDirection = ListSortDirection.Descending });
treeGrid.View.EndInit();
```

Note: View has properties (**EnableRecursiveChecking**, **LiveNodeUpdateMode**, **RecursiveCheckingMode**,...) that already defined in SfTreeGrid. It is recommended to set those properties via SfTreeGrid.

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Columns in WPF TreeGrid (SfTreeGrid)

SfTreeGrid allows you to add or remove columns using [SfTreeGrid.Columns](#) property. You can choose the columns to be added from built-in column types or you can create your own column and add to the [SfTreeGrid.Columns](#).

Below are the built-in column types supported in SfTreeGrid. Each column has its own properties to handle different types of data.

Column Type	Description
TreeGridTextColumn	Use to display the string data.
TreeGridNumericColumn	Use to display the numeric data
TreeGridCurrencyColumn	Use to display the currency value.
TreeGridPercentColumn	Use to display the percent value.
TreeGridMaskColumn	Use to display the data to be masked.
TreeGridDateTimeColumn	Use to display the date time value
TreeGridComboBoxColumn	Use to display the IEnumerable data using Combo Box.
TreeGridCheckBoxColumn	Use to display the Boolean type data
TreeGridHyperlinkColumn	Use to display the URI data
TreeGridTemplateColumn	Use to display the custom template-specified content.

Defining Columns

You can let the SfTreeGrid to create columns or you can manually defined columns to be displayed. Below sections explain 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 [SfTreeGrid.AutoGenerateColumns](#). Default value is `true`.

Columns are generated based on type of property. For example, [TreeGridNumericColumn](#) is added for `int` type property. Below are table shows data type and its column type. For remaining types, [TreeGridTextColumn](#) will be added.

Data Type	Column
-----------	--------

string, object, dynamic	TreeGridTextColumn
int, float, double, decimal and also itâ€™s nullable	TreeGridNumericColumn
DateTime, DateTimeOffset and also itâ€™s nullable	TreeGridDateTimeColumn
Uri, Uri?	TreeGridHyperLinkColumn
bool, bool?	TreeGridCheckBoxColumn

Note: The order of columns in the collection will determine the order of that they will appear in SfTreeGrid.

AutoGenerateColumns with different modes

Column auto generation is controlled using [SfTreeGrid.AutoGenerateColumnsMode](#) property.

The [SfTreeGrid.AutoGenerateColumnsMode](#) includes the following modes.

Mode	Behavior	When ItemsSource changed
Reset	Generates the columns based on the properties defined in the underlying data object.	Keeps the columns added manually. Clears the columns which are auto generated before and creates new columns based on new ItemsSource.
RetainOld	Generates the columns based on the properties defined in the underlying data object.	The same columns will be maintained when changing ItemsSource also. So filtering, sorting and grouping settings will be maintained.
ResetAll	Generates the columns based on the properties defined in the underlying data object.	Clear all the columns including the columns defined manually and creates new columns based on new ItemsSource.
None	Columns will not be generated.	Keeps old columns in TreeGrid.Columns collection.

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.treeGrid.AutoGeneratingColumn += TreeGrid_AutoGeneratingColumn;
private void TreeGrid_AutoGeneratingColumn(object sender,
TreeGridAutoGeneratingColumnEventArgs e)
{
}
```

[TreeGridAutoGeneratingColumnEventArgs](#) provides the information about the auto-generated column to the `AutoGeneratingColumn` event. [TreeGridAutoGeneratingColumnEventArgs.Column](#) property returns the newly created column.

Cancel column generation for particular property

You can cancel the specific column adding to the TreeGrid by handling `AutoGeneratingColumn` event.

In the below code, column generation for `ReportsTo` property is canceled by setting `Cancel` property to `true`.

C#

```
treeGrid.AutoGeneratingColumn += TreeGrid_AutoGeneratingColumn;
private void TreeGrid_AutoGeneratingColumn(object sender,
TreeGridAutoGeneratingColumnEventArgs e)
{
    if (e.Column.MappingName == "ReportsTo")
        e.Cancel = true;
}
```

Changing column type

You can change the type of column adding to SfTreeGrid by setting the instance of column you want to add in `AutoGeneratingColumn` event.

In the below code, column type for `Salary` property is changed to `TreeGridTextColumn` by setting instance of `TreeGridTextColumn` to `Column` property.

C#

```
treeGrid.AutoGeneratingColumn += TreeGrid_AutoGeneratingColumn;
private void TreeGrid_AutoGeneratingColumn(object sender,
TreeGridAutoGeneratingColumnEventArgs e)
{
    if (e.Column.MappingName == "Salary")
    {
        if (e.Column is TreeGridNumericColumn)
            e.Column = new TreeGridTextColumn() { MappingName = "Salary" };
    }
}
```

Changing property settings

You can change the column properties in `AutoGeneratingColumn` event handler.

C#

```
treeGrid.AutoGeneratingColumn += TreeGrid_AutoGeneratingColumn;
private void TreeGrid_AutoGeneratingColumn(object sender,
TreeGridAutoGeneratingColumnEventArgs e)
{
    if (e.Column.MappingName == "Salary")
    {
        e.Column.AllowEditing = false;
        e.Column.AllowSorting = false;
        e.Column.AllowFocus = true;
        e.Column.AllowResizing = false;
    }
}
```

```
e.Column.ColumnSizer = TreeColumnSizer.Star;  
e.Column.AllowDragging = true;  
}  
}
```

Setting template to auto-generated column

You can set [TreeGridColumn.HeaderTemplate](#) and [TreeGridColumn.CellTemplate](#) properties for auto-generated column in `AutoGeneratingColumn` event handler.

XML

```
<Window.Resources>  
<DataTemplate x:Key="headerTemplate">  
<TextBlock FontSize="10"  
Text="This is the first name of the employee"  
TextWrapping="Wrap" />  
</DataTemplate>  
</Window.Resources>
```

C#

```
treeGrid.AutoGeneratingColumn += TreeGrid_AutoGeneratingColumn;  
private void TreeGrid_AutoGeneratingColumn(object sender,  
TreeGridAutoGeneratingColumnEventArgs e)  
{  
    if (e.Column.MappingName == "FirstName")  
    {  
        e.Column.HeaderTemplate = this.Resources["headerTemplate"] as DataTemplate;  
    }  
}
```

Below screenshot shows the customized header template loaded on the header of FirstName column.

This is the first name of the employee	LastName	Availability	
⊕ Obama	bosh	<input type="checkbox"/>	
⊕ John	Adams	<input checked="" type="checkbox"/>	
⊕ Thomas	Jefferson	<input checked="" type="checkbox"/>	
⊕ Andrew	Madison	<input type="checkbox"/>	
⊕ Ulysses	Pierce	<input checked="" type="checkbox"/>	
⊕ Jimmy	Harrison	<input type="checkbox"/>	
⊕ Ronald	Fillmore	<input type="checkbox"/>	

Data Annotations with AutoGenerateColumns

SfTreeGrid support to generate the columns based on built-in [Data Annotation Attributes](#). Data Annotations ignored, when the `AutoGenerateColumns` is set to False.

Exclude column

You can skip the column generation using `AutoGenerateField` property or set the `Bindable` attribute to false.

C#

```
[Display(AutoGenerateField = false, Description = "Title field is not
generated in UI")]
public string Title
{
    get
    {
        return _title;
    }
    set
    {
        _title = value;
        RaisePropertyChanged("Title");
    }
}
```

Editing

You can change the value of the property using `Editable` attribute.

C#

```
[Editable(true)]
public string FirstName
```

```
{
    get
    {
        return _firstName;
    }
    set
    {
        _firstName = value;
        RaisePropertyChanged("FirstName");
    }
}
```

Change the HeaderText of column

You can customize header text of column using `Display.Name` property.

C#

```
[Display(Name = "FirstName of the employee", Description = "First Name is necessary for identification")]
public string FirstName
{
    get
    {
        return _firstName;
    }
    set
    {
        _firstName = value;
        RaisePropertyChanged("FirstName");
    }
}
```

Change the order of the columns

You can change the columns order using `DisplayAttribute.Order` property.

C#

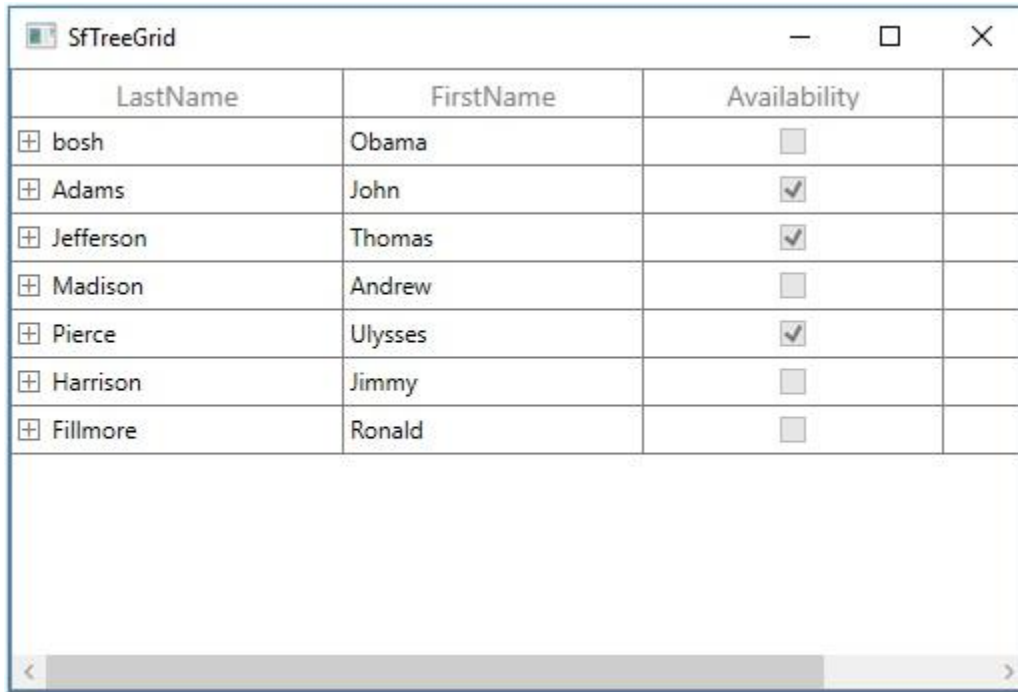
```
[Display(Order = 0)]
public string FirstName
{
    get { return _firstName; }
    set
    {
        _firstName = value;
        RaisePropertyChanged("FirstName");
    }
}

[Display(Order = -1)]
public string LastName
{
    get { return _lastName; }
    set
    {
        _lastName = value;
        RaisePropertyChanged("LastName");
    }
}
```

```
}

```

The FirstName and LastName column rearranged based on specified order.



LastName	FirstName	Availability	
+ bosh	Obama	<input type="checkbox"/>	
+ Adams	John	<input checked="" type="checkbox"/>	
+ Jefferson	Thomas	<input checked="" type="checkbox"/>	
+ Madison	Andrew	<input type="checkbox"/>	
+ Pierce	Ulysses	<input checked="" type="checkbox"/>	
+ Harrison	Jimmy	<input type="checkbox"/>	
+ Fillmore	Ronald	<input type="checkbox"/>	

Customizing data format

You can customize the data format using `DataTypeAttribute.DataType` property.

C#

```
[DataType(DataType.Currency)]
public double? Salary
{
    get
    {
        return _salary;
    }
    set
    {
        _salary = value;
        RaisePropertyChanged("Salary");
    }
}
```

Manually defining columns

SfTreeGrid control allows you to define the columns manually by adding desired column to the [SfTreeGrid.Columns](#) collection.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
    AutoGenerateColumns="False"
    ChildPropertyName="ReportsTo"
```

```

ItemsSource="{Binding EmployeeInfo}"
ParentPropertyName="ID"
SelfRelationRootValue="-1">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName" />
<syncfusion:TreeGridTextColumn HeaderText="Last Name" MappingName="LastName"
/>
<syncfusion:TreeGridTextColumn HeaderText="Employee ID" MappingName="ID" />
<syncfusion:TreeGridTextColumn MappingName="Title" />
<syncfusion:TreeGridNumericColumn MappingName="Salary" />
<syncfusion:TreeGridTextColumn MappingName="ReportsTo" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

C#

```

treeGrid.Columns.Add(new TreeGridTextColumn() { MappingName = "FirstName",
HeaderText = "First Name" });
treeGrid.Columns.Add(new TreeGridTextColumn() { MappingName = "LastName",
HeaderText = "Last Name" });
treeGrid.Columns.Add(new TreeGridTextColumn() { MappingName = "ID",
HeaderText = "Employee ID" });
treeGrid.Columns.Add(new TreeGridTextColumn() { MappingName = "Title" });
treeGrid.Columns.Add(new TreeGridNumericColumn() { MappingName = "Salary"
});
treeGrid.Columns.Add(new TreeGridTextColumn() { MappingName = "ReportsTo",
HeaderText = "Reports To" });

```

You can refer more information about handling the column level operations for manually defined columns in [Column types](#) section.

Column manipulation

You can get the columns (added or auto-generated) from [SfTreeGrid.Columns](#) property.

Adding column

You can add column at runtime by adding instance of column to `SfTreeGrid.Columns` property.

C#

```

treeGrid.Columns.Add(new TreeGridTextColumn() { MappingName = "FirstName",
HeaderText = "First Name" });

```

Accessing column

You can access the column through its column index or [TreeGridColumn.MappingName](#) from the `SfTreeGrid.Columns` collection.

C#

```

TreeGridColumn column = treeGrid.Columns[1];
//OR
TreeGridColumn column = treeGrid.Columns["FirstName"];

```

Clearing or Removing Column

You can remove all the columns by clearing the `SfTreeGrid.Columns` property.

C#

```
this.treeGrid.Columns.Clear();
```

You can remove a column using `Remove` and `RemoveAt` methods.

C#

```
treeGrid.Columns.Remove(column);  
//OR  
treeGrid.Columns.RemoveAt(1);
```

Resizing Columns

SfTreeGrid allows to resize the columns like in excel by resizing column header. This can be enabled or disabled by setting `SfTreeGrid.AllowResizingColumns` or `TreeGridColumn.AllowResizing` property.

Note: Resizing considers `MinWidth` and `MaxWidth` of column.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"  
AllowResizingColumns="True"  
AutoGenerateColumns="False"  
ChildPropertyName="ReportsTo"  
ItemsSource="{Binding EmployeeInfo}"  
ParentPropertyName="ID"  
SelfRelationRootValue="-1"/>
```

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.

Column Reizing				
First Name	Last Name	Id	Salary	
<input type="checkbox"/> Chester	Buchanan	0	\$100000.00	
<input checked="" type="checkbox"/> Abraham	Fillmore	1	\$100000.00	
<input checked="" type="checkbox"/> Rutherford	Clinton	8	\$50000.00	
<input checked="" type="checkbox"/> Chester	Polk	15	\$50000.00	
<input checked="" type="checkbox"/> Chester	Monroe	22	\$90000.00	
<input checked="" type="checkbox"/> George	Truman	29	\$50000.00	
<input checked="" type="checkbox"/> Warner	Roosevelt	36	\$40000.00	
<input type="checkbox"/> Dwight	Nixon	43	\$80000.00	
<input checked="" type="checkbox"/> Grover	Harrison	44	\$90000.00	
<input checked="" type="checkbox"/> Warren	Harrison	51	\$90000.00	

Hidden column resizing

SfTreeGrid shows indication for hidden columns in column header and also allows end-users to resize the hidden columns when setting [SfTreeGrid.AllowResizingHiddenColumns](#) property to **true**.

First Name	Last Name	DOB	Salary
Chester	Buchanan	11/12/2004	\$100000.00
Abraham	Fillmore	6/6/1998	\$100000.00
Rutherford	Clinton	1/31/1998	\$50000.00
Chester	Polk	1/16/2004	\$50000.00
Chester	Monroe	1/7/1998	\$90000.00
George	Truman	9/2/2001	\$50000.00
Warner	Roosevelt	7/25/2004	\$40000.00
Dwight	Nixon	4/20/1995	\$80000.00
Grover	Harrison	5/20/2003	\$90000.00
Warren	Harrison	9/16/1996	\$90000.00

Disable resizing

You can cancel resizing of particular column by setting [TreeGridColumn.AllowResizing](#) property to **false**. In another way, you can cancel the resizing by handling [SfTreeGrid.ResizingColumns](#) event. The **ResizingColumns** event occurs when you start dragging by resizing cursor on headers.

[ResizingColumnsEventArgs](#) of **ResizingColumns** provides information about the columns's index and width.

C#

```
treeGrid.ResizingColumns += TreeGrid_ResizingColumns;
private void TreeGrid_ResizingColumns(object sender,
ResizingColumnsEventArgs e)
{
    if (e.ColumnIndex == 1)
        e.Cancel = true;
}
```

Identify resizing of the column gets completed

SfTreeGrid allows you to identify the progress of the resizing of columns through **ResizingColumnsEventArgs.Reason** property. You can get the width of the column after resizing completed by getting **ResizingColumnsEventArgs.Width** when **ResizingColumnsEventArgs.Reason** is **ColumnResizingReason.Resized** in **ResizingColumns** event.

C#

```
this.treeGrid.ResizingColumns += OnResizingColumns;
void OnResizingColumns(object sender, ResizingColumnsEventArgs e)
{
    if (e.Reason == Syncfusion.UI.Xaml.Grid.ColumnResizingReason.Resized)
    {
        var resizedWidth = e.Width;
    }
}
```

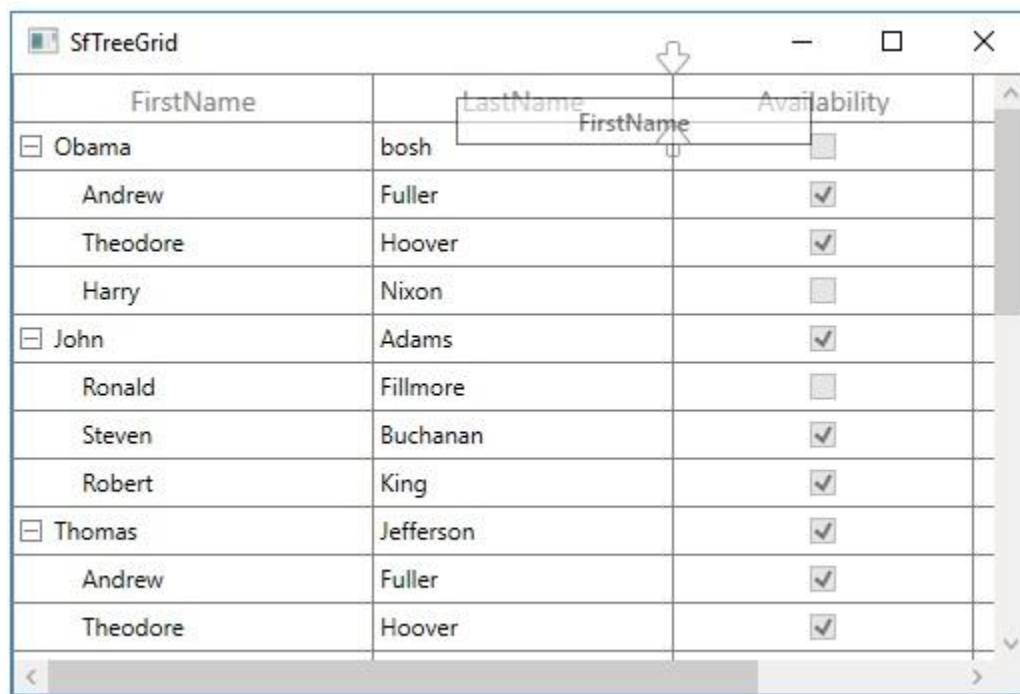
```
}
}
```

Column drag and drop

You can allow end-users to rearrange the columns by drag and drop the column headers by setting [SfTreeGrid.AllowDraggingColumns](#) to `true`.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowDraggingColumns="True"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding EmployeeInfo}"
ParentPropertyName="ID"
SelfRelationRootValue="-1"/>
```



You can enable or disable dragging on particular column using [TreeGridColumn.AllowDragging](#) property.

XML

```
<syncfusion:TreeGridTextColumn AllowDragging="True"
HeaderText="First Name"
MappingName="FirstName" />
```

Disable column reordering

You can cancel the particular column dragging by handling [SfTreeGrid.ColumnDragging](#). `ColumnDragging` event occurs when you start dragging the column header.

[TreeGridColumnDraggingEventArgs](#) of [ColumnDragging](#) event provides information about the column triggered this event.

[TreeGridColumnDraggingEventArgs.From](#) - It returns the index of column triggered this event.

[TreeGridColumnDraggingEventArgs.To](#) - It returns the index where you try to drop the column.

[TreeGridColumnDraggingEventArgs.Reason](#) - It returns column dragging details by [QueryColumnDraggingReason](#).

C#

```
treeGrid.ColumnDragging += TreeGrid_ColumnDragging;
private void TreeGrid_ColumnDragging(object sender,
TreeGridColumnDraggingEventArgs e)
{
    var column = treeGrid.Columns[e.From];
    if(column.MappingName=="FirstName" &&
e.Reason==QueryColumnDraggingReason.Dropping)
    {
        e.Cancel = true;
    }
}
```

Freezing Columns

You can freeze the columns in view at the left and right side like in excel by setting [SfTreeGrid.FrozenColumnCount](#) and [SfTreeGrid.FooterColumnCount](#) properties.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
FrozenColumnCount="2"
FooterColumnCount="2"
ItemsSource="{Binding EmployeeDetails}">
```

C#

```
this.treeGrid.FrozenColumnCount = 2;
this.treeGrid.FooterColumnCount = 2;
```


Order ID	Order Date	Customer ID	Customer Area	Price	Discount
[-] 1000	9/7/2000	DB	London	\$32000.00	10.
[+] 1000	3/7/2007	B	NewYork	\$33000.00	10.
[+] 1001	10/20/2000	NP	Rome	\$34000.00	10.
[+] 1002	1/16/2004	EP	Cardiff	\$36000.00	6.
[+] 1003	3/22/2001	LI	Canberra	\$31000.00	7.
[+] 1004	7/7/2004	DD	Durban	\$30000.00	5.
[+] 1005	1/5/2000	DD	Brisbane	\$35000.00	8.
[+] 1006	4/30/2002	DB	Liverpool	\$38000.00	7.
[-] 1001	2/14/2005	FE	Birmingham	\$30000.00	10.
[+] 1000	9/7/2006	FE	Delhi	\$32000.00	15.

Stacked Headers

SfTreeGrid supports additional unbound header rows known as **stacked header rows** that span across the TreeGrid columns using [StackedHeaderRows](#). You can group one or more columns under each stacked header.

Each [StackedHeaderRow](#) contains the [StackedColumns](#) where each [StackedColumn](#) contains a number of child columns. [StackedColumn.ChildColumns](#) property returns the columns which are grouped under the stacked header row. [StackedColumn.HeaderText](#) returns the text that displays in stacked header row.

XML

```
<syncfusion:SfTreeGrid.StackedHeaderRows>
  <syncfusion:StackedHeaderRow>
    <syncfusion:StackedHeaderRow.StackedColumns>
      <syncfusion:StackedColumn
        ChildColumns="OrderID,OrderDate,CustomerID,CustomerArea,UnitPrice,Discount"
        HeaderText="Sales Details" />
    </syncfusion:StackedHeaderRow.StackedColumns>
  </syncfusion:StackedHeaderRow>
  <syncfusion:StackedHeaderRow>
    <syncfusion:StackedHeaderRow.StackedColumns>
      <syncfusion:StackedColumn ChildColumns="OrderID,OrderDate" HeaderText="Order
        Details" />
      <syncfusion:StackedColumn ChildColumns="CustomerID,CustomerArea"
        HeaderText="Customer Details" />
      <syncfusion:StackedColumn ChildColumns="UnitPrice,Discount"
        HeaderText="Price Details" />
    </syncfusion:StackedHeaderRow.StackedColumns>
  </syncfusion:StackedHeaderRow>
</syncfusion:SfTreeGrid.StackedHeaderRows>
```

C#

```
var stackedHeaderRow = new StackedHeaderRow();
```

```

stackedHeaderRow.StackedColumns.Add(new StackedColumn() { ChildColumns =
"OrderID,OrderDate,CustomerID,CustomerArea,UnitPrice,Discount", HeaderText =
"Sales Details" });
this.treeGrid.StackedHeaderRows.Add(stackedHeaderRow);
var stackedHeaderRow1 = new StackedHeaderRow();
stackedHeaderRow1.StackedColumns.Add(new StackedColumn() { ChildColumns =
"OrderID,OrderDate", HeaderText = "Order Details" });
stackedHeaderRow1.StackedColumns.Add(new StackedColumn() { ChildColumns =
"CustomerID,CustomerArea", HeaderText = "Customer Details" });
stackedHeaderRow1.StackedColumns.Add(new StackedColumn() { ChildColumns =
"UnitPrice,Discount", HeaderText = "Price Details" });
this.treeGrid.StackedHeaderRows.Add(stackedHeaderRow1);

```

Sales Details					
Order Details		Customer Details		Price Details	
Order ID	Order Date	Customer ID	Customer Area	Price	Discount
[-] 1000	9/9/2000	SIMOB	London	\$32000.00	10.20%
[+] 1000	3/9/2007	FURIB	NewYork	\$33000.00	10.20%
[+] 1001	10/22/2000	BLONP	Rome	\$34000.00	10.20%
[+] 1002	1/18/2004	MEREP	Cardiff	\$36000.00	6.00%
[+] 1003	3/24/2001	WELLI	Canberra	\$31000.00	7.70%
[+] 1004	7/9/2004	LINOD	Durban	\$30000.00	5.50%
[+] 1005	1/7/2000	LINOD	Brisbane	\$35000.00	8.20%
[+] 1006	5/2/2002	SIMOB	Liverpool	\$38000.00	7.70%
[-] 1001	2/16/2005	VAFFE	Birmingham	\$30000.00	10.20%
[+] 1000	9/9/2006	VAFFE	Delhi	\$32000.00	15.00%
[+] 1001	7/24/1997	MEREP	Madrid	\$35000.00	15.00%

Adding ChildColumns

You can add the child columns in particular stacked header directly.

C#

```

var childColumn =
this.treeGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns;
this.treeGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns =
childColumn + ", " + "Discount";

```

Removing ChildColumns

Similarly, you can remove the child columns from particular stacked header directly.

C#

```

var removingColumns =
this.treeGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns.Split(',')
.ToList<string>();
string childColumns = string.Empty;
foreach (var stackedColumnName in removingColumns.ToList())
{

```

```

if (stackedColumnName.Equals("OrderID"))
{
    removingColumns.Remove(stackedColumnName);
}
else
{
    childColumns = childColumns + stackedColumnName + ",";
}
this.treeGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns =
childColumns;
}

```

Changing Stacked Header Row Height

You can change the height of StackedHeaderRows by using [GetTreePanel.RowHeights](#) property.

C#

```

this.treeGrid.Loaded += TreeGrid_Loaded;
private void TreeGrid_Loaded(object sender, RoutedEventArgs e)
{
    var getTreePanel = this.treeGrid.GetTreePanel();
    int count = this.treeGrid.StackedHeaderRows.Count;
    for (int i = 0; i < count; i++)
    {
        getTreePanel.RowHeights[i] = 50;
    }
    getTreePanel.InvalidateMeasure();
}

```

Binding column properties with ViewModel

SfTreeGrid provides MVVM support for binding `TreeGridColumn` properties with ViewModel properties.

C#

```

public class ViewModel
{
    private bool _allowEditing = true;
    public bool AllowEditing
    {
        get { return _allowEditing; }
        set { _allowEditing = value; }
    }
}

```

Below code, binds the `ViewModel.AllowEditing` property to `TreeGridColumn.AllowEditing` property.

XML

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<syncfusion:SfTreeGrid Name="treeGrid"
AllowEditing="False"
AutoExpandMode="AllNodesExpanded"

```

```

AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
ParentPropertyName="ID">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn AllowEditing="{Binding AllowEditing}"
MappingName="ID" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Column Types in WPF TreeGrid (SfTreeGrid)

SfTreeGrid provides support for various built-in column types. Each column has its own properties and renderer to handle different types of data.

You can also add or override existing columns and renderers as you need.

Column Type	Renderer	Description
TreeGridTextColumn	TreeGridCellTextBoxRenderer	Use to display the string data.
TreeGridNumericColumn	TreeGridCellNumericRenderer	Use to display the numeric data.
TreeGridCurrencyColumn	TreeGridCellCurrencyRenderer	Use to display the currency value.
TreeGridPercentColumn	TreeGridCellPercentageRenderer	Use to display the percent value.
TreeGridMaskColumn	TreeGridCellMaskRenderer	Use to display the data to be masked.
TreeGridDateTimeColumn	TreeGridCellDateTimeRenderer	Use to display the date time value.
TreeGridComboBoxColumn	TreeGridCellComboBoxRenderer	Use to display the IEnumerable data using ComboBox.
TreeGridCheckBoxColumn	TreeGridCellCheckBoxRenderer	Use to display the boolean type data.
TreeGridHyperlinkColumn	TreeGridCellHyperLinkRenderer	Use to display the URI data.
TreeGridTemplateColumn	TreeGridCellTemplateRenderer	Use to display the custom template-specified content.

TreeGridColumn

TreeGridColumn is an abstract class provides base functionalities for all the column types in SfTreeGrid.

Mapping column to particular property

Column can be bound to a property in data object using [TreeGridColumn.MappingName](#) property. In addition, it supports to format or bind different property for display and edit mode separately via [TreeGridColumn.DisplayBinding](#) and [TreeGridColumn.ValueBinding](#).

When you set `MappingName`, `DisplayBinding` and `ValueBinding` are created based on `MappingName`, if these properties are not defined explicitly.

You can use `DisplayBinding` property to format the column in display, by setting `StringFormat` or `Converter` property of `Binding`.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
    AllowEditing="True"
    AutoExpandMode="RootNodesExpanded"
    AutoGenerateColumns="False"
    ChildPropertyName="Children"
    ItemsSource="{Binding EmployeeDetails}">
    <syncfusion:SfTreeGrid.Columns>
    <syncfusion:TreeGridTextColumn MappingName="Salary" DisplayBinding="{Binding
    Path=Salary, StringFormat='{0:C}'}"/>
    </syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

In the below screenshot, `Salary` column display value is formatted to currency by setting `DisplayBinding` property.

First Name	Last Name	Person ID	Salary
<input type="checkbox"/> Chester	Buchanan	0	\$100,000.00
<input checked="" type="checkbox"/> Abraham	Fillmore	1	100000
<input checked="" type="checkbox"/> Rutherford	Clinton	8	\$50,000.00
<input checked="" type="checkbox"/> Chester	Polk	15	\$50,000.00
<input checked="" type="checkbox"/> Chester	Monroe	22	\$90,000.00
<input checked="" type="checkbox"/> George	Truman	29	\$50,000.00
<input checked="" type="checkbox"/> Warner	Roosevelt	36	\$40,000.00
<input type="checkbox"/> Dwight	Nixon	43	\$80,000.00
<input checked="" type="checkbox"/> Grover	Harrison	44	\$90,000.00
<input checked="" type="checkbox"/> Warren	Harrison	51	\$90,000.00
<input checked="" type="checkbox"/> Abraham	Nixon	58	\$50,000.00

CellTemplate in TreeGridColumn

You can load any WPF control in the display mode for all columns by setting [TreeGridColumn.CellTemplate](#) property. In edit mode, corresponding editor will be loaded based on column type.

In the below code snippet, `TreeGridCurrencyColumn` is loaded with `ProgressBar` and `TextBlock`. When you start editing `DoubleTextBox` will be loaded as Editor.











XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
    AllowEditing="True"
    AllowSorting="True"
```

```

AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ColumnSizer="Star"
ItemsSource="{Binding EmployeeDetails}"
LiveNodeUpdateMode="AllowDataShaping">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridColumn CurrencyColumn MappingName="Salary" >
<syncfusion:TreeGridColumn.CurrencyColumn.CellTemplate>
<DataTemplate>
<Grid>
<ProgressBar x: Name="progressBar"
Background="Transparent"
BorderThickness="0"
Maximum="200000"
Minimum="0"
Visibility="Visible"
Value="{Binding Salary}" />
<TextBlock HorizontalAlignment="Right"
VerticalAlignment="Center"
Text="{Binding Salary,
Converter={StaticResource CurrencyConverter}}"
TextAlignment="Center" />
</Grid>
</DataTemplate>
</syncfusion:TreeGridColumn.CurrencyColumn.CellTemplate>
</syncfusion:TreeGridColumn.CurrencyColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

First Name	Last Name	Person ID	Salary	Availability
<input type="checkbox"/> Chester	Buchanan	0	 \$100,000.00	<input checked="" type="checkbox"/>
<input type="checkbox"/> Abraham	Fillmore	1	 \$100,000.00	<input type="checkbox"/>
<input type="checkbox"/> Rutherford	Clinton	8	 \$50,000.00	<input checked="" type="checkbox"/>
<input type="checkbox"/> Chester	Polk	15	 \$50,000.00	<input type="checkbox"/>
<input type="checkbox"/> Chester	Monroe	22	 \$90,000.00	<input checked="" type="checkbox"/>
<input type="checkbox"/> George	Truman	29	 \$50,000.00	<input type="checkbox"/>
<input type="checkbox"/> Warner	Roosevelt	36	 \$40,000.00	<input type="checkbox"/>
<input type="checkbox"/> Dwight	Nixon	43	 \$80,000.00	<input type="checkbox"/>
<input type="checkbox"/> Grover	Harrison	44	 \$90,000.00	<input type="checkbox"/>
<input type="checkbox"/> Warren	Harrison	51	 \$90,000.00	<input checked="" type="checkbox"/>

CellTemplate is not support by [TreeGridHyperlinkColumn](#) and [TreeGridCheckBoxColumn](#) columns.

Reusing same DataTemplate for multiple columns

By default, underlying record is `DataContext` for CellTemplate. So you have to define, template for each column to display values based on `MappingName`.

You can use the same [DataTemplate](#) for all columns to display value based on MappingName by setting [TreeGridColumn.SetCellBoundValue](#) property to `true`. Setting `SetCellBoundValue` to true, changes the `DataContext` for CellTemplate to `DataContextHelper` which has the following members,

- **Value** - Return the value base on MappingName.
- **Record** - Returns the underlying data object.

XML

```
<DataTemplate x:Key="cellTemplate">
<TextBlock Foreground="Red"
Margin="3,0,0,0"
Text="{Binding Path=Value}"/>
</DataTemplate>
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ColumnSizer="Star"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName"
SetCellBoundValue="True"
CellTemplate="{StaticResource cellTemplate}"/>
<syncfusion:TreeGridTextColumn HeaderText="Last Name" MappingName="LastName"
/>
<syncfusion:TreeGridNumericColumn HeaderText="Person ID" MappingName="Id"
NumberDecimalDigits="0"
SetCellBoundValue="True"
CellTemplate="{StaticResource cellTemplate}"/>
<syncfusion:TreeGridTextColumn MappingName="Salary" DisplayBinding="{Binding
Path=Salary, StringFormat='{0:C}'}"/>
<syncfusion:TreeGridCheckBoxColumn HeaderText="Availability"
MappingName="IsAvailable" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

First Name	Last Name	Person ID	Salary
<input type="checkbox"/> Chester	Buchanan	0	\$100,000.00
<input type="checkbox"/> Abraham	Fillmore	1	\$100,000.00
<input type="checkbox"/> Rutherford	Clinton	8	\$50,000.00
<input type="checkbox"/> Chester	Polk	15	\$50,000.00
<input type="checkbox"/> Chester	Monroe	22	\$90,000.00
<input type="checkbox"/> George	Truman	29	\$50,000.00
<input type="checkbox"/> Warner	Roosevelt	36	\$40,000.00
<input type="checkbox"/> Dwight	Nixon	43	\$80,000.00
<input type="checkbox"/> Grover	Harrison	44	\$90,000.00
<input type="checkbox"/> Warren	Harrison	51	\$90,000.00

Setting CellTemplate based on custom logic using TemplateSelector

TreeGridColumn provides support to choose different [DataTemplate](#) based on underlying data object using [TreeGridColumn.CellTemplateSelector](#) property.

For example, two different templates loaded alternatively in **Salary** column.

XML

```

<Window.Resources>
<local:CustomCellTemplateSelector x:Key="cellTemplateSelector"/>
<DataTemplate x:Key="DefaultTemplate">
<TextBlock Background="Wheat"
Foreground="Red"
Text="{Binding Path=Id}"
TextAlignment="Center" />
</DataTemplate>
<DataTemplate x:Key="AlternateTemplate">
<TextBlock Background="Beige"
Foreground="Green"
Text="{Binding Path=Id}"
TextAlignment="Center" />
</DataTemplate>
</Window.Resources>

```

Below code returns the **DefaultTemplate** and **AlternateTemplate** based on Salary? value.

C#

```

public class CustomCellTemplateSelector : DataTemplateSelector
{
public override System.Windows.DataTemplate SelectTemplate(object item,
System.Windows.DependencyObject container)
{
if (item == null)
return null;
var data = item as Employee;
if (data.Id % 2 == 0)
return Application.Current.MainWindow.FindResource("AlternateTemplate") as
DataTemplate;
else
return Application.Current.MainWindow.FindResource("DefaultTemplate") as
DataTemplate;
}
}

```

In the below code, the custom template selector set to **TreeGridColumn.CellTemplateSelector**.

XML

```

<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ColumnSizer="Star"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="Person ID" MappingName="Id"
CellTemplateSelector="{StaticResource cellTemplateSelector}"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```


Note: Non-Editable columns does not support CellTemplate.

First Name	Last Name	Person ID	Salary
<input type="checkbox"/> Chester	Buchanan	0	\$100,000.00
<input checked="" type="checkbox"/> Abraham	Fillmore	1	\$100,000.00
<input checked="" type="checkbox"/> Rutherford	Clinton	8	\$50,000.00
<input checked="" type="checkbox"/> Chester	Polk	15	\$50,000.00
<input checked="" type="checkbox"/> Chester	Monroe	22	\$90,000.00
<input checked="" type="checkbox"/> George	Truman	29	\$50,000.00
<input checked="" type="checkbox"/> Warner	Roosevelt	36	\$40,000.00
<input type="checkbox"/> Dwight	Nixon	43	\$80,000.00
<input checked="" type="checkbox"/> Grover	Harrison	44	\$90,000.00
<input checked="" type="checkbox"/> Warren	Harrison	51	\$90,000.00

Binding ViewModel properties with CellTemplate

You can bind properties in ViewModel with the controls in CellTemplate.

Below command defined in ViewModel is bound to **Button** inside **CellTemplate**. Below code, denote the base command.

C#

```
public class BaseCommand : ICommand
{
    #region Fields
    readonly Action<object> _execute;
    readonly Predicate<object> _canExecute;
    public event EventHandler CanExecuteChanged;
    #endregion
    #region Constructors
    /// <summary>
    /// Creates a new command that always execute.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    public BaseCommand(Action<object> execute)
    : this(execute, null)
    {
    }

    /// <summary>
    /// Creates a new command.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    /// <param name="canExecute">The execution status logic.</param>
    public BaseCommand(Action<object> execute, Predicate<object> canExecute)
    {
        if (execute == null)
            throw new ArgumentNullException("execute");
        _execute = execute;
        _canExecute = canExecute;
    }
    #endregion
}
```

```
bool ICommand.CanExecute(object parameter)
{
    return _canExecute == null ? true : _canExecute(parameter);
}
void ICommand.Execute(object parameter)
{
    _execute(parameter);
}
```

Below code, defines the command for **Button** in **ViewModel**.

C#

```
public class ViewModel
{
    private BaseCommand deleteRecord;
    public BaseCommand DeleteRecord
    {
        get
        {
            if (deleteRecord == null)
                deleteRecord = new BaseCommand(OnDeleteRecordClicked, OnCanDelete);
            return deleteRecord;
        }
    }
    private static bool OnCanDelete(object obj)
    {
        return true;
    }
    private void OnDeleteRecordClicked(object obj)
    {
        //TODO ACTION.
    }
}
```

In the below code, Button inside CellTemplate bound to the command in ViewModel.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
    AutoExpandMode="AllNodesExpanded"
    ChildPropertyName="ReportsTo"
    ItemsSource="{Binding Employees}"
    ParentPropertyName="ID">
    <syncfusion:SfTreeGrid.Columns>
        <syncfusion:TreeGridTextColumn HeaderText="ID" MappingName="ID">
            <syncfusion:TreeGridTextColumn.CellTemplate>
                <DataTemplate>
                    <Button Command="{Binding DeleteRecord,
                        Source={StaticResource viewModel}}"
                        CommandParameter="{Binding}"
                        HorizontalAlignment="Stretch"
                        VerticalAlignment="Stretch"
                        Content="Delete" />
                </DataTemplate>
            </syncfusion:TreeGridTextColumn.CellTemplate>
        </syncfusion:TreeGridTextColumn>
    </syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

```

<Button Command="{Binding DataContext.DeleteRecord,
ElementName=treeGrid}"
CommandParameter="{Binding}"
Content="Delete" />
</DataTemplate>
</syncfusion:TreeGridTextColumn.CellTemplate>
</syncfusion:TreeGridTextColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

Data Formatting

TreeGridColumn supports to format the data using [Converter](#) properties, by defining TreeGridColumn.DisplayBinding and TreeGridColumn.ValueBinding.

TreeGridColumn.DisplayBinding formats the data in display mode. TreeGridColumn.ValueBinding formats the data in edit mode.

Format column using StringFormat

You can apply format for the column using [StringFormat](#) property by defining

DisplayBinding.StringFormat applies to TreeGridTextColumn alone. Refer the Converter section to format the other column types.

XML

```

<syncfusion:SfTreeGrid Name="treeGrid"
AllowEditing="True"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn MappingName="Salary" DisplayBinding="{Binding
Path=Salary, StringFormat='{0:C}'}"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

When column is auto-generated, you can set the 'StringFormat' by handling 'AutoGeneratingColumn' event.

C#

```

void treeGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs
e)
{
    if (e.Column.MappingName == "Salary")
    {
        if (e.Column is TreeGridNumericColumn)
        {
            e.Column = new TreeGridTextColumn() { MappingName = "Salary", HeaderText =
"Salary" };
        }
        e.Column.DisplayBinding = new Binding() { Path = new
PropertyPath(e.Column.MappingName), StringFormat = "{0:C}" };
        e.Column.ValueBinding = new Binding() { Path = new
PropertyPath(e.Column.MappingName), StringFormat="{0:C}" };
    }
}

```

```
}

```

Format column using Converter

You can format the column using **Converter** property by defining **DisplayBinding**.

XML

```
<Window.Resources>
<local:CurrencyFormatConverter x:Key="CurrencyConverter" />
</Window.Resources>
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ColumnSizer="Star"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridNumericColumn MappingName="Salary"
DisplayBinding="{Binding Path=Salary, Converter={StaticResource
CurrencyConverter}}"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

{% tabs%}

C#

```
public class CurrencyFormatConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        string language)
    {
        return string.Format("{0:C2}", value);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        string language)
    {
        return value;
    }
}
```

When column is auto-generated, you can set the **Converter** by handling [AutoGeneratingColumn](#) event

C#

```
treeGrid.AutoGeneratingColumn += TreeGrid_AutoGeneratingColumn;
private void TreeGrid_AutoGeneratingColumn(object sender,
TreeGridAutoGeneratingColumnEventArgs e)
{
    if (e.Column.MappingName == "Salary")
    {
        if (e.Column is TreeGridNumericColumn)
        {
            e.Column = new TreeGridTextColumn() { MappingName = "Salary" };
        }
    }
}
```

```
e.Column.DisplayBinding = new Binding() { Path = new
PropertyPath(e.Column.MappingName), Converter = new
CurrencyFormatConverter() };
}
}
```

	Id	FirstName	LastName	DOB	Salary
[-]	0	Chester	Buchanan	11/11/2004	\$100,000.00
[+]	1	Abraham	Fillmore	6/5/1998	\$100,000.00
[+]	8	Rutherford	Clinton	1/30/1998	\$50,000.00
[+]	15	Chester	Polk	1/15/2004	\$50,000.00
[+]	22	Chester	Monroe	1/6/1998	\$90,000.00
[+]	29	George	Truman	9/1/2001	\$50,000.00
[+]	36	Warner	Roosevelt	7/24/2004	\$40,000.00
[-]	43	Dwight	Nixon	4/19/1995	\$80,000.00
[+]	44	Grover	Harrison	5/19/2003	\$90,000.00
[+]	51	Warren	Harrison	9/15/1996	\$90,000.00

Styling TreeGridColumn

TreeGridColumn support to customize the style of particular column using [TreeGridColumn.CellStyle](#) property.

Change the font setting

You can change the font settings such as **FontSize**, **FontFamily**, **FontWeight** etc. by writing style of TargetType **TreeGridCell** or **TreeGridColumn.CellStyle** property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="AllNodesExpanded"
AutoGenerateColumns="True"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName">
<syncfusion:TreeGridTextColumn.CellStyle>
<Style TargetType="syncfusion:TreeGridCell">
<Setter Property="FontSize" Value="12" />
<Setter Property="FontFamily" Value="Segoe UI" />
<Setter Property="FontWeight" Value="Bold" />
<Setter Property="FontStyle" Value="Italic" />
<Setter Property="FontStretch" Value="Condensed" />
</Style>
</syncfusion:TreeGridTextColumn.CellStyle>
</syncfusion:TreeGridTextColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

When column is auto-generated, you can style the column by handling `AutoGeneratingColumn` event

C#

```
this.treeGrid.AutoGeneratingColumn += TreeGrid_AutoGeneratingColumn;
private void TreeGrid_AutoGeneratingColumn(object sender,
TreeGridAutoGeneratingColumnEventArgs e)
{
    if (e.Column.MappingName == "FirstName")
    {
        e.Column.CellStyle=this.Resources["cellStyle"] as Style;
    }
}
```

Last Name	First Name	Id	Salary
<input type="checkbox"/> Buchanan	<i>Chester</i>	0	\$100000.00
<input type="checkbox"/> Fillmore	<i>Abraham</i>	1	\$100000.00
<input type="checkbox"/> Clinton	<i>Rutherford</i>	8	\$50000.00
<input type="checkbox"/> Polk	<i>Chester</i>	15	\$50000.00
<input type="checkbox"/> Monroe	<i>Chester</i>	22	\$90000.00
<input type="checkbox"/> Truman	<i>George</i>	29	\$50000.00
<input type="checkbox"/> Roosevelt	<i>Warner</i>	36	\$40000.00
<input type="checkbox"/> Nixon	<i>Dwight</i>	43	\$80000.00
<input type="checkbox"/> Harrison	<i>Grover</i>	44	\$90000.00
<input type="checkbox"/> Harrison	<i>Warren</i>	51	\$90000.00

Styles based on custom logic

You can apply the styles to columns based on certain condition using [TreeGridColumn.CellStyleSelector](#) property.

Below code creates two different styles by TargetType `TreeGridCell`.

XML

```
<Application.Resources>
<Style x:Key="cellStyle1" TargetType="syncfusion:TreeGridCell">
<Setter Property="Background" Value="Bisque" />
</Style>
<Style x:Key="cellStyle2" TargetType="syncfusion:TreeGridCell">
<Setter Property="Background" Value="Aqua" />
</Style>
</Application.Resources>
```

In the below code, returns the style based on `ID` value. Using `Container` you can format the columns data based on `TreeGridCell`.

C#

```

public class CustomCellStyleSelector : StyleSelector
{
    protected override Style SelectStyleCore(object item, DependencyObject container)
    {
        var treeGridCell = container as TreeGridCell;
        var mappingName = treeGridCell.ColumnBase.TreeGridColumn.MappingName;
        var record = treeGridCell.DataContext;
        var cellValue = record.GetType().GetProperty(mappingName).GetValue(record);
        if (mappingName.Equals("ID"))
        {
            if (Convert.ToInt16(cellValue)%3==0)
            return App.Current.Resources["cellStyle1"] as Style;
            else
            return App.Current.Resources["cellStyle2"] as Style;
        }
        return base.SelectStyle(item, container);
    }
}

```

Below code, set the customized style selector to `TreeGridColumn.CellStyleSelector` property.

XML

```

<Window.Resources>
<local:CustomCellStyleSelector x:Key="cellStyleSelector">
</Window.Resources>
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="AllNodesExpanded"
AutoGenerateColumns="True"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn MappingName="ID"
CellStyleSelector="{StaticResource cellStyleSelector}" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

When column is auto-generated, you can style the column by handling `AutoGeneratingColumn` event

C#

```

this.treeGrid.AutoGeneratingColumn += TreeGrid_AutoGeneratingColumn;
private void TreeGrid_AutoGeneratingColumn(object sender,
TreeGridAutoGeneratingColumnEventArgs e)
{
    if (e.Column.MappingName == "ID")
    {
        e.Column.CellStyleSelector = new CustomCellStyleSelector();
    }
}

```

First Name	Last Name	Id	Salary
<input type="checkbox"/> Chester	Buchanan	0	\$100000.00
<input type="checkbox"/> Abraham	Fillmore	1	\$100000.00
<input type="checkbox"/> Rutherford	Clinton	8	\$50000.00
<input type="checkbox"/> Chester	Polk	15	\$50000.00
<input type="checkbox"/> Chester	Monroe	22	\$90000.00
<input type="checkbox"/> George	Truman	29	\$50000.00
<input type="checkbox"/> Warner	Roosevelt	36	\$40000.00
<input type="checkbox"/> Dwight	Nixon	43	\$80000.00
<input type="checkbox"/> Grover	Harrison	44	\$90000.00
<input type="checkbox"/> Warren	Harrison	51	\$90000.00

UI Interaction

Hide Column

You can hide or unhide the particular column programmatically by setting [TreeGridColumn.IsHidden](#) property. For allowing end-user to hide or unhide column in UI refer [Resizing Columns](#) section.

Disable column

You can disable column by setting [TreeGridColumn.AllowFocus](#) property. Therefore, that column can't be selected or edited.

Width, alignment and padding settings

Width

The width of `TreeGridColumn` can be changed by setting `Width` property. Column width set based on [TreeGridColumn.MinimumWidth](#) and [TreeGridColumn.MaximumWidth](#) properties.

Note: If the `TreeGridColumn.Width` is defined explicitly takes priority than `TreeGridColumn.ColumnSizer`.

Padding

`TreeGridColumn` allows you to change the padding of cell content by setting `Padding` property.

Alignment

`TreeGridColumn` allows you to change the alignment of `TreeGridCell` and `TreeGridHeaderCellControl` content using `TextAlignment`, `VerticalAlignment` and `HorizontalHeaderContentAlignment` properties.

TreeGridTextColumnBase

[TreeGridTextColumnBase](#) is the abstract class derived from `TreeGridColumn`. The following columns are derived from the `TreeGridTextColumnBase`.

1. `TreeGridTextColumn`
2. `TreeGridDateTimeColumn`
3. `TreeGridNumericColumn`
4. `TreeGridMaskColumn`
5. `TreeGridTemplateColumn`

TreeGridTextColumnBase properties

- Text decorations - You can [decorate](#) column? data using [TextDecorations](#) property.
- Text trimming - You can [\[trim\]\(\)](#) the column? data using [TextTrimming](#) property.
- Text wrapping - You can [wrap](#) the column? data using [TextWrapping](#) property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName"
TextDecorations="StrikeThrough"
TextTrimming="WordEllipsis"
TextWrapping="Wrap"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

Last Name	First Name	Person ID	Salary
<input type="checkbox"/> Buchanan	Chester	0.00	100000
<input checked="" type="checkbox"/> Fillmore	Abraham	1.00	100000
<input checked="" type="checkbox"/> Clinton	Rutherford	8.00	50000
<input checked="" type="checkbox"/> Polk	Chester	15.00	50000
<input checked="" type="checkbox"/> Monroe	Chester	22.00	90000
<input checked="" type="checkbox"/> Truman	George	29.00	50000
<input checked="" type="checkbox"/> Roosevelt	Warner	36.00	40000
<input type="checkbox"/> Nixon	Dwight	43.00	80000
<input checked="" type="checkbox"/> Harrison	Grover	44.00	90000
<input checked="" type="checkbox"/> Harrison	Warren	51.00	90000
<input checked="" type="checkbox"/> Nixon	Abraham	58.00	50000

TreeGridTextColumn

TreeGridTextColumn derived from TreeGridTextColumnBase which hosts TextBox in edit mode.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName"
/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

```
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
treeGrid.Columns.Add(new TreeGridTextColumn() { MappingName = "FirstName",
HeaderText = "First Name" });
```

TreeGridNumericColumn

[TreeGridNumericColumn](#) derived from [TreeGridTextColumnBase](#) which displays columns data as numeric. It hosts [SfNumericTextBox](#) in editing mode.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridNumericColumn MappingName="Salary" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
this.treeGrid.Columns.Add(new TreeGridNumericColumn() { MappingName =
"Salary" });
```

First Name	Last Name	Person ID	Salary
<input type="checkbox"/> Chester	Buchanan	0.00	100000
<input checked="" type="checkbox"/> Abraham	Fillmore	1.00	100000
<input checked="" type="checkbox"/> Rutherford	Clinton	8.00	50000
<input checked="" type="checkbox"/> Chester	Polk	15.00	50000
<input checked="" type="checkbox"/> Chester	Monroe	22.00	90000
<input checked="" type="checkbox"/> George	Truman	29.00	50000
<input checked="" type="checkbox"/> Warner	Roosevelt	36.00	40000
<input type="checkbox"/> Dwight	Nixon	43.00	80000
<input checked="" type="checkbox"/> Grover	Harrison	44.00	90000
<input checked="" type="checkbox"/> Warren	Harrison	51.00	90000

Data formatting

[TreeGridNumericColumn](#) allows you to format the numeric data with culture-specific information.

- [NumberDecimalDigits](#) - You can change the [Number of decimal digits](#) to be displayed after the decimal point using [NumberDecimalDigits](#) property.

- [NumberDecimalSeparator](#) - By default, the dot (.) operator [separates the decimal part](#) of numeric value .You can use any operator as decimal separator using `NumberDecimalSeparator` property.
- [NumberGroupSeparator](#) - By default, the comma (,) [separates group of digits](#) before the decimal point. You can use any operator as group separator using `NumberGroupSeparator` property.
- [NumberGroupSizes](#) - You can change the [number of digits in each group](#) before the decimal point on numeric values using `NumberGroupSizes` property.
- [Formatting negative pattern](#) - You can format the [pattern of negative](#) numeric values using `NumberNegativePattern`.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridNumericColumn MappingName="Salary"
NumberDecimalDigits="2"
NumberDecimalSeparator="."
NumberGroupSeparator=","
NumberGroupSizes="3"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

First Name	Last Name	Person ID	Salary
<input type="checkbox"/> Chester	Buchanan	0	100,000.00
<input checked="" type="checkbox"/> Abraham	Fillmore	1	100,000.00
<input checked="" type="checkbox"/> Rutherford	Clinton	8	50,000.00
<input checked="" type="checkbox"/> Chester	Polk	15	50,000.00
<input checked="" type="checkbox"/> Chester	Monroe	22	90,000.00
<input checked="" type="checkbox"/> George	Truman	29	50,000.00
<input checked="" type="checkbox"/> Warner	Roosevelt	36	40,000.00
<input type="checkbox"/> Dwight	Nixon	43	80,000.00
<input checked="" type="checkbox"/> Grover	Harrison	44	90,000.00
<input checked="" type="checkbox"/> Warren	Harrison	51	90,000.00
<input checked="" type="checkbox"/> Abraham	Nixon	58	50,000.00

TreeGridCurrencyColumn

`TreeGridCurrencyColumn` derived from `GridEditorColumn` and it displays columns data as currency. It hosts `CurrencyTextBox` element in editing mode.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
```

```
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridCurrencyColumn MappingName="Salary" HeaderText =
"Salary" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
this.treeGrid.Columns.Add(new TreeGridCurrencyColumn() { MappingName =
"Salary", HeaderText = "Salary" });
```

Data formatting

TreeGridCurrencyColumn allows you to format the parsing currency data with culture-specific information.

- [CurrencySymbol](#) - By default, the currency symbol will be displayed based on culture. You can change the symbol using **CurrencySymbol** property.
- [CurrencyDecimalDigits](#) - You can change the [number of decimal digits](#) to be displayed after the decimal point on currency values using **CurrencyDecimalDigits** property.
- [CurrencyDecimalSeparator](#) - By default, the dot (.) operator [separates the decimal part](#) of currency value .You can use any operator as decimal separator through **CurrencyDecimalSeparator** property.
- [CurrencyGroupSeparator](#) - By default, the comma (,) [separates the group](#) of digits before the decimal point on currency value .You can use any operator as group separator through **CurrencyGroupSeparator** property.
- [CurrencyGroupSizes](#) - You can specify [the number of digits in each group](#) before the decimal point on currency value using **CurrencyGroupSizes** property.
- **Pattern** - You can format the pattern for both [positive](#) and [negative](#) currency values through [CurrencyPositivePattern](#) and [CurrencyNegativePattern](#).

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridCurrencyColumn MappingName="Salary"
CurrencyDecimalDigits="2"
CurrencyDecimalSeparator="."
CurrencyGroupSeparator=","
CurrencyGroupSizes="3"
CurrencyPositivePattern="1"
CurrencySymbol="$" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

First Name	Last Name	Person ID	Salary
[-] Chester	Buchanan	0	100,000.00\$
[+] Abraham	Fillmore	1	100,000.00\$
[+] Rutherford	Clinton	8	50,000.00\$
[+] Chester	Polk	15	50,000.00\$
[+] Chester	Monroe	22	90,000.00\$
[+] George	Truman	29	50,000.00\$
[+] Warner	Roosevelt	36	40,000.00\$
[-] Dwight	Nixon	43	80,000.00\$
[+] Grover	Harrison	44	90,000.00\$
[+] Warren	Harrison	51	90,000.00\$

TreeGridPercentColumn

TreeGridPercentColumn derived from **GridEditorColumn** and it displays columns data as percent. It hosts **PercentTextBox** element in editing mode.

You can display data as percent value or double value using **PercentEditMode** property.

PercentEditMode.PercentMode returns the value as percentage. **PercentEditMode.DoubleMode** returns the value as numeric.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridPercentColumn MappingName="Hike" HeaderText = "Hike"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
this.treeGrid.Columns.Add(new TreeGridPercentColumn() { MappingName =
"Hike", HeaderText = "Hike" });
```

Data formatting

TreeGridPercentColumn allows you to format the parsing percent data with culture-specific information.

- **PercentSymbol** - By default, the percent operator (%) will be loaded with the value. You can change the symbol using **PercentSymbol** property.
- **PercentDecimalDigits** - You can change the **number of decimal digits** to be displayed after the decimal point on percent value can be specified using **PercentDecimalDigits** property.

- [PercentDecimalSeparator](#) - By default, the dot (.) operator [separates the decimal part](#) of percent value .You can use any operator as decimal separator using [PercentDecimalSeparator](#) property.
- [PercentGroupSeparator](#) - By default, the comma (,) operator [separates the group](#) of digits left to the decimal point on currency value .You can use any operator as group separator using [PercentGroupSeparator](#) property.
- [PercentGroupSizes](#) - You can specify [the number of digits in each group](#) before the decimal point through [PercentGroupSizes](#) property.
- Pattern - You can specify the pattern for both [positive](#) and [negative](#) percent values through [PercentPositivePattern](#) and [PercentNegativePattern](#).

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridPercentColumn MappingName="Hike"
PercentDecimalDigits="2"
PercentDecimalSeparator="."
PercentEditMode="PercentMode"
PercentGroupSeparator=","
PercentGroupSizes="2"
PercentPositivePattern="0"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

First Name	Last Name	Person ID	Salary	Hike
<input type="checkbox"/> Chester	Buchanan	0	\$100000.00	15,00.00 %
<input checked="" type="checkbox"/> Abraham	Fillmore	1	\$100000.00	6,80.00 %
<input checked="" type="checkbox"/> Rutherford	Clinton	8	\$50000.00	7,70.00 %
<input checked="" type="checkbox"/> Chester	Polk	15	\$50000.00	16,00.00 %
<input checked="" type="checkbox"/> Chester	Monroe	22	\$90000.00	11,00.00 %
<input checked="" type="checkbox"/> George	Truman	29	\$50000.00	14,00.00 %
<input checked="" type="checkbox"/> Warner	Roosevelt	36	\$40000.00	10,20.00 %
<input type="checkbox"/> Dwight	Nixon	43	\$80000.00	10,20.00 %
<input checked="" type="checkbox"/> Grover	Harrison	44	\$90000.00	10,00.00 %
<input checked="" type="checkbox"/> Warren	Harrison	51	\$90000.00	8,20.00 %

TreeGridDateTimeColumn

[TreeGridDateTimeColumn](#) derived from [TreeGridTextColumnBase](#) and it displays columns data as date time. It hosts [SfDatePicker](#) element in editing mode.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
ChildPropertyName="ReportsTo">
```

```
ItemsSource="{Binding Employees}"
ParentPropertyName="ID">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridDateTimeColumn MappingName="DOB" HeaderText="Employee
DOB"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
treeGrid.Columns.Add(new TreeGridDateTimeColumn() { MappingName = "DOB",
HeaderText = "Employee DOB" });
```

Change the pattern of date time value

You can format the date time value using [TreeGridDateTimeColumn.Pattern](#) property, which contains the set of predefined date time patterns,

Pattern	Expected
LongDate	Monday, June 15, 2016
LongTime	1:45:30 PM
ShortDate	6/15/2016
ShortTime	1:45 PM
FullDateTime	Monday, June 15, 2016 1:45 PM
MonthDay	June 15
RFC1123	Mon, 15 Jun 2016 20:45:30 GMT
SortableDateTime	2016-06-15T13:45:30
UniversalSortableDateTime	2016-06-15 13:45:30Z
YearMonth	June, 2016

When the predefined **Pattern** does not meet your requirement, you can set the custom pattern using [CustomPattern](#) property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
ColumnSizer="Star"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridDateTimeColumn MappingName="DOB" HeaderText="Date of
Birth"
CustomPattern="dd-mm-yyyy hh:mm:ss"
Pattern="CustomPattern" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

```
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

You can also change the format of standard date time pattern such as short pattern, long pattern, etc. by using [GridDateTimeColumn.DateTimeFormat](#) property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
ColumnSizer="Star"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridDateTimeColumn MappingName="DOB" HeaderText="Date of
Birth"
Pattern="LongDate">
<syncfusion:TreeGridDateTimeColumn.DateTimeFormat>
<global:DateTimeFormatInfo LongDatePattern="dd-MM-yyyy hh:mm:ss" />
</syncfusion:TreeGridDateTimeColumn.DateTimeFormat>
</syncfusion:TreeGridDateTimeColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

AutoIncrement

You can allow end-user to increment or decrement the value when 'MouseWheel' or pressing up and down arrow keys by setting [AllowScrollingOnCircle](#) property to `true`.

Null value support

[TreeGridDateTimeColumn](#) provides support to restrict or allow null value in columns based on [AllowNullValue](#) property. Instead of displaying null values, you can display hint text using [NullText](#) or or you can set the default value using [NullValue](#) property.

XML

```
<syncfusion:TreeGridDateTimeColumn HeaderText="Employee DOB"
MappingName="DOB"
AllowNullValue="True"
NullValue="17/11/2016" />
```

Setting date time value range

You can restrict and display the input value with in the range using [MinDate](#) and [MaxDate](#) properties.

Editing support

By default, the user can input the date time value by selecting through Calendar in dropdown. You allow users to input or delete the date time value from the key board by setting [TreeGridDateTimeColumn.CanEdit](#) to `true`.

Disable date selection

You can disable the date selection on dropdown popup by setting the [DisableDateSelection](#) property to `true`, so dropdown calender displays months to select.

EnableBackSpaceKey and EnableDeleteKey

You can delete using backspace and delete keys by setting [EnableBackSpaceKey](#) and [EnableDeleteKey](#) properties to `true`.

Enable clock and calendar

By default, the Calendar displayed in dropdown popup. You can enable both Calendar and Clock control in dropdown popup by setting the [EnableClassicStyle](#) to `true`.

Show repeat button

You can increment or decrement the selected part of date time value by enabling the repeat button through [TreeGridDateTimeColumn.ShowRepeatButton](#) property.

Format using Converter

[TreeGridDateTimeColumn](#) allows you to set different cultures by setting [ConverterCulture](#) property in [DisplayBinding](#) and [ValueBinding](#).

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridDateTimeColumn HeaderText="Date of Birth"
DisplayBinding="{Binding Path=DOB,
ConverterCulture=fr-FR,
Converter={StaticResource converter}}"
ValueBinding="{Binding Path=DOB,
ConverterCulture=fr-FR}" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

Note: By default, short date pattern is applied in [TreeGridDateTimeColumn](#).

Below code, returns the culture-specified date time value instead of default pattern.

C#

```
public class CultureFormatConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        return value;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

[TreeGridCheckBoxColumn](#)

[TreeGridCheckBoxColumn](#) derived from [TreeGridColumn](#) and it used display and edit `Boolean` type data. It hosts `CheckBox` element as [TreeGridCell](#) content.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridCheckBoxColumn MappingName="AvailabilityStatus"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
treeGrid.Columns.Add(new TreeGridCheckBoxColumn() { MappingName =
"AvailabilityStatus", HeaderText = "Available Status" });
```

First Name	Last Name	Person ID	Availability	Salary
<input type="checkbox"/> Chester	Buchanan	0	<input checked="" type="checkbox"/>	\$100000.00
<input type="checkbox"/> Abraham	Fillmore	1	<input type="checkbox"/>	\$100000.00
<input type="checkbox"/> Rutherford	Clinton	8	<input checked="" type="checkbox"/>	\$50000.00
<input type="checkbox"/> Chester	Polk	15	<input type="checkbox"/>	\$50000.00
<input type="checkbox"/> Chester	Monroe	22	<input checked="" type="checkbox"/>	\$90000.00
<input type="checkbox"/> George	Truman	29	<input type="checkbox"/>	\$50000.00
<input type="checkbox"/> Warner	Roosevelt	36	<input type="checkbox"/>	\$40000.00
<input type="checkbox"/> Dwight	Nixon	43	<input type="checkbox"/>	\$80000.00
<input type="checkbox"/> Grover	Harrison	44	<input type="checkbox"/>	\$90000.00
<input type="checkbox"/> Warren	Harrison	51	<input checked="" type="checkbox"/>	\$90000.00

TreeGridCheckBoxColumn allows you to customize check box state and its alignment.

- [IsThreeState](#) - By default, the **TreeGridCheckBoxColumn** has **Checked** and **Unchecked** state. You can enable another **Intermediate** state setting **IsThreeState** property to **true**.
- [HorizontalAlignment](#) - You can change the horizontal alignment of **CheckBox** using **HorizontalAlignment** property.

TreeGridTemplateColumn

[TreeGridTemplateColumn](#) derived from **TreeGridTextColumnBase** and it displays the template-specified cell content. You can load any WPF control in the display mode for all columns by setting [CellTemplate](#) and [EditTemplate](#) properties.

Using **CellTemplate**, you can format data or conditionally change the properties using [DataTrigger](#).

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID">
<syncfusion:SfTreeGrid.Columns>
```

```

<syncfusion:TreeGridTemplateColumn MappingName="FirstName" HeaderText="First
Name">
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<TextBlock Text="{Binding FirstName}"/>
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.CellTemplate>
<syncfusion:TreeGridTemplateColumn.EditTemplate>
<DataTemplate>
<TextBox Text="{Binding FirstName}"
Syncfusion:FocusManagerHelper.FocusedElement="True"/>
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.EditTemplate>
</syncfusion:TreeGridTemplateColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

C#

```

DataTemplate cellTemplate = new DataTemplate();
FrameworkElementFactory frameworkElement1 = new
FrameworkElementFactory(typeof(TextBlock));
Binding displayBinding = new Binding() { Path = new
PropertyPath("FirstName") };
frameworkElement1.SetValue(TextBlock.TextProperty, displayBinding);
cellTemplate.VisualTree = frameworkElement1;
//EditTemplate creation.
DataTemplate editTemplate = new DataTemplate();
FrameworkElementFactory frameworkElement2 = new
FrameworkElementFactory(typeof(TextBox));
Binding editBinding = new Binding() { Path = new PropertyPath("FirstName"),
Mode = BindingMode.TwoWay };
frameworkElement2.SetValue(TextBox.TextProperty, editBinding);
editTemplate.VisualTree = frameworkElement2;
this.treeGrid.Columns.Add(new TreeGridTemplateColumn() { HeaderText = "First
Name", MappingName = "FirstName", CellTemplate = cellTemplate, EditTemplate
= editTemplate, SetCellBoundValue = false });

```

Mouse interaction for UIElement loaded inside template

You can allow **UIElement** loaded inside **CellTemplate** or **EditTemplate** to handle mouse interaction in required cases by setting **VisualContainer.WantsMouseInput** attached property to the particular **UIElement** inside template.

XML

```

<syncfusion:TreeGridTemplateColumn MappingName="City">
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<ComboBox ItemsSource="{Binding CityCollection, Source={StaticResource
viewModel}}" syncfusion:VisualContainer.WantsMouseInput="True" />
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.CellTemplate>
</syncfusion:TreeGridTemplateColumn>

```

Keyboard interaction for UIElement loaded inside CellTemplate

You can allow **UIElement** loaded inside **CellTemplate** to handle keyboard interaction by setting [FocusManagerHelper.WantsKeyInput](#) attached property to **TreeGridColumn**. You can use this when loading edit element in CellTemplate.

In this case SfTreeGrid handles the below key operations and other keys are handled by UIElement loaded inside **CellTemplate**.

- Tab
- Enter
- PageUp
- PageDown

XML

```
<syncfusion:TreeGridTemplateColumn MappingName="FirstName"
syncfusion:FocusManagerHelper.WantsKeyInput="True"
HeaderText="First Name">
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<TextBlock Text="{Binding FirstName}"/>
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.CellTemplate>
</syncfusion:TreeGridTemplateColumn>
```

Setting focus to particular element inside Template when cell gets activated or edited

You can allow logical focus to specific UIElement loaded inside EditTemplate or CellTemplate by setting ``FocusManagerHelper.FocusedElement`` attached property.

You can use this property to start editing the template column value as like normal column when the user gets into edit mode.

XML

```
xmlns:sync="using:Syncfusion.UI.Xaml.Grid"
xmlns:syncfusion="using:Syncfusion.UI.Xaml.TreeGrid"
<syncfusion:SfTreeGrid Name="treeGrid"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTemplateColumn MappingName="FirstName" HeaderText="First
Name">
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<TextBlock Text="{Binding FirstName}"/>
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.CellTemplate>
<syncfusion:TreeGridTemplateColumn.EditTemplate>
<DataTemplate>
<TextBox Text="{Binding FirstName}"
sync:FocusManagerHelper.FocusedElement="True"/>
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.EditTemplate>
```

```

</syncfusion:TreeGridTemplateColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

Binding CellTemplate and EditTemplate based on MappingName

By default, underlying record is `DataContext` for `CellTemplate`. So you have to define, template for each column to display values based on `MappingName`.

You can use the same [DataTemplate](#) for all columns to display value based on `MappingName` by setting [SetCellBoundValue](#) property to `true`. Setting `SetCellBoundValue` to `true`, changes the `DataContext` for `CellTemplate` to [DataContextHelper](#) which has the following members,

- `Value` - Return the value base on `MappingName`.
- `Record` - Returns the underlying data object.

Note: `EditTemplate` support available only for `TreeGridTemplateColumn`.

XML

```

<syncfusion:SfTreeGrid Name="treeGrid"
AllowEditing="True"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID">
  <syncfusion:SfTreeGrid.Columns>
    <syncfusion:TreeGridTemplateColumn HeaderText="First Name"
MappingName="FirstName"
SetCellBoundValue="True">
      <syncfusion:TreeGridTemplateColumn.CellTemplate>
        <DataTemplate>
          <Grid>
            <TextBlock Text="{Binding FirstName}" />
          </Grid>
        </DataTemplate>
      </syncfusion:TreeGridTemplateColumn.CellTemplate>
      <syncfusion:TreeGridTemplateColumn.EditTemplate>
        <DataTemplate>
          <Grid>
            <TextBox sync:FocusManagerHelper.FocusedElement="True" Text="{Binding
FirstName}" />
          </Grid>
        </DataTemplate>
      </syncfusion:TreeGridTemplateColumn.EditTemplate>
    </syncfusion:TreeGridTemplateColumn>
  </syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

Set EditTemplate based on custom logic

`TreeGridTemplateColumn` provides support to load different edit elements based on underlying data object using [TreeGridTemplateColumn.EditTemplateSelector](#) property.

Below code returns the **DefaultTemplate** and **AlternateTemplate** based on ID? value.

XML

```
<Window.Resources>
<local:CustomCellTemplateSelector x:Key="cellTemplateSelector"/>
<local:CustomEditTemplateSelector x:Key="editTemplateSelector"/>
<DataTemplate x:Key="DefaultCellTemplate">
<TextBlock VerticalAlignment="Center"
Foreground="Red"
Text="{Binding Path=Id}"
TextAlignment="Center" />
</DataTemplate>
<DataTemplate x:Key="AlternateCellTemplate">
<TextBlock VerticalAlignment="Center"
Foreground="Green"
Text="{Binding Path=Id}"
TextAlignment="Center" />
</DataTemplate>
<DataTemplate x:Key="DefaultEditTemplate">
<TextBox Height="45"
VerticalAlignment="Center"
Foreground="Red"
Text="{Binding Path=Id}"
TextAlignment="Center" />
</DataTemplate>
<DataTemplate x:Key="AlternateEditTemplate">
<TextBox Height="45"
VerticalAlignment="Center"
Foreground="Green"
Text="{Binding Path=Id}"
TextAlignment="Center" />
</DataTemplate>
</Window.Resources>
```

C#

```
public class CustomEditTemplateSelector : DataTemplateSelector
{
    public override System.Windows.DataTemplate SelectTemplate(object item,
        System.Windows.DependencyObject container)
    {
        if (item == null)
            return null;
        var data = item as Employee;
        if (data.Id % 2 == 0)
            return App.Current.Resources["AlternateEditTemplate"] as DataTemplate;
        else
            return App.Current.Resources["DefaultEditTemplate"] as DataTemplate;
    }
}

public class CustomCellTemplateSelector : DataTemplateSelector
{
    public override System.Windows.DataTemplate SelectTemplate(object item,
        System.Windows.DependencyObject container)
    {
        if (item == null)
```

```

return null;
var data = item as Employee;
if (data.Id % 2 == 0)
return Application.Current.MainWindow.FindResource("AlternateCellTemplate")
as DataTemplate;
else
return Application.Current.MainWindow.FindResource("DefaultCellTemplate") as
DataTemplate;
}
}

```

In the below code, custom template selector set to `TreeGridTemplateColumn.EditTemplateSelector`.

XML

```

<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
ColumnSizer="Star"
AllowEditing="True"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTemplateColumn MappingName="Id"
CellTemplateSelector="{StaticResource cellTemplateSelector}"
EditTemplateSelector="{StaticResource editTemplateSelector}"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

First Name	Last Name	Id	Salary
<input type="checkbox"/> Chester	Buchanan	0	\$100000.00
<input checked="" type="checkbox"/> Abraham	Fillmore	1	\$100000.00
<input checked="" type="checkbox"/> Rutherford	Clinton	8	\$50000.00
<input checked="" type="checkbox"/> Chester	Polk	15	\$50000.00
<input checked="" type="checkbox"/> Chester	Monroe	22	\$90000.00
<input checked="" type="checkbox"/> George	Truman	29	\$50000.00
<input checked="" type="checkbox"/> Warner	Roosevelt	36	\$40000.00
<input type="checkbox"/> Dwight	Nixon	43	\$80000.00
<input checked="" type="checkbox"/> Grover	Harrison	44	\$90000.00
<input checked="" type="checkbox"/> Warren	Harrison	51	\$90000.00

TreeGridComboBoxColumn

[TreeGridComboBoxColumn](#) derived from [TreeGridColumn](#) which hosts [ComboBox](#) as edit element. The data source to [ComboBox](#) can be set by using [TreeGridComboBoxColumn.ItemsSource](#) property.

By default, [TreeGridComboBoxColumn](#) displays the value using [MappingName](#) property. You can set [DisplayMemberPath](#) which denotes the path to a value on the source object ([TreeGridComboBoxColumn.ItemsSource](#)) to serve as the visual representation of object. You can set the [SelectedValuePath](#) which denotes the path to get the [SelectedValue](#) from the [SelectedItem](#).

XML

```
<syncfusion:SfTreeGrid x:Name="treeGrid"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridComboBoxColumn ItemsSource="{Binding CityCollection,
Source={StaticResource viewModel}}"
MappingName="City" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
treeGrid.Columns.Add(new TreeGridComboBoxColumn() { MappingName = "City",
ItemsSource = viewModel.CityCollection });
```

SfTreeGrid triggers, [CurrentCellDropDownSelectionChanged](#) event, when the SelectedValue is changed. [CurrentCellDropDownSelectionChangedEventArgs](#) of [CurrentCellDropDownSelectionChanged](#) event provides the information about the changed cell value.

SelectedIndex property returns the index of selected item.

SelectedItem property returns the selected item from drop down list.

First Name	Last Name	Id	City	Salary
<input type="checkbox"/> Chester	Buchanan	0	Sydney	\$100000.00
<input type="checkbox"/> Abraham	Fillmore	1	Canberra	\$100000.00
<input type="checkbox"/> Rutherford	Clinton	8	Sydney	\$50000.00
<input type="checkbox"/> Chester	Polk	15	London	\$50000.00
<input type="checkbox"/> Chester	Monroe	22	Manchester	\$90000.00
<input type="checkbox"/> George	Truman	29	Birmingham	\$50000.00
<input type="checkbox"/> Warner	Roosevelt	36	Liverpool	\$40000.00
<input type="checkbox"/> Dwight	Nixon	43	Cardiff	\$80000.00
<input type="checkbox"/> Grover	Harrison	44	Adelaide	\$90000.00
<input type="checkbox"/> Warren	Harrison	51	Perth	\$90000.00
<input type="checkbox"/> Abraham	Nixon	58	Zurich	\$50000.00
<input type="checkbox"/> Franklin	Johnson	65	Madrid	\$50000.00
			Barcelona	\$60000.00

Keep the dropdown to be opened

You can keep the drop-down control open when start editing on the text box of **ComboBox** also by setting [StaysOpenOnEdit](#) property to true.

Improving dropdown opening time

You can improve the drop-down opening time on loading by setting [VirtualizingStackPanel](#) as [ItemsPanelTemplate](#) of **ComboBox**, when the large number of items loaded in it.

XML

```
<Window.Resources>
<Style TargetType="ComboBox">
<Setter Property="ItemsPanel">
```



```

<Setter.Value>
<ItemsPanelTemplate>
<VirtualizingStackPanel />
</ItemsPanelTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>

```

Opening dropdown popup in single-click

You can open the drop down within single click by setting [ComboBox.IsDropDownOpen](#) property to `true` in [OnEditElement](#) method by overriding existing renderer.

Below code, creates `TreeGridCellComboBoxRendererExt` to set `IsDropDownOpen` property. Replace the default renderer with created renderer in [SfTreeGrid.CellRenderers](#) collection.

C#

```

treeGrid.CellRenderers.Remove("ComboBox");
treeGrid.CellRenderers.Add("ComboBox", new
TreeGridCellComboBoxRendererExt());
public class TreeGridCellComboBoxRendererExt : TreeGridCellComboBoxRenderer
{
    protected override void OnEditElementLoaded(object sender,
    System.Windows.RoutedEventArgs e)
    {
        base.OnEditElementLoaded(sender, e);
        var combobox = sender as ComboBox;
        combobox.IsDropDownOpen = true;
    }
}

```

Note: This is applicable when `SfTreeGrid.EditTrigger` is `OnTap`

Loading Different ItemSource for each row of TreeGridComboBoxColumn

You can load the different `ItemSource` to each row of `TreeGridComboBoxColumn` by setting [SfTreeGrid.ItemsSourceSelector](#) property.

Implementing IItemsSourceSelector

`ItemsSourceSelector` needs to implement [IItemsSourceSelector](#) interface which requires you to implement [GetItemsSource](#) method which receives the below parameters,

```

<ul>
<li> <b>Record</b> – data object associated with row.</li>
<li> <b>Data Context</b> – Data context of data grid.</li>
</ul>

```

In the below code, `ItemSource` for `ShipCity` column returned based on `ShipCountry` column value using the record and data context of data grid passed to `GetItemsSource` method.

XML

```

<Window.Resources>

```

```

<local:ItemsSourceSelector x:Key="itemSourceSelector" />
</Window.Resources>
<syncfusion:SfTreeGrid x:Name="treeGrid"
AllowEditing="True"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ItemsSource="{Binding OrderDetails}"
ColumnSizer="Star">
<interactivity:Interaction.Behaviors>
<local:ItemsSourceSelectorBehavior />
</interactivity:Interaction.Behaviors>
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridComboBoxColumn MappingName="ShipCountry"
ItemsSource="{Binding Path=DataContext.CountryList, ElementName=treeGrid}"/>
<syncfusion:TreeGridComboBoxColumn HeaderText="ShipCity"
DisplayMemberPath="ShipCityName"
ItemsSourceSelector="{StaticResource itemSourceSelector}"
MappingName="ShipCityID" SelectedValuePath="ShipCityID"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

C#

```

/// <summary>
/// Implementation class for ItemsSourceSelector interface
/// </summary>
public class ItemsSourceSelector : IItemsSourceSelector
{
    public IEnumerable GetItemsSource(object record, object dataContext)
    {
        if (record == null)
            return null;
        var orderinfo = record as OrderDetails;
        var countryName = orderinfo.ShipCountry;
        var viewModel = dataContext as ViewModel;
        //Returns ShipCity collection based on ShipCountry.
        if (viewModel.ShipCities.ContainsKey(countryName))
        {
            ObservableCollection<ShipCityDetails> shipCities = null;
            viewModel.ShipCities.TryGetValue(countryName, out shipCities);
            return shipCities.ToList();
        }
        return null;
    }
}

```

The following screenshot illustrates the different ShipCity ItemsSource bound to each row of the ComboBox based on the Country Name.

OrderID	CustomerID	ProductName	NoOfOrders	ShipCountry	ShipCity
[-] 1000	FRANS	Konbu	10	Argentina	Bergamo
1001	SEVES	Alice Mutton	10	Austria	austriaAachen ▾
1002	FOLIG	Konbu	10	Belgium	austriaAachen
1003	SIMOB	Alice Mutton	25	Brazil	Cork
1004	RISCU	Raclette Courdavault	25	Canada	Århus
[-] 1005	WARTH	Konbu	15	Denmark	Montréal
1006	FRANS	Wimmers gute Semmel	15	Finland	Graz
1007	FOLKO	Alice Mutton	15	Italy	Bruxelles
1008	FURIB	Konbu	11	US	Campinas
1009	FOLKO	Wimmers gute Semmel	11	Belgium	Campinas
[-] 1010	SIMOB	Alice Mutton	11	Brazil	Aachen
1011	FOLIG	Wimmers gute Semmel	7	Denmark	Bruxelles
1012	WARTH	Wimmers gute Semmel	7	Argentina	Graz

OrderID	CustomerID	ProductName	NoOfOrders	ShipCountry	ShipCity
[-] 1000	FRANS	Konbu	10	Argentina	Bergamo
1001	SEVES	Alice Mutton	10	Austria	austriaAachen
1002	FOLIG	Konbu	10	Belgium	Bruxelles ▾
1003	SIMOB	Alice Mutton	25	Brazil	Bruxelles
1004	RISCU	Raclette Courdavault	25	Canada	Campinas
[-] 1005	WARTH	Konbu	15	Denmark	Lille
1006	FRANS	Wimmers gute Semmel	15	Finland	Bergamo
1007	FOLKO	Alice Mutton	15	Italy	Bruxelles
1008	FURIB	Konbu	11	US	Campinas
1009	FOLKO	Wimmers gute Semmel	11	Belgium	Campinas
[-] 1010	SIMOB	Alice Mutton	11	Brazil	Aachen
1011	FOLIG	Wimmers gute Semmel	7	Denmark	Bruxelles
1012	WARTH	Wimmers gute Semmel	7	Argentina	Graz

You can download the sample from [here](#).

TreeGridHyperlinkColumn

[TreeGridHyperlinkColumn](#) derived from [TreeGridTextColumn](#) and it displays columns data as hyperlink. It hosts [HyperlinkButton](#) element as [TreeGridCell](#) content.

XML

```
<syncfusion:SfTreeGrid x:Name="treeGrid"
    AutoExpandMode="RootNodesExpanded"
    AllowEditing="True"
    AutoGenerateColumns="False"
    ChildPropertyName="Children"
    ItemsSource="{Binding EmployeeDetails}">
    <syncfusion:SfTreeGrid.Columns>
```

```
<syncfusion:TreeGridHyperlinkColumn HeaderText="City"
MappingName="CityDescription" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
this.treeGrid.Columns.Add(new TreeGridHyperlinkColumn() { MappingName =
"CityDescription", HeaderText = "City" });
```

First Name	Last Name	Id	City	Salary
☐ Chester	Buchanan	0	San Francisco	\$100000.00
☐ Abraham	Fillmore	1	Durban	\$100000.00
☐ Rutherford	Clinton	8	Tokyo	\$50000.00
☐ Chester	Polk	15	NewYork	\$50000.00
☐ Chester	Monroe	22	Barcelona	\$90000.00
☐ George	Truman	29	San Francisco	\$50000.00
☐ Warner	Roosevelt	36	Delhi	\$40000.00
☐ Dwight	Nixon	43	Manchester	\$80000.00
☐ Grover	Harrison	44	Durban	\$90000.00
☐ Warren	Harrison	51	Sydney	\$90000.00

You can allow end-user to navigate the [Uri](#) when the cell value contains valid [Uri](#) address or using [CurrentCellRequestNavigate](#) event. The [CurrentCellRequestNavigate](#) occurs when the current cell in [TreeGridHyperLinkColumn](#) is clicked for navigation.

[CurrentCellRequestNavigateEventArgs](#) of [CurrentCellRequestNavigate](#) event provide information about the hyperlink triggered this event. [CurrentCellRequestNavigateEventArgs.NavigateText](#) returns the value using [ValueBinding](#) or [MappingName](#) to navigate.

C#

```
this.treeGrid.CurrentCellRequestNavigate +=
TreeGrid_CurrentCellRequestNavigate;
private async void TreeGrid_CurrentCellRequestNavigate(object sender,
Syncfusion.UI.Xaml.Grid.CurrentCellRequestNavigateEventArgs args)
{
var URI = string.Format("https://en.wikipedia.org/wiki/" +
args.NavigateText);
Windows.System.Launcher.LaunchUriAsync(new Uri(URI));
}
```

Cancel the navigation

You can cancel the navigation when clicking hyperlink by setting [CurrentCellRequestNavigateEventArgs.Handled](#) to [false](#).

C#

```
this.treeGrid.CurrentCellRequestNavigate +=  
TreeGrid_CurrentCellRequestNavigate;  
private async void TreeGrid_CurrentCellRequestNavigate(object sender,  
Syncfusion.UI.Xaml.Grid.CurrentCellRequestNavigateEventArgs args)  
{  
    args.Handled = true;  
}
```

Customize Hyperlink

Change the alignment

You can change the horizontal alignment of `TreeGridHyperlinkColumn` using [HorizontalAlignment](#) property.

Change the foreground color

You can change the foreground color of `TreeGridHyperlinkColumn` by writing the style with target type `HyperlinkButton`.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"  
    AutoGenerateColumns="False"  
    AutoExpandMode="RootNodesExpanded"  
    ChildPropertyName="Children"  
    ItemsSource="{Binding EmployeeDetails}">  
    <syncfusion:SfTreeGrid.Columns>  
        <syncfusion:TreeGridHyperlinkColumn HeaderText="City"  
            MappingName="CityDescription">  
            <syncfusion:TreeGridHyperlinkColumn.CellStyle>  
                <Style>  
                    <Style.Resources>  
                        <Style TargetType="Hyperlink">  
                            <Setter Property="Foreground" Value="Green" />  
                        </Style>  
                    </Style.Resources>  
                </Style>  
            </syncfusion:TreeGridHyperlinkColumn.CellStyle>  
        </syncfusion:TreeGridHyperlinkColumn>  
    </syncfusion:SfTreeGrid.Columns>  
</syncfusion:SfTreeGrid>
```

First Name	Last Name	Id	City	Salary
☐ Chester	Buchanan	0	San Francisco	\$100000.00
☐ Abraham	Fillmore	1	Durban	\$100000.00
☐ Rutherford	Clinton	8	Tokyo	\$50000.00
☐ Chester	Polk	15	NewYork	\$50000.00
☐ Chester	Monroe	22	Barcelona	\$90000.00
☐ George	Truman	29	San Francisco	\$50000.00
☐ Warner	Roosevelt	36	Delhi	\$40000.00
☐ Dwight	Nixon	43	Manchester	\$80000.00
☐ Grover	Harrison	44	Durban	\$90000.00
☐ Warren	Harrison	51	Sydney	\$90000.00

TreeGridMaskColumn

TreeGridMaskColumn derived from **GridTextColumnBase** and it displays columns data with specified mask pattern. It hosts **MaskedTextBox** element in editing mode.

You can set the input mask at runtime by setting [TreeGridMaskColumn.Mask](#) property.

For example,

In the below code snippet, **Mask** applied to format and validate the user input to get five digit numeric value for phone number.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridMaskColumn HeaderText="Contact Number"
MappingName="ContactNumber"
Mask="(99)-9999"
TextAlignment="Left" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
this.treeGrid.Columns.Add(new TreeGridMaskColumn() { HeaderText = "Contact
Number", MappingName = "ContactNumber", Mask = "(99)999" });
```

First Name	Last Name	Id	Contact Number	Salary
<input type="checkbox"/> Chester	Buchanan	0	(99)-9116	\$100000.00
<input type="checkbox"/> Abraham	Fillmore	1	(99)-9118	\$100000.00
<input type="checkbox"/> Rutherford	Clinton	8	(99)-9114	\$50000.00
<input type="checkbox"/> Chester	Polk	15	(99)-9111	\$50000.00
<input type="checkbox"/> Chester	Monroe	22	(99)-9111	\$90000.00
<input type="checkbox"/> George	Truman	29	(99)-9112	\$50000.00
<input type="checkbox"/> Warner	Roosevelt	36	(99)-9112	\$40000.00
<input type="checkbox"/> Dwight	Nixon	43	(99)-9118	\$80000.00
<input type="checkbox"/> Grover	Harrison	44	(99)-9114	\$90000.00
<input type="checkbox"/> Warren	Harrison	51	(99)-9111	\$90000.00

Mask for numeric value not exceeds two digits to the left of the decimal point.

In the below code snippet, **Mask** applied to format and validate the user input to enter one digit double value.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridMaskColumn HeaderText="Contact Number"
MappingName="ContactNumber"
Mask="####.00" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

First Name	Last Name	Id	Contact Number	Salary
<input type="checkbox"/> Chester	Buchanan	0	9991.16	\$100000.00
<input type="checkbox"/> Abraham	Fillmore	1	9991.18	\$100000.00
<input type="checkbox"/> Rutherford	Clinton	8	9991.14	\$50000.00
<input type="checkbox"/> Chester	Polk	15	9991.11	\$50000.00
<input type="checkbox"/> Chester	Monroe	22	9991.11	\$90000.00
<input type="checkbox"/> George	Truman	29	9991.12	\$50000.00
<input type="checkbox"/> Warner	Roosevelt	36	9991.12	\$40000.00
<input type="checkbox"/> Dwight	Nixon	43	9991.18	\$80000.00
<input type="checkbox"/> Grover	Harrison	44	9991.14	\$90000.00
<input type="checkbox"/> Warren	Harrison	51	9991.11	\$90000.00

Specifying prompt character

By default, an underscore (_) is displayed when the user input is absent. This can be changed by setting [TreeGridMaskColumn.PromptChar](#) property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridMaskColumn HeaderText="Contact Number"
MappingName="ContactNumber"
Mask="#####"
PromptChar="^" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

Setting mask format

You can decide whether the value contains the literals and prompt characters in it through [TreeGridMaskColumn.MaskFormat](#) property.

The [TreeGridMaskColumn.MaskFormat](#) property has the following format.

- ExcludePromptAndLiterals
- IncludePrompt
- IncludeLiterals
- IncludePromptAndLiterals

Custom column support

SfTreeGrid allows you to create your own column by overriding predefined column type or creating a new custom column.

Creating column from existing column

You can create your own column by overriding the [predefined](#) column types in SfTreeGrid.

For example, the [TreeGridDateTimeColumn](#) loads the [DateTime](#) value by default. If you want to display [DateTimeOffset](#) value, you can create a new column by overriding the [TreeGridDateTimeColumn](#) class.

In the below code snippet, converter created to format the [DateTimeOffset](#) value to [DateTime](#) by defining [ValueBinding](#) (edit) and [DisplayBinding](#) (non-edit).

C#

```
public class DateTimeOffsetFormatConverter : IValueConverter
{
    private TreeGridDateTimeOffsetColumn cachedColumn;
    public DateTimeOffsetFormatConverter(TreeGridDateTimeOffsetColumn column)
    {
        cachedColumn = column;
    }
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        value = ((DateTime)value);
        var column = cachedColumn as TreeGridDateTimeColumn;
        if (value == null || DBNull.Value == value)
        {
```



```

if (column.AllowNullValue && column.MaxDateTime != System.DateTime.MaxValue
&& column.NullText == string.Empty)
return column.MaxDateTime;
if (column.AllowNullValue && column.NullValue != null)
return column.NullValue;
else if (column.AllowNullValue && column.NullText != string.Empty)
return column.NullText;
if (column.MaxDateTime != System.DateTime.MaxValue)
return column.MaxDateTime;
}
DateTime _columnValue;
_columnValue = (DateTime)value;
if (_columnValue < column.MinDateTime)
_columnValue = column.MinDateTime;
if (_columnValue > column.MaxDateTime)
_columnValue = column.MaxDateTime;
return DateTimeFormatString(_columnValue, column);
}
private string DateTimeFormatString(DateTime columnValue,
TreeGridDateTimeColumn column)
{
switch (column.Pattern)
{
case DateTimePattern.ShortDate:
return columnValue.ToString("d", column.DateTimeFormat);
case DateTimePattern.LongDate:
return columnValue.ToString("D", column.DateTimeFormat);
case DateTimePattern.LongTime:
return columnValue.ToString("T", column.DateTimeFormat);
case DateTimePattern.ShortTime:
return columnValue.ToString("t", column.DateTimeFormat);
case DateTimePattern.FullDateTime:
return columnValue.ToString("F", column.DateTimeFormat);
case DateTimePattern.RFC1123:
return columnValue.ToString("R", column.DateTimeFormat);
case DateTimePattern.SortableDateTime:
return columnValue.ToString("s", column.DateTimeFormat);
case DateTimePattern.UniversalSortableDateTime:
return columnValue.ToString("u", column.DateTimeFormat);
case DateTimePattern.YearMonth:
return columnValue.ToString("Y", column.DateTimeFormat);
case DateTimePattern.MonthDay:
return columnValue.ToString("M", column.DateTimeFormat);
case DateTimePattern.CustomPattern:
return columnValue.ToString(column.CustomPattern, column.DateTimeFormat);
default:
return columnValue.ToString("MMMM", column.DateTimeFormat);
}
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
throw new NotImplementedException();
}
}
public class DateTimeOffsetToDateTimeConverter : IValueConverter
{

```

```

public object Convert(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
{
    if (value == null)
        return null;
    return value != null ? ((DateTimeOffset)value).DateTime : DateTime.Now;
}
public object ConvertBack(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
{
    if (value == null)
        return null;
    return value != null ? (new DateTimeOffset((DateTime)value)) : new
    DateTimeOffset(DateTime.Now);
}
}

```

In the below code snippet, `TreeGridDateTimeOffsetColumn` column created from `TreeGridDateTimeColumn`.

C#

```

public class TreeGridDateTimeOffsetColumn : TreeGridDateTimeColumn
{
    protected override void SetDisplayBindingConverter()
    {
        if ((DisplayBinding as Binding).Converter == null)
            (DisplayBinding as Binding).Converter = new
            DateTimeOffsetFormatConverter(this);
        if ((ValueBinding as Binding).Converter == null)
            (ValueBinding as Binding).Converter = new
            DateTimeOffsetToDateTimeConverter();
    }
}

```

In the below code snippet, created `TreeGridDateTimeOffsetColumn` added to [SfTreeGrid.Columns](#) collection and specify the full date-time pattern in `FormatString` as `F`. Since the `ShortDate` is the default pattern of `TreeGridDateTimeColumn`.

XML

```

<local:TreeGridDateTimeOffsetColumn HeaderText="Employee DOB"
    MappingName="DOB"
    Pattern="FullDateTime" />

```

C#

```

this.treeGrid.Columns.Add(new TreeGridDateTimeOffsetColumn() { MappingName =
    "DOB", Pattern = DateTimePattern.FullDateTime });

```

Customize column renderer

SfTreeGrid allows you to customize the column related operations like key navigation and UI related interactions by overriding the corresponding renderer associated with the column. Each column has its own renderer with set of virtual methods for handling the column level operations.

Below table lists the available cell types for unbound row and its renderers.

Column Name	Renderer	Cell Type
TreeGridTextColumn	TreeGridCellTextBoxRenderer	TextBox
TreeGridNumericColumn	TreeGridCellNumericRenderer	Numeric
TreeGridCurrencyColumn	TreeGridCellCurrencyRenderer	Currency
TreeGridPercentColumn	TreeGridCellPercentageRenderer	Percent
TreeGridMaskColumn	TreeGridCellMaskRenderer	Mask
TreeGridCheckBoxColumn	TreeGridCheckBoxCellRenderer	CheckBox
TreeGridTemplateColumn	TreeGridCellTemplateRenderer	Template
TreeGridComboBoxColumn	TreeGridCellComboBoxRenderer	ComboBox
TreeGridDateTimeColumn	TreeGridCellDateTimeRenderer	DateTime
TreeGridHyperlinkColumn	TreeGridCellHyperlinkRenderer	HyperlinkButton

Below code, creates the `TreeGridCellTextBoxRendererExt` to change the fore ground of `FirstName` column and replacing created renderer to

First Name	Last Name	Id	Salary
<input type="checkbox"/> Chester	Buchanan	0	\$100000.00
<input type="checkbox"/> Abraham	Fillmore	1	\$100000.00
<input type="checkbox"/> Rutherford	Clinton	8	\$50000.00
<input type="checkbox"/> Chester	Polk	15	\$50000.00
<input type="checkbox"/> Chester	Monroe	22	\$90000.00
<input type="checkbox"/> George	Truman	29	\$50000.00
<input type="checkbox"/> Warner	Roosevelt	36	\$40000.00
<input type="checkbox"/> Dwight	Nixon	43	\$80000.00
<input type="checkbox"/> Grover	Harrison	44	\$90000.00
<input type="checkbox"/> Warren	Harrison	51	\$90000.00

CellRenderers.

C#

```
this.treeGrid.CellRenderers.Remove("TextBox");
this.treeGrid.CellRenderers.Add("TextBox", new
TreeGridCellTextBoxRendererExt());
```

```

public class TreeGridCellTextBoxRendererExt : TreeGridCellTextBoxRenderer
{
    public override void OnInitializeDisplayElement(TreeDataColumnBase
dataColumn, TextBlock uiElement, object dataContext)
    {
        base.OnInitializeDisplayElement(dataColumn, uiElement, dataContext);
        if (dataColumn.TreeGridColumn.MappingName.Equals("FirstName"))
            uiElement.Foreground = new SolidColorBrush(Colors.Blue);
    }
    public override void OnUpdateDisplayBinding(TreeDataColumnBase dataColumn,
TextBlock uiElement, object dataContext)
    {
        base.OnUpdateDisplayBinding(dataColumn, uiElement, dataContext);
        if (dataColumn.TreeGridColumn.MappingName.Equals("FirstName"))
            uiElement.Foreground = new SolidColorBrush(Colors.Blue);
    }
}

```

![WPF TreeGrid Column with Custom Renderer](ColumnTypes_images/wpf-treegrid-custom-renderer.png)

Create the renderer of existing column

You can change the renderer of existing column by removing the predefined cell type value from [CellRenderers](#) collection and add the newly derived renderer from [TreeGridVirtualizingCellRenderer](#).

Below code creates the new `TreeGridComboBoxRenderer` with `ComboBoxAdv` as edit element for `TreeGridComboBoxColumn` and replacing created renderer to `CellRenderers`.

C#

```

this.treeGrid.CellRenderers.Remove("ComboBox");
this.treeGrid.CellRenderers.Add("ComboBox", new
TreeGridComboBoxRendererExt());
public class TreeGridComboBoxRendererExt :
TreeGridVirtualizingCellRenderer<ContentControl, ComboBoxAdv>
{
    public TreeGridComboBoxRendererExt()
    {
    }
    /// <summary>
    /// Create new display element.
    /// </summary>
    /// <returns></returns>
    protected override ContentControl OnCreateDisplayUIElement()
    {
        return new ContentControl();
    }
    /// <summary>
    /// Create new edit element.
    /// </summary>
    /// <returns></returns>
    protected override ComboBoxAdv OnCreateEditUIElement()
    {
        return new ComboBoxAdv();
    }
    /// <summary>

```

```

/// Initialize binding for display element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnInitializeDisplayElement(TreeDataColumnBase
dataColumn, ContentControl uiElement, object dataContext)
{
    SetDisplayBinding(uiElement, dataColumn.TreeGridColumn, dataContext);
}
/// <summary>
/// custom binding for display element.
/// </summary>
/// <param name="element"></param>
/// <param name="column"></param>
/// <param name="dataContext"></param>
private static void SetDisplayBinding(ContentControl element, TreeGridColumn
column, object dataContext)
{
    var comboBoxColumn = (TreeGridColumnComboBox)column;
    var binding = new Binding
    {
        Path = new PropertyPath(comboBoxColumn.MappingName),
        Mode = BindingMode.TwoWay,
        UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
        Converter = (comboBoxColumn.DisplayBinding as Binding).Converter,
    };
    element.SetBinding(ContentControl.ContentProperty, binding);
}
/// <summary>
/// Update binding for display element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnUpdateDisplayBinding(TreeDataColumnBase dataColumn,
ContentControl uiElement, object dataContext)
{
    SetDisplayBinding(uiElement, dataColumn.TreeGridColumn, dataContext);
}
/// <summary>
/// Initialize binding for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnInitializeEditElement(TreeDataColumnBase dataColumn,
ComboBoxAdv uiElement, object dataContext)
{
    SetEditBinding(uiElement, dataColumn.TreeGridColumn, dataContext);
}
/// <summary>
/// Update binding for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>

```

```

public override void OnUpdateEditBinding(TreeDataColumnBase dataColumn,
ComboBoxAdv element, object dataContext)
{
    SetEditBinding(element, dataColumn.TreeGridColumn, dataContext);
}
/// <summary>
/// custom binding for display element.
/// </summary>
/// <param name="element"></param>
/// <param name="column"></param>
/// <param name="dataContext"></param>
private static void SetEditBinding(ComboBoxAdv element, TreeGridColumn
column, object dataContext)
{
    var comboboxColumn = (TreeGridComboBoxColumn)column;
    var binding = new Binding
    {
        Source = dataContext,
        Path = new PropertyPath(comboboxColumn.MappingName),
        Mode = BindingMode.TwoWay,
        UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
    };
    element.SetBinding(ComboBoxAdv.SelectedValueProperty, binding);
    var itemsSourceBinding = new Binding { Path = new
PropertyPath("ItemsSource"), Mode = BindingMode.TwoWay, Source =
comboboxColumn };
    element.SetBinding(ComboBoxAdv.ItemsSourceProperty, itemsSourceBinding);
    var displayMemberBinding = new Binding { Path = new
PropertyPath("DisplayMemberPath"), Mode = BindingMode.TwoWay, Source =
comboboxColumn };
    element.SetBinding(ComboBoxAdv.DisplayMemberPathProperty,
displayMemberBinding);
    var selectedValuePathBinding = new Binding { Path = new
PropertyPath("SelectedValuePath"), Mode = BindingMode.TwoWay, Source =
comboboxColumn };
    element.SetBinding(ComboBoxAdv.SelectedValuePathProperty,
selectedValuePathBinding);
    var itemTemplateBinding = new Binding { Path = new
PropertyPath("ItemTemplate"), Mode = BindingMode.TwoWay, Source =
comboboxColumn };
    element.SetBinding(ComboBoxAdv.ItemTemplateProperty, itemTemplateBinding);
}
/// <summary>
/// Let Renderer decide whether the parent grid should be allowed to handle
keys and prevent
/// the key event from being handled by the visual UIElement for this
renderer.
/// </summary>
/// <param name="e">A <see cref="KeyRoutedEventArgs" /> object.</param>
/// <returns>
/// True if the parent grid should be allowed to handle keys; false
otherwise.
/// </returns>
protected override bool
ShouldGridTryToHandleKeyDown(System.Windows.Input.KeyEventArgs e)
{
    if (!HasCurrentCellState || !IsInEditing)

```

```
return true;
switch (e.Key)
{
case Key.End:
case Key.Home:
case Key.Enter:
case Key.Escape:
return !((ComboBoxAdv)CurrentCellRenderElement).IsDropDownOpen;
case Key.Down:
case Key.Up:
case Key.Left:
case Key.Right:
return !((ComboBoxAdv)CurrentCellRenderElement).IsDropDownOpen;
}
return base.ShouldGridTryToHandleKeyDown(e);
}
/// <summary>
/// Gets the control value.
/// </summary>
public override object GetControlValue()
{
if (!HasCurrentCellState)
return base.GetControlValue();
return CurrentCellRenderElement.GetValue(IsInEditing ?
ComboBoxAdv.SelectedValueProperty : ContentControl.ContentProperty);
}
/// <summary>
/// Set the control value.
/// </summary>
/// <param name="value">The value.</param>
public override void SetControlValue(object value)
{
if (!HasCurrentCellState)
return;
if (IsInEditing)
((ComboBoxAdv)CurrentCellRenderElement).SelectedValue = value;
else
throw new Exception("Value cannot be Set for Unloaded Editor");
}
}
```

First Name	Last Name	Id	City	Salary
[-] Chester	Buchanan	0	Rome	\$100000.00
[+] Abraham	Fillmore	1	Rome	\$100000.00
[+] Rutherford	Clinton	8	Durban	\$50000.00
[+] Chester	Polk	15	Canberra	\$50000.00
[+] Chester	Monroe	22	Sydney	\$90000.00
[+] George	Truman	29	London	\$50000.00
[+] Warner	Roosevelt	36	Manchester	\$40000.00
[-] Dwight	Nixon	43	Birmingham	\$80000.00
[+] Grover	Harrison	44	Liverpool	\$90000.00
[+] Warren	Harrison	51	Cardiff	\$90000.00
[+] Abraham	Nixon	58	Adelaide	\$50000.00
[+] Franklin	Johnson	65	Perth	\$60000.00
			Zurich	
			Madrid	
			Canberra	

Creating new column and renderer

You can create a new column by deriving [TreeGridColumn](#), rendered in UI using customized [CellType](#) using [GridVirtualizingCellRenderer](#).

Below steps to create custom column in SfDataGrid.

- Creating custom column
- Creating renderer.
- Adding the custom renderer to SfDataGrid.CellRenderers collection.
- Defining custom column.

Creating custom column

You can create custom column by overriding a new class from `TreeGridColumn` class.

Below code creates the converter to format the date time value.

C#

```
public class CustomConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        if (string.IsNullOrEmpty(value.ToString()))
            return null;
        return new ConvertToDateTimeClass().ConvertToDateTime(value);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        return new ConvertToDateTimeClass().ConvertToDateTime(value);
    }
}
public class ConvertToDateTimeClass
{

```



```
public string ConvertToDateTime(object value)
{
    DateTime date = Convert.ToDateTime(value);
    return date.Year + "/" + date.Month + "/" + date.Day;
}
}
```

In the below code, new column created with converter using `SetDisplayBindingConverter` method.

C#

```
public class DatePickerColumn : TreeGridColumn
{
    public DatePickerColumn()
    {
        SetCellType("DatePickerRenderer");
    }
    protected override void SetDisplayBindingConverter()
    {
        (this.DisplayBinding as Binding).Converter = new CustomConverter();
    }
    protected override Freezable CreateInstanceCore()
    {
        return new DatePickerColumn();
    }
}
```

Creating renderer

After creating custom column, you need to create renderer for the custom column. Below are the steps to create custom renderer. You can create custom renderer either by deriving `TreeGridVirtualizingCellRenderer` class or overriding existing renderers.

In the below code snippet, display and edit `UIElement` defined via `TreeGridVirtualizingCellRenderer` parameter.

C#

```
/// <summary>
/// CustomRenderer Creation
/// </summary>
/// <param name="TextBlock">Display Control</param>
/// <param name="DatePicker">Edit Control</param>
public class DatePickerRenderer :
    TreeGridVirtualizingCellRenderer<TextBlock, DatePicker>
{
    public DatePickerRenderer()
    {
    }
}
```

With the below code snippet, you can create the display and edit element for renderer by overriding [OnCreateDisplayUIElement](#) and [OnCreateEditUIElement](#) methods.

C#

```

/// <summary>
/// Create new display element.
/// </summary>
/// <returns></returns>
protected override TextBlock OnCreateDisplayUIElement()
{
    return new TextBlock();
}
/// <summary>
/// Create new edit element.
/// </summary>
/// <returns></returns>
protected override DatePicker OnCreateEditUIElement()
{
    return new DatePicker();
}

```

With the below code snippet, you can initialize the binding for display element by overriding the [OnInitializeDisplayElement](#) method.

C#

```

/// <summary>
/// Initialize binding for display element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnInitializeDisplayElement(TreeDataColumnBase
dataColumn, TextBlock uiElement, object dataContext)
{
    base.OnInitializeDisplayElement(dataColumn, uiElement, dataContext);
    SetDisplayBinding(uiElement, dataColumn.TreeGridColumn, dataContext);
}
/// <summary>
/// custom binding for display element.
/// </summary>
/// <param name="element"></param>
/// <param name="column"></param>
/// <param name="dataContext"></param>
private static void SetDisplayBinding(TextBlock element, TreeGridColumn
column, object dataContext)
{
    var customColumn = (DatePickerColumn)column;
    var binding = new Binding
    {
        Path = new PropertyPath(customColumn.MappingName),
        Mode = BindingMode.TwoWay,
        UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
        Converter = (column.DisplayBinding as Binding).Converter,
    };
    element.SetBinding(TextBlock.TextProperty, binding);
}

```

With the below code snippet, updates the binding while UI interaction by overriding [OnUpdateDisplayBinding](#) method.

C#

```

/// <summary>
/// Update binding for display element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnUpdateDisplayBinding(TreeDataColumnBase dataColumn,
TextBlock uiElement, object dataContext)
{
    base.OnUpdateDisplayBinding(dataColumn, uiElement, dataContext);
    SetDisplayBinding(uiElement, dataColumn.TreeGridColumn, dataContext);
}

```

Similarly, you can initialize and update the binding for edit element by overriding [OnInitializeEditElement](#) and [OnUpdateEditBinding](#) methods.

C#

```

/// <summary>
/// Initialize binding for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnInitializeEditElement(TreeDataColumnBase dataColumn,
DatePicker uiElement, object dataContext)
{
    base.OnInitializeEditElement(dataColumn, uiElement, dataContext);
    SetEditBinding(uiElement, dataColumn.TreeGridColumn, dataContext);
}

/// <summary>
/// Update binding for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnUpdateEditBinding(TreeDataColumnBase dataColumn,
DatePicker element, object dataContext)
{
    base.OnUpdateEditBinding(dataColumn, element, dataContext);
    SetEditBinding(element, dataColumn.TreeGridColumn, dataContext);
}

/// <summary>
/// custom binding for display element.
/// </summary>
/// <param name="element"></param>
/// <param name="column"></param>
/// <param name="dataContext"></param>
private static void SetEditBinding(DatePicker element, TreeGridColumn
column, object dataContext)
{
    var customColumn = (DatePickerColumn)column;
}

```

```

var binding = new Binding
{
    Source = dataContext,
    Path = new PropertyPath(customColumn.MappingName),
    Mode = BindingMode.TwoWay,
    UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
};
element.SetBinding(DatePicker.TextProperty, binding);
}

```

You can customize the editor control while loading by overriding [OnEditElementLoaded](#) method.

C#

```

/// <summary>
/// Handling operations on edit mode UIElement.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected override void OnEditElementLoaded(object sender, RoutedEventArgs e)
{
    var datePicker = (sender as DatePicker);
    datePicker.Focus();
    DatePickerTextBox datePickerTextBox =
        (DatePickerTextBox)GridUtil.FindDescendantChildByType(datePicker,
            typeof(DatePickerTextBox));
    if (this.TreeGrid.EditorSelectionBehavior ==
        Syncfusion.UI.Xaml.Grid.EditorSelectionBehavior.SelectAll)
    {
        datePickerTextBox.SelectAll();
    }
    else
    {
        datePickerTextBox.Select(datePickerTextBox.SelectedText.Length, 0);
    }
    PreviewInputText = null;
}

```

With the below code snippet, you can customize the keyboard interactions for the custom column by overriding [ShouldGridTryToHandleKeyDown](#) method.

C#

```

/// <summary>
/// Let Renderer decide whether the parent grid should be allowed to handle
/// keys and prevent
/// the key event from being handled by the visual UIElement for this
/// renderer.
/// </summary>
/// <param name="e">A <see cref="KeyEventArgs" /> object.</param>
/// <returns>
/// True if the parent grid should be allowed to handle keys; false
/// otherwise.
/// </returns>

```

```
protected override bool
ShouldGridTryToHandleKeyDown(System.Windows.Input.KeyEventArgs e)
{
    if (!HasCurrentCellState || !IsInEditing)
        return true;
    DatePickerTextBox datePickerTextBox =
        (DatePickerTextBox)
        GridUtil.FindDescendantChildByType(CurrentCellRendererElement as DatePicker,
        typeof(DatePickerTextBox));
    switch (e.Key)
    {
        case Key.End:
        case Key.Home:
        case Key.Enter:
        case Key.Escape:
            return !((DatePicker)CurrentCellRendererElement).IsDropDownOpen;
        case Key.Down:
        case Key.Up:
        case Key.Left:
        case Key.Right:
            return !((DatePicker)CurrentCellRendererElement).IsDropDownOpen;
    }
    return base.ShouldGridTryToHandleKeyDown(e);
}
```

You can handle the cell value for the custom renderer by overriding [GetControlValue](#) and [SetControlValue](#) methods.

C#

```
/// <summary>
/// Gets the control value.
/// </summary>
public override object GetControlValue()
{
    if (!HasCurrentCellState)
        return base.GetControlValue();
    return CurrentCellRendererElement.GetValue(IsInEditing ?
    DatePicker.TextProperty : TextBlock.TextProperty);
}
/// <summary>
/// Set the control value.
/// </summary>
/// <param name="value">The value.</param>
public override void SetControlValue(object value)
{
    if (!HasCurrentCellState)
        return;
    if (IsInEditing)
        ((TextBox)CurrentCellRendererElement).Text = value.ToString();
    else
        throw new Exception("Value cannot be Set for Unloaded Editor");
}
```

Adding the custom renderer to SfTreeGrid.CellRenderers collection

By below code, you can add the previous created custom renderer to [SfTreeGrid.CellRenderers](#) collection.

C#

```
this.treeGrid.CellRenderers.Add("DatePickerRenderer", new
DatePickerRenderer());
```

Loading custom column

By below code, you can define the custom column in SfTreeGrid.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowEditing="True"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<local:DatePickerColumn MappingName="DOB"/>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
this.treeGrid.Columns.Add(new DatePickerColumn() { AllowEditing = true,
MappingName = "DOB" });
```

First Name	Last Name	Id	DOB	Salary
<input type="checkbox"/> Chester	Buchanan	0	2004/11/12	\$100000.00
<input checked="" type="checkbox"/> Abraham	Fillmore	1	1998/6/6	\$100000.00
<input checked="" type="checkbox"/> Rutherford	Clinton	8	1/7/1998	\$50000.00
<input checked="" type="checkbox"/> Chester	Polk			\$50000.00
<input checked="" type="checkbox"/> Chester	Monroe			\$90000.00
<input checked="" type="checkbox"/> George	Truman			\$50000.00
<input checked="" type="checkbox"/> Warner	Roosevelt			\$40000.00
<input type="checkbox"/> Dwight	Nixon			\$80000.00
<input checked="" type="checkbox"/> Grover	Harrison			\$90000.00
<input checked="" type="checkbox"/> Warren	Harrison			\$90000.00

How To*Restrict the input content length*

You can restrict the range of input using **MaxLength** property on **TreeGridColumn** in below ways.

- Using Converter property in [DisplayBinding](#) and [ValueBinding](#)
- Using control style
- Overriding existing cell types

Using Converter

You can restrict the length of user input in both display and edit element using **Converter** using **DisplayBinding** and **ValueBinding**.

XML

```
<Window.Resources>
<local:MaxLengthConverter x:Key="maxLengthConverter"/>
</Window.Resources>
<syncfusion:SfTreeGrid x:Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AllowEditing="True"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName"
DisplayBinding="{Binding FirstName,Converter={StaticResource
maxLengthConverter}}"
ValueBinding="{Binding FirstName,Converter={StaticResource
maxLengthConverter}}" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
public class MaxLengthConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
string language)
    {
        // Define max length for column.
        int maxLength = 5;
        // Get the ColumnValue
        var columnValue = System.Convert.ToString(value);
        if (columnValue.Length < maxLength)
            return columnValue;
        else
            return columnValue.Substring(0, maxLength);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
string language)
    {
        return value;
    }
}
```

Using control style

You can set the **MaxLength** property in edit mode by writing style of **TargetType** edit element of the corresponding column.

Note: **TextBlock** does not have the **MaxLength** property. Therefore, you can use the converter to format in display.

XML

```
<Window.Resources>
<Style TargetType="TextBox">
<Setter Property="MaxLength" Value="7" />
</Style>
</Window.Resources>
<syncfusion:SfTreeGrid x:Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AllowEditing="True"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

Overriding existing cell types

You can set the `MaxLength` property to the edit element of the particular column by overriding existing cell types.

Below code, overrides the `OnInitializeEditElement` method of the corresponding renderer and set the `MaxLength` to the `UIElement` and add the renderer to `SfTreeGrid.CellRenderers` collection.

C#

```
this.treeGrid.CellRenderers.Remove("TextBox");
this.treeGrid.CellRenderers.Add("TextBox", new
TreeGridCellTextBoxRendererExt());
public class TreeGridCellTextBoxRendererExt : TreeGridCellTextBoxRenderer
{
public override void OnInitializeEditElement(TreeDataColumnBase dataColumn,
TextBox uiElement, object dataContext)
{
if (dataColumn.TreeGridColumn != null &&
dataColumn.TreeGridColumn.MappingName == "FirstName")
{
uiElement.MaxLength = 7;
}
else
{
uiElement.MaxLength = 0;
}
base.OnInitializeEditElement(dataColumn, uiElement, dataContext);
}
}
```

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Column Sizing in WPF TreeGrid (SfTreeGrid)

SfTreeGrid allows you to set the column widths based on certain logic using [SfTreeGrid.ColumnSizer](#) or [TreeGridColumn.ColumnSizer](#) property.

Below is the list of predefined column sizing options available.

Type	Column width
Star	Divides the total width equally for columns.
Auto	Calculates the width of column based on header and cell contents. So that header and cell content's are not truncated.
FillColumn	While setting the <code>TreeGridColumn.ColumnSizer</code> property, all column widths are calculated based on content of cell and last column fills the remaining space of grid. And possible to set any column to fill the remaining space instead of last column by setting <code>TreeGridColumn.ColumnSizer</code> as <code>FillColumn</code> for that particular column.
AutoFillColumn	While setting the <code>TreeGridColumn.ColumnSizer</code> property, all column widths are calculated based on content of cell and the last column fills the remaining column width as auto fill. And possible to set any column to fill the remaining space instead of last column by setting <code>TreeGridColumn.ColumnSizer</code> as <code>AutoFillColumn</code> for that particular column.
SizeToCells	Calculates the width of column based on cell contents. So that cell content's are not truncated.
SizeToHeader	Calculates the width of column based on header content. So that header content is not truncated.
None	Default column width or defined width set to column.

Note: ColumnSizer will not work when the column width defined explicitly. ColumnSizer calculates column width based on `MinWidth` and `MaxWidth` properties.

Below code, applies `GridLengthUnitType.Star` to equally set width for `SfTreeGrid.Columns`.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="True"
ChildPropertyName="Children"
ColumnSizer="Star"
ExpanderColumn="FirstName"
ItemsSource="{Binding PersonDetails}">
```

First Name	Last Name	Person ID	Salary
<input type="checkbox"/> Chester	Buchanan	0	\$100,000.00
<input checked="" type="checkbox"/> Abraham	Fillmore	1	100000
<input checked="" type="checkbox"/> Rutherford	Clinton	8	\$50,000.00
<input checked="" type="checkbox"/> Chester	Polk	15	\$50,000.00
<input checked="" type="checkbox"/> Chester	Monroe	22	\$90,000.00
<input checked="" type="checkbox"/> George	Truman	29	\$50,000.00
<input checked="" type="checkbox"/> Warner	Roosevelt	36	\$40,000.00
<input type="checkbox"/> Dwight	Nixon	43	\$80,000.00
<input checked="" type="checkbox"/> Grover	Harrison	44	\$90,000.00
<input checked="" type="checkbox"/> Warren	Harrison	51	\$90,000.00
<input checked="" type="checkbox"/> Abraham	Nixon	58	\$50,000.00

Note: The `TreeGridColumn.ColumnSizer` takes higher priority than the `SfTreeGrid.ColumnSizer`.

Refreshing ColumnSizer at runtime

You can refresh the `ColumnSizer` at runtime by calling `SfTreeGrid.TreeGridColumnSizer.Refresh` method.

SfTreeGrid support to recalculates the column auto width by calling reset methods of `TreeGridColumnSizer`. `TreeGridColumnSizer.ResetAutoCalculationforAllColumns` method reset widths to all columns. `TreeGridColumnSizer.ResetAutoCalculation` method reset the width to particular column.

Note: The `TreeGridColumnSizer.ResetAutoCalculationforAllColumns` or `TreeGridColumnSizer.ResetAutoCalculation` methods applicable for Auto, FillColumn, AutoFillColumn, SizeToCells types.

For example, you can refresh all the column's width based on the cell contents of newly added records at runtime.

C#

```
var viewModel = this.treeGrid.DataContext as ViewModel;
viewModel.PersonDetails.Add(new PersonInfo("Smith", "Anders", "Red", new
DateTime(2008, 10, 26), null));
this.treeGrid.TreeGridColumnSizer.ResetAutoCalculationforAllColumns();
this.treeGrid.TreeGridColumnSizer.Refresh();
```

Resetting column width to apply ColumnSizer

When the width of the column is explicitly defined or column is resized, then column width is not changed based on `TreeGridColumnSizer`. You can reset `TreeGridColumn.Width` by setting `double.NaN` to apply column width based on column sizer.

C#

```
foreach (var column in treeGrid.Columns)
{
    if (!double.IsNaN(column.Width))
        column.Width = double.NaN;
```

```
}
this.treeGrid.TreeGridColumnSizer.Refresh();
```

Customizing built-in column sizing logic

SfTreeGrid process column sizing operations in [TreeGridColumnSizer](#) class. You can customize the column sizing operations by overriding [GridColumnSizer](#) and set it to [SfTreeGrid.TreeGridColumnSizer](#).

C#

```
this.treeGrid.TreeGridColumnSizer = new TreeGridColumnSizerExt(treeGrid);
public class TreeGridColumnSizerExt:GridColumnSizer
{
    public TreeGridColumnSizerExt(SfTreeGrid treeGrid)
    :base(treeGrid)
    {
    }
    // Calculate Width for column when ColumnSizer is SizeToCells.
    protected override double CalculateCellWidth(TreeGridColumn column)
    {
        return base.CalculateCellWidth(column);
    }
    //Calculate Width for the column when ColumnSizer is SizeToHeader
    protected override double CalculateHeaderWidth(TreeGridColumn column)
    {
        return base.CalculateHeaderWidth(column);
    }
}
```

Auto width calculation based on font settings

By default, the ColumnSizer calculates column's width based on fixed [FontSize](#), [FontFamily](#), [Margin](#), [SortIconWidth](#). You can change the calculation by customized settings.

Changing SortIcon width

You can change the filter icon and sort icon widths for column width calculation by setting [TreeGridColumnSizer.SortIconWidth](#) properties.

C#

```
this.treeGrid.TreeGridColumnSizer.SortIconWidth = 20;
```

Changing Font settings for SfTreeGrid

You can change the font settings for column width calculation by setting [TreeGridColumnSizer.FontSize](#), [TreeGridColumnSizer.FontFamily](#) and [TreeGridColumnSizer.Margin](#) properties. This settings will be considered for all columns.

C#

```
this.treeGrid.TreeGridColumnSizer.FontSize = 10.0;
this.treeGrid.TreeGridColumnSizer.FontFamily = new
FontFamily("TimesNewRoman");
this.treeGrid.TreeGridColumnSizer.Margin = new Thickness(9, 3, 1, 3);
```

Star column sizer ratio support

You can customize the `ColumnSizer.Star` width calculation logic by overriding [SetStarWidth](#) method of [TreeGridColumnSizer](#).

For example, you can calculate the column width, with specified ratios instead of dividing equal width for all columns in Star calculation using `ColumnRatio` attached property.

C#

```
public static class StarRatio
{
    public static int GetColumnRatio(DependencyObject obj)
    {
        return (int)obj.GetValue(ColumnRatioProperty);
    }
    public static void SetColumnRatio(DependencyObject obj, int value)
    {
        obj.SetValue(ColumnRatioProperty, value);
    }
    public static readonly DependencyProperty ColumnRatioProperty =
        DependencyProperty.RegisterAttached("ColumnRatio", typeof(int),
        typeof(StarRatio), new PropertyMetadata(1, null));
}
```

Below code to define the star width calculation based on the `ColumnRatio`.

C#

```
//Assign the customized TreeGridColumnSizerExt to
SfTreeGrid.TreeGridColumnSizer
this.treeGrid.TreeGridColumnSizer = new TreeGridColumnSizerExt(treeGrid);
public class TreeGridColumnSizerExt : TreeGridColumnSizer
{
    public TreeGridColumnSizerExt(SfTreeGrid treeGrid) : base(treeGrid)
    {
    }
    protected override void SetStarWidth(double remainingColumnWidth,
    IEnumerable<TreeGridColumn> remainingColumns)
    {
        var removedColumn = new List<TreeGridColumn>();
        var columns = remainingColumns.ToList();
        var totalRemainingStarValue = remainingColumnWidth;
        double removedWidth = 0;
        bool isRemoved;
        while (columns.Count > 0)
        {
            isRemoved = false;
            removedWidth = 0;
            var columnsCount = 0;
            columns.ForEach((col) =>
            {
                columnsCount += StarRatio.GetColumnRatio(col);
            });
            double starWidth = Math.Floor((totalRemainingStarValue / columnsCount));
            var column = columns.First();
            starWidth *= StarRatio.GetColumnRatio(column);
```

```
double computedWidth = SetColumnWidth(column, starWidth);
if (starWidth != computedWidth && starWidth > 0)
{
    isRemoved = true;
    columns.Remove(column);
    foreach (var remColumn in removedColumn)
    {
        if (!columns.Contains(remColumn))
        {
            removedWidth += remColumn.ActualWidth;
            columns.Add(remColumn);
        }
    }
    removedColumn.Clear();
    totalRemainingStarValue += removedWidth;
}
totalRemainingStarValue = totalRemainingStarValue - computedWidth;
if (!isRemoved)
{
    columns.Remove(column);
    if (!removedColumn.Contains(column))
        removedColumn.Add(column);
}
}
```

Below code uses the **ColumnRatio** to apply the defined star width for each column.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ColumnSizer="Star"
ExpanderColumn="FirstName"
ItemsSource="{Binding PersonDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="Person Id"
MappingName="Id"
TextAlignment="Left"
local:StarRatio.ColumnRatio="1" />
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName"
local:StarRatio.ColumnRatio="2" />
<syncfusion:TreeGridTextColumn HeaderText="Last Name"
MappingName="LastName"
local:StarRatio.ColumnRatio="3"/>
<syncfusion:TreeGridCheckBoxColumn HeaderText="Availability"
MappingName="LikesCake" />
<syncfusion:TreeGridCurrencyColumn MappingName="Salary" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

Person Id	First Name	Last Name	Availability	Salary
0	<input type="checkbox"/> Chester	Buchanan	<input checked="" type="checkbox"/>	\$70000.00
1	<input checked="" type="checkbox"/> Abraham	Buchanan	<input checked="" type="checkbox"/>	\$40000.00
157	<input checked="" type="checkbox"/> Warner	Buchanan	<input checked="" type="checkbox"/>	\$90000.00
313	<input checked="" type="checkbox"/> Herbert	Buchanan	<input checked="" type="checkbox"/>	\$60000.00
469	<input checked="" type="checkbox"/> Gerald	Buchanan	<input checked="" type="checkbox"/>	\$90000.00
625	<input checked="" type="checkbox"/> John	Buchanan	<input checked="" type="checkbox"/>	\$110000.00
781	<input type="checkbox"/> Herbert	Taylor	<input checked="" type="checkbox"/>	\$100000.00
782	<input checked="" type="checkbox"/> Grover	Taylor	<input checked="" type="checkbox"/>	\$70000.00
938	<input checked="" type="checkbox"/> George	Taylor	<input checked="" type="checkbox"/>	\$50000.00
1094	<input checked="" type="checkbox"/> Zachary	Taylor	<input checked="" type="checkbox"/>	\$30000.00
1250	<input checked="" type="checkbox"/> James	Taylor	<input checked="" type="checkbox"/>	\$100000.00
1406	<input checked="" type="checkbox"/> Theodore	Taylor	<input checked="" type="checkbox"/>	\$60000.00
1562	<input type="checkbox"/> Theodore	Washington	<input checked="" type="checkbox"/>	\$60000.00

Change the width of TreeGridComboBoxColumn based on its ItemsSource

By default, the `ColumnSizer` calculates auto width based on the column content. You can change the auto width calculation for `TreeGridComboBoxColumn` based on its items source by overriding the `CalculateCellWidth` virtual method.

Below code creates `CustomColumnSizer` to change the width of `TreeGridComboboxColumn` and set to `SfTreeGrid.TreeGridColumnSizer`.

C#

```

this.TreeGrid.TreeGridColumnSizer = new CustomColumnSizer(this.treeGrid);
public class CustomColumnSizer : TreeGridColumnSizer
{
    public CustomColumnSizer(SfTreeGrid treeGrid)
    : base(treeGrid)
    {
    }
    protected override double CalculateCellWidth(TreeGridColumn column)
    {
        if (column is TreeGridComboBoxColumn)
        {
            double colWidth = double.MaxValue;
            var source = (column as TreeGridComboBoxColumn).ItemsSource;
            string maximumComboItemsText = string.Empty;
            var clientSize = new Size(colWidth, TreeGrid.RowHeight);
            foreach (var comboItems in source)
            {
                string comboItemText = (string)comboItems;
                if (maximumComboItemsText.Length < comboItemText.Length)
                    maximumComboItemsText = comboItemText;
            }
            var measureSize = MeasureText(clientSize, maximumComboItemsText, column,
            null, Syncfusion.UI.Xaml.Grid.GridQueryBounds.Width);
        }
    }
}

```

```
return measureSize.Width + SystemParameters.ScrollWidth;
}
else
return base.CalculateCellWidth(column);
}
}
```

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Sorting in WPF TreeGrid (SfTreeGrid)

SfTreeGrid allows you to sort the data against one or more columns either in ascending or descending order. When sorting is applied, the rows are rearranged based on sort criteria. You can allow users to sort the data by touching or clicking the column header using [SfTreeGrid.AllowSorting](#) property to `true`.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowSorting="True"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
```

C#

```
this.treeGrid.AllowSorting = true;
```

In another way, you can enable or disable the sorting for particular column by setting the [TreeGridColumn.AllowSorting](#) property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowSorting="False"
AutoGenerateColumns="False"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn AllowSorting="True" MappingName="FirstName"
/>
<syncfusion:TreeGridTextColumn AllowSorting="False" MappingName="LastName"
/>
<syncfusion:TreeGridTextColumn MappingName="Id" />
<syncfusion:TreeGridNumericColumn MappingName="Salary" />
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>
```

C#

```
this.treeGrid.Columns["FirstName"].AllowSorting = true;
this.treeGrid.Columns["LastName"].AllowSorting = false;
```

Note: The [TreeGridColumn.AllowSorting](#) takes higher priority than [SfTreeGrid.AllowSorting](#) property.

End users can sort the column by clicking column header cell. Once the columns get sorted, the sort indicator will be displayed on the right side of the column header.

First Name ▼	Last Name	Id	Salary
[-] Warner	Taft	215	\$90000.00
[+] Warner	Nixon	216	\$50000.00
[+] Warner	Buchanan	230	\$40000.00
[+] Ulysses	Coolidge	251	\$100000.00
[+] Grover	Taft	244	\$90000.00
[+] George	Buchanan	223	\$50000.00
[+] Abraham	Kennedy	237	\$70000.00
[-] Ulysses	Stogner	86	\$50000.00
[+] Warner	Coolidge	87	\$30000.00

Sort column in double click

By default, column gets sorted when column header clicked. You can change this behavior to sort the column in double click action by setting [SfTreeGrid.SortClickAction](#) property to **DoubleClick**.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowSorting="True"
SortClickAction="DoubleClick"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
```

C#

```
this.treeGrid.AllowSorting = true;
this.treeGrid.SortClickAction = SortClickAction.DoubleClick;
```

Sorting order

By default, the data is sorted in ascending or descending order when clicking column header. You can rearrange the data to its initial order from descending, when clicking column header by setting [SfTreeGrid.AllowTriStateSorting](#) property.

Following are the sequence of sorting orders when clicking column header,

- Sorts the data in ascending order
- Sorts the data in descending order
- Clears the sorting and records displayed in its initial order

Multi column sorting

SfTreeGrid control allows you sort more than one column, where sorting is applied one column against other columns. To apply sorting on multiple columns, user have to click the column header by pressing the Ctrl key.

In the below screen shot, the First Name column sorted. Then the Employee ID column is sorted against the First Name data by clicking column header by pressing Ctrl key. The sorting state of First Name column is preserved and Employee ID column sorted against First Name column.

First Name ^	Last Name	Id v	Salary
[-] Warner	Taft	215	\$90000.00
[+] Ulysses	Coolidge	251	\$100000.00
[+] Grover	Taft	244	\$90000.00
[+] Abraham	Kennedy	237	\$70000.00
[+] Warner	Buchanan	230	\$40000.00
[+] George	Buchanan	223	\$50000.00
[+] Warner	Nixon	216	\$50000.00
[-] Andrew	Polk	172	\$110000.00
[+] Andrew	Monroe	208	\$80000.00
[+] Harry	Carter	201	\$60000.00

Display sort order

It is also possible to display sorted order of columns in header by setting [SfTreeGrid.ShowSortNumbers](#) property to true.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowSorting="True"
ShowSortNumbers="True"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
```

C#

```
this.treeGrid.ShowSortNumbers = true;
```

First Name ▼ ¹	Last Name	Id ▲ ²	Salary
[-] Warner	Taft	215	\$90000.00
[+] Warner	Nixon	216	\$50000.00
[+] Warner	Buchanan	230	\$40000.00
[+] Ulysses	Coolidge	251	\$100000.00
[+] Grover	Taft	244	\$90000.00
[+] George	Buchanan	223	\$50000.00
[+] Abraham	Kennedy	237	\$70000.00
[-] Ulysses	Stogner	86	\$50000.00
[+] Warner	Coolidge	87	\$30000.00
[+] Richard	Wilson	115	\$30000.00

Programmatic Sorting

You can sort the data programmatically by adding or removing the `SortColumnDescription` in [SfTreeGrid.SortColumnDescriptions](#) property.

Note: [SfTreeGrid.SortColumnChanging](#) and [SfTreeGrid.SortColumnChanged](#) events are not raised when the data sorted programmatically through `SfTreeGrid.SortColumnDescriptions`.

Adding sort columns

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowSorting="True"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
  <syncfusion:SfTreeGrid.SortColumnDescriptions>
    <sync:SortColumnDescription ColumnName="FirstName" SortDirection="Ascending"
/>
    <sync:SortColumnDescription ColumnName="Id" SortDirection="Descending"/>
  </syncfusion:SfTreeGrid.SortColumnDescriptions>
</syncfusion:SfTreeGrid>
```

C#

```
this.treeGrid.SortColumnDescriptions.Add(new SortColumnDescription() {
    ColumnName = "FirstName", SortDirection=ListSortDirection.Ascending });
this.treeGrid.SortColumnDescriptions.Add(new SortColumnDescription() {
    ColumnName = "Id", SortDirection = ListSortDirection.Descending });
```

Removing sort columns

You can unsort the data by removing the corresponding `SortColumnDescription` from the [SfTreeGrid.SortColumnDescriptions](#) property.

C#

```
var sortColumnDescription =  
this.treeGrid.SortColumnDescriptions.FirstOrDefault(col => col.ColumnName ==  
"FirstName");  
if (sortColumnDescription != null)  
this.treeGrid.SortColumnDescriptions.Remove(sortColumnDescription);
```

Clear sorting

You can clear sorting, by clearing the [SfTreeGrid.SortColumnDescriptions](#).

C#

```
this.treeGrid.SortColumnDescriptions.Clear();
```

Custom sorting

SfTreeGrid allows you to sort the columns based on the custom logic.

The custom sorting can be applied by adding the [SortComparer](#) instance to [SfTreeGrid.SortComparers](#).

The [SortComparer](#) have the following properties,

[PropertyName](#) - Gets or sets the name of the column to apply custom sorting.

[Comparer](#) - Gets or sets the custom comparer in which you can code to compare the data using custom logic.

You can implement [ISortDirection](#) interface in comparer to get the sort direction. So you can apply different custom logics for ascending and descending.

Follow the below steps to add custom comparer to sort using custom logic,

Define custom comparer with custom sort logic

In the below code snippet, `FirstName` property is compared based on its string length, instead of default string comparison.

C#

```
public class CustomSortComparer : IComparer<object>, ISortDirection  
{  
    public int Compare(object x, object y)  
    {  
        var item1 = x as EmployeeInfo;  
        var item2 = y as EmployeeInfo;  
        var value1 = item1.FirstName;  
        var value2 = item2.FirstName;  
        int c = 0;  
        if (value1 != null && value2 == null)  
        {  
            c = 1;  
        }  
        else if (value1 == null && value2 != null)  
        {  
            c = -1;  
        }  
        else if (value1 != null && value2 != null)  
        {  
            c = value1.Length.CompareTo(value2.Length);  
        }  
    }  
}
```

```

if (SortDirection == ListSortDirection.Descending)
{
    c = -c;
    return c;
}
//Get or Set the SortDirection value
private ListSortDirection _SortDirection;
public ListSortDirection SortDirection
{
    get { return _SortDirection; }
    set { _SortDirection = value; }
}
}

```

Adding custom comparer to SfTreeGrid

Custom comparer can be added to [SfTreeGrid.SortComparers](#) property. **SortComparers** maintains custom comparers and the custom comparer gets called when corresponding column gets sorted by clicking column header or programmatically.

XML

```

xmlns:data="clr-namespace:Syncfusion.Data;assembly=Syncfusion.Data.WPF"
<syncfusion:SfTreeGrid.SortComparers>
<data:SortComparer Comparer="{StaticResource sortComparer}"
PropertyName="FirstName" />
</syncfusion:SfTreeGrid.SortComparers>

```

C#

```

this.treeGrid.SortComparers.Add(new SortComparer() { Comparer = new
CustomSortComparer(), PropertyName = "FirstName" });

```

Sorting **FirstName** column sorts the data using custom sort comparer available in **SfTreeGrid.SortComparers**.

First Name ▼	Last Name	Id	Salary
<input type="checkbox"/> Chester	Buchanan	0	\$100000.00
<input checked="" type="checkbox"/> Rutherford	Clinton	8	\$50000.00
<input checked="" type="checkbox"/> Abraham	Fillmore	1	\$100000.00
<input checked="" type="checkbox"/> Chester	Polk	15	\$50000.00
<input checked="" type="checkbox"/> Chester	Monroe	22	\$90000.00
<input checked="" type="checkbox"/> George	Truman	29	\$50000.00
<input checked="" type="checkbox"/> Warner	Roosevelt	36	\$40000.00
<input type="checkbox"/> Ulysses	Stogner	86	\$50000.00
<input checked="" type="checkbox"/> Millard	Taft	94	\$110000.00
<input checked="" type="checkbox"/> Millard	Washington	108	\$100000.00

Handling events

SortColumnChanging event

[SfTreeGrid.SortColumnChanging](#) event occurs while sorting the columns by clicking column header.

[GridSortColumnsChangingEventArgs](#) has following members which provides information for SortColumnChanging event.

[Action](#) - Gets the action triggered this event.

[Cancel](#) - Setting value to `true`, cancels the triggered action.

[AddedItems](#) - Gets the list of new `SortColumnDescriptions` that are added.

[RemovedItems](#) - Gets the list of `SortColumnDescriptions` that are removed.

[CancelScroll](#) - Gets or sets a value that indicates, whether scroll and bring SelectedItem in view after sorting takes place.

You can prevent sorting for the particular column through [GridSortColumnsChangingEventArgs.Cancel](#) property of `SortColumnChanging` event.

C#

```
this.treeGrid.SortColumnsChanging += TreeGrid_SortColumnsChanging;
private void TreeGrid_SortColumnsChanging(object sender,
GridSortColumnsChangingEventArgs e)
{
    if (e.AddedItems[0].ColumnName == "FirstName")
    {
        e.Cancel = true;
    }
}
```

SortColumnChanged event

[SfTreeGrid.SortColumnChanged](#) event occurs when the sorting is applied to the column.

[GridSortColumnsChangedEventArgs](#) provides information for `SortColumnChanged` event.

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Filtering in WPF TreeGrid (SfTreeGrid)

Filtering is the process of retrieving the values from a collection that satisfies the specified condition. In SfTreeGrid, filtering can be applied through the UI as well as the programmatic filters.

FilterLevel

You can filter the nodes based on level using the [SfTreeGrid.FilterLevel](#) property.

C#

```
treeGrid.FilterLevel = FilterLevel.All;
```

- Root - Filter will be applied to root nodes only in SfTreeGrid.
- All - Filter will be applied to all the nodes in SfTreeGrid.

- Extended - Filter will be applied to all the nodes. If a node matches the filter condition, its all ancestors will be displayed even though the parent node does not match the filter condition.

Root

Filter will be applied to root nodes only in SfTreeGrid. For other nodes, `IsFiltered` value will be false, and they always will be displayed in view.

All

Filter will be applied to all the nodes in SfTreeGrid. If a parent node does not match the filter condition, filter will not be applied for child nodes. Else, filter will be applied to its child nodes also.

Extended

Filter will be applied to all the nodes. If a node matches the filter condition, its all ancestors will also be displayed even though the parent node does not match the filter condition, and parent node's `IsFiltered` value will be set to false.

Note: You can change the `FilterLevel` at run time.

Programmatic filtering

The programmatic filtering can be applied to SfTreeGrid using the following methods:

View Filtering Column Filtering

View filtering

View filtering can be achieved by setting the [SfTreeGrid.View.Filter](#) delegate and calling the [SfTreeGrid.View.RefreshFilter](#) method.

C#

```
public bool FilerNodes(object o)
{
    var data = o as Employee;
    if (data.Salary > 70000)
        return true;
    return false;
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    treeGrid.View.Filter = FilerNodes;
    treeGrid.View.RefreshFilter();
}
```

Here, `FilterNodes` delegate filters the data based on Salary. `FilterNodes` delegate is assigned to [SfTreeGrid.View.Filter](#) predicate to filter the tree grid. After that, [SfTreeGrid.View.RefreshFilter](#) method is called to refresh the nodes. If the node satisfies the filter conditions, true will be returned. Else false will be returned.

First Name	Last Name	Employee ID	City	Salary
Chester	Buchanan	0	Barcelona	\$90000.00
Woodrow	Bush	5556	Canberra	\$110000.00
Harry	Grant	8889	Manchester	\$90000.00
Ronald	Jones	33333	Sydney	\$110000.00
Millard	Harrison	55555	Tokyo	\$90000.00
Zachary	Washington	56667	London	\$80000.00
Andrew	Van Buren	60000	Adelaide	\$90000.00
Chester	Adams	61111	Perth	\$80000.00
Woodrow	Stogner	63333	Brisbane	\$90000.00
Chester	Hoover	66666	Birmingham	\$100000.00
Abraham	Grant	88888	Rome	\$90000.00
Calvin	Monroe	88889	Adelaide	\$90000.00
Theodore	Wilson	91111	Manchester	\$80000.00

While filtering, if the node satisfies filter condition, `IsFiltered` property of `TreeNode` will be set as false. Else, it will be true. If `IsFiltered` value is True, the node will not be displayed in view, else it will be displayed in view.

Note: SfTreeGrid refreshes the filtering on property change if `SfTreeGrid.LiveNodeUpdateMode` property is set as `AllowDataShaping`.

You can download the sample from [here](#).

Clear filters

You can clear the view filters applied in tree grid by setting the `SfTreeGrid.View.Filter` delegate to null and calling the `SfTreeGrid.View.RefreshFilter` method.

C#

```
treeGrid.View.Filter = null;
treeGrid.View.RefreshFilter();
```

HasVisibleChildNodes

You can find whether a particular node has child node(s) displayed in a view (matches filtering criteria) using the `HasVisibleChildNodes` property in `TreeNode`.

C#

```
var treeNode=treeGrid.View.Nodes[0];
var hasVisibleChildNodes = treeNode.HasVisibleChildNodes;
```

Column filtering

The column filtering can be achieved by adding the `FilterPredicate` to the `TreeGridColumn.FilterPredicates` property.

C#

```
this.sfTreeGrid.Columns["FirstName"].FilterPredicates.Add(  
new Syncfusion.Data.FilterPredicate()  
{  
    FilterType = Syncfusion.Data.FilterType.Equals,  
    FilterValue = "Chester"  
});
```

Filter behavior

The [FilterBehavior](#) property is used to specify whether to consider the [FilterValue](#) as string or specific data type.

- **StringTyped** - Records are filtered without considering the type and it takes FilterValue type as string.
- **StronglyTyped** - Records are filtered by considering the FilterValue underlying type.

Clear filtering

The filters applied to SfTreeGrid can be removed by clearing the [FilterPredicates](#) added for the columns. This can be achieved using the following methods:

- [SfTreeGrid.ClearFilters](#) - Clears filters for all the columns programmatically.
- [SfTreeGrid.ClearFilter\(String columnName\)](#) - Clears the filter for a specific column that has the columnName as MappingName.
- [SfTreeGrid.ClearFilter\(TreeGridColumn column\)](#) - Clears the filter for a specific column alone.

C#

```
this.sfTreeGrid.ClearFilters();  
this.sfTreeGrid.ClearFilter("FirstName");  
this.sfTreeGrid.ClearFilter(this.sfTreeGrid.Columns["FirstName"]);
```

Adding multiple filter predicates for a column

The [PredicateType](#) property is used to apply multiple FilterPredicates for a column.

- **And**: Performs And operation in filters.
- **AndAlso**: Performs AndAlso operation in filters.
- **Or**: Performs Or operation in filters.
- **OrElse**: Performs OrElse operation in filters.

C#

```
this.sfTreeGrid.Columns["FirstName"].FilterPredicates.Add(  
new Syncfusion.Data.FilterPredicate()  
{  
    FilterType = Syncfusion.Data.FilterType.Equals,  
    FilterValue = "Chester",  
    PredicateType = Syncfusion.Data.PredicateType.Or  
});  
this.sfTreeGrid.Columns["FirstName"].FilterPredicates.Add(  
new Syncfusion.Data.FilterPredicate()
```



```
{
    FilterType = Syncfusion.Data.FilterType.Equals,
    FilterValue = "Martin",
    PredicateType = Syncfusion.Data.PredicateType.Or
});
```

First Name	Last Name	Employee ID	Date of Joining	Contact Number	City
<input type="checkbox"/> Chester	Buchanan	0	11/15/2010	999116	Perth
<input checked="" type="checkbox"/> Abraham	Fillmore	1	1/20/2011	999111	Barcelona
<input checked="" type="checkbox"/> Ulysses	Reagan	8	11/6/2011	999118	San Francisco
<input checked="" type="checkbox"/> Zachary	Harding	15	1/10/2008	999116	Birmingham
<input checked="" type="checkbox"/> Theodore	Kennedy	22	5/17/2008	999118	Madrid
<input checked="" type="checkbox"/> Ulysses	Madison	29	1/11/2008	999115	San Francisco
<input checked="" type="checkbox"/> Ronald	Johnson	36	8/14/2009	999118	Liverpool
<input type="checkbox"/> Martin	Buchanan	86	9/11/2010	999115	Adelaide
<input checked="" type="checkbox"/> Franklin	Garfield	87	10/5/2010	999113	Canberra
<input checked="" type="checkbox"/> Theodore	Grant	94	7/21/2010	999118	Cardiff
<input checked="" type="checkbox"/> William	Bush	101	4/24/2008	999112	London
<input checked="" type="checkbox"/> Warren	Stogner	108	3/13/2008	999115	Manchester
<input checked="" type="checkbox"/> Andrew	Clinton	115	3/18/2010	999112	NewYork
<input checked="" type="checkbox"/> William	Roosevelt	122	1/4/2010	999116	Perth
<input type="checkbox"/> Chester	Tyler	129	8/3/2008	999116	Madrid
<input checked="" type="checkbox"/> Andrew	McKinley	130	1/8/2010	999113	Durban
<input checked="" type="checkbox"/> Larry	Wilson	137	10/10/2010	999117	Brisbane
<input checked="" type="checkbox"/> Warren	Hayes	144	8/6/2009	999115	London
<input checked="" type="checkbox"/> Harry	Bush	151	1/20/2009	999111	Manchester
<input checked="" type="checkbox"/> Benjamin	Nixon	158	1/1/2011	999112	NewYork

Filter DataColumn with range between two dates

A [TreeGridDateTimeColumn](#) can be filtered with a range between two dates by applying two [FilterPredicate](#) for the same column. The [FilterType](#) for the [FilterPredicate](#) with start date should be [GreaterThanOrEqual](#) and end date should be [LessThanOrEqual](#).

UI filtering

SfTreeGrid provides Excel-like filtering UI and advanced filter UI to filter the data easily. UI filtering can be enabled by setting the [SfTreeGrid.AllowFiltering](#) property to [true](#). This allows to open the filter UI by clicking the filter icon on the column header to filter the nodes.

The filtering can be enabled or disabled for a specific column by setting the [TreeGridColumn.AllowFiltering](#) property.

C#

```
// Enable UI filtering for SfTreeGrid.
this.sfTreeGrid.AllowFiltering = true;
// Enable UI filtering for EmployeeID column.
this.sfTreeGrid.Columns["EmployeeID"].AllowFiltering = true;
```

Note: `TreeGridColumn.AllowFiltering` has higher priority than the `SfTreeGrid.AllowFiltering` property.

Built-in UI views

The SfTreeGrid provides the following types of filter pop-up modes:

- Check box filter: Provides Excel-like filter interface with a list of check boxes.
- Advanced filter: Provides advanced filter options to filter the data.
- Both: Both check box filter and advanced filter are loaded while opening the filter pop-up.

By default, the filter pop-up mode of the column is set to Both. The check box and the advanced filters can be switched using the Advanced Filter button.

Checkbox filtering UI

First Name	Last Name	Employee ID	Date of Joining	Contact Number	City
Ches			11/15/2010	999116	Perth
Al			1/20/2011	999111	Barcelona
UI			11/6/2011	999118	San Francisco
Za			1/10/2008	999116	Birmingham
Th			5/17/2008	999118	Madrid
UI			1/11/2008	999115	San Francisco
Re			8/14/2009	999118	Liverpool
Grov			1/10/2009	999117	San Francisco
Ge			9/18/2008	999112	Zurich
Za			11/25/2011	999113	London
W			7/1/2008	999113	Sydney
Ge			5/16/2010	999117	San Francisco
M			4/16/2011	999114	NewYork
M			1/27/2008	999112	NewYork
Mart			9/11/2010	999115	Adelaide
Fr			10/5/2010	999113	Canberra
Th			7/21/2010	999118	Cardiff
W			4/24/2008	999112	London
W			3/13/2008	999115	Manchester
Andrew	Clinton	115	3/18/2010	999112	NewYork

Advanced filtering UI

First Name	Last Name	Employee ID	Date of Joining	Contact Number	City
Ches			11/15/2010	999116	Perth
Al			1/20/2011	999111	Barcelona
UI			11/6/2011	999118	San Francisco
Za			1/10/2008	999116	Birmingham
Th			5/17/2008	999118	Madrid
UI			1/11/2008	999115	San Francisco
Ro			8/14/2009	999118	Liverpool
Gro			1/10/2009	999117	San Francisco
Ge			9/18/2008	999112	Zurich
Za			11/25/2011	999113	London
W			7/1/2008	999113	Sydney
Ge			5/16/2010	999117	San Francisco
M			4/16/2011	999114	NewYork
M			1/27/2008	999112	NewYork
Mart			9/11/2010	999115	Adelaide
Fr			10/5/2010	999113	Canberra
Th			7/21/2010	999118	Cardiff
W			4/24/2008	999112	London
W			3/13/2008	999115	Manchester
Andrew	Clinton	115	3/18/2010	999112	NewYork

Changing filter UI for grid

Filter UI view can be changed for all the columns in grid by changing the [FilterMode](#) in [TreeGridFilterControl](#) by writing style and assign it to [SfTreeGrid.FilterPopupStyle](#).

XML

```
<Style x:Key="filterControlStyle"
TargetType="syncfusion:TreeGridFilterControl">
<Setter Property="FilterMode" Value="AdvancedFilter" />
</Style>
<syncfusion:SfTreeGrid Name="sfTreeGrid"
FilterPopupStyle="{StaticResource filterControlStyle}"
AllowFiltering="True"
ItemsSource="{Binding EmployeeDetails}" />
```

Changing filter UI for a column

Filter UI view can be changed for a specific column by changing the [FilterMode](#) in [TreeGridFilterControl](#) by writing style and assigning it to [TreeGridColumn.FilterPopupStyle](#).

XML

```
<Style x:Key="filterControlStyle"
TargetType="syncfusion:TreeGridFilterControl">
<Setter Property="FilterMode" Value="AdvancedFilter" />
</Style>
```

```
<syncfusion:TreeGridTextColumn MappingName="FirstName"
FilterPopupStyle="{StaticResource filterControlStyle}"/>
```

Setting default filter popup style for a specific column

You can skip the [TreeGridFilterControl](#) styling for a specific column in [SfTreeGrid.FilterPopupStyle](#) by setting [TreeGridColumn.FilterPopupStyle](#) to null.

XML

```
<Style x:Key="filterControlStyle"
TargetType="syncfusion:TreeGridFilterControl">
<Setter Property="FilterMode" Value="AdvancedFilter" />
</Style>
<syncfusion:SfTreeGrid Name="sfTreeGrid"
FilterPopupStyle="{StaticResource filterControlStyle}"
AllowFiltering="True"
ItemsSource="{Binding EmployeeDetails}" />
<syncfusion:TreeGridTextColumn MappingName="LastName"
FilterPopupStyle="{x:Null}"/>
```

Check box filtering

The check box filtering is the same as Excel-like filter popup, which displays a search text box and a list of check boxes with unique items from the expanded tree nodes.

The items in the checked state will be visible in the view, and the other items will be filtered out of the view.

This filtering operation is performed based on the value of [SfTreeGrid.FilterLevel](#) property.



Advanced filtering

Advanced filter UI provides multiple filter options to filter the data easily. Filter menu options are loaded based on advanced filter type by automatically detecting the underlying data type.

The following built-in filter types are supported in SfTreeGrid:

- Text filter: Loads various menu options to filter the display text effectively.
- Number filter: Loads various menu options to filter the numeric data.
- Date filter: Loads various menu options and [DatePicker](#) to filter date-time type columns.

Text filters	Number filters	Date filters
When the string value is bound to the [TreeGridColumn](https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.TreeGrid.TreeGridColumn.html) or the items source is dynamic , then TextFilters are loaded in TreeGridAdvancedFilterControl .	When integer, double, short, decimal, byte, or long are bound to the TreeGridColumn , then Number Filters are loaded in [TreeGridAdvancedFilterControl](https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.TreeGrid.Filtering.TreeGridAdvancedFilterControl.html).	When the DateTime type value is bound to the TreeGridColumn , then Date Filters are loaded in TreeGridAdvancedFilterControl .

		cedFilterControl' markdownify}}.
		
Filter menu options	Filter menu options	Filter menu options
<ol style="list-style-type: none"> 1. Equals 2. Does Not Equal 3. Begins With 4. Ends With 5. Contains 6. Does Not Contain 7. Empty 8. Not Empty 9. Null 10. Not Null 	<ol style="list-style-type: none"> 1. Equals 2. Does Not Equal 3. Null 4. Not Null 5. Less Than 6. Less Than or Equal 7. Greater Than 8. Greater Than or Equal 	<ol style="list-style-type: none"> 1. Equals 2. Does Not Equal 3. Before 4. Before Or Equal 5. After 6. After Or Equal 7. Null 8. Not Null

Note: The **Null** and **Not Null** options are available only when **AllowBlankFilters** is set to **True**.

Instant filtering

By default, filters are applied to the columns when OK button is clicked in UI filtering. To update the filters immediately whenever update in filter popup, set [TreeGridColumn.ImmediateUpdateColumnFilter](#) to **true**.

XML

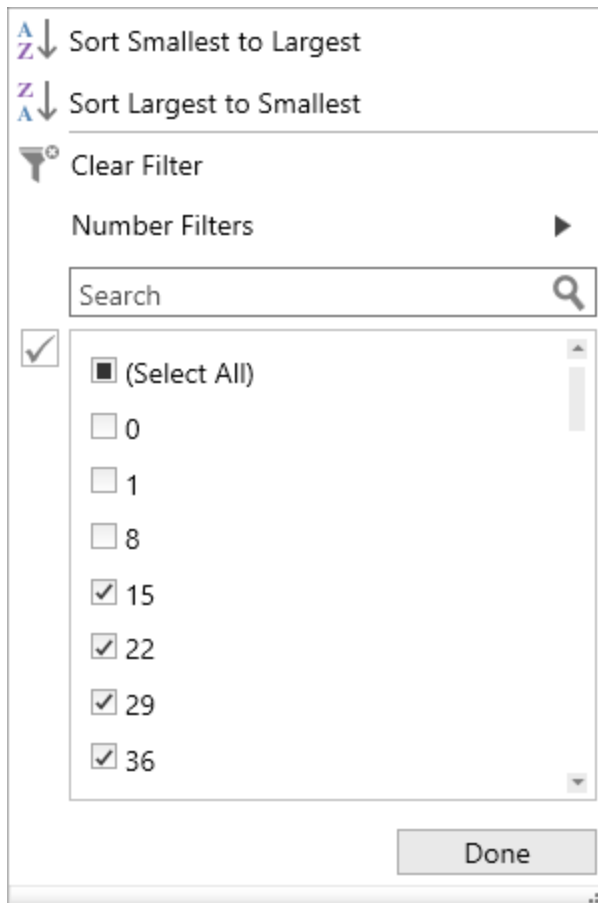
```
<syncfusion:TreeGridTextColumn MappingName="EmployeeID"
    ImmediateUpdateColumnFilter="True"/>
```

C#

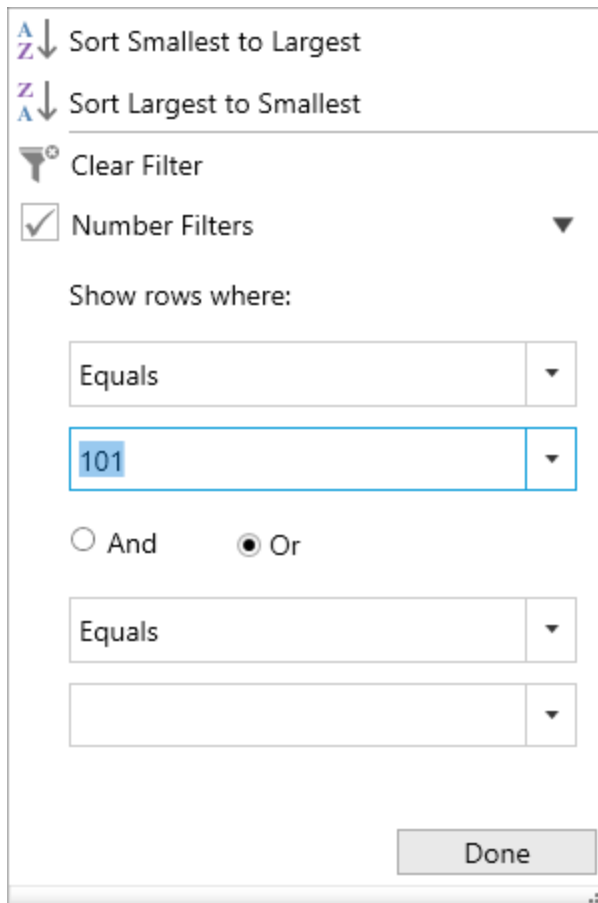
```
this.sfTreeGrid.Columns["EmployeeID"].ImmediateUpdateColumnFilter = true;
```

Here, the OK and Cancel buttons are unavailable and Done button is available to just close the popup.

The following screenshot illustrates the check box filter when **ImmediateUpdateColumnFilter** is set to **true**.



The following screenshot illustrates the advanced filter when `ImmediateUpdateColumnFilter` is set to `true`.



Note: In check box filter, the `SelectAll` option is not reflected in the filter updates if [ImmediateUpdateColumnFilter](#) is true.

Filtering null values

To filter the null values, the [TreeGridColumn.AllowBlankFilters](#) property should be enabled. Enabling `AllowBlankFilters` includes null values into the filter items list.

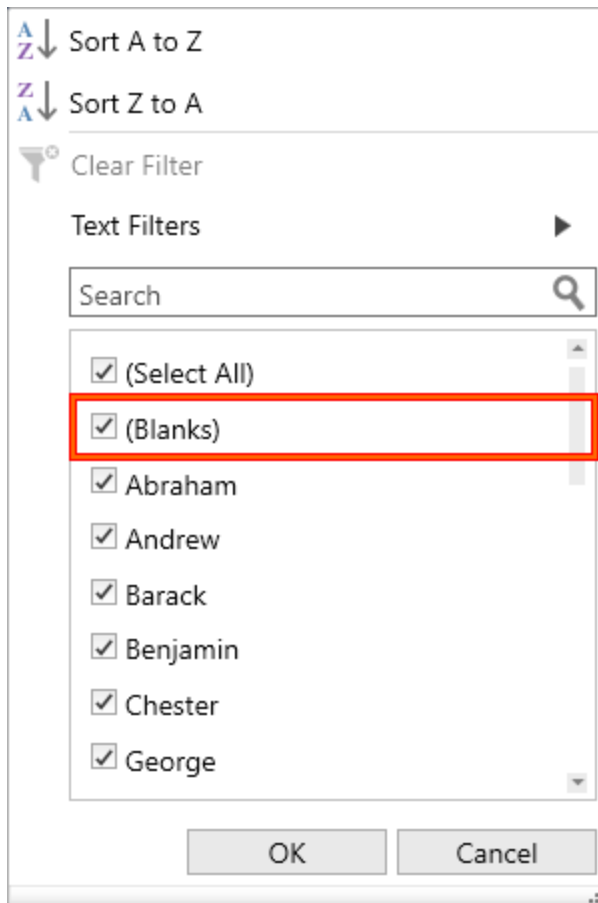
XML

```
<syncfusion:TreeGridTextColumn MappingName="FirstName"
    AllowBlankFilters="True"/>
```

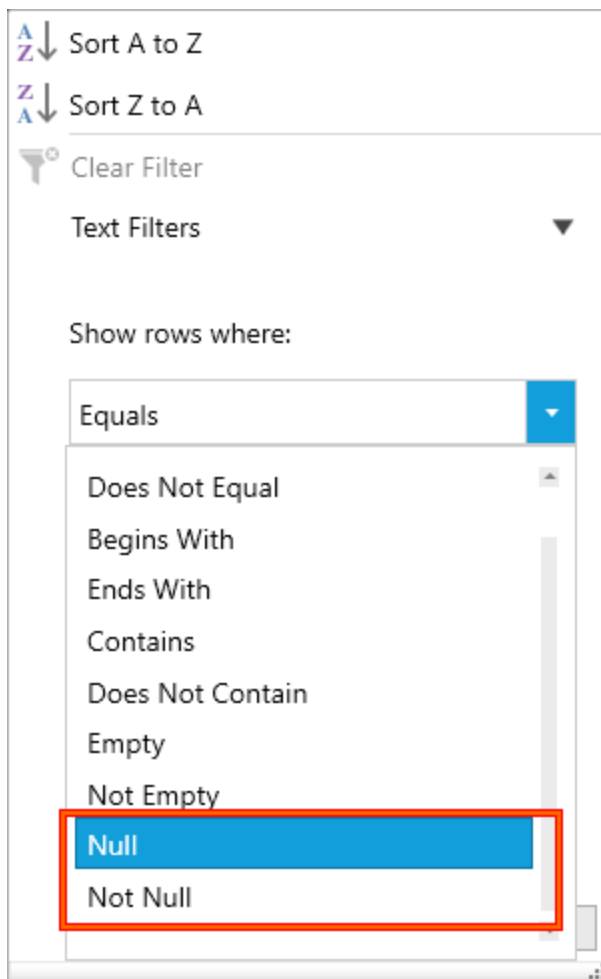
C#

```
this.sfTreeGrid.Columns["FirstName"].AllowBlankFilters = true;
```

The following screenshot illustrates the check box filter when `AllowBlankFilters` is set to `true`.



The following screenshot illustrates advanced filter when `AllowBlankFilters` is set to `true`.



Changing AdvancedFilter type when loading dynamic ItemsSource

By default, the text filters will be loaded for the columns if `ItemsSource` is [dynamic](#). The [TreeGridColumn.ColumnMemberType](#) property loads number filters or date filters based on the column values.

C#

```
this.sfTreeGrid.Columns["EmployeeID"].ColumnMemberType = typeof(double);
```

Customization using events

Loading text filter for number or date column

The [SfTreeGrid.FilterItemsPopulating](#) event is used to load text filters for the columns that have number or date value as underlying type by setting value of the `TreeGridFilterItemsPopulatingEventArgs.FilterControl.AdvancedFilterType` property to `AdvancedFilterType.TextFilter`.

C#

```
this.sfTreeGrid.FilterItemsPopulating += OnSfTreeGridFilterItemsPopulating;  
private void OnSfTreeGridFilterItemsPopulating(object sender,  
TreeGridFilterItemsPopulatingEventArgs e)  
{
```

```
if (e.Column.MappingName == "EmployeeID")
e.FilterControl.AdvancedFilterType = AdvancedFilterType.TextFilter;
}
```

Customizing filter predicates

The filter predicates can be customized using the [SfTreeGrid.FilterChanging](#) event. This event occurs when applying filter using the filter control. Here, [FilterValue](#) is changed based on some conditions.

C#

```
this.sfTreeGrid.FilterChanging += OnSfTreeGridFilterChanging;
private void OnSfTreeGridFilterChanging(object sender,
TreeGridFilterChangingEventArgs e)
{
if (e.FilterPredicates == null || e.Column.MappingName != "FirstName")
return;
if (e.FilterPredicates[0].FilterValue.Equals("Chester"))
e.FilterPredicates[0].FilterValue = "Abraham";
}
```

Customizing Excel-like filter ItemsSource

The [TreeGridFilterControl.ItemsSource](#) can be customized to restrict some data from filtering using the [SfTreeGrid.FilterItemsPopulated](#) event. Here, the [FilterElement](#) that has actual value as 0 is removed from [ItemsSource](#).

C#

```
this.sfTreeGrid.FilterItemsPopulated += OnSfTreeGridFilterItemsPopulated;
private void OnSfTreeGridFilterItemsPopulated(object sender,
TreeGridFilterItemsPopulatedEventArgs e)
{
if (e.Column.MappingName == "EmployeeID")
{
var itemsSource = e.ItemsSource as List<FilterElement>;
// Get the FilterElement to remove from itemsSource.
var filterElement = itemsSource.FirstOrDefault(items =>
items.ActualValue.Equals(0));
// Remove the FilterElement from itemsSource.
itemsSource.Remove(filterElement);
}
}
```

Changing filter UI

Filter UI can be changed either for all the columns or for a specific column in SfTreeGrid by changing the [FilterMode](#) property value using the [SfTreeGrid.FilterItemsPopulating](#) event.

Here, filter UI is changed to [AdvancedFilter](#) only for [EmployeeID](#) column.

C#

```
this.sfTreeGrid.FilterItemsPopulating += SfTreeGrid_FilterItemsPopulating;
private void SfTreeGrid_FilterItemsPopulating(object sender,
TreeGridFilterItemsPopulatingEventArgs e)
{
if (e.Column.MappingName == "EmployeeID")
```

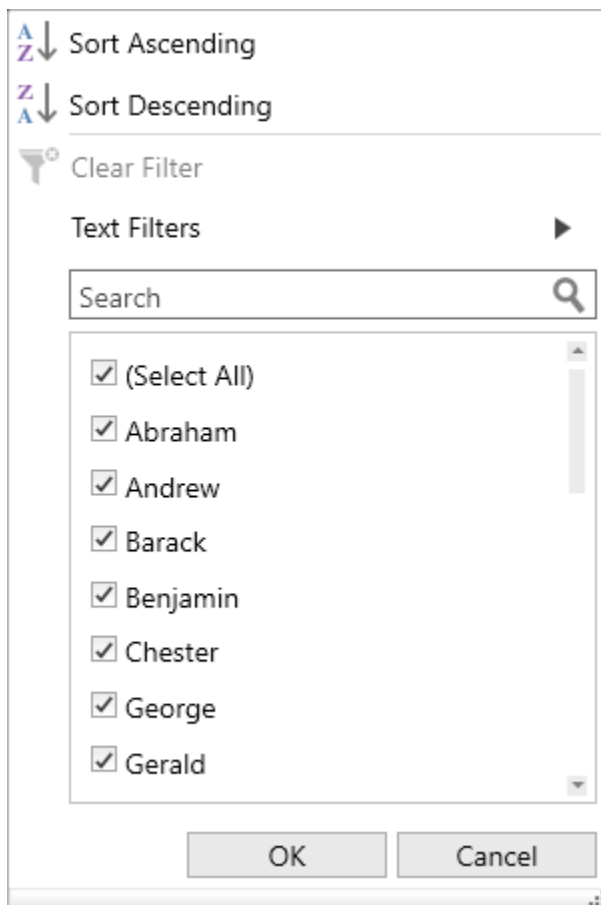
```
e.FilterControl.FilterMode = FilterMode.AdvancedFilter;
}
```

Customizing sort options text

Sort options text can be customized by changing the value of [AscendingSortString](#) and [DescendingSortString](#) properties in the [TreeGridFilterControl](#) using the [SfTreeGrid.FilterItemsPopulating](#) event.

C#

```
this.sfTreeGrid.FilterItemsPopulating += OnSfTreeGridFilterItemsPopulating;
private void OnSfTreeGridFilterItemsPopulating(object sender,
TreeGridFilterItemsPopulatingEventArgs e)
{
    if (e.Column.MappingName == "FirstName")
    {
        e.FilterControl.AscendingSortString = "Sort Ascending";
        e.FilterControl.DescendingSortString = "Sort Descending";
    }
}
```



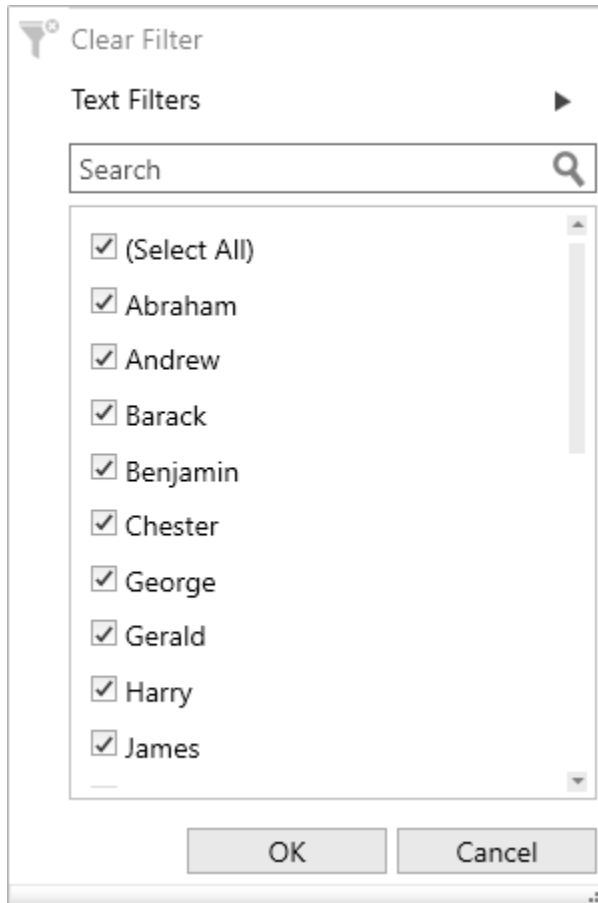
Appearance

Hiding sort options

Sort options can be collapsed by setting the [SortOptionVisibility](#) property in [TreeGridFilterControl](#).

XML

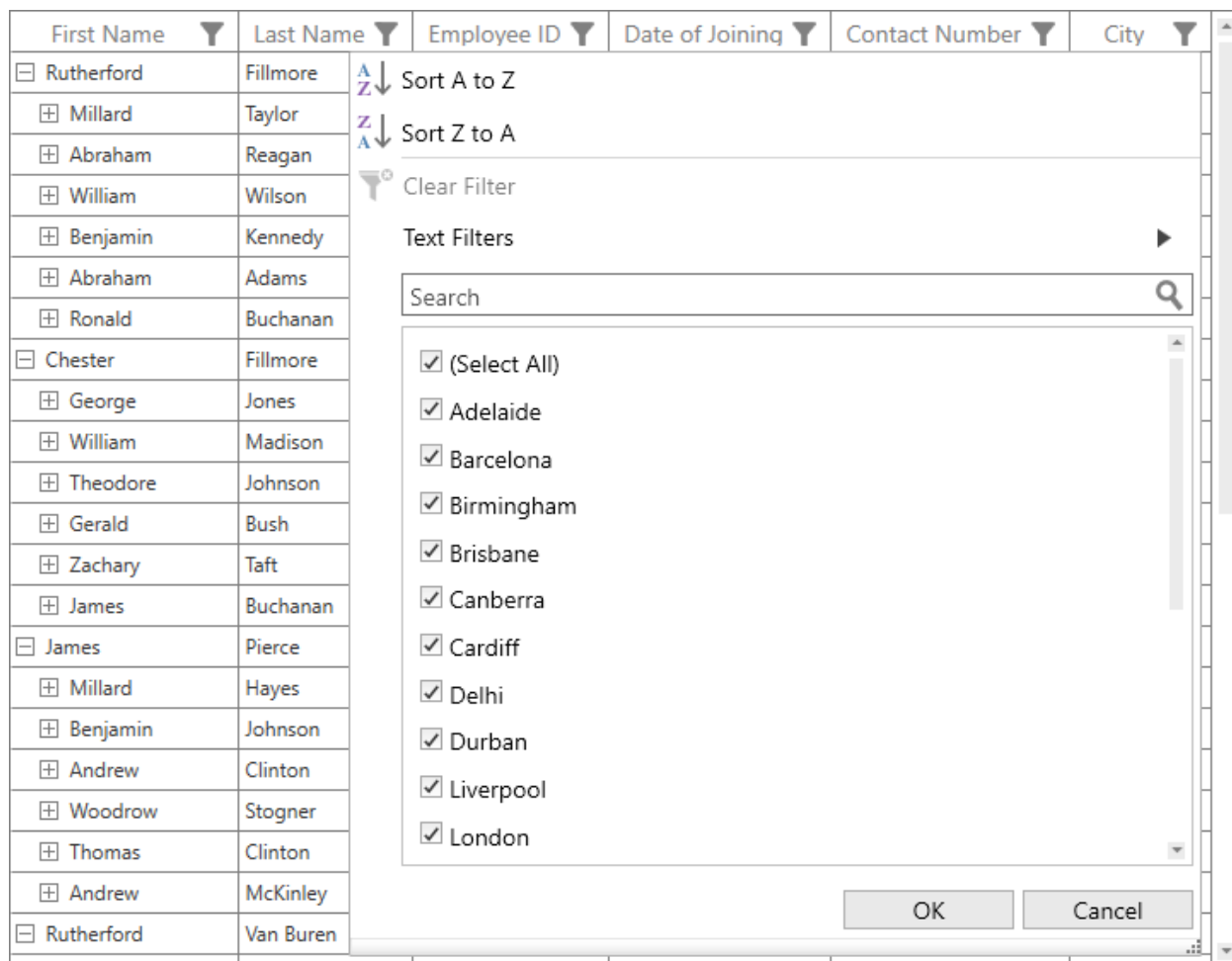
```
<Style TargetType="syncfusion:TreeGridFilterControl"
x:Key="filterControlStyle">
<Setter Property="SortOptionVisibility" Value="Collapsed"/>
</Style>
<syncfusion:SfTreeGrid Name="sfTreeGrid"
FilterPopupStyle="{StaticResource filterControlStyle}"
AllowFiltering="True"
ItemsSource="{Binding EmployeeDetails}" />
```

*Customizing the filter popup size*

The size of the filter popup can be changed using the [FilterPopupHeight](#) and [FilterPopupWidth](#) properties in [TreeGridFilterControl](#).

XML

```
<Window.Resources>
<Style TargetType="syncfusion:TreeGridFilterControl">
<Setter Property="FilterPopupHeight" Value="530"/>
<Setter Property="FilterPopupWidth" Value="500"/>
</Style>
</Window.Resources>
```



Changing filter icon style after applying filters

The filter icon style can be changed by writing style with TargetType as [FilterToggleButton](#).

C#

```
<Style TargetType="syncfusion:FilterToggleButton">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="syncfusion:FilterToggleButton">
        <Grid SnapsToDevicePixels="True">
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
              <VisualState x:Name="Normal" />
              <VisualState x:Name="MouseOver" />
              <VisualState x:Name="Pressed" />
              <VisualState x:Name="Disabled" />
            </VisualStateGroup>
            <VisualStateGroup x:Name="FilterStates">
              <VisualState x:Name="Filtered">
                <Storyboard>
                  <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="PART_FilterToggleButtonIndicator"
                    Storyboard.TargetProperty="Data">
                      <DiscreteObjectKeyFrame KeyTime="0">
```

```

























<DiscreteObjectKeyFrame.Value>
<Geometry>M2.1299944,9.9798575L55.945994,9.9798575 35.197562,34.081179
35.197562,
62.672859 23.428433,55.942383 23.428433,33.52121z M1.3001332,0L56.635813,
0C57.355887,0,57.935946,0.5891428,57.935946,1.3080959L57.935946,
2.8258877C57.935946,3.5448422,57.355887,4.133985,56.635813,4.133985L1.300133
2,
4.133985C0.58005941,4.133985,-2.3841858E-07,3.5448422,0,2.8258877L0,
1.3080959C-2.3841858E-07,0.5891428,0.58005941,0,1.3001332,0z
</Geometry>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
<ColorAnimation Duration="0:0:0:1"
Storyboard.TargetName="PathFillColor"
Storyboard.TargetProperty="Color"
To="Red" />
</Storyboard>
</VisualState>
<VisualState x:Name="UnFiltered">
<Storyboard>
<ObjectAnimationUsingKeyFrames
Storyboard.TargetName="PART_FilterToggleButtonIndicator"
Storyboard.TargetProperty="Data">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Geometry>F1M-2124.61,-1263.65L-2131.54,-1263.72 -2145.51,-1263.84 -2152.41,
-1263.9C-2155.99,-1263.93,-2157.48,-1262.16,-2155.7,-1259.96L-2152.05,
-1255.43C-2150.28,-1253.23,-2147.38,-1249.62,-2145.61,-1247.42L-2143.25,
-1244.5 -2143.25,-1230.24C-2143.25,-1229.23,-2142.43,-1228.42,-2141.42,
-1228.42L-2135.64,-1228.42C-2134.63,-1228.42,-2133.81,-1229.23,-2133.81,
-1230.24L-2133.81,-1244.78 -2131.7,-1247.3C-2129.89,-1249.47,-2126.93,-
1253.02,
-2125.12,-1255.18L-2121.39,-1259.65C-2119.57,-1261.82,-2121.02,-1263.62,-
2124.61,-1263.65z
</Geometry>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
<ColorAnimation Duration="0:0:0:1"
Storyboard.TargetName="PathFillColor"
Storyboard.TargetProperty="Color"
To="Gray" />
</Storyboard>
</VisualState>
</VisualStateManager>
</VisualStateManager.VisualStateGroups>
<Border Width="{TemplateBinding Width}"
Height="{TemplateBinding Height}"
Background="{TemplateBinding Background}"
SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}">
<Path Name="PART_FilterToggleButtonIndicator"
Margin="3"
Data="F1M-2124.61,-1263.65L-2131.54,-1263.72 -2145.51,-1263.84 -2152.41,
-1263.9C-2155.99,-1263.93,-2157.48,-1262.16,-2155.7,
-1259.96L-2152.05,-1255.43C-2150.28,-1253.23,-2147.38,
-1249.62,-2145.61,-1247.42L-2143.25,-1244.5 -2143.25,

```

```

-1230.24C-2143.25,-1229.23,-2142.43,-1228.42,-2141.42,
-1228.42L-2135.64,-1228.42C-2134.63,-1228.42,-2133.81,
-1229.23,-2133.81,-1230.24L-2133.81,-1244.78 -2131.7,
-1247.3C-2129.89,-1249.47,-2126.93,-1253.02,-2125.12,
-1255.18L-2121.39,-1259.65C-2119.57,-1261.82,-2121.02,
-1263.62,-2124.61,-1263.65z"
SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}"
Stretch="Fill">
<Path.Fill>
<SolidColorBrush x:Name="PathFillColor"
Color="Gray" />
</Path.Fill>
</Path>
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

First Name 	Last Name 	Employee ID 	Date of Joining 	Contact Number 	City 
 Rutherford	Fillmore	0	11/15/2010	999116	Perth
 Abraham	Reagan	8	11/6/2011	999118	San Francisco
Benjamin	Kennedy	22	5/17/2008	999118	Madrid
 Abraham	Adams	29	1/11/2008	999115	San Francisco
 Chester	Fillmore	43	1/10/2009	999117	San Francisco
 George	Jones	44	9/18/2008	999112	Zurich
 Gerald	Bush	65	5/16/2010	999117	San Francisco
 James	Pierce	86	9/11/2010	999115	Adelaide
 Benjamin	Johnson	94	7/21/2010	999118	Cardiff
 Andrew	Clinton	101	4/24/2008	999112	London
 Andrew	McKinley	122	1/4/2010	999116	Perth
 Rutherford	Van Buren	129	8/3/2008	999116	Madrid
 Larry	Taft	137	10/10/2010	999117	Brisbane
 Chester	Nixon	158	1/1/2011	999112	NewYork
 John	Ford	165	7/17/2009	999116	Durban
 William	Smith	172	5/25/2009	999114	London
 Barack	Bush	173	7/5/2010	999111	Cardiff
John	Wilson	187	8/1/2009	999118	Barcelona
 Gerald	Bush	194	9/27/2011	999117	Manchester
 George		208	4/22/2008	999113	NewYork

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Editing in WPF TreeGrid (SfTreeGrid)

SfTreeGrid provides support for editing and it can be enabled or disabled by setting [SfTreeGrid.AllowEditing]() property.

XML

```
<syncfusion:SfTreeGrid x:Name="treeGrid"
    AllowEditing="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding EmployeeDetails}" />
```

C#

```
this.treeGrid.AllowEditing = true;
```

You can enable or disable editing for particular column by setting [TreeGridColumn.AllowEditing](#) property.

XML

```
<syncfusion:TreeGridColumn AllowEditing="True"
    HeaderText="First Name"
    MappingName="FirstName" />
```

C#

```
this.treeGrid.Columns["FirstName"].AllowEditing = true;
```

Note: TreeGridColumn.AllowEditing takes higher priority than SfTreeGrid.AllowEditing

First Name	Last Name	Id	Salary
<input type="checkbox"/> Chester	Buchanan	0	\$100000.00
<input type="checkbox"/> Abraham	Fillmore	1	\$100000.00
<input type="checkbox"/> Rutherford	Clinton	8	\$50000.00
<input type="checkbox"/> Chester	Polk	15	\$50000.00
<input type="checkbox"/> Chester	Monroe	22	\$90000.00
<input type="checkbox"/> George	Truman	29	\$50000.00
<input type="checkbox"/> Warner	Roosevelt	36	\$40000.00
<input type="checkbox"/> Dwight	Nixon	43	\$80000.00
<input type="checkbox"/> Grover	Harrison	44	\$90000.00
<input type="checkbox"/> Warren	Harrison	51	\$90000.00

Note: It is mandatory to set the NavigationMode to Cell to enable CurrentCell navigation and editing.

Entering into edit mode

You can enter into edit mode by pressing F2 key or clicking (touch also supported) the cell. You can allow users to edit the cell in single click (OnTap) or double click (OnDoubleTap) by setting [SfTreeGrid.EditTrigger](#) property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
EditTrigger="OnTap"
AllowEditing="True"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding EmployeeInfo}"
ParentPropertyName="ID"/>
```

C#

```
this.treeGrid.EditTrigger = EditTrigger.OnTap;
```

Cursor placement

When the cell enters into edit mode, cursor is placed based on [SfTreeGrid.EditorSelectionBehavior](#) property.

SelectAll^o selects the text of edit element loaded inside cell.

MoveLast^o places the cursor at the last of edit element loaded inside cell.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowEditing="True"
EditorSelectionBehavior="SelectAll"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="ReportsTo"
EditTrigger="OnTap"
ParentPropertyName="ID"
ItemsSource="{Binding EmployeeInfo}" />
```

C#

```
this.treeGrid.EditorSelectionBehavior = EditorSelectionBehavior.SelectAll;
```

Support for IEditableObject

SfTreeGrid supports to commit and roll back the changes in row level when underlying data object implements [IEditableObject](#) interface.

The editing changes in a row will be committed only when user move to next row or pressing enter key in [EndEdit](#). Also when user press Esc key, then the changes made in a row will be reverted in [CancelEdit](#).

[IEditableObject](#) has the following methods to capture editing,

[BeginEdit](#) - Gets called to begin edit on underlying data object when cell in a row get into edit mode.

CancelEdit - Gets called when user press the Esc key to discard the changes in a row since last **BeginEdit** call.

EndEdit - Gets called when user move to the next row or press Enter key to commit changes in underlying data object since last **BeginEdit** call.

In the below code snippet explains the simple implementation of **IEditableObject**.

C#

```
public class EmployeeInfo : IEditableObject, INotifyPropertyChanged
{
    int _id;
    /// <summary>
    /// Gets or sets the ID.
    /// </summary>
    /// <value>The ID.</value>
    public int ID
    {
        get
        {
            return _id;
        }
        set
        {
            _id = value;
            RaisePropertyChanged("ID");
        }
    }
    string _firstName;
    /// <summary>
    /// Gets or sets the first name.
    /// </summary>
    /// <value>The first name.</value>
    public string FirstName
    {
        get { return _firstName; }
        set
        {
            _firstName = value;
            RaisePropertyChanged("FirstName");
        }
    }
    string _lastName;
    /// <summary>
    /// Gets or sets the last name.
    /// </summary>
    /// <value>The last name.</value>
    public string LastName
    {
        get { return _lastName; }
        set
        {
            _lastName = value;
            RaisePropertyChanged("LastName");
        }
    }
}
```

```
private string _title;
/// <summary>
/// Gets or sets the title.
/// </summary>
/// <value>The title.</value>
public string Title
{
    get
    {
        return _title;
    }
    set
    {
        _title = value;
        RaisePropertyChanged("Title");
    }
}
double? _salary;
/// <summary>
/// Gets or sets the salary.
/// </summary>
/// <value>The salary.</value>
public double? Salary
{
    get
    {
        return _salary;
    }
    set
    {
        _salary = value;
        RaisePropertyChanged("Salary");
    }
}
int _reportsTo;
/// <summary>
/// Gets or sets the reports to.
/// </summary>
/// <value>The reports to.</value>
public int ReportsTo
{
    get
    {
        return _reportsTo;
    }
    set
    {
        _reportsTo = value;
        RaisePropertyChanged("ReportsTo");
    }
}
protected Dictionary<string, object> BackUp()
{
    var dictionary = new Dictionary<string, object>();
    var itemProperties = this.GetType().GetTypeInfo().DeclaredProperties;
    foreach (var pDescriptor in itemProperties)
    {

```

```

if (pDescriptor.CanWrite)
dictionary.Add(pDescriptor.Name, pDescriptor.GetValue(this));
}
return dictionary;
}
private Dictionary<string, object> storedValues;
public void BeginEdit()
{
this.storedValues = this.BackUp();
}
public void CancelEdit()
{
if (this.storedValues == null)
return;
foreach (var item in this.storedValues)
{
var itemProperties = this.GetType().GetTypeInfo().DeclaredProperties;
var pDesc = itemProperties.FirstOrDefault(p => p.Name == item.Key);
if (pDesc != null)
pDesc.SetValue(this, item.Value);
}
}
public void EndEdit()
{
if (this.storedValues != null)
{
this.storedValues.Clear();
this.storedValues = null;
}
}
public event PropertyChangedEventHandler PropertyChanged;
public void RaisePropertyChanged(string propertyName)
{
if (PropertyChanged != null)
PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
}

```

Events

SfTreeGrid triggers the following events during editing.

CurrentCellBeginEdit Event

[CurrentCellBeginEdit](#) event occurs when the [CurrentCell](#) enter into edit mode.

[TreeCurrentCellBeginEditEventArgs](#) has following members which provides information for [CurrentCellBeginEdit](#) event.

- [Cancel](#) : When set to `true`, the event is canceled and the [CurrentCell](#) does not enter into the edit mode.
- [RowColumnIndex](#) : Gets the current row column index of the TreeGrid.
- [Column](#) : Gets the Tree Grid Column of the SfTreeGrid.

C#

```

this.treeGrid.CurrentCellBeginEdit += TreeGrid_CurrentCellBeginEdit;

```

```
void TreeGrid_CurrentCellBeginEdit(object sender,
TreeGridCurrentCellBeginEditEventArgs args)
{
}
```

CurrentCellEndEdit Event

[CurrentCellEndEdit](#) event occurs when the [CurrentCell](#) exits the edit mode.

[CurrentCellEndEditEventArgs]() has following members which provides information for [CurrentCellEndEdit](#) event.

- [RowColumnIndex](#) : Gets the value for the current row column index.

C#

```
this.treeGrid.CurrentCellEndEdit += TreeGrid_CurrentCellEndEdit;
void TreeGrid_CurrentCellEndEdit(object sender, CurrentCellEndEditEventArgs
args)
{
}
```

CurrentCellValueChanged Event

[CurrentCellValueChanged](#) event occurs whenever a value changes in TreeGridColumn that supports editing.

[TreeGridCurrentCellValueChangedEventArgs](#) has following members which provides information for [CurrentCellValueChanged](#) event.

- [Column](#) : Gets the Grid Column of the SfTreeGrid.
- [RowColumnIndex](#) : Gets the value of the current RowColumnIndex.

C#

```
this.treeGrid.CurrentCellValueChanged += TreeGrid_CurrentCellValueChanged;
void TreeGrid_CurrentCellValueChanged(object sender,
TreeGridCurrentCellValueChangedEventArgs args)
{
}
```

Note: For TreeGridComboBoxColumn, you have to use the 'CurrentCellDropDownSelectionChanged' event.

CurrentCellDropDownSelectionChanged Event

[CurrentCellDropDownSelectionChanged](#) event occurs whenever the [SelectedItem](#) of [TreeGridComboBoxColumn](#) column changed.

[CurrentCellDropDownSelectionChangedEventArgs](#) has following members which provides information for [CurrentCellDropDownSelectionChanged](#) event.

- [RowColumnIndex](#) - Gets the RowColumnIndex of the corresponding item that were selected from the drop-down control.

- [SelectedIndex](#) - Gets the index of the corresponding item that were selected from the drop-down control.
- [SelectedItem](#) - Gets the data item that were selected from the drop-down control.

C#

```
this.treeGrid.CurrentCellDropDownSelectionChanged +=
TreeGrid_CurrentCellDropDownSelectionChanged;
void TreeGrid_CurrentCellDropDownSelectionChanged(object sender,
CurrentCellDropDownSelectionChangedEventArgs args)
{
}
```

Programmatically edit the cell

BeginEdit

SfTreeGrid allows you to edit the cell programmatically by calling the [BeginEdit](#) method. Initially the [CurrentCell](#) need to set before calling the [BeginEdit](#) method when the CurrentCell value is null.

C#

```
this.treeGrid.Loaded += TreeGrid_Loaded;
void TreeGrid_Loaded(object sender, RoutedEventArgs e)
{
    RowColumnIndex rowColumnIndex = new RowColumnIndex(2, 3);
    treeGrid.SelectionController.MoveCurrentCell(rowColumnIndex);
    treeGrid.SelectionController.CurrentCellManager.BeginEdit();
}
```

EndEdit

You can call the [EndEdit](#) method to programmatically end edit.

C#

```
this.treeGrid.Loaded += TreeGrid_Loaded;
void TreeGrid_Loaded(object sender, RoutedEventArgs e)
{
    RowColumnIndex rowColumnIndex = new RowColumnIndex(2, 3);
    treeGrid.SelectionController.MoveCurrentCell(rowColumnIndex);
    treeGrid.SelectionController.CurrentCellManager.EndEdit();
}
```

CancelEdit

You can use the [CurrentCellBeginEdit](#) event to cancel the editing operation for the corresponding cell.

C#

```
this.treeGrid.CurrentCellBeginEdit += TreeGrid_CurrentCellBeginEdit;
void TreeGrid_CurrentCellBeginEdit(object sender,
TreeGridCurrentCellBeginEditEventArgs args)
{
    var mappingName =
    treeGrid.Columns[args.RowColumnIndex.ColumnIndex].MappingName;
    var node = treeGrid.View.GetNodeAt(args.RowColumnIndex.RowIndex);
    if (args.RowColumnIndex == new RowColumnIndex(2, 2))
```

```
args.Cancel = true;  
}
```

ReadOnly

You can prevent users from modifying the contents of a treegrid cell by setting the [SfTreeGrid.IsReadOnly](#) property, but the user can able to perform copy and selection operation.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"  
AllowEditing="True"  
IsReadOnly="True"  
AutoGenerateColumns="True"  
ItemsSource="{Binding EmployeeDetails}"/>
```

C#

```
this.treeGrid.IsReadOnly = true;
```

You can enable or disable editing for particular column by setting [TreeGridColumn.IsReadOnly](#) property.

XML

```
<syncfusion:TreeGridTextColumn HeaderText="Order ID"  
MappingName="OrderID"  
IsReadOnly="True"/>
```

C#

```
this.treeGrid.Columns["OrderID"].IsReadOnly = true;
```

Note: We should set the AllowEditing property to achieve the IsReadOnly behavior.

[TreeGridColumn.IsReadOnly](#) takes higher priority than [SfTreeGrid.IsReadOnly](#).

Mouse and Keyboard operations for UIElement inside Template

You can directly load edit element using [TreeGridTemplateColumn.CellTemplate](#) property. In this case, you can provide focus and control to the UIElement inside CellTemplate in the below ways,

Providing focus to the control inside the Template

You can focus to the particular [UIElement](#) loaded inside template when cell gets activated by setting [FocusedManager.FocusedElement](#) attached property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"  
AutoGenerateColumns="False"  
ColumnSizer="Star"  
AllowEditing="True"  
AutoExpandMode="RootNodesExpanded"  
ChildPropertyName="Children"  
ItemsSource="{Binding EmployeeDetails}">  
<syncfusion:SfTreeGrid.Columns>
```

```

<syncfusion:TreeGridTemplateColumn HeaderText="First Name"
MappingName="FirstName" >
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<TextBlock Syncfusion:FocusManagerHelper.FocusedElement="True"
FontStyle="Italic"
FontWeight="SemiBold"
Padding="2,0"
Text="{Binding FirstName}" />
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.CellTemplate>
</syncfusion:TreeGridTemplateColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

First Name	Last Name	Id	Salary
<input type="checkbox"/> <i>Chester</i>	Buchanan	0	\$100000.00
<input type="checkbox"/> <i>Abraham</i>	Fillmore	1	\$100000.00
<input type="checkbox"/> <i>Rutherford</i>	Clinton	8	\$50000.00
<input type="checkbox"/> <i>Chester</i>	Polk	15	\$50000.00
<input type="checkbox"/> <i>Chester</i>	Monroe	22	\$90000.00
<input type="checkbox"/> <i>George</i>	Truman	29	\$50000.00
<input type="checkbox"/> <i>Warner</i>	Roosevelt	36	\$40000.00
<input type="checkbox"/> <i>Dwight</i>	Nixon	43	\$80000.00
<input type="checkbox"/> <i>Grover</i>	Harrison	44	\$90000.00
<input type="checkbox"/> <i>Warren</i>	Harrison	51	\$90000.00

Providing keyboard control to UIElement inside CellTemplate

You can allow UIElement loaded inside CellTemplate to handle keyboard interaction by setting [FocusManagerHelper.WantsKeyInput](#) attached property to `TreeGridColumn`.

XML

```

<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
AllowEditing="True"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTemplateColumn HeaderText="First Name"
MappingName="FirstName"
syncfusion:FocusManagerHelper.WantsKeyInput="True">
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<Grid>
<TextBox x:Name="text" Text="{Binding FirstName}"/>
</Grid>
</DataTemplate>

```



```

</syncfusion:TreeGridTemplateColumn.CellTemplate>
</syncfusion:TreeGridTemplateColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

	First Name	Last Name	Id	Salary
[-]	Chester	Buchanan	0	\$100000.00
[+]	Abraham	Fillmore	1	\$100000.00
[+]	Rutherford	Clinton	8	\$50000.00
[+]	Chester	Polk	15	\$50000.00
[+]	Chester	Monroe	22	\$90000.00
[+]	George	Truman	29	\$50000.00
[+]	Warner	Roosevelt	36	\$40000.00
[-]	Dwight	Nixon	43	\$80000.00
[+]	Grover	Harrison	44	\$90000.00
[+]	Warren	Harrison	51	\$90000.00

Note: Enter and Tab keys are always handled by SfTreeGrid only.

Providing mouse control to UIElement inside template

You can allow **UIElement** loaded inside template to handle mouse interaction in required cases by setting [VisualContainer.WantsMouseInput](#) attached property to **TreeGridColumn**.

XML

```

<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
AllowEditing="True"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}">
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTemplateColumn HeaderText="City" MappingName="City" >
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<ComboBox ItemsSource="{Binding CityCollection, Source={StaticResource
viewModel}}">
syncfusion:VisualContainer.WantsMouseInput="True"/>
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.CellTemplate>
</syncfusion:TreeGridTemplateColumn>
</syncfusion:SfTreeGrid.Columns>
</syncfusion:SfTreeGrid>

```

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Selection in WPF TreeGrid (SfTreeGrid)

SfTreeGrid allows you select one or more rows.

Current cell navigation

Keyboard navigation through the rows is determined based on the [NavigationMode](#) property.

[NavigationMode.Cell](#) allows you navigate between the cells in a row and navigate between rows.

[NavigationMode.Row](#) allows you navigate only between rows.

Selection modes

The [SelectionMode](#) property defines the behavior of selection in tree grid. If SelectionMode is [Single](#), you can select a single row, and if the SelectionMode is Extended or Multiple, you can select multiple rows. If you want to disable the selection, set SelectionMode to None.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
NavigationMode="Row"
ChildPropertyName="Children"
SelectionMode="Single"
ColumnSizer="Star"
ExpanderColumn="FirstName"
ItemsSource="{Binding PersonDetails}"
>
```

First Name	Id	Last Name	Availability	DOB
<input type="checkbox"/> Chester	0	Buchanan	<input checked="" type="checkbox"/>	3/3/1998
<input checked="" type="checkbox"/> Abraham	1	Fillmore	<input checked="" type="checkbox"/>	2/18/2003
<input checked="" type="checkbox"/> Martin	157	Lincoln	<input checked="" type="checkbox"/>	5/26/2005
<input checked="" type="checkbox"/> Gerald	313	Arthur	<input checked="" type="checkbox"/>	7/9/1999
<input checked="" type="checkbox"/> Herbert	469	Reagan	<input checked="" type="checkbox"/>	1/15/1997
<input checked="" type="checkbox"/> Theodore	625	Madison	<input checked="" type="checkbox"/>	6/14/2007
<input type="checkbox"/> George	781	Madison	<input checked="" type="checkbox"/>	6/7/2006
<input checked="" type="checkbox"/> George	782	Harding	<input checked="" type="checkbox"/>	12/21/2004
<input checked="" type="checkbox"/> Benjamin	938	Jackson	<input checked="" type="checkbox"/>	6/16/1997
<input checked="" type="checkbox"/> Barack	1094	Harrison	<input checked="" type="checkbox"/>	2/27/2003
<input checked="" type="checkbox"/> Ulysses	1250	Cleveland	<input checked="" type="checkbox"/>	7/25/2002
<input checked="" type="checkbox"/> James	1406	Clinton	<input checked="" type="checkbox"/>	2/27/2001
<input type="checkbox"/> Zachary	1562	Buchanan	<input checked="" type="checkbox"/>	5/11/1998
<input checked="" type="checkbox"/> Herbert	1563	Carter	<input checked="" type="checkbox"/>	6/30/2004

Disable selection for rows and columns

You can disable selection and navigation on particular column by setting the [GridColumn.AllowFocus](#) property. You can disable selection on particular row or column by handling the [CurrentCellActivating](#) event.

Multiple row selection

The tree grid allows you select multiple rows by setting the [SelectionMode](#) property to [Extended](#) or [Multiple](#), where you can select multiple rows by dragging the mouse on tree grid and also using the key modifiers.

When using Extended, you can select multiple rows by pressing the key modifiers Ctrl and Shift.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
NavigationMode="Row"
ChildPropertyName="Children"
SelectionMode="Extended"
ColumnSizer="Star"
ExpanderColumn="FirstName"
ItemsSource="{Binding PersonDetails}"
>
```

First Name	Id	Last Name	Availability	DOB
<input type="checkbox"/> Chester	0	Buchanan	<input checked="" type="checkbox"/>	3/3/1998
<input checked="" type="checkbox"/> Abraham	1	Fillmore	<input checked="" type="checkbox"/>	2/18/2003
<input checked="" type="checkbox"/> Martin	157	Lincoln	<input checked="" type="checkbox"/>	5/26/2005
<input checked="" type="checkbox"/> Gerald	313	Arthur	<input checked="" type="checkbox"/>	7/9/1999
<input checked="" type="checkbox"/> Herbert	469	Reagan	<input checked="" type="checkbox"/>	1/15/1997
<input checked="" type="checkbox"/> Theodore	625	Madison	<input checked="" type="checkbox"/>	6/14/2007
<input type="checkbox"/> George	781	Madison	<input checked="" type="checkbox"/>	6/7/2006
<input checked="" type="checkbox"/> George	782	Harding	<input checked="" type="checkbox"/>	12/21/2004
<input checked="" type="checkbox"/> Benjamin	938	Jackson	<input checked="" type="checkbox"/>	6/16/1997
<input checked="" type="checkbox"/> Barack	1094	Harrison	<input checked="" type="checkbox"/>	2/27/2003
<input checked="" type="checkbox"/> Ulysses	1250	Cleveland	<input checked="" type="checkbox"/>	7/25/2002
<input checked="" type="checkbox"/> James	1406	Clinton	<input checked="" type="checkbox"/>	2/27/2001
<input type="checkbox"/> Zachary	1562	Buchanan	<input checked="" type="checkbox"/>	5/11/1998
<input checked="" type="checkbox"/> Herbert	1563	Carter	<input checked="" type="checkbox"/>	6/30/2004
<input checked="" type="checkbox"/> Chester	1719	Fillmore	<input checked="" type="checkbox"/>	11/18/1997

Note: When [SelectionMode](#) is [Multiple](#), you can select or deselect multiple rows by clicking the respective row. In multiple selection, pressing the navigation keys moves only the current cell, and you can select or deselect by pressing the Space key.

Get selected rows

The [SelectedItem](#) property returns the data object of the selected row, and the [SelectedIndex](#) property returns the index of the SelectedItem in tree grid. The SelectedItem denotes the first selected row in multiple selection.

The [CurrentItem](#) returns the data object that currently has focus, and the [CurrentColumn](#) denotes the [GridColumn](#) that currently has focus.

The [CurrentCellInfo](#) returns an instance [GridCellInfo](#), which contains information about the cell that currently has focus.

Row selection

You can get all the selected records using the [SelectedItems](#) property, and you can also get all the selected rows' information using [SfTreeGrid.SelectionController.SelectedRows](#), which is a collection of [GridRowInfo](#).

CurrentItem vs SelectedItem

Both [SelectedItem](#) and [CurrentItem](#) return the same data object when a single row is selected in tree grid. When you select more than one rows or cells, the record that had been selected initially is maintained in [SelectedItem](#), and the record that currently has focus is maintained in [CurrentItem](#).

Programmatic selection

Process selection using properties

You can select a single row by setting the [SelectedItem](#) property or [SelectedIndex](#) property.

C#

```
var recordIndex = this.treeGrid.ResolveToNodeIndex(6);  
this.treeGrid.SelectedIndex = recordIndex;
```

C#

```
var node = this.treeGrid.GetNodeAtRowIndex(6);  
this.treeGrid.SelectedItem = node.Item;
```

In row selection, you can select multiple rows by adding data objects to the [SelectedItems](#) property.

C#

```
var viewModel = this.treeGrid.DataContext as ViewModel;  
foreach (var order in viewModel.PersonDetails)  
{  
    if (order.LastName == "Buchanan")  
        this.treeGrid.SelectedItems.Add(order);  
}
```

First Name	Id	Last Name	Availability	DOB
<input type="checkbox"/> Chester	0	Buchanan	<input checked="" type="checkbox"/>	3/3/1998
<input type="checkbox"/> Abraham	1	Fillmore	<input checked="" type="checkbox"/>	2/18/2003
<input type="checkbox"/> Martin	157	Lincoln	<input checked="" type="checkbox"/>	5/26/2005
<input type="checkbox"/> Gerald	313	Arthur	<input checked="" type="checkbox"/>	7/9/1999
<input type="checkbox"/> Herbert	469	Reagan	<input checked="" type="checkbox"/>	1/15/1997
<input type="checkbox"/> Theodore	625	Madison	<input checked="" type="checkbox"/>	6/14/2007
<input type="checkbox"/> George	781	Madison	<input checked="" type="checkbox"/>	6/7/2006
<input type="checkbox"/> George	782	Harding	<input checked="" type="checkbox"/>	12/21/2004
<input type="checkbox"/> Benjamin	938	Jackson	<input checked="" type="checkbox"/>	6/16/1997
<input type="checkbox"/> Barack	1094	Harrison	<input checked="" type="checkbox"/>	2/27/2003
<input type="checkbox"/> Ulysses	1250	Cleveland	<input checked="" type="checkbox"/>	7/25/2002
<input type="checkbox"/> James	1406	Clinton	<input checked="" type="checkbox"/>	2/27/2001
<input type="checkbox"/> Zachary	1562	Buchanan	<input checked="" type="checkbox"/>	5/11/1998
<input type="checkbox"/> Herbert	1563	Carter	<input checked="" type="checkbox"/>	6/30/2004

Process selection using methods

You can select a range of rows using the [SelectRows](#) method in row selection.

C#

```
this.treeGrid.SelectRows(3, 7);
```

First Name	Id	Last Name	Availability	DOB
<input type="checkbox"/> Chester	0	Buchanan	<input checked="" type="checkbox"/>	3/3/1998
<input type="checkbox"/> Abraham	1	Fillmore	<input checked="" type="checkbox"/>	2/18/2003
<input type="checkbox"/> Martin	157	Lincoln	<input checked="" type="checkbox"/>	5/26/2005
<input type="checkbox"/> Gerald	313	Arthur	<input checked="" type="checkbox"/>	7/9/1999
<input type="checkbox"/> Herbert	469	Reagan	<input checked="" type="checkbox"/>	1/15/1997
<input type="checkbox"/> Theodore	625	Madison	<input checked="" type="checkbox"/>	6/14/2007
<input type="checkbox"/> George	781	Madison	<input checked="" type="checkbox"/>	6/7/2006
<input type="checkbox"/> George	782	Harding	<input checked="" type="checkbox"/>	12/21/2004
<input type="checkbox"/> Benjamin	938	Jackson	<input checked="" type="checkbox"/>	6/16/1997
<input type="checkbox"/> Barack	1094	Harrison	<input checked="" type="checkbox"/>	2/27/2003
<input type="checkbox"/> Ulysses	1250	Cleveland	<input checked="" type="checkbox"/>	7/25/2002
<input type="checkbox"/> James	1406	Clinton	<input checked="" type="checkbox"/>	2/27/2001
<input type="checkbox"/> Zachary	1562	Buchanan	<input checked="" type="checkbox"/>	5/11/1998
<input type="checkbox"/> Herbert	1563	Carter	<input checked="" type="checkbox"/>	6/30/2004

Process current cell

When you set [CurrentItem](#) to a particular record, the [CurrentCell](#) will be moved to the corresponding record. When [SelectionMode](#) is [Multiple](#) or [Extended](#), the selection will added to a particular record item. When [SelectionMode](#) is [Single](#), the selection will added to a single cell.

You can move the [CurrentCell](#) to a particular rowColumnIndex using the [MoveCurrentCell](#) method.

C#

```
this.treeGrid.SelectionController.MoveCurrentCell(new RowColumnIndex(3, 2), false);
```

Clear selection

You can clear the selection using the [ClearSelections](#) method. In row selection, you can remove the selection by setting null to the [SelectedItem](#) or clearing the [SelectedItems](#) property.

C#

```
this.treeGrid.SelectionController.ClearSelections(true);
```

Scrolling rows and columns

Automatic scrolling on drag selection

SfTreeGrid scrolls rows and columns automatically when you try to perform the drag selection like in Excel. You can enable or disable AutoScrolling by setting the [AutoScroller.AutoScrolling](#) property.

C#

```
this.treeGrid.AutoScroller.AutoScrolling = AutoScrollOrientation.Both;
```

Scroll to a particular row or column index

You can scroll programmatically to particular cell using the [ScrollInView](#) method by passing row and column index.

C#

```
int rowIndex = this.treeGrid.GetLastDataRowIndex();
int columnIndex = this.treeGrid.GetLastColumnIndex();
this.treeGrid.ScrollInView(new RowColumnIndex(rowIndex, columnIndex));
```

Scroll to selected item

You can scroll programmatically to the SelectedItem using the ScrollInView method.

C#

```
var rowIndex = this.treeGrid.ResolveToRowIndex(this.treeGrid.SelectedItem);
var columnIndex = this.treeGrid.ResolveToStartColumnIndex();
this.treeGrid.ScrollInView(new RowColumnIndex(rowIndex, columnIndex));
```

Mouse and keyboard behaviors

Keyboard behavior

Key or Key combinations	Description
DownArrow	Moves CurrentCell directly below the active current cell. If the CurrentCell is in the last row, pressing the Down arrow does nothing.

UpArrow	Moves the CurrentCell directly above the active current cell. If the CurrentCell is in the first row, pressing the Up arrow does nothing.
LeftArrow	Moves the current cell to previous to the active current cell. If the CurrentCell is in the first cell, pressing the Left arrow does nothing. If the focused row is group header, the group will be collapsed when it is in expanded state.
RightArrow	Moves the current cell to next to the active current cell. If the CurrentCell is in the last cell, pressing the Right arrow does nothing. If the focused row is group header, the group will be expanded when it is in collapsed state.
Home / Ctrl + LeftArrow	Moves the current cell to the first cell of the current row.
End / Ctrl + RightArrow	Moves the current cell to the last cell of the current row.
PageDown	The tree grid will be scrolled to the next set of rows that is not displayed in view, including the row that is partially displayed, and the current cell is set to the last row.
PageUp	The tree grid will be scrolled to the previous set of rows that is not displayed in view, including the row that is partially displayed, and the current cell is set to the first row.
Tab	Moves the current cell to next to the active current cell. If the active current cell is the last cell of the current row, the focus will be moved to the first cell of the row next to the current row. If the active current cell is the last cell of the last row, the focus will be moved to next control in the tab order of the parent container.
Shift + Tab	Moves the current cell to previous cell to the active current cell. If the active current cell is the first cell of the current row, the current cell will be moved to the last cell of the row previous to the current row. If the active current cell is the first cell of the first row, the focus will be moved to the previous control in the tab order of the parent container.
Ctrl + DownArrow	Moves the current cell to the current column of the last row.

Ctrl+UpArrow	Moves the current cell to the current column of the first row.
Ctrl+Home	Moves the current cell to the first cell of the first row.
Ctrl+End	Moves the current cell to the last cell of the last row.
Enter	If the active current cell is in edit mode, the changes will be committed, and moves the current cell to below the active current cell. If the active current cell is in the last row, commits changes only and retains the same cell.
Ctrl+Enter	Commits only the changes when the current cell is in edit mode and retains the focus in same cell.
F2	If the TreeGrid property is true, and the GridColumn.AllowEditing property is true for the current column, the current cell enters into edit mode.
Esc	If the current cell is in edit mode, reverts the changes that had been done in the current cell. If the underlying source implements IEditableObject , clicking the Esc key for the second time cancels the edit mode for entire row.
Ctrl+A	All rows or cells will be selected.

Shift key combinations

When [SelectionMode](#) is set to [Extended](#), you can select multiple rows using the navigation keys along with the Shift key. Before starting navigation, the current cell will be marked as a pressed cell, and the selection will be done in all rows between the pressed cell and current cell.

Key combinations
Shift + DownArrow
Shift + UpArrow
Shift + LeftArrow
Shift + Home
Shift + End
Shift + PageDown
Shift + PageUp

Shift + Ctrl + DownArrow
Shift + Ctrl + UpArrow
Shift + Ctrl + RightArrow
Shift + Ctrl + LeftArrow
Shift + Ctrl + Home
Shift + Ctrl + End
Shift + Ctrl + PageDown
Shift + Ctrl + PageUp

Mouse behavior

You can enable or disable the selection when the mouse button is in the pressed state by setting the [AllowSelectionOnPointerPressed](#) property.

When [SelectionMode](#) is set to [Extended](#), you can select multiple rows by clicking any cell along with **ctrl** and **Shift** keys. When you click a cell along with **Ctrl** key, you can select or deselect a particular row. When you click a cell along with **Shift** key, you can select the range rows from the pressed cell to the current cell.

Customize mouse and keyboard behaviors

You can customize the mouse and keyboard behaviors by overriding the selection controller. Refer to the Customizing Selection Behaviors section to learn about override the selection controller.

Events

CurrentCellActivating

[ActivationTrigger](#): Returns the reason for moving the current cell.

[CurrentRowColumnIndex](#): [RowColumnIndex](#) of the cell where the current cell need to move.

[PreviousRowColumnIndex](#): [RowColumnIndex](#) of the cell from where the current cell moved.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
Grid.Row="0"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
CurrentCellActivating="TreeGrid_CurrentCellActivating"
ItemsSource="{Binding EmployeeInfo}"
LiveNodeUpdateMode="AllowDataShaping"
Converter="{StaticResource SelectionModeConverter}">
```

C#

```
this.treeGrid.CurrentCellActivating += TreeGrid_CurrentCellActivating;
private void TreeGrid_CurrentCellActivating(object sender,
Syncfusion.UI.Xaml.Grid.CurrentCellActivatingEventArgs e)
{
```

```
}

```

You can cancel the current cell moving process within this event by setting [GridCurrentCellActivatingEventArgs.Cancel](#) to true.

C#

```
private void TreeGrid_CurrentCellActivating(object sender,
Syncfusion.UI.Xaml.Grid.CurrentCellActivatingEventArgs e)
{
    var provider = this.treeGrid.View.GetPropertyAccessProvider();
    var record =
    this.treeGrid.GetNodeAtRowIndex(e.CurrentRowColumnIndex.RowIndex).Item;
    if (record == null)
        return;
    var column =
    this.treeGrid.Columns[this.treeGrid.ResolveToGridVisibleColumnIndex(e.Curren
tRowColumnIndex.ColumnIndex)];
    var cellValue = provider.GetValue(record, column.MappingName);
    if (cellValue.ToString() == "1001")
        e.Cancel = true;
}
```

CurrentCellActivated

The [CurrentCellActivated](#) event occurs after the current cell moves to the corresponding cell. [CurrentCellActivatedEventArgs](#) has the following members, which provide information to the CurrentCellActivated event:

[ActivationTrigger](#): Returns the reason of the current cell movement.

[CurrentRowColumnIndex](#): RowColumnIndex of the cell where the current cell move.

[PreviousRowColumnIndex](#): RowColumnIndex of the cell where the current cell moved.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
Grid.Row="0"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
CurrentCellActivated="TreeGrid_CurrentCellActivated"
ItemsSource="{Binding EmployeeInfo}"
LiveNodeUpdateMode="AllowDataShaping"
Converter="{StaticResource SelectionModeConverter}">
```

C#

```
this.treeGrid.CurrentCellActivated += TreeGrid_CurrentCellActivated;
private void TreeGrid_CurrentCellActivated(object sender,
Syncfusion.UI.Xaml.Grid.CurrentCellActivatedEventArgs e)
{
}
```

SelectionChanging

The [SelectionChanging](#) event occurs before processing the selection to a particular row or cell. This event is triggered only to the keyboard and mouse interactions. [GridSelectionChangingEventArgs](#) has the following members, which provide information to the SelectionChanging event.

[AddedItems](#): Collection of [GridRowInfo](#) or [GridCellInfo](#) where the selection is going to be processed.

[RemovedItems](#): Collection of [GridRowInfo](#) or [GridCellInfo](#) where the selection is going to be removed.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
Grid.Row="0"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
SelectionChanging="TreeGrid_SelectionChanging"
ItemsSource="{Binding EmployeeInfo}"
LiveNodeUpdateMode="AllowDataShaping"
Converter="{StaticResource SelectionModeConverter}">
```

C#

```
this.treeGrid.SelectionChanging += TreeGrid_SelectionChanging;
private void TreeGrid_SelectionChanging(object sender,
Syncfusion.UI.Xaml.Grid.GridSelectionChangingEventArgs e)
{
}
```

SelectionChanged

The [SelectionChanged](#) event occurs after the selection process is completed for a particular row or cell in tree grid. [GridSelectionChangedEventArgs](#) has the following members, which provide information to the SelectionChanged event:

[AddedItems](#): Collection of [GridRowInfo](#) or [GridCellInfo](#) where the selection has been processed.

[RemovedItems](#): Collection of [GridRowInfo](#) or [GridCellInfo](#) where the selection has been removed.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
Grid.Row="0"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="ReportsTo"
SelectionChanged="TreeGrid_SelectionChanged"
ItemsSource="{Binding EmployeeInfo}"
LiveNodeUpdateMode="AllowDataShaping"
Converter="{StaticResource SelectionModeConverter}">
```

C#

```
this.treeGrid.SelectionChanged += TreeGrid_SelectionChanged;
private void TreeGrid_SelectionChanged(object sender,
Syncfusion.UI.Xaml.Grid.GridSelectionChangedEventArgs e)
{
}
```

```
}
```

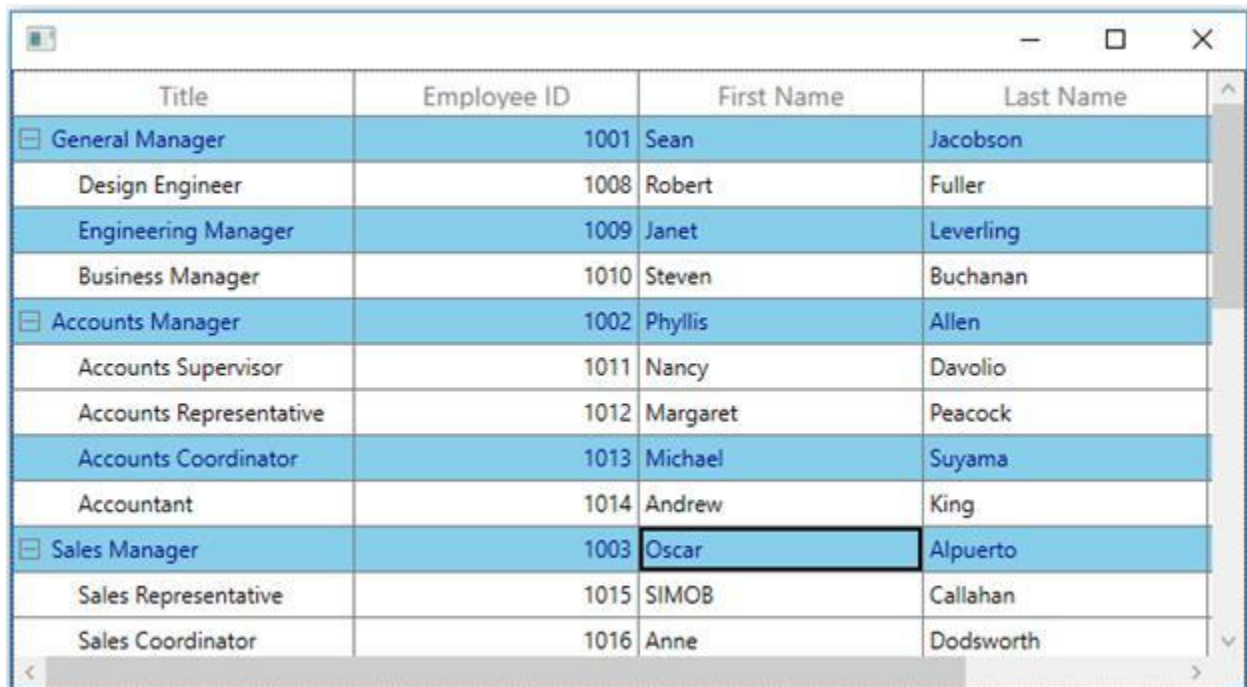
Appearance

Change selection background and foreground

You can change the selection background and foreground using the [SelectionBackGround](#) and [SelectionForeground](#) properties.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
Grid.Row="0"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
SelectionMode="Multiple"
ChildPropertyName="ReportsTo"
SelectionChanged="TreeGrid_SelectionChanged"
ItemsSource="{Binding EmployeeInfo}"
SelectionBackground="SkyBlue"
SelectionForeground="DarkBlue"
>
```



Title	Employee ID	First Name	Last Name
General Manager	1001	Sean	Jacobson
Design Engineer	1008	Robert	Fuller
Engineering Manager	1009	Janet	Leverling
Business Manager	1010	Steven	Buchanan
Accounts Manager	1002	Phyllis	Allen
Accounts Supervisor	1011	Nancy	Davolio
Accounts Representative	1012	Margaret	Peacock
Accounts Coordinator	1013	Michael	Suyama
Accountant	1014	Andrew	King
Sales Manager	1003	Oscar	Alpuerto
Sales Representative	1015	SIMOB	Callahan
Sales Coordinator	1016	Anne	Dodsworth

Change current cell border style

You can change the current cell border thickness and border color using the [CurrentCellBorderThickness](#) and [CurrentCellBorderBrush](#) properties.

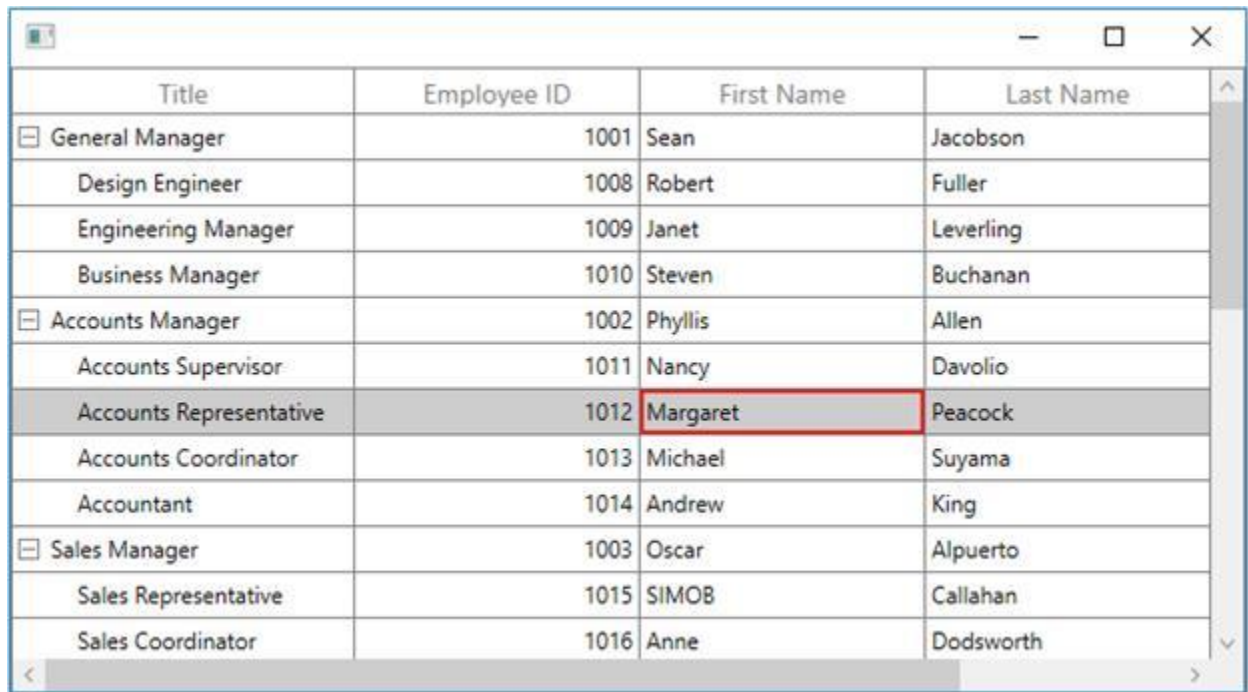
XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
Grid.Row="0"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
```

```

SelectionMode="Multiple"
ChildPropertyName="ReportsTo"
SelectionChanged="TreeGrid_SelectionChanged"
ItemsSource="{Binding EmployeeInfo}"
LiveNodeUpdateMode="AllowDataShaping"
ParentPropertyName="ID"
SelectedIndex="0"
CurrentCellBorderBrush="Red"
CurrentCellBorderThickness="1.6"
>

```



Title	Employee ID	First Name	Last Name
General Manager	1001	Sean	Jacobson
Design Engineer	1008	Robert	Fuller
Engineering Manager	1009	Janet	Leverling
Business Manager	1010	Steven	Buchanan
Accounts Manager	1002	Phyllis	Allen
Accounts Supervisor	1011	Nancy	Davolio
Accounts Representative	1012	Margaret	Peacock
Accounts Coordinator	1013	Michael	Suyama
Accountant	1014	Andrew	King
Sales Manager	1003	Oscar	Alpuerto
Sales Representative	1015	SIMOB	Callahan
Sales Coordinator	1016	Anne	Dodsworth

Customize row selection border

You can customize the row selection by editing the control template of TreeGridRowControl.

XML

```

<Style TargetType="syncfusion:TreeGridRowControl">
  <Setter Property="BorderBrush" Value="{StaticResource ContentBorderBrush}" />
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="syncfusion:TreeGridRowControl">
        <Grid>
          <Border x:Name="PART_BackgroundBorder"
            Margin="{TemplateBinding IndentMargin}"
            Background="{TemplateBinding Background}"
            Visibility="Visible">
            <Rectangle x:Name="PART_FocusRect"
              Margin="1,2,2,2"
              Fill="Transparent"
              Stroke="DarkGray"
              StrokeDashArray="2,2"
              StrokeThickness="1"

```

```

Visibility="Collapsed" />
</Border>
<!-- Adding new border to show border for whole selected row -->
<Border x:Name="PART_SelectionBorder"
Margin="{TemplateBinding IndentMargin}"
Background="{TemplateBinding SelectionBackground}"
Visibility="Collapsed"
BorderBrush="Red"
BorderThickness="1.5,1.5,1.5,2.5"
Opacity="0.75"/>
<ContentPresenter />
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="SelectionStates">
<VisualState x:Name="Unselected" />
<VisualState x:Name="Selected">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_SelectionBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="SelectedPointerOver">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_SelectionBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="SelectedPressed">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_SelectionBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="UnselectedPointerOver" />
<VisualState x:Name="UnselectedPressed" />
<VisualState x:Name="Focused">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_FocusRect"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>

```

```

</VisualStateManager.VisualStateGroups>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

Customize selection behaviors

The tree grid processes the selection operations in selection controller. [GridSelectionController](#) processes selection operations when selection unit is row.

You can customize the default row selection behaviors by overriding the GridSelectionController class and set it to SfTreeGrid.SelectionController.

C#

```

this.treeGrid.SelectionController = new
GridSelectionControllerExt(this.treeGrid);
public class GridSelectionControllerExt : TreeGridRowSelectionController
{
    public GridSelectionControllerExt(SfTreeGrid treeGrid) : base(treeGrid)
    {
    }
}

```

Change enter key behavior

By default, when pressing the **Enter** key, the current cell will be moved to the next focused cell in the same column. You can change the **Enter** key behavior by overriding the [ProcessKeyDown](#) method in selection controller. In this method, you have to create a new [KeyEventArgs](#), which refers to the **Tab** key and processes the **Tab** key action.

C#

```

public class GridSelectionControllerExt : TreeGridRowSelectionController
{
    public GridSelectionControllerExt(SfTreeGrid treeGrid) : base(treeGrid)
    {
    }
    protected override void ProcessKeyDown(KeyEventArgs args)
    {
        if (args.Key == Key.Enter)
        {
            //Creates new KeyEventArgs to refer the Tab key.
            KeyEventArgs arguments = new KeyEventArgs(args.KeyboardDevice,
            args.InputSource, args.Timestamp, Key.Tab) { RoutedEvent = args.RoutedEvent
            };
            //Base ProcessKeyDown is invoked to process Tab key operations.
            base.ProcessKeyDown(arguments);
            args.Handled = arguments.Handled;
            return;
        }
        base.ProcessKeyDown(args);
    }
}

```

Scroll and select the record programmatically

You can scroll to the record programmatically using the [ScrollInView](#) method by passing the row index of the record. You can get the row index of the record using the [ResolveToRowIndex](#) extension method present in [Syncfusion.UI.Xaml.TreeGrid.Helpers](#).

You can select the record programmatically by setting the [SelectedItem](#) property in tree grid.

C#

```
private void Treegrid_Loaded(object sender, System.Windows.RoutedEventArgs e)
{
    var selectedItem = (this.treegrid.DataContext as ViewModel).SelectedItem;
    var rowindex = this.treegrid.ResolveToRowIndex(selectedItem);
    var columnindex = this.treegrid.ResolveToStartColumnIndex();
    //Make the row in to available on the view.
    this.treegrid.ScrollInView(new RowColumnIndex(rowindex, columnindex));
    //Set the SelectedItem in SfTreeGrid.
    this.treegrid.SelectedItem = selectedItem;
}
```

You can [download](#) the sample.

Prevent the selection when right-click

You can prevent the selection when right-clicking in tree grid by customizing the [SelectionController](#) and overriding the [ProcessPointerPressed](#).

C#

```
protected override void ProcessPointerPressed(MouseButtonEventArgs args,
RowColumnIndex rowColumnIndex)
{
    if (args.ChangedButton == MouseButton.Right)
    {
        args.Handled = true;
    }
    else
    {
        base.ProcessPointerPressed(args, rowColumnIndex);
    }
}
```

You can [download](#) the sample.

Change the checkbox column values based on row selection

In tree grid, you can select multiple rows using the [SelectionMode](#) property. You can process the [CheckBoxSelection](#) using [TreeGridCheckBoxColumn](#) and a boolean property called [IsSelected](#) in Model. You can also select all the rows in tree grid by defining the [CheckBox](#) in header cell of [GridCheckBoxColumn](#) using [GridColumn.HeaderTemplate](#).

C#

```
public static class Commands
{
    static Commands ()
    {

```



```
CommandManager.RegisterClassCommandBinding(typeof(CheckBox), new
CommandBinding(CheckAndUnCheck, OnCheckUnCheckCommand,
OnCanExecuteCheckAndUnCheck));
}
public static RoutedCommand CheckAndUnCheck = new
RoutedCommand("CheckAndUnCheck", typeof(CheckBox));
private static void OnCheckUnCheckCommand(object sender,
ExecutedRoutedEventArgs args)
{
var treegrid = (args.Parameter as SfTreeGrid);
var viewmodel = (treegrid.DataContext as ViewModel);
var checkbox = (sender as CheckBox).IsChecked;
if (viewmodel != null)
{
if (checkbox == true)
{
treegrid.SelectAll();
foreach (var collection in viewmodel.EmployeeInfo)
{
if (collection.IsSelected == false)
collection.IsSelected = true;
}
}
else if (checkbox == false)
{
treegrid.ClearSelections(false);
foreach (var collection in viewmodel.EmployeeInfo)
{
if (collection.IsSelected == true)
collection.IsSelected = false;
}
}
}
private static void OnCanExecuteCheckAndUnCheck(object sender,
CanExecuteRoutedEventArgs args)
{
args.CanExecute = true;
}
}
```

You can download the [sample](#).

Select the rows based on cell value

In tree grid, you can select the rows based on cell value by adding the corresponding records to SelectedItems. You can get the cell value of a particular cell using the View.GetPropertyAccess provider method.

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
if (treeGrid.View != null)
reflector = treeGrid.View.GetPropertyAccessProvider();
else
reflector = null;
}
```

```
var totalRowIndex = treeGrid.View.Nodes.Count;
var totalColumnIndex = treeGrid.Columns.Count;
for (int recordIndex = 0; recordIndex < totalRowIndex; recordIndex++)
{
    for (int colindex = 0; colindex < totalColumnIndex; colindex++)
    {
        var record = this.treeGrid.View.Nodes[recordIndex];
        var mappingName = treeGrid.Columns[colindex].MappingName;
        //Get the cell value based on mappingName.
        var currentValue = reflector.GetValue(record.Item, mappingName);
        if (currentCellValue == "Steven")
        {
            object node = treeGrid.View.Nodes[recordIndex];
            //selected rows should be added here.
            treeGrid.SelectedItems.Add((node as TreeNode).Item);
        }
    }
}
```

Search and select the record

You can search and select a record in tree grid based on the searched text using the TextChanged event of TextBox.

C#

```
private void TextBox_TextChanged(object sender,
System.Windows.Controls.TextChangedEventArgs e)
{
    var textBox = sender as TextBox;
    var treeGrid = this.AssociatedObject.treeGrid;
    if (textBox.Text == "")
    treeGrid.SelectedItems.Clear();
    for (int i = 0; i < treeGrid.View.Nodes.Count; i++)
    {
        if (Provider == null)
        Provider = treeGrid.View.GetPropertyAccessProvider();
        if (treeGrid.View.Nodes[i].HasChildNodes &&
            treeGrid.View.Nodes[i].ChildNodes.Count == 0)
        {
            treeGrid.BeginInit();
            treeGrid.ExpandNode(treeGrid.View.Nodes[i]);
            treeGrid.CollapseNode(treeGrid.View.Nodes[i]);
            treeGrid.EndInit();
        }
        else if (treeGrid.View.Nodes[i].HasChildNodes)
        {
            dataRow = (treeGrid.View.Nodes[i].Item as EmployeeInfo);
            FindMatchText(dataRow);
            GetChildNodes(treeGrid.View.Nodes[i]);
        }
        else
        {
            dataRow=(treeGrid.View.Nodes[i].Item as EmployeeInfo);
            FindMatchText(dataRow);
        }
    }
}
```

```
}
}
```

You can download the [sample](#).

Read cell values from selected items

You can get the cell values of [SelectedItems](#) using [SfTreeGrid.SelectedItems](#) and internal reflector, which reflects the field value from data object based on field name.

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    listBox.Items.Clear();
    // Get the selected items of SfTreeGrid
    var reflector = this.treeGrid.View.GetPropertyAccessProvider();
    foreach (var row in this.treeGrid.SelectedItems)
    {
        foreach (var column in treeGrid.Columns)
        {
            //Get the value from data object based on MappingName
            var cellvalue = reflector.GetValue(row, column.MappingName);
            //Returns the display value of the cell from data object based on
            MappingName
            //var displayValue = reflector.GetFormattedValue(row, column.MappingName);
            listBox.Items.Add(cellvalue.ToString());
        }
    }
}
```

Show the selection of row/cell when setting the background

The Row/Cell selection border is behind the grid cell content. So, when you apply the background for a row, the selection is not displayed in UI. You can overcome this by setting opacity in TreeGridCell.

XML

```
<Style TargetType="syncfusion:TreeGridCell">
<Setter Property="Background">
<Setter.Value>
<SolidColorBrush Color="Blue" Opacity="0.5"/>
</Setter.Value>
</Setter>
</Style>
```

Set the current cell on a particular row when tree grid is loaded

You can set the current cell in tree grid using the [treeGrid.SelectionController.MoveCurrentCell](#) method.

C#

```
private void TreeGrid_Loaded(object sender, RoutedEventArgs e)
{
    var viewModel = this.treeGrid.DataContext as ViewModel;
    //find the RowIndex for particular record
    var RowIndex = this.treeGrid.ResolveToRowIndex(viewModel.EmployeeInfo[5]);
    // find the ColumnIndex for that row.
```

```
var ColumnIndex = this.treeGrid.ResolveToScrollColumnIndex(2);  
// CurrentCell is set if MappingName is EmployeeID  
if (this.treeGrid.Columns[ColumnIndex].MappingName == "FirstName")  
this.treeGrid.SelectionController.MoveCurrentCell(new  
RowColumnIndex(RowIndex, ColumnIndex));  
}
```

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Clipboard Operations in WPF TreeGrid (SfTreeGrid)

SfTreeGrid provides support to the clipboard operations such as cut, copy, and paste the data within control and between other applications such as Notepad and Excel. The clipboard operations of copy and paste are enabled by default. You can copy selected nodes/cells from tree grid by clicking **Ctrl+C** and paste the content from the [Clipboard](#) to tree grid by clicking **Ctrl+V**.

Copy

The copy operation works based on the [GridCopyOption](#) property.

The GridCopyOption property provides the following options:

- [None](#): Disables copy in tree grid.
- [CopyData](#): Enables copy in tree grid.
- [IncludeHeaders](#): Copies the column header along with data.
- [IncludeFormat](#): Copies the display text with format instead of actual value.
- [IncludeHiddenColumn](#): Copies hidden column to the clipboard.

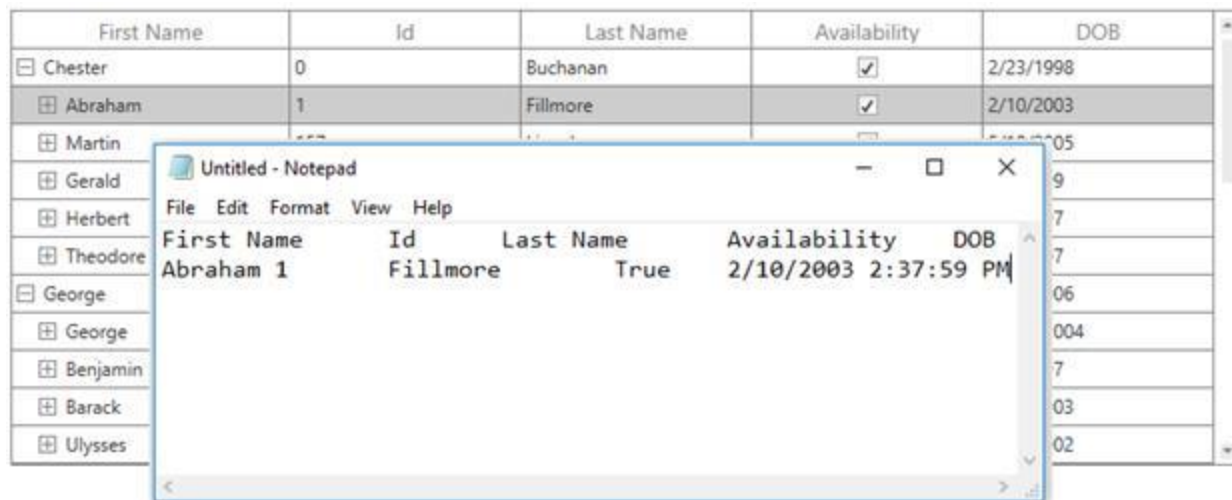
You can use the IncludeHeaders, IncludeFormat, and IncludeHiddenColumn options along with CopyData option.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"  
AutoExpandMode="RootNodesExpanded"  
AutoGenerateColumns="False"  
GridCopyOption="CopyData, IncludeHeaders"  
ChildPropertyName="Children"  
ColumnSizer="Star"  
ExpanderColumn="FirstName"  
ItemsSource="{Binding PersonDetails}"  
NavigationMode="Row">
```

C#

```
this.treeGrid.GridCopyOption = GridCopyOption.CopyData |  
GridCopyOption.IncludeHeaders;
```



Paste

The paste operation works based on the [GridPasteOption](#) property.

The GridPasteOption property provides the following options:

- [None](#): Disables paste in tree grid.
- [PasteData](#): Enables paste in tree grid. When an incompatible value is pasted into a record/cell, the pasting operation is skipped for that particular record/cell.
- [ExcludeFirstLine](#): Pastes the data copied with [IncludeHeader](#) copy option.
- [IncludeHiddenColumn](#): Pastes the values in hidden columns.

You can use the ExcludeFirstLine and IncludeHiddenColumn options along with the PasteData option.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
NavigationMode="Cell"
ChildPropertyName="Children"
ColumnSizer="Star"
GridCopyOption="CopyData, IncludeHeaders"
GridPasteOption="PasteData, ExcludeFirstLine"
ExpanderColumn="FirstName"
ItemsSource="{Binding PersonDetails}"
>
```

C#

```
this.treeGrid.GridCopyOption = GridCopyOption.CopyData |
GridCopyOption.IncludeHeaders;
this.treeGrid.GridPasteOption = GridPasteOption.PasteData |
GridPasteOption.ExcludeFirstLine;
```

First Name	Id	Last Name	Availability	DOB
<input type="checkbox"/> Chester	0	Buchanan	<input checked="" type="checkbox"/>	2/24/1998
<input type="checkbox"/> Abraham	1	Fillmore	<input checked="" type="checkbox"/>	2/11/2003
<input type="checkbox"/> Franklin	157	Fillmore	<input checked="" type="checkbox"/>	5/19/2005
<input type="checkbox"/> Franklin	313	Fillmore	<input checked="" type="checkbox"/>	7/2/1999
<input type="checkbox"/> Herbert	469	Reagan	<input checked="" type="checkbox"/>	1/8/1997
<input type="checkbox"/> Theodore	625	Madison	<input checked="" type="checkbox"/>	6/7/2007
<input type="checkbox"/> George	781	Madison	<input checked="" type="checkbox"/>	5/31/2006
<input type="checkbox"/> George	782	Harding	<input checked="" type="checkbox"/>	12/14/2004
<input type="checkbox"/> Benjamin	938	Jackson	<input checked="" type="checkbox"/>	6/9/1997
<input type="checkbox"/> Barack	1094	Harrison	<input checked="" type="checkbox"/>	2/20/2003
<input type="checkbox"/> Ulysses	1250	Cleveland	<input checked="" type="checkbox"/>	7/18/2002
<input type="checkbox"/> James	1406	Clinton	<input checked="" type="checkbox"/>	2/20/2001

Cut

The cut operation works based on the [GridCopyOption](#) property.

The GridCopyOption property provides the following options:

- [None](#): Disables cut in tree grid.
- [CutData](#): Enables cut in tree grid.
- [IncludeHeaders](#): Copies column header also along with data.
- [IncludeFormat](#): Cuts the display text with format instead of actual value.
- [IncludeHiddenColumn](#): Cuts the hidden column also to the clipboard.

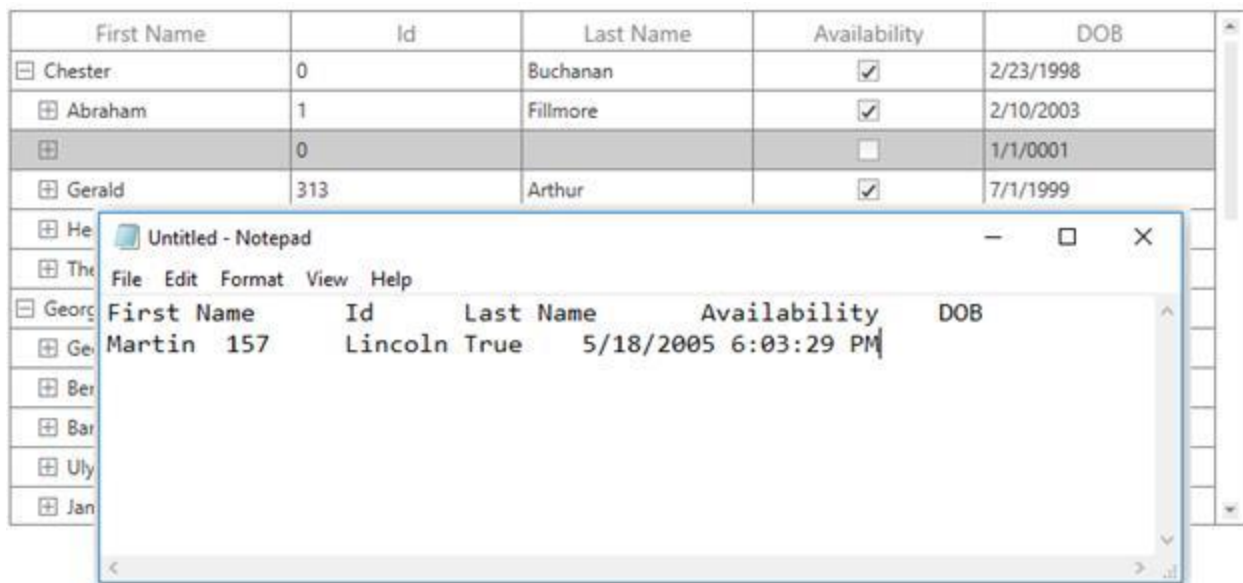
You can use the IncludeHeaders, IncludeFormat, and IncludeHiddenColumn options along with the CutData option.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
GridCopyOption="CutData,IncludeHeaders"
ChildPropertyName="Children"
ColumnSizer="Star"
ExpanderColumn="FirstName"
ItemsSource="{Binding PersonDetails}"
NavigationMode="Row">
```

C#

```
this.treeGrid.GridCopyOption = GridCopyOption.CutData |
GridCopyOption.IncludeHeaders;
```



Events

CopyContent

The [CopyContent](#) event occurs when copy/cut the cells in tree grid. The [GridCopyPasteEventArgs](#) provides information to the [CopyContent](#) event. You can cancel the copy operation by handling this event.

C#

```
this.treeGrid.CopyContent += TreeGrid_CopyContent;
private void TreeGrid_CopyContent(object sender, GridCopyPasteEventArgs e)
{
    if (((e.OriginalSender as SfTreeGrid).SelectedItem as PersonInfo).Id == 1094)
    {
        e.Handled = true;
    }
}
```

PasteContent

The [PasteContent](#) event occurs when paste the clipboard value into tree grid. The [GridCopyPasteEventArgs](#) provides information to the [PasteContent](#) event. You can cancel paste operation by handling this event.

C#

```
this.treeGrid.PasteContent += TreeGrid_PasteContent;
private void TreeGrid_PasteContent(object sender, GridCopyPasteEventArgs e)
{
    if (((e.OriginalSender as SfTreeGrid).SelectedItem as PersonInfo).Id == 1094)
    {
        e.Handled = true;
    }
}
```

CopyCellContent

The [CopyGridCellContent](#) event occurs when a cell is being copied/cut. The [GridCopyPasteCellEventArgs](#) provides information to the [CopyGridCellContent](#) event, which has following members:

- [ClipboardValue](#): Returns the cell value.
- [Column](#): Returns the corresponding GridColumn of a cell.
- [RowData](#): Returns the corresponding RowData of a cell.
- [OriginalSender](#): Returns SfTreeGrid.

You can change the text copied to the clipboard by changing the ClipboardValue.

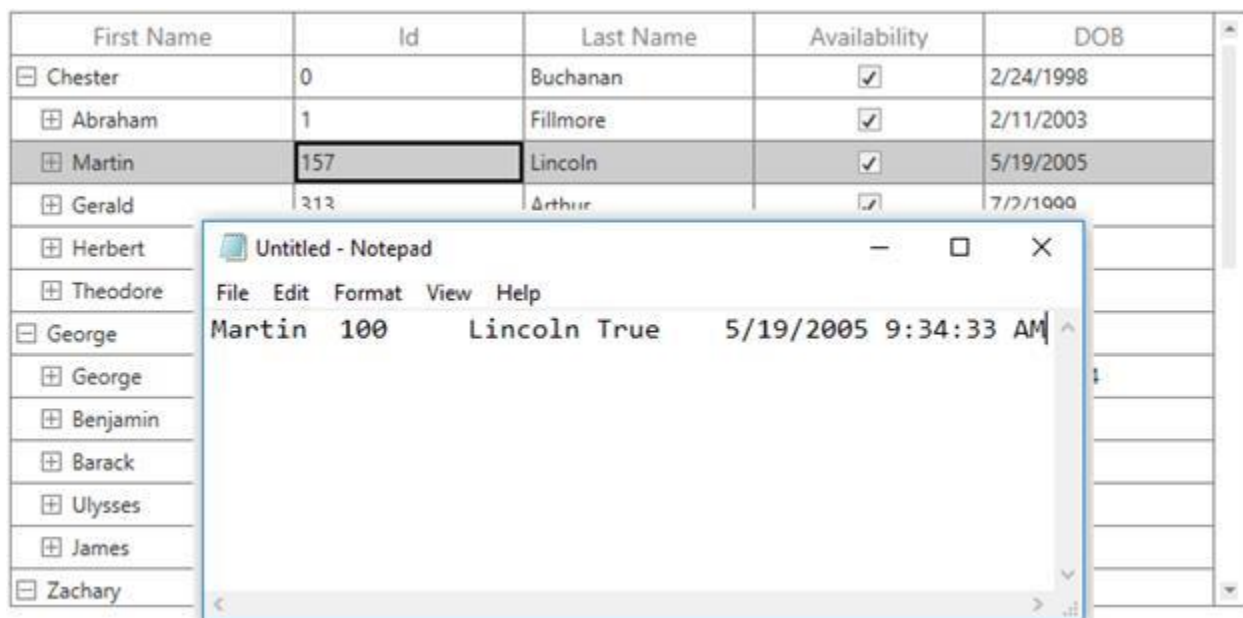
C#

```
this.treeGrid.CopyCellContent += TreeGrid_CopyCellContent;
private void TreeGrid_CopyCellContent(object sender,
TreeGridCopyPasteCellEventArgs e)
{
}
```

The following code example changes the clipboard value to 100 instead of cell value 1094 in tree grid.

C#

```
private void TreeGrid_CopyCellContent(object sender,
TreeGridCopyPasteCellEventArgs e)
{
    if (e.Column.MappingName == "Id" && (e.RowData as PersonInfo).Id == 157)
        e.ClipBoardValue = 100;
}
```



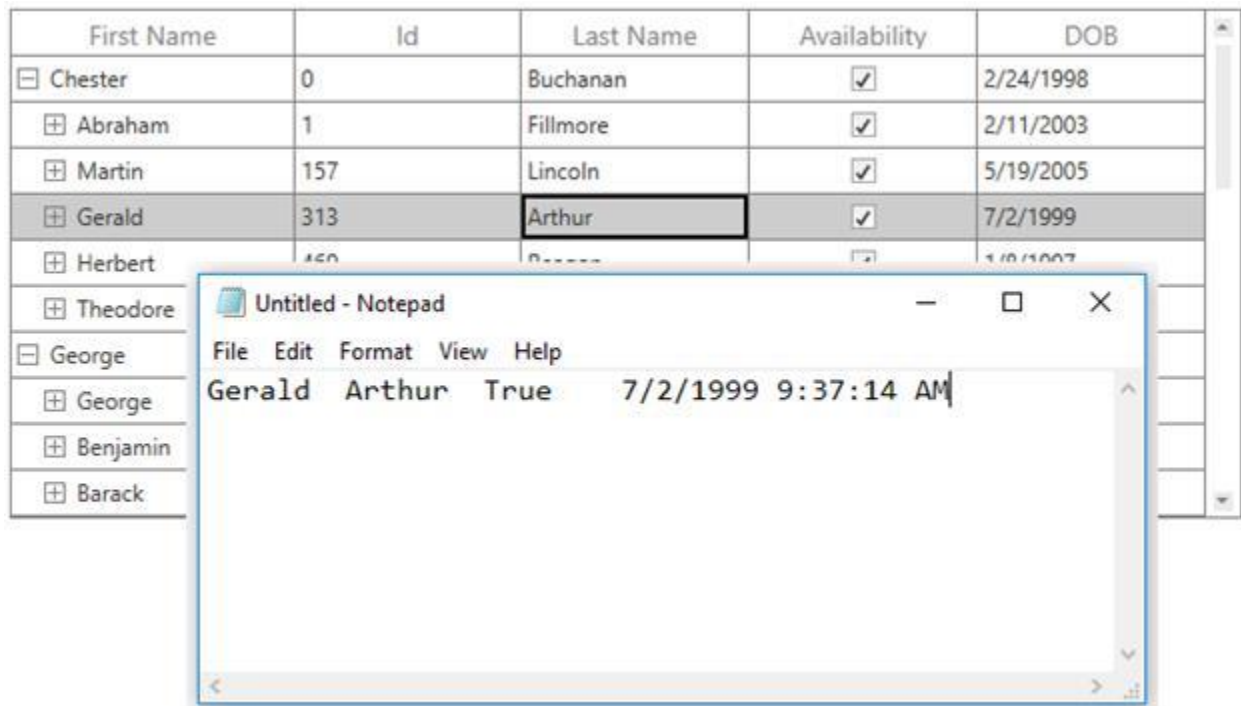
The following code example demonstrates how to handle the copy operation when MappingName of a column is Id.

C#

```
private void TreeGrid_CopyCellContent(object sender,
TreeGridCopyPasteCellEventArgs e)
{
    if (e.Column.MappingName == "Id")
```



```
e.Handled = true;
}
```



PasteCellContent

The [PasteGridCellContent](#) event occurs when a cell is being pasted. The [GridCopyPasteCellEventArgs](#) provides information to the [PasteGridCellContent](#) event, which has the following members:

- [ClipboardValue](#): Returns the clipboard value of a particular cell.
- [Column](#): Returns the corresponding GridColumn of a cell.
- [RowData](#): Returns the corresponding RowData of a cell.
- [OriginalSender](#): Returns SfTreeGrid.

You can paste the text to tree grid by changing the ClipboardValue.

C#

```
this.treeGrid.PasteCellContent += TreeGrid_PasteCellContent;
private void TreeGrid_PasteCellContent(object sender,
TreeGridCopyPasteCellEventArgs e)
{
}
```

The following code example demonstrates how to change the clipboard value to Test instead of Martin.

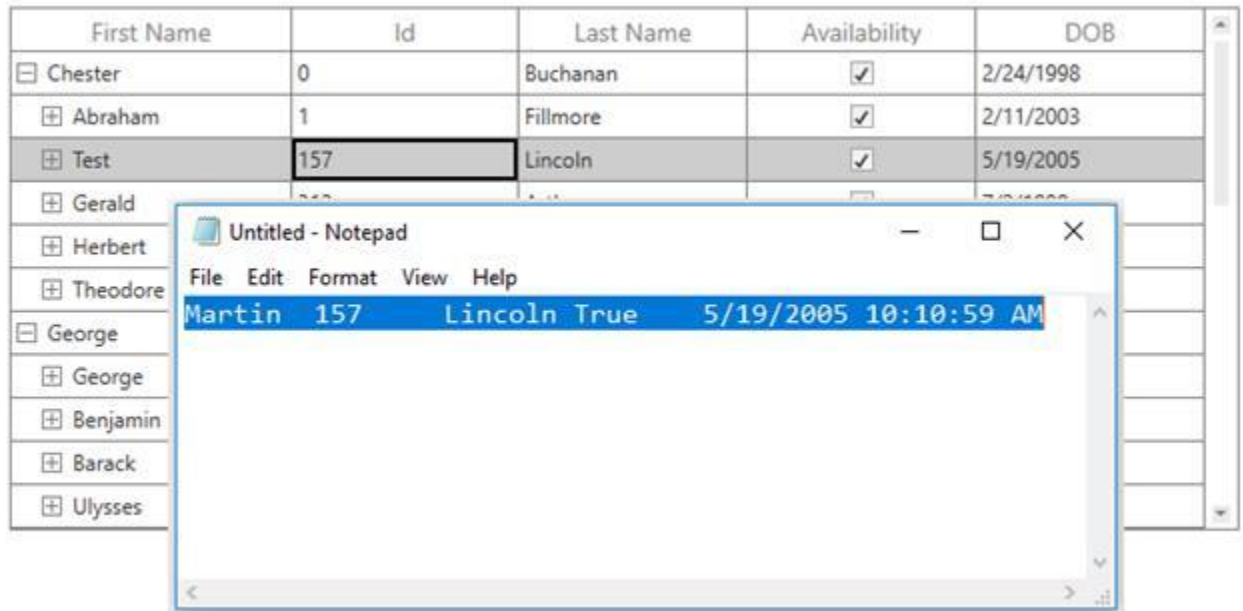
C#

```
private void TreeGrid_PasteCellContent(object sender,
TreeGridCopyPasteCellEventArgs e)
{
}
```

```

if (e.Column.MappingName == "FirstName" && (e.RowData as
PersonInfo).FirstName == "Martin")
e.ClipBoardValue = "Test";
}

```



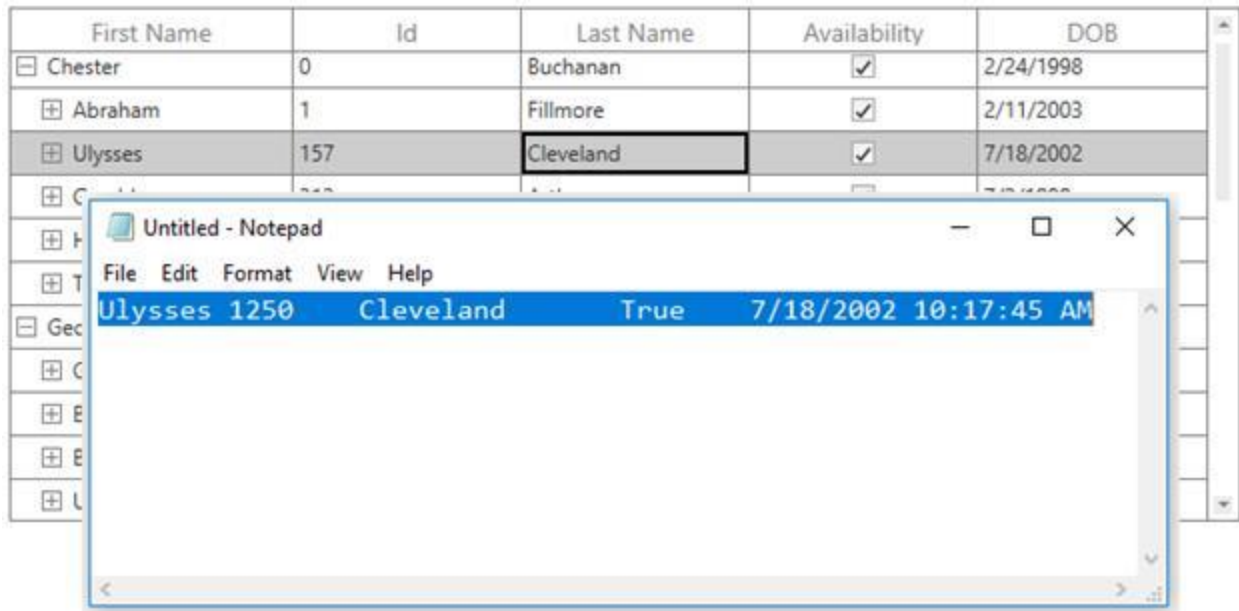
The following code example demonstrates how to handle the paste operation when MappingName of column is Id.

C#

```

private void TreeGrid_PasteCellContent(object sender,
TreeGridCopyPasteCellEventArgs e)
{
    if (e.Column.MappingName == "Id")
        e.Handled = true;
}

```



Handle the clipboard operations programmatically

Copy the node

You can copy the selected nodes in tree grid using the [Copy](#) method in [TreeGridCopyPaste](#) of tree grid.

C#

```
this.treeGrid.TreeGridCopyPaste.Copy();
```

Copy a record by selecting the record using the [MoveCurrentCell](#) method and [Copy](#) method in [TreeGridCopyPaste](#) of treegrid.

C#

```
RowColumnIndex rowColumnIndex = new RowColumnIndex();
rowColumnIndex.RowIndex = 2;
rowColumnIndex.ColumnIndex = 2;
this.treeGrid.SelectionController.MoveCurrentCell(rowColumnIndex);
this.treeGrid.TreeGridCopyPaste.Copy();
```

Copy the multiple records by selecting a group of records using the [SelectRows](#) method and [Copy](#) method in [TreeGridCopyPaste](#) of tree grid.

C#

```
this.treeGrid.SelectionController.SelectRows(2, 10);
this.treeGrid.TreeGridCopyPaste.Copy();
```

Paste to tree grid

Paste the clipboard value into tree grid using the [Paste](#) method in [TreeGridCopyPaste](#) of tree grid.

C#

```
this.treeGrid.TreeGridCopyPaste.Paste();
```

Paste the clipboard value into the selected record by selecting a record using the `MoveCurrentCell` method and `Paste` method in [TreeGridCopyPaste](#) of tree grid.

C#

```
RowColumnIndex rowColumnIndex = new RowColumnIndex();
rowColumnIndex.RowIndex = 1;
rowColumnIndex.ColumnIndex = 1;
this.treeGrid.SelectionController.MoveCurrentCell(rowColumnIndex);
this.treeGrid.TreeGridCopyPaste.Paste();
```

Cut from tree grid

Cut the selected records/cells in the tree grid using the `Cut` method in [TreeGridCopyPaste](#) of tree grid.

C#

```
this.treeGrid.TreeGridCopyPaste.Cut();
```

Cut the entire record in tree grid by selecting the whole tree grid using the [SelectAll](#) method and `Cut` method in [TreeGridCopyPaste](#) of tree grid.

C#

```
this.treeGrid.SelectionController.SelectAll();
this.treeGrid.TreeGridCopyPaste.Cut();
```

Customize the copy and paste behaviors

The tree grid processes the clipboard operations in the [TreeGridCutCopyPaste](#) class. You can customize the default copy paste behaviors by overriding the `TreeGridCutCopyPaste` class and set it to `SfTreeGrid.TreeGridCopyPaste`.

C#

```
public class CustomCopyPaste : TreeGridCutCopyPaste
{
    public CustomCopyPaste(SfTreeGrid sfTreeGrid) : base(sfTreeGrid)
    {
    }
}
```

C#

```
public MainWindow()
{
    InitializeComponent();
    this.treeGrid.TreeGridCopyPaste = new CustomCopyPaste(this.treeGrid);
}
```

Paste a record into selected rows

By default, you can copy a row and paste it into another row in treegrid. The following code demonstrates how to copy a row and paste it into all the selected rows by overriding the [PasteRow](#) method in the [TreeGridCutCopyPaste](#) class.

C#

```
public class CustomCopyPaste : TreeGridCutCopyPaste
{
    public CustomCopyPaste(SfTreeGrid sfTreeGrid) : base(sfTreeGrid)
    {
    }
    protected override void PasteRow(object clipboardcontent, object
selectedRecords)
    {
        var text = Clipboard.GetText();
        string[] clipBoardText = Regex.Split(text, @"\r\n");
        // Get the clipBoardText and check if the clipBoardText is more than one
        row.
        //means call the base.
        if (clipBoardText.Count() > 1)
        {
            base.PasteRow(clipboardcontent, selectedRecords);
            return;
        }
        var selectedRecord = this.TreeGrid.SelectedItems;
        for (int i = 0; i < selectedRecord.Count; i++)
        {
            // Get the selected records for paste the copied row.
            selectedRecords = selectedRecord[i];
            // Call the PasteRow method with clipBoardContent and selectedRecords.
            base.PasteRow(clipboardcontent, selectedRecords);
        }
    }
}
```

Select the pasted records

By default, after pasting the clipboard value to tree grid, the selection is maintained in previously selected records. The following code demonstrates how to select the pasted records after the paste operation by overriding the [PasteToRow](#) method in the [TreeGridCutCopyPaste](#) class.

C#

```
public class CustomCopyPaste : TreeGridCutCopyPaste
{
    public CustomCopyPaste(SfTreeGrid sfTreeGrid) : base(sfTreeGrid)
    {
    }
    protected override void PasteRow(object clipboardcontent, object
selectedRecords)
    {
        base.PasteRow(clipboardcontent, selectedRecords);
        // Add the selected record to list.
        this.TreeGrid.SelectedItems.Add(selectedRecords);
    }
}
```

Create new records while pasting

By default, when pasting the clipboard value to tree grid, it changes the values of the already existing records. The following code example demonstrates how to add the copied records as new rows in the tree grid by overriding the [PasteRows](#) method in the [TreeGridCutCopyPaste](#) class.

C#

```
public class CustomCopyPaste : TreeGridCutCopyPaste
{
    public CustomCopyPaste(SfTreeGrid sfTreeGrid) : base(sfTreeGrid)
    {
    }
    protected override void PasteRows(object clipBoardRows)
    {
        var copiedRecord = (string[])clipBoardRows;
        int copiedRecordsCount = copiedRecord.Count();
        // Based on the clipboard count, add the new record for paste.
        if (copiedRecordsCount > 0)
        {
            //Get the viewModel for adding the record.
            var record = this.TreeGrid.DataContext as ViewModel;
            for (int i = 0; i < copiedRecordsCount; i++)
            {
                // Create a new instance for Model for adding the new record.
                PersonInfo entity = new PersonInfo();
                for (int j = 0; j < this.TreeGrid.Columns.Count; j++)
                {
                    string[] values = Regex.Split(copiedRecord[i], @"\t");
                    // Add a new record using the PasteToCell method by passing the created
                    // data, particular column, and clipboard value.
                    this.PasteCell(entity, this.TreeGrid.Columns[j], values[j]);
                }
                // Add the pasted record in collection.
                record.PersonDetails.Add(entity);
            }
        }
    }
}
```

Paste data by custom column order

By default, the data can be pasted only from the first column. However, you can paste the copied data anywhere in the grid by deriving a new class from [TreeGridCutCopyPaste](#) and overriding the [PasteRow](#) virtual method.

C#

```
public class CustomCopyPaste : TreeGridCutCopyPaste
{
    public CustomCopyPaste(SfTreeGrid sfTreeGrid) : base(sfTreeGrid)
    {
    }
    protected override void PasteRow(object clipboardcontent, object
selectedRecords)
    {
        // Split the row into number of cells by using \t.
        clipboardcontent = Regex.Split(clipboardcontent.ToString(), @"\t");
    }
}
```

```

var copyValue = (string[])clipboardcontent;
int columnIndex = 0;
// Get the currentcell column index.
var index =
this.TreeGrid.SelectionController.CurrentCellManager.CurrentCell.ColumnIndex
;
foreach (var column in TreeGrid.Columns)
{
if (index >= TreeGrid.Columns.Count)
return;
if (copyValue.Count() <= this.TreeGrid.Columns.IndexOf(column))
break;
// Call the PasteToCell method, pass the copied data, and paste the column
index.
PasteCell(selectedRecords, this.TreeGrid.Columns[index],
copyValue[columnindex]);
index++;
columnindex++;
}
}
}

```

Copy the column and paste it as a new column

You can copy a column and paste it into a new position using the context menu option in tree grid.

XML

```

<syncfusion:SfTreeGrid.HeaderContextMenu>
<ContextMenu ItemsSource="{Binding Menu,Source={StaticResource viewmodel}}">
>
<ContextMenu.ItemContainerStyle>
<Style TargetType="MenuItem">
<Setter Property="Command" Value="{Binding MyCommand,Source={StaticResource
viewmodel}}"></Setter>
<Setter Property="CommandParameter" >
<Setter.Value>
<MultiBinding Converter="{StaticResource ResourceKey=converter}">
<Binding RelativeSource="{RelativeSource Self}"/>
<Binding />
</MultiBinding>
</Setter.Value>
</Setter>
</Style>
</ContextMenu.ItemContainerStyle>
</ContextMenu>
</syncfusion:SfTreeGrid.HeaderContextMenu>

```

C#

```

private static void OnCopyColumn(object obj)
{
if (obj is TreeGridColumnContextMenuInfo)
{
// The selected column is stored into CopiedColumn.
CopiedColumn = (obj as TreeGridColumnContextMenuInfo).Column;
}
}

```

```

}
private static void OnPasteColumn(object obj)
{
    if (obj is TreeGridColumnContextMenuInfo && CopiedColumn != null)
    {
        var grid = (obj as TreeGridColumnContextMenuInfo).TreeGrid;
        // Get the index for corresponding column.
        var index = grid.Columns.IndexOf((obj as TreeGridColumnContextMenuInfo).Column);
        // Copy the column and insert based on the index position.
        grid.Columns.Insert(index + 1, new TreeGridTextColumn() { MappingName = CopiedColumn.MappingName });
    }
}

```

Copy the ID column using context menu

FirstName	LastName	ID	Title	Salary	ReportsTo
Ferando	Joseph	2	Management	2000000.00	-1
Andrew	Fuller	9	Vice President	1200000.00	2
Janet	Leverling	10	GM	1000000.00	2
Steven	Buchanan	11	Manager	900000.00	2
John		3	Accounts	2000000.00	-1
Nancy	Devolio	12	Accounts Manager	850000.00	3
Margaret	Peacock	13	Accountant	700000.00	3
Michael	Suyama	14	Accountant	700000.00	3
Robert	King	15	Accountant	650000.00	3
Ferando1	Joseph1	4	Management1	2000000.00	-1
Nancy1	Devolio1	16	Accounts Manager1	850000.00	4
Margaret1	Peacock1	17	Accountant1	700000.00	4
Michael1	Suyama1	18	Accountant1	700000.00	4
John1		5	Accounts1	2000000.00	-1
Robert1	King1	19	Accountant1	650000.00	5
Nancy2	Devolio2	20	Accounts Manager2	850000.00	5
Margaret2	Peacock2	21	Accountant2	700000.00	5
Michael2	Suyama2	22	Accountant2	700000.00	5

Paste the ID column after ReportsTo column

FirstName	LastName	ID	Title	PartsTo
Ferando	Joseph	2	Management	-1
Andrew	Fuller	9	Vice President	2
Janet	Leverling	10	GM	2
Steven	Buchanan	11	Manager	2
John		3	Accounts	-1
Nancy	Devolio	12	Accounts Manager	3
Margaret	Peacock	13	Accountant	3
Michael	Suyama	14	Accountant	3
Robert	King	15	Accountant	3
Ferando1	Joseph1	4	Management1	-1
Nancy1	Devolio1	16	Accounts Manager1	4
Margaret1	Peacock1	17	Accountant1	4
Michael1	Suyama1	18	Accountant1	4
John1		5	Accounts1	-1
Robert1	King1	19	Accountant1	5
Nancy2	Devolio2	20	Accounts Manager2	5

You can download the [sample](#).

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Node CheckBox in WPF TreeGrid (SfTreeGrid)

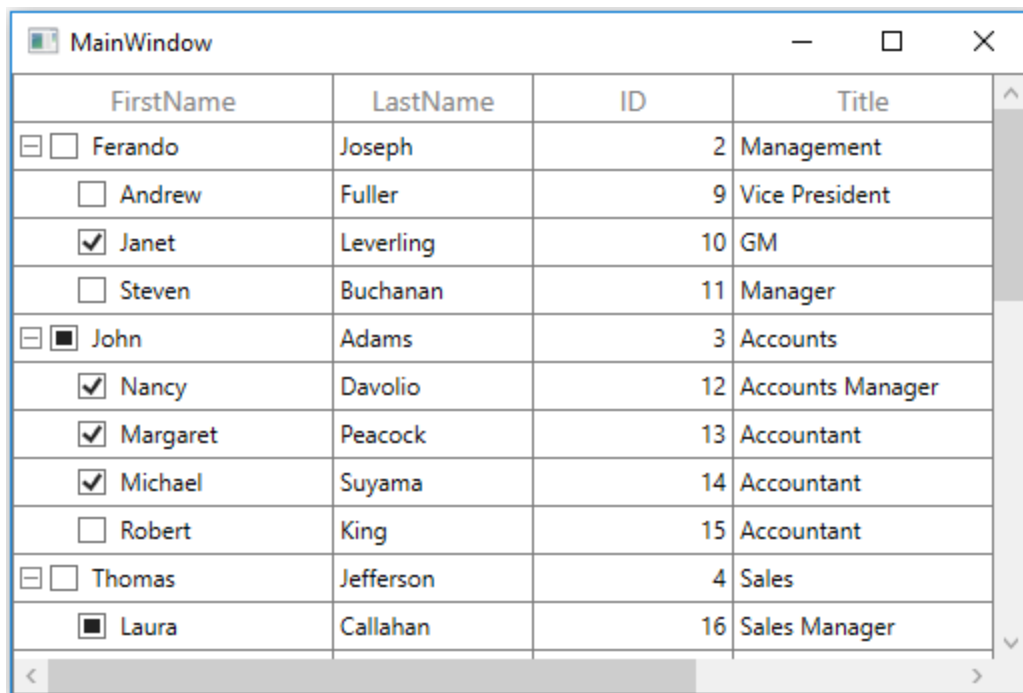
SfTreeGrid provides support for loading **CheckBox** in the expander cell of each node, which allows the user to check/uncheck the corresponding node. You can display check box in each node by setting [SfTreeGrid.ShowCheckBox](#) property as **true**. It also provides support to process the selection in the context of state of the checkbox based on [SfTreeGrid.CheckBoxSelectionMode](#) property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
    ChildPropertyName="Children"
    ItemsSource="{Binding PersonDetails}"
    CheckBoxSelectionMode="Default"
    ShowCheckBox="True"/>
```

C#

```
treeGrid.ShowCheckBox = true;
treeGrid.CheckBoxSelectionMode = CheckBoxSelectionMode.Default;
```



Indeterminate State Support

You can enable or disable the indeterminate state for node CheckBox using [SfTreeGrid.AllowTriStateChecking](#) property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AllowTriStateChecking="True"
ChildPropertyName="Children"
ItemsSource="{Binding PersonDetails}"
ShowCheckBox="True"/>
```

C#

```
treeGrid.AllowTriStateChecking = true;
```

Recursive Checking

SfTreeGrid provides support for recursive checking where the checked state of parent node and child nodes is changed recursively based on the state of currently changed node. You can enable recursive checking by setting [SfTreeGrid.EnableRecursiveChecking](#) property as `true`.

- A tree node will be checked only if all its child nodes are checked.
- A tree node will be unchecked if all its child nodes are unchecked.
- The tree node will be in Indeterminate state in other combinations of its children.

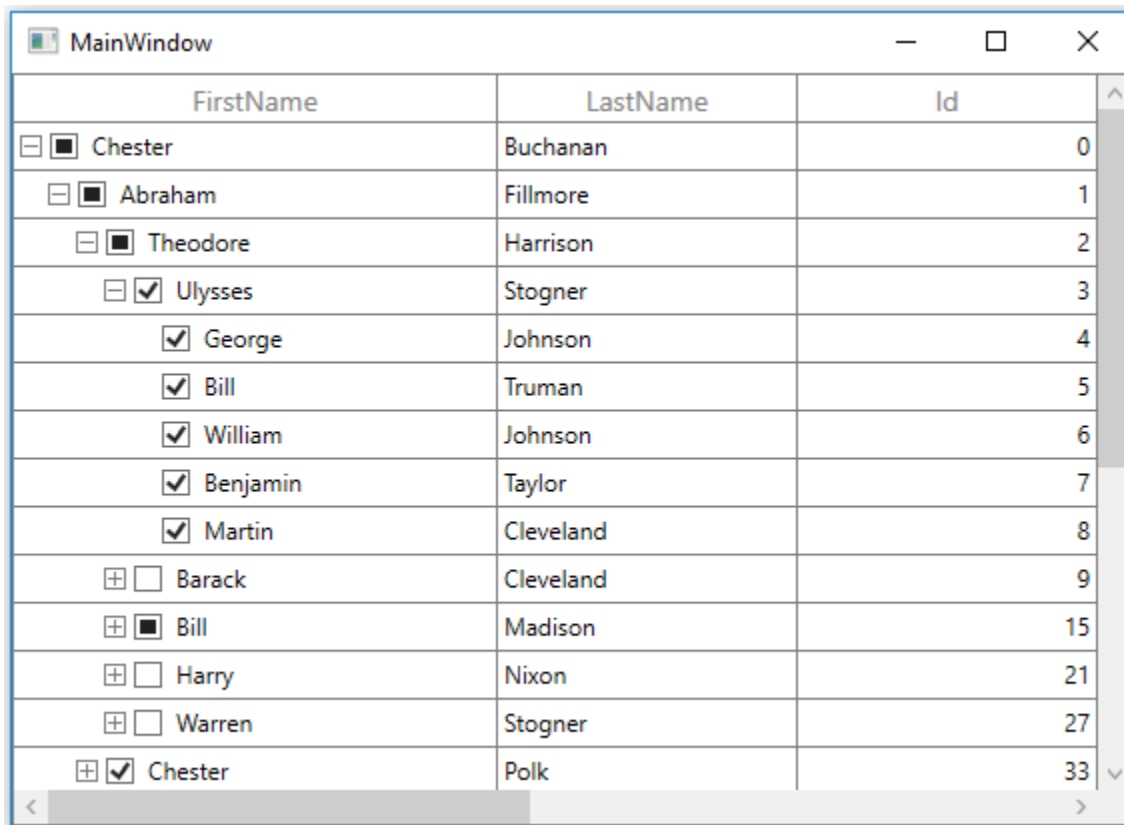
XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
EnableRecursiveChecking="True"
ChildPropertyName="Children"
```

```
ItemsSource="{Binding PersonDetails}"
ShowCheckBox="True"/>
```

C#

```
treeGrid.EnableRecursiveChecking = true;
```



Note: Even though [SfTreeGrid.AllowTriStateChecking](#) is **false** if [SfTreeGrid.EnableRecursiveChecking](#) is **true**, **CheckBox** can be in indeterminate state.

Saving and loading Node **CheckBox** state from the property in data object

You can bind state of node checkbox to the bool property in underlying data object by using [SfTreeGrid.CheckBoxMappingName](#) property. TreeGrid updates the checked state of checkbox when underlying data object property gets changed and vice versa.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
CheckBoxMappingName="IsSelected"
ChildPropertyName="Children"
ItemsSource="{Binding PersonDetails}"
ShowCheckBox="True"/>
```

C#

```
treeGrid.CheckBoxMappingName = "IsSelected";
```

Disable Recursive Checking when data object property changed

By default, recursive checking will be applied, whenever node's `IsChecked` property gets changed. You can disable the recursive checking on property value change (which is mapped with `CheckBoxMappingName`) by setting `SfTreeGrid.RecursiveCheckingMode` as `OnCheck`.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
    EnableRecursiveChecking="True"
    ChildPropertyName="Children"
    RecursiveCheckingMode="OnCheck"
    ItemsSource="{Binding PersonDetails}"
    ShowCheckBox="True"/>
```

C#

```
treeGrid.RecursiveCheckingMode = RecursiveCheckingMode.OnCheck;
```

Disabling CheckBox for certain nodes

You can disable CheckBox by writing style for `TreeGridExpanderCell` and setting `IsCheckBoxEnabled` property as 'false'

XML

```
<Style TargetType="syncfusion:TreeGridExpanderCell">
    <Setter Property="IsCheckBoxEnabled" Value="{Binding Path=HasChildNodes,
        RelativeSource={RelativeSource Self}}" />
</Style>
```

In the below screenshot, node CheckBox is disabled for leaf nodes.

FirstName	LastName	Id
<input type="checkbox"/> Chester	Buchanan	0
<input type="checkbox"/> Abraham	Fillmore	1
<input type="checkbox"/> Theodore	Harrison	2
<input type="checkbox"/> Ulysses	Stogner	3
<input type="checkbox"/> George	Johnson	4
<input type="checkbox"/> Bill	Truman	5
<input type="checkbox"/> William	Johnson	6
<input type="checkbox"/> Benjamin	Taylor	7
<input type="checkbox"/> Martin	Cleveland	8
<input checked="" type="checkbox"/> Barack	Cleveland	9
<input checked="" type="checkbox"/> Bill	Madison	15
<input checked="" type="checkbox"/> Harry	Nixon	21
<input checked="" type="checkbox"/> Warren	Stogner	27
<input checked="" type="checkbox"/> Chester	Polk	33

Collapsing CheckBox for certain nodes

You can collapse node CheckBox for certain nodes by editing the control template of [TreeGridExpanderCell](#) and changing the Checkbox visibility based on condition.

XML

```
<Window.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.SfGrid.WPF;component/Control/Themes/Generic.xaml" />
</ResourceDictionary.MergedDictionaries>
<local:BoolToVisibilityConverter x:Key="boolToVisibilityConverter" />
<Style TargetType="syncfusion:TreeGridExpanderCell">
<Setter Property="Background" Value="Transparent" />
<Setter Property="BorderThickness" Value="0,0,1,1" />
<Setter Property="BorderBrush" Value="Gray" />
<Setter Property="Padding" Value="0" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:TreeGridExpanderCell">
<Grid x:Name="Root">
<Border x:Name="PART_GridCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<Grid Margin="{TemplateBinding IndentMargin}">
<Grid.ColumnDefinitions>
```

```

<ColumnDefinition Width="14" />
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<syncfusion:TreeGridExpander x:Name="PART_ExpanderCell"
Grid.Column="0"
Width="12"
Height="12"
Margin="2,1,0,1"
HorizontalAlignment="Center"
VerticalAlignment="Center"
IsExpanded="{Binding RelativeSource={RelativeSource TemplatedParent},
Path=IsExpanded,
Mode=TwoWay}"
Visibility="{Binding RelativeSource={RelativeSource TemplatedParent},
Path=HasChildNodes,
Converter={StaticResource boolToVisibilityConverter},
Mode=TwoWay}" />
<CheckBox Name="PART_SelectCheckBox"
Grid.Column="1"
Width="16"
Height="16"
MinWidth="16"
Margin="3,0,0,0"
VerticalAlignment="Center"
Syncfusion:VisualContainer.WantsMouseInput="True"
IsEnabled="{Binding RelativeSource={RelativeSource TemplatedParent},
Path=IsCheckBoxEnabled,
Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
IsThreeState="True"
Visibility="{Binding Path=HasChildNodes,
RelativeSource={RelativeSource TemplatedParent},
Converter={StaticResource boolToVisibilityConverter},
Mode=TwoWay}" />
<Grid Grid.Column="2"
Margin="3,0,0,0"
Background="{TemplateBinding Background}">
<ContentPresenter />
<Border x:Name="PART_CurrentCellBorder"
Margin="0,0,0,0"
Background="Transparent"
BorderBrush="{TemplateBinding CurrentCellBorderBrush}"
BorderThickness="{TemplateBinding CurrentCellBorderThickness}"
IsHitTestVisible="False"
Visibility="Collapsed" />
<Border x:Name="PART_InvalidCellBorder"
Width="10"
Height="10"
HorizontalAlignment="Right"
VerticalAlignment="Top"
SnapsToDevicePixels="True"
Visibility="Collapsed">
<ToolTipService.ToolTip>
<ToolTip Background="#FFDB000C"
Placement="Right"
PlacementRectangle="20,0,0,0"

```

```

Tag="{TemplateBinding ErrorMessage}"
Template="{StaticResource ValidationToolTipTemplate}" />
</ToolTipService.ToolTip>
<Path Data="M0.5,0.5 L12.652698,0.5 12.652698,12.068006 z"
Fill="Red"
SnapsToDevicePixels="True"
Stretch="Fill" />
</Border>
</Grid>
</Grid>
</Border>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="IndicationStates">
<VisualState x:Name="NoError" />
<VisualState x:Name="HasError">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CurrentStates">
<VisualState x:Name="Regular" />
<VisualState x:Name="Current">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_CurrentCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>
</Window.Resources>

```

C#

```

public class BoolToVisibilityConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        if ((bool) value)
            return Visibility.Visible;
    }
}

```

```

return Visibility.Collapsed;
}
public object ConvertBack(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
{
    if ((Visibility)value == Visibility.Visible)
        return true;
    return false;
}
}

```

Here, node CheckBox is collapsed for leaf nodes.



Handling Selection based on CheckBox State

SfTreeGrid has following modes for processing selection based on check box state.

1. Default – Selection and state of checkbox works independent of each other.
2. SelectOnCheck – Row can be selected or deselected based on state of checkbox.
3. SynchronizeSelection – Row can be selected or deselected based on state of checkbox and vice versa.

Default mode

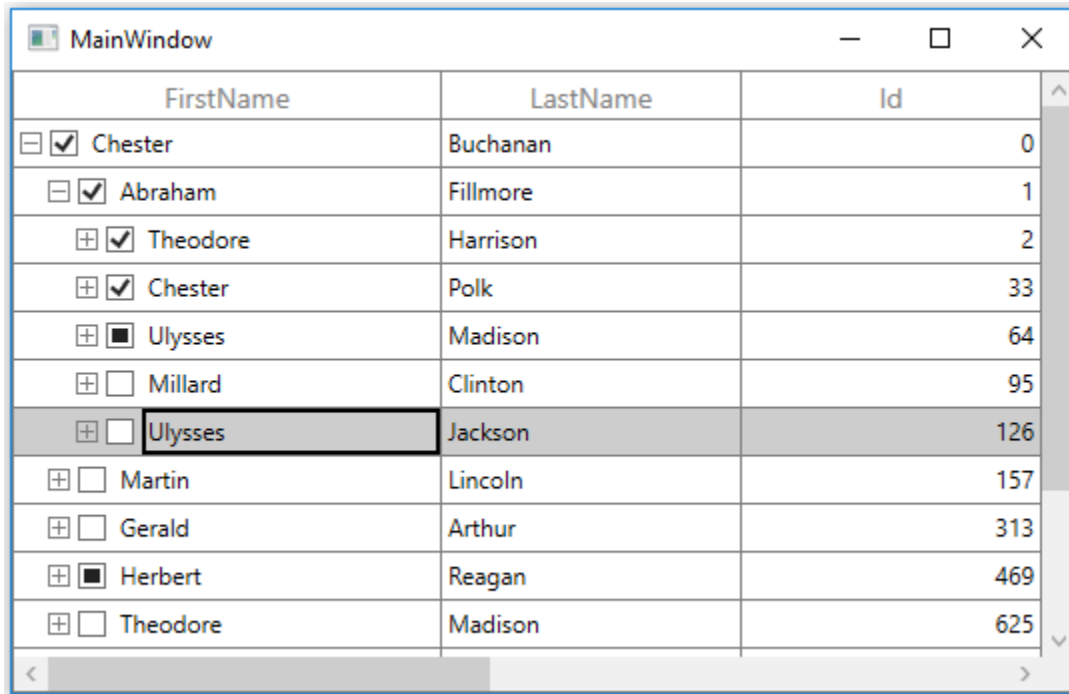
If you don't want to affect the selection while checking/unchecking the node CheckBox, you need to set [SfTreeGrid.CheckBoxSelectionMode](#) as Default.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
  CheckBoxSelectionMode="Default"
  ItemsSource="{Binding PersonDetails}"
  ShowCheckBox="True"/>
```

C#

```
treeGrid.CheckBoxSelectionMode = CheckBoxSelectionMode.Default;
```

**SelectOnCheck**

If you want to select/deselect the rows using node CheckBox only, you need to set [SfTreeGrid.CheckBoxSelectionMode](#) as [SelectOnCheck](#).

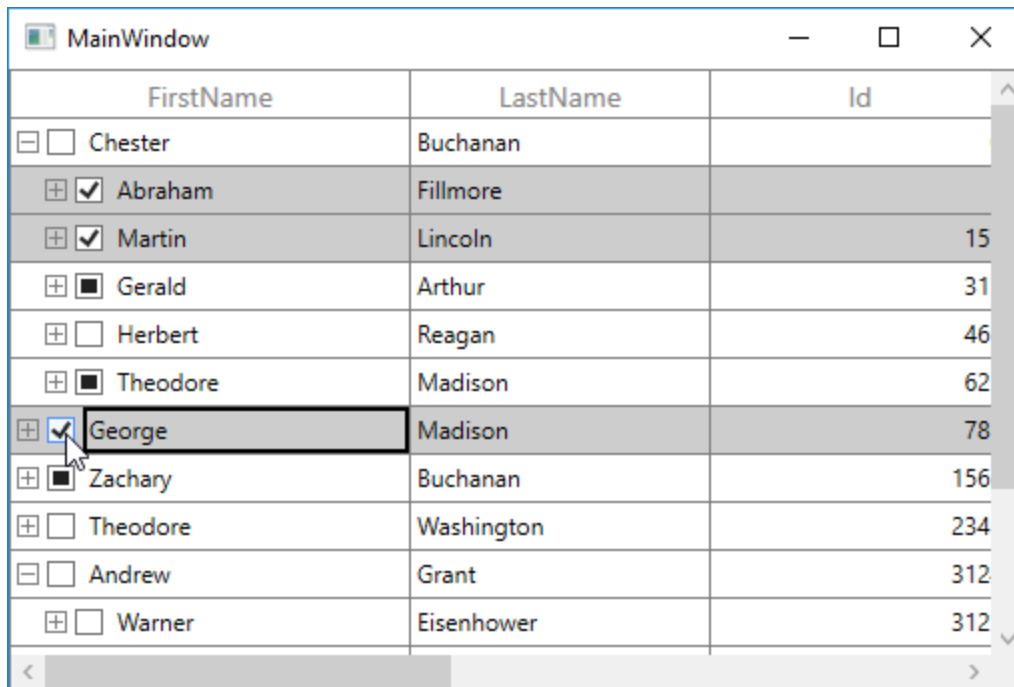
Navigation, editing and programmatic selection are not supported in this mode.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
  CheckBoxSelectionMode="SelectOnCheck"
  ItemsSource="{Binding PersonDetails}"
  ShowCheckBox="True"/>
```

C#

```
treeGrid.CheckBoxSelectionMode = CheckBoxSelectionMode.SelectOnCheck;
```



FirstName	LastName	Id
<input type="checkbox"/> Chester	Buchanan	
<input checked="" type="checkbox"/> Abraham	Fillmore	
<input checked="" type="checkbox"/> Martin	Lincoln	15
<input checked="" type="checkbox"/> Gerald	Arthur	31
<input type="checkbox"/> Herbert	Reagan	46
<input checked="" type="checkbox"/> Theodore	Madison	62
<input checked="" type="checkbox"/> George	Madison	78
<input checked="" type="checkbox"/> Zachary	Buchanan	156
<input type="checkbox"/> Theodore	Washington	234
<input type="checkbox"/> Andrew	Grant	312
<input type="checkbox"/> Warner	Eisenhower	312

SynchronizeSelection

If you want to synchronize the selection with node CheckBox's IsChecked state, you need to set [SfTreeGrid.CheckBoxSelectionMode](#) as `SynchronizeSelection`. In this mode, you can select by checking checkbox and selecting/deselecting the row will check/uncheck the corresponding node checkbox.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
    CheckBoxSelectionMode="SynchronizeSelection"
    ItemsSource="{Binding PersonDetails}"
    ShowCheckBox="True"/>
```

C#

```
treeGrid.CheckBoxSelectionMode = CheckBoxSelectionMode.SynchronizeSelection;
```

FirstName	LastName	Id
<input checked="" type="checkbox"/> Chester	Buchanan	
<input checked="" type="checkbox"/> Abraham	Fillmore	
<input type="checkbox"/> Martin	Lincoln	1
<input type="checkbox"/> Gerald	Arthur	3
<input checked="" type="checkbox"/> Herbert	Reagan	4
<input checked="" type="checkbox"/> Theodore	Madison	6
<input type="checkbox"/> George	Madison	7
<input checked="" type="checkbox"/> Zachary	Buchanan	15
<input type="checkbox"/> Theodore	Washington	23
<input type="checkbox"/> Andrew	Grant	31
<input type="checkbox"/> Warner	Eisenhower	31

Note:

- Recursive checking is not supported when selection mode is single.
- CheckBox selection is not supported if selection mode in None.

Events

[NodeCheckStateChanged](#)

[NodeCheckStateChanged](#) event triggered when user check or uncheck the node check box.

C#

```
treeGrid.NodeCheckStateChanged += TreeGrid_NodeCheckStateChanged;
private void TreeGrid_NodeCheckStateChanged(object sender,
NodeCheckStateChangedEventArgs e)
{
    var node = e.Node;
}
```

Programmatically Processing Node CheckBox

You can change the state of node checkbox programmatically by calling [SetCheckedState](#) method as below,

C#

```
var treeNode = treeGrid.View.Nodes[0];
treeNode.SetCheckedState(true);
```

If you want to restrict the `IsChecked` update of the parent and child nodes (when [SfTreeGrid.EnableRecursiveChecking](#) is `true`), you can pass default parameter values as `false` in [SetCheckedState](#) method.

C#

```
var treeNode = treeGrid.View.Nodes[0];
treeNode.SetCheckedState(true, false, false);
```

Getting Checked nodes

You can get the checked nodes collection using [GetCheckedNodes](#) method.

C#

```
var nodes = treeGrid.GetCheckedNodes();
```

If you want to get all the checked nodes even though they are not in view, you can pass parameter as 'true' in [GetCheckedNodes](#) method.

C#

```
var nodes = treeGrid.GetCheckedNodes(true);
```

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Data Validation in WPF TreeGrid (SfTreeGrid)

SfTreeGrid allows you to validate the data and display hints in case of validation is not passed. In case of invalid data, error icon is displayed at the top right corner of [TreeGridCell](#). When mouse over the error icon, error information will be displayed in tooltip.

Built-in validations

Built-in validations through [IDataErrorInfo](#), [INotifyDataErrorInfo](#) and Data annotation attributes, can be enabled by setting [SfTreeGrid.GridValidationMode](#) or [TreeGridColumn.GridValidationMode](#) properties. [TreeGridColumn.GridValidationMode](#) takes priority than [SfTreeGrid.GridValidationMode](#).

- [GridValidation.InEdit](#) - display error icon & tips and doesn't allows the users to commit the invalid data without allowing users to edit other cells.
- [GridValidation.InView](#) - displays error icons and tips alone.
- [GridValidation.None](#) - disables built-in validation support.

Built-in validation using IDataErrorInfo / INotifyDataErrorInfo

SfTreeGrid provides support to validate the data based on [IDataErrorInfo](#)/[INotifyDataErrorInfo](#).

Using IDataErrorInfo

You can validate the data by inheriting the [IDataErrorInfo](#) interface in model class.

C#

```
public class OrderInfo : IDataErrorInfo
{
    private string country;
    public string Country
    {
        get { return country; }
    }
}
```

```
set { country = value; }
}
[Display(AutoGenerateField = false)]
public string Error
{
    get
    {
        return string.Empty;
    }
}
public string this[string columnName]
{
    get
    {
        if (!columnName.Equals("Country"))
            return string.Empty;
        if (this.Country.Contains("Germany") || this.Country.Contains("UK"))
            return "Delivery not available for the country " + this.Country;
        return string.Empty;
    }
}
}
```

Enable built-in validation support by setting `SfTreeGrid.GridValidationMode` or `TreeGridColumn.GridValidationMode` property to `InEdit` or `InView`.

XML

```
<syncfusion:SfTreeGrid x:Name="treeGrid"
    AllowEditing="True"
    ChildPropertyName="Children"
    GridValidationMode="InView"
    ItemsSource="{Binding PersonDetails}" />
```

C#

```
treeGrid.GridValidationMode = GridValidationMode.InView;
```

FirstName	Country	LastName
⊕ Obama	Italy	Bosh
⊕ John	Germany	Adams
⊕ Thomas	Venezuela	Jefferson
⊕ Andrew	Finland	Madison
⊕ Ulysses	France	Pierce
⊖ Jimmy	Argentina	Harrison
Ronald	Denmark	Fillmore
Steven	Finland	Buchanan
Robert	Italy	King
⊖ Ronald	Sweden	Fillmore
Andrew	Finland	Fuller

INotifyDataErrorInfo

You can validate the data by implementing the [INotifyDataErrorInfo](#) interface in model class.

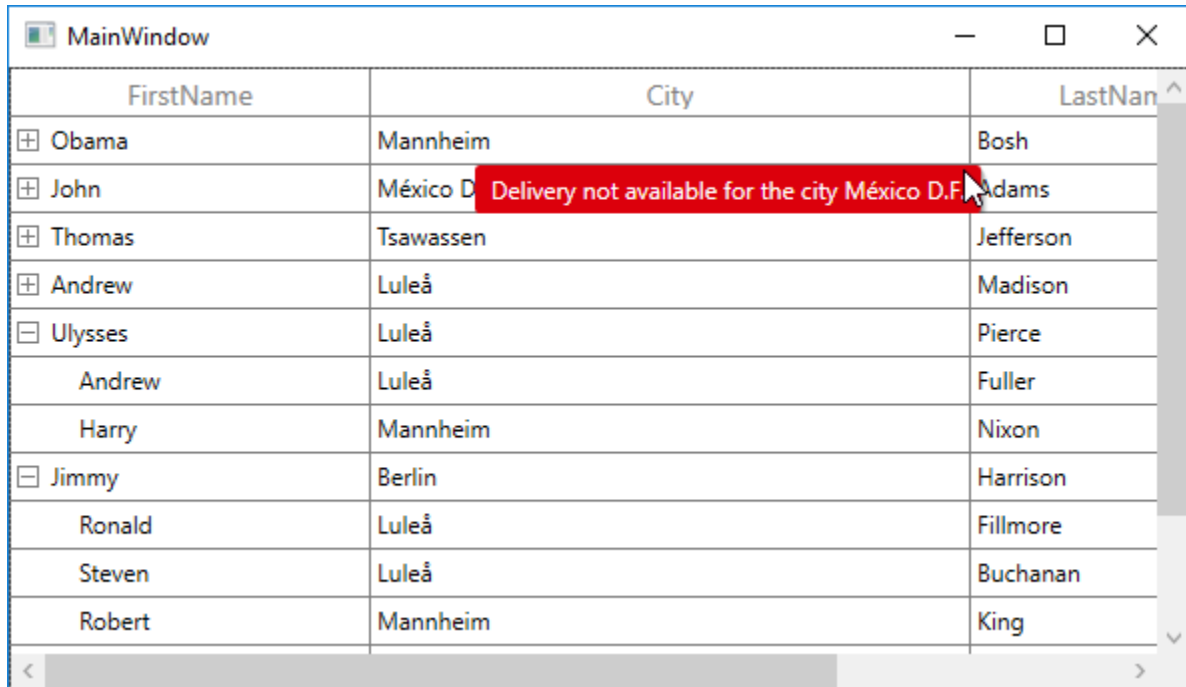
C#

```
public class OrderInfo : INotifyDataErrorInfo
{
    private List<string> errors = new List<string>();
    private string shippingCity;
    public string ShipCity
    {
        get { return shippingCity; }
        set { shippingCity = value; }
    }
    public System.Collections.IEnumerable GetErrors(string propertyName)
    {
        if (!propertyName.Equals("ShipCity"))
            return null;
        if (this.ShipCity.Contains("Mexico D.F. "))
            errors.Add("Delivery not available for the city " + this.ShipCity);
        return errors;
    }
    [Display(AutoGenerateField = false)]
    public bool HasErrors
    {
        get
        {
            return false;
        }
    }
    public event EventHandler<DataErrorsChangedEventArgs> ErrorsChanged;
}
```

Enable built-in validation support by setting `SfTreeGrid.GridValidationMode` or `TreeGridColumn.GridValidationMode` property to `InEdit` or `InView`.

XML

```
<syncfusion:SfTreeGrid x:Name="treeGrid"
    AllowEditing="True"
    ChildPropertyName="Children"
    GridValidationMode="InView"
    ItemsSource="{Binding PersonDetails}" />
```



FirstName	City	LastName
Obama	Mannheim	Bosh
John	México D.F.	Adams
Thomas	Tsawassen	Jefferson
Andrew	Luleå	Madison
Ulysses	Luleå	Pierce
Andrew	Luleå	Fuller
Harry	Mannheim	Nixon
Jimmy	Berlin	Harrison
Ronald	Luleå	Fillmore
Steven	Luleå	Buchanan
Robert	Mannheim	King

Built-in validation using Data Annotation

You can validate the data using **data annotation attributes** by setting [SfTreeGrid.GridValidationMode](#) or [TreeGridColumn.GridValidationMode](#) property to `InEdit` or `InView`.

Using different annotations

The numeric type like int, double, decimal properties can be validated using [Range attributes](#).

C#

```
private int orderID;
[Range(1001, 1005, ErrorMessage = "OrderID between 1001 and 1005 alone
processed")]
public int OrderID
{
    get { return orderID; }
    set { orderID = value; }
}
private decimal price;
[Range(typeof(decimal), "12", "20")]
public decimal Price
{
    get { return price; }
    set { price = value; }
}
```

```
get { return price; }
set { price = value; }
}
```

The string type property can be validated using [Required](#), [String Length attributes](#)

C#

```
private string shippingCity;
[Required]
public string ShipCity
{
    get { return shippingCity; }
    set { shippingCity = value; }
}
private string customerName;
[StringLength(17)]
public string CustomerName
{
    get { return customerName; }
    set { customerName = value; }
}
```

The data that has heterogeneous type (combination of number, special character) can be validated using [RegularExpressions](#).

C#

```
[RegularExpressionAttribute(@"^[a-zA-Z]{1,40}$", ErrorMessage="Numbers and
special characters not allowed")]
public string CustomerID
{
    get { return customerId; }
    set { customerId = value; }
}
```

Custom validation through events

You can validate the cells and rows using [CurrentCellValidating](#) and [RowValidating](#) events. SfTreeGrid will not allow user to edit other cell / row if validation failed.

Cell Validation

You can validate the cells using [CurrentCellValidating](#) event when the cell is edited.

[CurrentCellValidating](#) event occurs when the edited cell tries to commit the data or lose the focus.

[TreeGridCurrentCellValidatingEventArgs](#) provides information to [CurrentCellValidating](#) event for validating the cell.

[TreeGridCurrentCellValidatingEventArgs.NewValue](#) returns the edited value and you can set the validation status using [TreeGridCurrentCellValidatingEventArgs.IsValid](#) property.

C#

```
treeGrid.CurrentCellValidating += treeGrid_CurrentCellValidating;
void treeGrid_CurrentCellValidating(object sender,
TreeGridCurrentCellValidatingEventArgs e)
```



```
{
    if (e.NewValue.ToString().Equals("1004"))
    {
        e.IsValid = false;
        e.ErrorMessage = "OrderID 1004 is invalid";
    }
}
```

[SfTreeGrid.CurrentCellValidated](#) event triggered when the cell has finished validating with valid data.

C#

```
treeGrid.CurrentCellValidated += treeGrid_CurrentCellValidated;
void treeGrid_CurrentCellValidated(object sender,
TreeGridCurrentCellValidatedEventArgs e)
{
}
```

Row Validation

You can validate the row using [RowValidating](#) event when the cell is edited. The [RowValidating](#) event occurs when the edited cells tries to commit the row data or lose the focus.

[TreeGridRowValidatingEventArgs](#) provides information to [RowValidating](#) event for validating row.

[TreeGridRowValidatingEventArgs.RowData](#) returns the edited value and you can set the validation status using [TreeGridRowValidatingEventArgs.IsValid](#) property.

C#

```
treeGrid.RowValidating += treeGrid_RowValidating;
void treeGrid_RowValidating(object sender, TreeGridRowValidatingEventArgs e)
{
    var data = e.RowData.GetType().GetProperty("FirstName").GetValue(e.RowData);
    if (data.ToString().Equals("Andrew"))
    {
        e.IsValid = false;
        e.ErrorMessages.Add("FirstName", "FirstName Andrew is invalid");
    }
}
```

[SfTreeGrid.RowValidated](#) event triggered when the row has finished validating with valid row data.

C#

```
treeGrid.RowValidated += treeGrid_RowValidated;
void treeGrid_RowValidated(object sender, TreeGridRowValidatedEventArgs e)
{
}
```

Error icon and tip customization

Customizing error icon

You can customize the error icon by editing [TreeGridCell](#) control template. If you want to customize the error icon in expander column, you need to edit the control template of [TreeGridExpanderCell](#).

Change the shape of error icon

You can change the validation error template shape of the **TreeGridCell** by changing the **Data** property of the path in the **PART_InValidCellBorder** of **TreeGridCell**.

XML

```
<Window.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.SfGrid.WPF;component/Control/Themes/Generic.xaml" />
</ResourceDictionary.MergedDictionaries>
<Style TargetType="syncfusion:TreeGridCell">
<Setter Property="Background" Value="Transparent" />
<Setter Property="BorderBrush" Value="Gray" />
<Setter Property="BorderThickness" Value="0,0,1,1" />
<Setter Property="Padding" Value="0" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:TreeGridCell">
<Grid x:Name="Root">
<Border x:Name="PART_GridCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<ContentPresenter />
</Border>
<Border x:Name="PART_CurrentCellBorder"
Margin="0,0,1,1"
Background="Transparent"
BorderBrush="{TemplateBinding CurrentCellBorderBrush}"
BorderThickness="{TemplateBinding CurrentCellBorderThickness}"
IsHitTestVisible="False"
Visibility="Collapsed" />
<Border x:Name="PART_InValidCellBorder"
Width="10"
Height="10"
HorizontalAlignment="Right"
VerticalAlignment="Top"
SnapsToDevicePixels="True"
Visibility="Collapsed">
<ToolTipService.ToolTip>
<ToolTip Background="#FFDB00C"
Placement="Right"
PlacementRectangle="20,0,0,0"
Tag="{TemplateBinding ErrorMessage}"
Template="{StaticResource ValidationToolTipTemplate}" />
</ToolTipService.ToolTip>
<Path Data="M15.396557,23.044006C14.220558,23.044006 13.268559,23.886993
13.268559,24.927994 13.268559,25.975006 14.220558,26.817001
15.396557,26.817001 16.572557,26.817001 17.523547,25.975006
17.523547,24.927994 17.523547,23.886993 16.572557,23.044006
15.396557,23.044006z M15.467541,5.181999C15.447552,5.181999
15.436566,5.182998 15.436566,5.182998 13.118533,5.5049973
13.055545,7.3330002 13.055545,7.3330002L13.055545,9.2929993
13.626531,16.539001C13.983558,18.357002 14.243538,19.020004
```

```

14.243538,19.020004 15.275555,19.975006 16.203567,19.25 16.203567,19.25
16.976548,18.565994 17.028552,16.962997 17.028552,16.962997
17.956563,9.2929993 17.696553,7.1029968 17.696553,7.1029968
17.608571,5.2839966 15.823561,5.1849976 15.490551,5.1819992
15.481549,5.1819992 15.473553,5.1819992 15.467541,5.1819992z
M15.56355,0C15.56355,0 21.710574,4.1259995 31.581613,2.8030014
31.581613,2.8030014 33.634629,26.556992 15.56355,32 15.56355,32 -
0.10249132,27.548004 0.00050565118,2.9670029 0.0005058694,2.9670029
10.72555,3.6309967 15.56355,0z"
Fill="Red"
SnapsToDevicePixels="True"
Stretch="Fill" />
</Border>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="IndicationStates">
<VisualState x:Name="NoError" />
<VisualState x:Name="HasError">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CurrentStates">
<VisualState x:Name="Regular" />
<VisualState x:Name="Current">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_CurrentCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>
</Window.Resources>

```

FirstName	Country	LastName
⊕ Obama	Italy	Bosh
⊕ John	Germany	Adams
⊖ Thomas	Venezuela	Jefferson
Andrew	Finland	Fuller
Theodore	Germany	Hoover
Harry	Germany	Nixon
⊕ Andrew	Finland	Madison
⊕ Ulysses	France	Pierce
⊕ Jimmy	Argentina	Harrison
⊖ Ronald	Sweden	Fillmore
Andrew	Finland	Fuller
Theodore	Germany	Hoover

Change the color of error icon

You can change the validation error template color of the `TreeGridCell` by changing the `Fill` property of the path in the `PART_InValidCellBorder` of `TreeGridCell`.


XML

```
<Window.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.SfGrid.WPF;component/Control/Themes/Generic.xaml" />
</ResourceDictionary.MergedDictionaries>
<Style TargetType="syncfusion:TreeGridCell">
<Setter Property="Background" Value="Transparent" />
<Setter Property="BorderBrush" Value="Gray" />
<Setter Property="BorderThickness" Value="0,0,1,1" />
<Setter Property="Padding" Value="0" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:TreeGridCell">
<Grid x:Name="Root">
<Border x:Name="PART_GridCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<ContentPresenter />
</Border>
<Border x:Name="PART_CurrentCellBorder"
Margin="0,0,1,1"
Background="Transparent"
BorderBrush="{TemplateBinding CurrentCellBorderBrush}"
BorderThickness="{TemplateBinding CurrentCellBorderThickness}"
```

```

IsHitTestVisible="False"
Visibility="Collapsed" />
<Border x:Name="PART_InValidCellBorder"
Width="10"
Height="10"
HorizontalAlignment="Right"
VerticalAlignment="Top"
SnapsToDevicePixels="True"
Visibility="Collapsed">
<ToolTipService.ToolTip>
<ToolTip Background="#FFDB000C"
Placement="Right"
PlacementRectangle="20,0,0,0"
Tag="{TemplateBinding ErrorMessage}"
Template="{StaticResource ValidationToolTipTemplate}" />
</ToolTipService.ToolTip>
<Path Data="M0.5,0.5 L12.652698,0.5 12.652698,12.068006 z"
Fill="Orange"
SnapsToDevicePixels="True"
Stretch="Fill" />
</Border>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="IndicationStates">
<VisualState x:Name="NoError" />
<VisualState x:Name="HasError">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CurrentStates">
<VisualState x:Name="Regular" />
<VisualState x:Name="Current">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_CurrentCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>
</Window.Resources>

```



FirstName	Country	LastName
⊕ Obama	Italy	Bosh
⊕ John	Germany	Adams
⊖ Thomas	Venezuela	Jefferson
Andrew	Finland	Fuller
Theodore	Germany	Hoover
Harry	Germany	Nixon
⊕ Andrew	Finland	Madison
⊕ Ulysses	France	Pierce
⊕ Jimmy	Argentina	Harrison
⊖ Ronald	Sweden	Fillmore
Andrew	Finland	Fuller
Theodore	Germany	Hoover

[Change the cursor on error icon](#)

You can change the validation error template cursor of the `TreeGridCell` by changing the `Cursor` property of the path codes in the `PART_InvalidCellBorder` of `TreeGridCell`.

XML

```
<Window.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.SfGrid.WPF;component/Control/Themes/Generic.xaml" />
</ResourceDictionary.MergedDictionaries>
<Style TargetType="syncfusion:TreeGridCell">
<Setter Property="Background" Value="Transparent" />
<Setter Property="BorderBrush" Value="Gray" />
<Setter Property="BorderThickness" Value="0,0,1,1" />
<Setter Property="Padding" Value="0" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:TreeGridCell">
<Grid x:Name="Root">
<Border x:Name="PART_GridCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<ContentPresenter />
</Border>
<Border x:Name="PART_CurrentCellBorder"
Margin="0,0,1,1"
Background="Transparent"
```

```

BorderBrush="{TemplateBinding CurrentCellBorderBrush}"
BorderThickness="{TemplateBinding CurrentCellBorderThickness}"
IsHitTestVisible="False"
Visibility="Collapsed" />
<Border x:Name="PART_InValidCellBorder"
Width="10"
Height="10"
HorizontalAlignment="Right"
VerticalAlignment="Top"
SnapsToDevicePixels="True"
Visibility="Collapsed">
<ToolTipService.ToolTip>
<ToolTip Background="#FFDB000C"
Placement="Right"
PlacementRectangle="20,0,0,0"
Tag="{TemplateBinding ErrorMessage}"
Template="{StaticResource ValidationToolTipTemplate}" />
</ToolTipService.ToolTip>
<Path Cursor="Hand"
Data="M0.5,0.5 L12.652698,0.5 12.652698,12.068006 z"
Fill="Red"
SnapsToDevicePixels="True"
Stretch="Fill" />
</Border>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="IndicationStates">
<VisualState x:Name="NoError" />
<VisualState x:Name="HasError">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CurrentStates">
<VisualState x:Name="Regular" />
<VisualState x:Name="Current">
<Storyboard>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_CurrentCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

```
</ResourceDictionary>
</Window.Resources>
```

FirstName	Country	LastName
Obama	Italy	Bosh
John	Germany	Adams
Thomas	Venezuela	Jefferson
Andrew	Finland	Madison
Ulysses	France	Pierce
Jimmy	Argentina	Harrison
Ronald	Denmark	Fillmore
Steven	Finland	Buchanan
Robert	Italy	King
Ronald	Sweden	Fillmore
Andrew	Finland	Fuller
Theodore	Germany	Hoover

Customizing error tip

You can customize the error tip by editing the style of `ValidationToolTipTemplate`. Get the style of `ValidationToolTipTemplate` by editing the `TreeGridCell` style.

Change the background and foreground color of error tip

You can change the error tip background color by setting `Background` property of the border in `ValidationToolTipTemplate`. The error tip foreground color can be changed by setting `Foreground` property of the `TextBlock` in 'ValidationToolTipTemplate'.

XML

```
<ControlTemplate x:Key="ValidationToolTipTemplate">
  <Grid x:Name="Root"
    Margin="5,0"
    Opacity="0"
    RenderTransformOrigin="0,0">
    <Grid.RenderTransform>
      <TranslateTransform x:Name="transform" X="-25" />
    </Grid.RenderTransform>
    <Border Margin="4,4,-4,-4"
      Background="#052A2E31"
      CornerRadius="5" />
    <Border Margin="3,3,-3,-3"
      Background="#152A2E31"
      CornerRadius="4" />
    <Border Margin="2,2,-2,-2"
      Background="#252A2E31"
      CornerRadius="3" />
  </Grid>
</ControlTemplate>
```



```

<Border Margin="1,1,-1,-1"
Background="#352A2E31"
CornerRadius="2" />
<Border Background="Orange" CornerRadius="2" />
<Border CornerRadius="2">
<TextBlock MaxWidth="250"
Margin="8,4,8,4"
Foreground="Black"
Text="{TemplateBinding Tag}"
TextWrapping="Wrap"
UseLayoutRounding="false" />
</Border>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup Name="OpenStates">
<VisualStateGroup.Transitions>
<VisualTransition GeneratedDuration="0" />
<VisualTransition GeneratedDuration="0:0:0.2" To="Open">
<Storyboard>
<DoubleAnimation Duration="0:0:0.2"
Storyboard.TargetName="transform"
Storyboard.TargetProperty="X"
To="0">
<DoubleAnimation.EasingFunction>
<BackEase Amplitude=".3" EasingMode="EaseOut" />
</DoubleAnimation.EasingFunction>
</DoubleAnimation>
<DoubleAnimation Duration="0:0:0.2"
Storyboard.TargetName="Root"
Storyboard.TargetProperty="Opacity"
To="1" />
</Storyboard>
</VisualTransition>
</VisualStateGroup.Transitions>
<VisualState x:Name="Closed">
<Storyboard>
<DoubleAnimation Duration="0"
Storyboard.TargetName="Root"
Storyboard.TargetProperty="Opacity"
To="0" />
</Storyboard>
</VisualState>
<VisualState x:Name="Open">
<Storyboard>
<DoubleAnimation Duration="0"
Storyboard.TargetName="transform"
Storyboard.TargetProperty="X"
To="0" />
<DoubleAnimation Duration="0"
Storyboard.TargetName="Root"
Storyboard.TargetProperty="Opacity"
To="1" />
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</Grid>
</ControlTemplate>

```

FirstName	Country	LastName
⊕ Obama	Italy	Bosh
⊕ John	Germany	Adams
⊖ Thomas	Venezuela	Jefferson
Andrew	Finland	Fuller
Theodore	Germany	Hoover
Harry	Germany	Nixon
⊕ Andrew	Finland	Madison
⊕ Ulysses	France	Pierce
⊕ Jimmy	Argentina	Harrison
⊖ Ronald	Sweden	Fillmore
Andrew	Finland	Fuller

Showing error details in RowHeader

SfTreeGrid supports to show the error icon in [TreeGridRowHeaderCell](#) based on [IDataErrorInfo.Error](#) or [INotifyDataErrorInfo.HasErrors](#) property.

Using IDataErrorInfo

You can show the error information in row header by setting [IDataErrorInfo.Error](#). `IDataErrorInfo.Error` will be displayed as error message in tooltip.

C#

```
[Display(AutoGenerateField = false)]
public string Error
{
    get
    {
        if (this.Country.Contains("Germany") || this.Country.Contains("UK"))
            return "Delivery not available for the country " + this.Country;
        return string.Empty;
    }
}
```

	FirstName	Country	LastName
+	Obama	Italy	Bosh
✗			Adams
-	Thomas	Venezuela	Jefferson
	Andrew	Finland	Fuller
✗	Theodore	Germany	Hoover
✗	Harry	Germany	Nixon
+	Andrew	Finland	Madison
+	Ulysses	France	Pierce
+	Jimmy	Argentina	Harrison
-	Ronald	Sweden	Fillmore
	Andrew	Finland	Fuller
✗	Theodore	Germany	Hoover

Using INotifyDataErrorInfo

You can show the error information in row header by setting [INotifyDataErrorInfo.HasErrors](#). By default, error message "Row Containing Error" will be displayed. You can change this by changing `RowErrorMessage` in the **resx** file.

C#

```
[Display(AutoGenerateField = false)]
public bool HasErrors
{
    get
    {
        if (this.ShipCity.Contains("Mexico D.F. "))
            return true;
        return false;
    }
}
```

	FirstName	City	LastName
+	Obama	Mannheim	Bosh
X	Thomas	México D.F.	Adams
-	Thomas	Tsawassen	Jefferson
	Andrew	Luleå	Fuller
	Harry	Mannheim	Nixon
-	Andrew	Luleå	Madison
	Ronald	Luleå	Fillmore
	Steven	Luleå	Buchanan
	Robert	Mannheim	King
+	Ulysses	Luleå	Pierce
+	Jimmy	Berlin	Harrison

Validation with CheckBox column

SfTreeGrid doesn't support to validate the [TreeGridCheckBoxColumn](#) through validating events. You can validate the check box column value by setting `TreeGridValidationHelper.IsCurrentCellValidated` and `TreeGridValidationHelper.IsCurrentRowValidated` static properties by calling [SetCurrentRowValidated](#) and [SetCurrentCellValidated](#) methods from [TreeGridValidationHelper](#).

C#

```
using Syncfusion.UI.Xaml.TreeGrid.Helpers;
treeGrid.CurrentCellValidating += treeGrid_CurrentCellValidating;
void treeGrid_CurrentCellValidating(object sender,
TreeGridCurrentCellValidatingEventArgs e)
{
    if (!(bool)e.NewValue)
    {
        e.IsValid = false;
        e.ErrorMessage = "Unavailable";
    }
}
treeGrid.RowValidating += treeGrid_RowValidating;
void treeGrid_RowValidating(object sender, TreeGridRowValidatingEventArgs e)
{
    var status =
e.RowData.GetType().GetProperty("Availability").GetValue(e.RowData);
    if (!(bool)status)
    {
        e.IsValid = false;
        e.ErrorMessages.Add("Availability", "Unavailable");
    }
}
treeGrid.CurrentCellValueChanged += treeGrid_CurrentCellValueChanged;
void treeGrid_CurrentCellValueChanged(object sender,
TreeGridCurrentCellValueChangedEventArgs e)
```

```

{
    int columnIndex =
    this.treeGrid.ResolveToGridVisibleColumnIndex(e.RowColumnIndex.ColumnIndex);
    //We are enabling the RowValidating, CellValidating event if the changes
    happen in TreeGridCheckBoxColumn
    if (this.treeGrid.Columns[columnIndex].CellType == "CheckBox")
    {
        this.treeGrid.GetValidationHelper().SetCurrentRowValidated(false);
        this.treeGrid.GetValidationHelper().SetCurrentCellValidated(false);
    }
}

```

FirstName	Availability	City	
Obama	<input type="checkbox"/>	Mannheim	B
John	<input type="checkbox"/> Unavailable	México D.F.	A
Thomas	<input checked="" type="checkbox"/>	Tsawassen	Je
Andrew	<input type="checkbox"/>	Luleå	M
Ulysses	<input checked="" type="checkbox"/>	Luleå	P
Andrew	<input checked="" type="checkbox"/>	Luleå	F
Harry	<input type="checkbox"/>	Mannheim	N
Jimmy	<input type="checkbox"/>	Berlin	H
Ronald	<input type="checkbox"/>	Luleå	F
Steven	<input checked="" type="checkbox"/>	Luleå	B
Robert	<input checked="" type="checkbox"/>	Mannheim	K

Limitations

1. Non-editable columns will not support custom validation except [TreeGridCheckBoxColumn](#).
2. [CurrentCellValidating](#) event will not triggered for [TreeGridTemplateColumn](#).

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Interactive Features in WPF TreeGrid (SfTreeGrid)

Context menu

SfTreeGrid provides an entirely customizable menu to expose the functionalities on user interface. You can create context menus for different rows in an efficient manner.

The following code sample shows the context menu with command bindings.

XML

```
<ContextMenu Style="{x:Null}">
```

```
<MenuItem Command="{Binding Copy, Source={StaticResource viewModel}}"
CommandParameter="{Binding}" Header="Copy">
</MenuItem>
</ContextMenu>
```

C#

```
public class BaseCommand : ICommand
{
    #region Fields
    readonly Action<object> _execute;
    readonly Predicate<object> _canExecute;
    #endregion
    #region Constructors
    /// <summary>
    /// Creates a new command that can always execute.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    public BaseCommand(Action<object> execute)
    : this(execute, null)
    {
    }
    /// <summary>
    /// Creates a new command.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    /// <param name="canExecute">The execution status logic.</param>
    public BaseCommand(Action<object> execute, Predicate<object> canExecute)
    {
        if (execute == null)
            throw new ArgumentNullException("execute");
        _execute = execute;
        _canExecute = canExecute;
    }
    #endregion
    #region ICommand Members
    public bool CanExecute(object parameter)
    {
        return _canExecute == null ? true : _canExecute(parameter);
    }
    public event EventHandler CanExecuteChanged
    {
        add { CommandManager.RequerySuggested += value; }
        remove { CommandManager.RequerySuggested -= value; }
    }
    public void Execute(object parameter)
    {
        _execute(parameter);
    }
    #endregion
}

public class EmployeeInfoViewModel : INotifyPropertyChanged
{
    public EmployeeInfoViewModel()
    {
        CopyCommand = new ContextMenuDemo.BaseCommand(ShowMessage);
    }
}
```

```
}  
private ICommand copyCommand  
public ICommand CopyCommand  
{  
    get  
    {  
        return copyCommand;  
    }  
    set  
    {  
        copyCommand = value;  
    }  
}  
public void ShowMessage(object obj)  
{  
    if (obj is GridRecordContextMenuInfo)  
    {  
        var grid = (obj as GridRecordContextMenuInfo).DataGrid;  
        grid.GridCopyPaste.Copy();  
    }  
}
```

ContextMenu based on rows

You can set different context menus to SfTreeGrid based on rows.

ContextMenu for nodes

You can set the context menu to data rows using the [SfTreeGrid.RecordContextMenu](#) property.

XML

```
<syncfusion:SfTreeGrid.RecordContextMenu>  
    <ContextMenu>  
        <MenuItem x:Name="Cut" Header="Cut" />  
        <MenuItem x:Name="Copy" Header="Copy" />  
        <MenuItem x:Name="Paste" Header="Paste" />  
        <MenuItem x:Name="Delete" Header="Delete" />  
    </ContextMenu>  
</syncfusion:SfTreeGrid.RecordContextMenu>
```

C#

```
this.treeGrid.RecordContextMenu = new ContextMenu();  
this.treeGrid.RecordContextMenu.Items.Add(new MenuItem() { Header = "Cut" });  
this.treeGrid.RecordContextMenu.Items.Add(new MenuItem() { Header = "Copy" });  
this.treeGrid.RecordContextMenu.Items.Add(new MenuItem() { Header = "Paste" });
```

Id	First Name	Last Name	Availability
0	<input type="checkbox"/> Chester	Buchanan	<input checked="" type="checkbox"/>
1	<input checked="" type="checkbox"/> Abraham	Buchanan	<input checked="" type="checkbox"/>
157	<input checked="" type="checkbox"/> Martin	Buchanan	<input checked="" type="checkbox"/>
313	<input checked="" type="checkbox"/> Gerald	Buchanan	<input checked="" type="checkbox"/>
469	<input checked="" type="checkbox"/> Herbert	Buchanan	<input checked="" type="checkbox"/>
625	<input checked="" type="checkbox"/> Theodore		<input checked="" type="checkbox"/>
781	<input type="checkbox"/> George		<input checked="" type="checkbox"/>
782	<input checked="" type="checkbox"/> George		<input checked="" type="checkbox"/>
938	<input checked="" type="checkbox"/> Benjamin	Madison	<input checked="" type="checkbox"/>
1094	<input checked="" type="checkbox"/> Barack	Madison	<input checked="" type="checkbox"/>
1250	<input checked="" type="checkbox"/> Ulysses	Madison	<input checked="" type="checkbox"/>
1406	<input checked="" type="checkbox"/> James	Madison	<input checked="" type="checkbox"/>
1562	<input type="checkbox"/> Zachary	Buchanan	<input checked="" type="checkbox"/>
1563	<input checked="" type="checkbox"/> Herbert	Buchanan	<input checked="" type="checkbox"/>
1719	<input checked="" type="checkbox"/> Chester	Buchanan	<input checked="" type="checkbox"/>

When binding the menu item using CommandBinding, you can get the command parameter as [TreeGridNodeContextMenuInfo](#), which contains nodes of the corresponding row.

XML

```
<syncfusion:SfTreeGrid.RecordContextMenu>
  <MenuItem Command="{Binding Copy, Source={StaticResource viewModel}}"
    CommandParameter="{Binding}"
    Header="Copy">
  </MenuItem>
</syncfusion:SfTreeGrid.RecordContextMenu>
```

C#

```
private static void OnCopyClicked(object obj)
{
    var contextMenuInfo = obj as TreeGridNodeContextMenuInfo;
    if (contextMenuInfo == null)
        return;
    var grid = contextMenuInfo.TreeGrid;
    grid.GridCopyOption = GridCopyOption.CopyData;
    grid.TreeGridCopyPaste.Copy();
}
```

ContextMenu for header

You can set the context menu to header using the [SfTreeGrid.HeaderContextMenu](#) property.

XML

```
<syncfusion:SfTreeGrid.HeaderContextMenu>
  <ContextMenu>
    <MenuItem x:Name=" SortAscending " Header="SortAscending" />
    <MenuItem x:Name=" SortDescending " Header="SortDescending" />
    <MenuItem x:Name=" ClearSorting " Header="ClearSorting" />
    <MenuItem x:Name=" ClearFiltering " Header="ClearFiltering" />
  </ContextMenu>
</syncfusion:SfTreeGrid.HeaderContextMenu>
```


C#

```

this.treeGrid.HeaderContextMenu = new ContextMenu();
this.treeGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header =
"SortAscending" });
this.treeGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header =
"SortDescending" });
this.treeGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header =
"ClearSorting" });
this.treeGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header =
"ClearFiltering " });

```

Id	First Name	Last Name	Availability
0	<input type="checkbox"/> Chester		<input checked="" type="checkbox"/>
1	<input type="checkbox"/> Abraham		<input checked="" type="checkbox"/>
157	<input type="checkbox"/> Martin		<input checked="" type="checkbox"/>
313	<input type="checkbox"/> Gerald		<input checked="" type="checkbox"/>
469	<input type="checkbox"/> Herbert	Buchanan	<input checked="" type="checkbox"/>
625	<input type="checkbox"/> Theodore	Buchanan	<input checked="" type="checkbox"/>
781	<input type="checkbox"/> George	Madison	<input checked="" type="checkbox"/>
782	<input type="checkbox"/> George	Madison	<input checked="" type="checkbox"/>
938	<input type="checkbox"/> Benjamin	Madison	<input checked="" type="checkbox"/>
1094	<input type="checkbox"/> Barack	Madison	<input checked="" type="checkbox"/>
1250	<input type="checkbox"/> Ulysses	Madison	<input checked="" type="checkbox"/>
1406	<input type="checkbox"/> James	Madison	<input checked="" type="checkbox"/>
1562	<input type="checkbox"/> Zachary	Buchanan	<input checked="" type="checkbox"/>

When binding the menu item using CommandBinding, you can get the parameter as [TreeGridColumnContextMenuInfo](#), which contains a particular GridColumn.

XML

```

<syncfusion:SfTreeGrid.HeaderContextMenu>
  <MenuItem Command="{Binding Source={x:Static <MenuItem Command="{Binding
SortAscending, Source={StaticResource viewModel}}}"
Header="Sort Ascending">
</MenuItem>
</syncfusion:SfTreeGrid.HeaderContextMenu>

```

C#

```

private static void OnSortAscendingClicked(object obj)
{
    var contextMenuInfo = obj as TreeGridColumnContextMenuInfo;
    if (contextMenuInfo == null)
        return;
    var grid = contextMenuInfo.TreeGrid;
    var column = contextMenuInfo.Column;
    grid.SortColumnDescriptions.Clear();
    grid.SortColumnDescriptions.Add(new SortColumnDescription() { ColumnName =
column.MappingName, SortDirection = ListSortDirection.Ascending });
}

```

ContextMenu for expander

You can set the context menu to header using the [SfTreeGrid.ExpanderContextMenu](#) property.

XML

```
<syncfusion:SfTreeGrid.ExpanderContextMenu>
<ContextMenu>
<MenuItem x:Name=" Expand " Header="SortAscending" />
<MenuItem x:Name=" Collapse" Header="Collapse" />
</ContextMenu>
</syncfusion:SfTreeGrid.ExpanderContextMenu>
```

C#

```
this.treeGrid.ExpanderContextMenu = new ContextMenu();
this.treeGrid.ExpanderContextMenu.Items.Add(new MenuItem() { Header = "
Expand" });
this.treeGrid.ExpanderContextMenu.Items.Add(new MenuItem() { Header =
"Collapse" });
```

When binding the menu item using CommandBinding, you can get the parameter as [TreeGridColumnContextMenuInfo](#), which contains the expander node.

XML

```
<syncfusion:SfTreeGrid.HeaderContextMenu>
<MenuItem Command="{Binding Expand, Source={StaticResource viewModel}}"
CommandParameter="{Binding}"
Header="Expand" />
</syncfusion:SfTreeGrid.HeaderContextMenu>
```

C#

```
private static void OnExpandClicked(object obj)
{
var contextMenuInfo = obj as TreeGridColumnContextMenuInfo;
if (contextMenuInfo == null)
return;
var grid = contextMenuInfo.TreeGrid;
grid.ExpandNode(contextMenuInfo.TreeNode);
}
```

Events

The [TreeGridColumnContextMenuOpening](#) event occurs when opening the context menu in SfTreeGrid. [TreeGridColumnContextMenuEventArgs](#) has the following members, which provide information about the TreeGridColumnContextMenuOpening event:

[ContextMenu](#) – Gets the corresponding context menu.

[ContextMenuInfo](#) – Returns the context menu info based on the row that opens the context menu.

[ContextMenuType](#) – Returns the type of context menu.

[RowColumnIndex](#) – RowColumnIndex of the context menu, which is currently going to be opened. The RowColumnIndex is updated only for the RecordContextMenu and remains empty.

[Handled](#) - Indicates whether the TreeGridContextMenuOpening event is handled or not.

Customizing ContextMenus

Changing the menu item when ContextMenu opens

You can use the [TreeGridContextMenuOpening](#) event to change the menu item when the context menu opens.

XML

```
<syncfusion:SfTreeGrid.RecordContextMenu>
<ContextMenu>
<MenuItem Command="{Binding Cut, Source={StaticResource viewModel}}"
CommandParameter="{Binding}"
Header="Cut">
</MenuItem>
<MenuItem Command="{Binding Copy, Source={StaticResource viewModel}}"
CommandParameter="{Binding}"
Header="Copy">
</MenuItem>
<MenuItem Command="{Binding Paste, Source={StaticResource viewModel}}"
CommandParameter="{Binding}"
Header="Paste">
</MenuItem>
</ContextMenu>
</syncfusion:SfTreeGrid.RecordContextMenu>
```

C#

```
this.treeGrid.TreeGridContextMenuOpening += treeGrid_
TreeGridContextMenuOpening;
void dataGrid_TreeGridContextMenuOpening (object sender,
TreeGridContextMenuEventArgs e)
{
e.ContextMenu.Items.Clear();
if(e.ContextMenuType == ContextMenuType.RecordCell)
{
e.ContextMenu.Items.Add(new MenuItem() { Header="Record"});
e.ContextMenu.Items.Add(new MenuItem() { Header = "Data" });
}
}
```

Id	First Name	Last Name	Availability
0	<input type="checkbox"/> Chester	Buchanan	<input checked="" type="checkbox"/>
1	<input checked="" type="checkbox"/> Abraham	Buchanan	<input checked="" type="checkbox"/>
157	<input checked="" type="checkbox"/> Martin	Buchanan	<input checked="" type="checkbox"/>
313	<input checked="" type="checkbox"/> Gerald	Buchanan	<input checked="" type="checkbox"/>
469	<input checked="" type="checkbox"/> Herbert		<input checked="" type="checkbox"/>
625	<input checked="" type="checkbox"/> Theodore		<input checked="" type="checkbox"/>
781	<input type="checkbox"/> George	Madison	<input checked="" type="checkbox"/>
782	<input checked="" type="checkbox"/> George	Madison	<input checked="" type="checkbox"/>
938	<input checked="" type="checkbox"/> Benjamin	Madison	<input checked="" type="checkbox"/>
1094	<input checked="" type="checkbox"/> Barack	Madison	<input checked="" type="checkbox"/>
1250	<input checked="" type="checkbox"/> Ulysses	Madison	<input checked="" type="checkbox"/>
1406	<input checked="" type="checkbox"/> James	Madison	<input checked="" type="checkbox"/>
1562	<input type="checkbox"/> Zachary	Buchanan	<input checked="" type="checkbox"/>
1563	<input checked="" type="checkbox"/> Herbert	Buchanan	<input checked="" type="checkbox"/>

Changing background to ContextMenu

You can change the appearance of the context menu by customizing the style with TargetType as ContextMenu.

XML

```
<Style x:Name="ToolTipStyle" TargetType="ContextMenu">
  <Setter Property="BorderThickness" Value="1,1,1,1" />
  <Setter Property="BorderBrush" Value="Red" />
  <Setter Property="Background" Value="LightGreen" />
</Style>
<ContextMenu>
  <MenuItem x:Name="Cut" Header="Expand" />
  <MenuItem x:Name="Copy" Header="Collapse" />
</ContextMenu>
```

Id	First Name	Last Name	Availability
0	<input type="checkbox"/> Chester	Buchanan	<input checked="" type="checkbox"/>
1	<input checked="" type="checkbox"/> Abraham	Buchanan	<input checked="" type="checkbox"/>
157	<input checked="" type="checkbox"/> Martin	Buchanan	<input checked="" type="checkbox"/>
313	<input checked="" type="checkbox"/> Gerald	Buchanan	<input checked="" type="checkbox"/>
469	<input checked="" type="checkbox"/> Herbert	Buchanan	<input checked="" type="checkbox"/>
625	<input checked="" type="checkbox"/> Theodore	Buchanan	<input checked="" type="checkbox"/>
781	<input type="checkbox"/> George	Madison	<input checked="" type="checkbox"/>
782	<input checked="" type="checkbox"/> George	Madison	<input checked="" type="checkbox"/>
938	<input checked="" type="checkbox"/> Benjamin	Madison	<input checked="" type="checkbox"/>
1094	<input checked="" type="checkbox"/> Barack	Madison	<input checked="" type="checkbox"/>
1250	<input checked="" type="checkbox"/> Ulysses	Madison	<input checked="" type="checkbox"/>
1406	<input checked="" type="checkbox"/> James	Madison	<input checked="" type="checkbox"/>
1562	<input type="checkbox"/> Zachary	Buchanan	<input checked="" type="checkbox"/>

Drag and drop row

SfTreeGrid allows drag and drop the rows within and between controls by setting the [AllowDraggingRows](#) and [AllowDrop](#) properties to true. It is also possible to drag and drop the rows between treegrid and other controls such as ListView and TreeView. SfTreeGrid allows dropping rows when [AllowDrop](#) is true and allows dragging when [AllowDraggingRows](#) is true.

XML

```
<syncfusion:SfTreeGrid Name="sfTreeGrid"
AllowDraggingRows="True"
AllowDrop="True"
ChildPropertyName="ReportsTo"
AutoGenerateColumns="False"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID"
SelfRelationRootValue="-1" />
```

C#

```
sfTreeGrid.AllowDraggingRows = true;
sfTreeGrid.AllowDrop = true;
```

	First Name	Last Name	Employee ID	Title	Salary	ReportsTo
<input type="checkbox"/>	Ferando	Joseph	2	Management	\$2000000.00	-1
	Andrew	Fuller	9	Vice President	\$1200000.00	2
	Janet	Leverling	10	GM	\$1000000.00	2
	Steven	Buchanan	11	Manager	\$900000.00	2
>	<input checked="" type="checkbox"/> Thomas	Jefferson	4	Sales	\$3000000.00	-1
<input type="checkbox"/>	John	Adams	3	Accounts	\$2000000.00	-1
	Nancy	Davolio	12	Accounts Manager	\$850000.00	3
	Margaret		13	Accountant	\$700000.00	3
	Michael		14	Accountant	\$700000.00	3
	Robert		15	Accountant	\$650000.00	3
<input type="checkbox"/>	Andrew	Madison	5	Marketing	\$4000000.00	-1
	Caroline	Patterson	21	Receptionist	\$800000.00	5
	Xavier	Martin	22	Mail Clerk	\$700000.00	5
<input type="checkbox"/>	Ulysses	Pierce	6	HumanResource	\$1500000.00	-1
	Laurent	Pereira	23	HR Manager	\$900000.00	6
	Syed	Abbas	24	HR Assistant	\$650000.00	6
	Amy	Alberts	25	HR Assistant	\$650000.00	6

When dropping, the dragged node(s) can be added above or below as a child node based on its drop position. For example, if you dropped node at the bottom of the targeted node, it will be added below the targeted node. If you drop over the targeted node, it will be added as a child of that targeted node.

	First Name	Last Name	Employee ID	Title	Salary	ReportsTo
[-]	Ferando	Joseph	2	Management	\$2000000.00	-1
	Andrew	Fuller	9	Vice President	\$1200000.00	2
	Janet	Leverling	10	GM	\$1000000.00	2
	Steven	Buchanan	11	Manager	\$900000.00	2
> [+]	Thomas	Jefferson	4	Sales	\$300000.00	-1
[-]	John	Adams	3	Accounts	\$2000000.00	-1
	Nancy	Davolio	12	Accounts Manager	\$850000.00	3
	Margaret	Peacock	13	Accountant	\$700000.00	3
	Michael	Sujama	14	Accountant	\$700000.00	3
	Robert	Kinn	15	Accountant	\$650000.00	3
[-]	Andrew	Madison	5	Marketing	\$4000000.00	-1
	Caroline	Patterson	21	Receptionist	\$800000.00	5
	Xavier	Martin	22	Mail Clerk	\$700000.00	5
[-]	Ulysses	Pierce	6	HumanResource	\$1500000.00	-1
	Laurent	Pereira	23	HR Manager	\$900000.00	6
	Syed	Abbas	24	HR Assistant	\$650000.00	6
	Amy	Alberts	25	HR Assistant	\$650000.00	6

Note: Drag indicators will not be shown when drop position is set to “Drop as child”.

Drag multiple nodes

SfTreeGrid allows users to drag multiple selected nodes. You can enable multiple selection by setting the [SfTreeGrid.SelectionMode](#) property to **Multiple** or **Extended**.

	First Name	Last Name	Employee ID	Title	Salary	ReportsTo
[-]	Ferando	Joseph	2	Management	\$2000000.00	-1
	Andrew	Fuller	9	Vice President	\$1200000.00	2
	Janet	Leverling	10	GM	\$1000000.00	2
> [-]	Steven	Buchanan	11	Manager	\$900000.00	2
[-]	John	Adams	3	Accounts	\$2000000.00	-1
	Nancy	Davolio	12	Accounts Manager	\$850000.00	3
	Margaret	Peacock	13	Accountant	\$700000.00	3
	Michael	Sujama	14	Accountant	\$700000.00	3
	Robert	Kinn	15	Accountant	\$650000.00	3
	Thomas	Jefferson	4	Sales	\$300000.00	-1
	Laura	Brid	16	Sales Manager	\$900000.00	4
	Anne	Dodsworth	17	Sales Representative	\$800000.00	4
	Albert	Hellstern	18	Sales Representative	\$750000.00	4
	Tim	Smith	19	Sales Representative	\$700000.00	4
	Justin	Brid	20	Sales Representative	\$700000.00	4
[-]	Andrew	Madison	5	Marketing	\$4000000.00	-1
	Caroline	Patterson	21	Receptionist	\$800000.00	5
	Xavier	Martin	22	Mail Clerk	\$700000.00	5
[-]	Ulysses	Pierce	6	HumanResource	\$1500000.00	-1
	Laurent	Pereira	23	HR Manager	\$900000.00	6
	Syed	Abbas	24	HR Assistant	\$650000.00	6
	Amy	Alberts	25	HR Assistant	\$650000.00	6

Events

SfTreeGrid triggers the following events when drag and drop:

DragStart

DragStart: Occurs when you start to drag the node in treegrid. The [TreeGridRowDragStartEventArgs](#) has the following member, which provides information for the DragStart event.

DraggingNodes: Gets the [TreeNode](#) that contains the data associated when dragging the rows.

C#

```
sfTreeGrid.RowDragDropController.DragStart +=  
RowDragDropController_DragStart;  
private void RowDragDropController_DragStart(object sender,  
Syncfusion.UI.Xaml.TreeGrid.TreeGridRowDragStartEventArgs e)  
{  
}
```

DragOver

DragOver: Occurs continuously when tree node is dragged within the the target treegrid. The [TreeGridRowDragStartEventArgs](#) has the following members, which provide information for the **DragOver** event:

Data: Gets or sets a data object that contains data associated when dragging the rows.

DropPosition: Gets a value that indicates the drop position based on the dropped location.

IsFromOutSideSource: Gets a value that indicates whether the dragging item is from the same TreeGrid.

ShowDragUI: Gets or sets a value that indicates the default Dragging UI.

TargetNode: Gets a value that indicates the target node, which is going to be dropped.

C#

```
sfTreeGrid.RowDragDropController.DragOver += RowDragDropController_DragOver;  
private void RowDragDropController_DragOver(object sender,  
Syncfusion.UI.Xaml.TreeGrid.TreeGridRowDragOverEventArgs e)  
{  
}
```

DragLeave

DragLeave: Occurs when perform a drag-and-drop operation. The [TreeGridRowDragStartEventArgs](#) has the following members, which provide information for the **DragLeave** event:

Data: Gets or sets a data object that contains the data associated when dragging the rows.

DropPosition: Gets a value that indicates the drop position based on the dropped location.

IsFromOutSideSource: Gets a value that indicates whether the dragging item is from the same TreeGrid.

TargetNode: Gets a value that indicates the target, node which is going to be dropped.

C#

```
this.sfTreeGrid.RowDragDropController.DragLeave +=  
RowDragDropController_DragLeave;  
private void RowDragDropController_DragLeave(object sender,  
Syncfusion.UI.Xaml.TreeGrid.TreeGridRowDragLeaveEventArgs e)  
{  
}
```

Drop

Drop: Occurs when a record is dropped within the target treegrid. The [TreeGridRowDragStartEventArgs](#) has the following members, which provide information for the **Drop** event:

[Data](#): Gets or sets a data object that contains the data associated while dragging the rows.

[DraggingNodes](#): Gets the tree node, which contains the data associated when dragging the rows.

[DropPosition](#): Gets a value that indicates the drop position based on the dropped location.

[IsFromOutSideSource](#): Gets a value that indicates whether the dragging item is from the same treegrid.

[TargetNode](#): Gets a value that indicates the target node, which is going to be dropped.

C#

```
this.sfTreeGrid.RowDragDropController.Drop += RowDragDropController_Drop;  
private void RowDragDropController_Drop(object sender,  
Syncfusion.UI.Xaml.TreeGrid.TreeGridRowDropEventArgs e)  
{  
}
```

Dropped

[Dropped](#): Occurs when a record is dropped within the target treegrid. The

[TreeGridRowDragStartEventArgs](#) has the following members, which provide information for the **Dropped** event:

[Data](#): Gets or sets a data object that contains the data associated when dragging the rows.

[DropPosition](#): Gets a value that indicates the drop position based on the dropped location.

[IsFromOutSideSource](#): Gets a value that indicates whether the dragging item is from the same treegrid.

[TargetNode](#): Gets a value that indicates the target node, which is going to be dropped.

C#

```
this.sfTreeGrid.RowDragDropController.Dropped +=  
RowDragDropController_Dropped;  
private void RowDragDropController_Dropped(object sender,  
Syncfusion.UI.Xaml.TreeGrid.TreeGridRowDroppedEventArgs e)  
{  
}
```

Drag and drop between SfTreeGrid and other controls

Drag and drop between ListView and SfTreeGrid

You can drag and drop the items between list view and treegrid by wiring the [Drop](#) event from the [TreeGridRowDragDropController](#) class.

C#

```
this.AssociatedObject.sfTreeGrid.RowDragDropController.Drop +=  
RowDragDropController_Drop;  
private void RowDragDropController_Drop(object sender,  
Syncfusion.UI.Xaml.TreeGrid.TreeGridRowDropEventArgs e)  
{  
    if (e.IsFromOutSideSource)  
    {  
        var item = e.Data.GetData("ListViewRecords") as  
ObservableCollection<object>;  
        var record = item[0] as EmployeeInfo;  
        var dropPosition = e.DropPosition.ToString();
```



```
var newItem = new EmployeeInfo();
var rowIndex =
AssociatedObject.sfTreeGrid.ResolveToRowIndex(e.TargetNode.Item);
int nodeIndex = (int)rowIndex;
if (dropPosition != "None" && rowIndex != -1)
{
    if (AssociatedObject.sfTreeGrid.View is TreeGridSelfRelationalView)
    {
        var treeNode = AssociatedObject.sfTreeGrid.GetNodeAtRowIndex(rowIndex);
        if (treeNode == null)
            return;
        var data = treeNode.Item;
        AssociatedObject.sfTreeGrid.SelectionController.SuspendUpdates();
        var itemIndex = -1;
        TreeNode parentNode = null;
        if (dropPosition == "DropBelow" || dropPosition == "DropAbove")
        {
            parentNode = treeNode.ParentNode;
            if (parentNode == null)
                newItem = new EmployeeInfo() { FirstName = record.FirstName, LastName =
                record.LastName, ID = record.ID, Salary = record.Salary, Title =
                record.Title, ReportsTo = -1 };
            else
            {
                var parent = parentNode.Item as EmployeeInfo;
                newItem = new EmployeeInfo() { FirstName = record.FirstName, LastName =
                record.LastName, ID = record.ID, Salary = record.Salary, Title =
                record.Title, ReportsTo = parent.ID };
            }
        }
        else if (dropPosition == "DropAsChild")
        {
            if (!treeNode.IsExpanded)
                AssociatedObject.sfTreeGrid.ExpandNode(treeNode);
            parentNode = treeNode;
            var parent = parentNode.Item as EmployeeInfo;
            newItem = new EmployeeInfo() { FirstName = record.FirstName, LastName =
            record.LastName, ID = record.ID, Salary = record.Salary, Title =
            record.Title, ReportsTo = parent.ID };
        }
        IList sourceCollection = null;
        if (dropPosition == "DropBelow" || dropPosition == "DropAbove")
        {
            if (treeNode.ParentNode != null)
            {
                var collection =
                AssociatedObject.sfTreeGrid.View.GetPropertyAccessProvider().GetValue(treeNo
                de.ParentNode.Item, AssociatedObject.sfTreeGrid.ChildPropertyName) as
                IEnumerable;
                sourceCollection = GetSourceListCollection(collection);
            }
            else
            {
                sourceCollection =
                GetSourceListCollection(AssociatedObject.sfTreeGrid.View.SourceCollection);
            }
            itemIndex = sourceCollection.IndexOf(data);
        }
    }
}
```

```

if (dropPosition == "DropBelow")
{
    itemIndex += 1;
}
}
else if (dropPosition == "DropAsChild")
{
    var collection =
AssociatedObject.sfTreeGrid.View.GetPropertyAccessProvider().GetValue(data,
AssociatedObject.sfTreeGrid.ChildPropertyName) as IEnumerable;
sourceCollection = GetSourceListCollection(collection);
if (sourceCollection == null)
{
    var list =
data.GetType().GetProperty(AssociatedObject.sfTreeGrid.ChildPropertyName).Pr
opertyType.CreateNew() as IList;
if (list != null)
{
    AssociatedObject.sfTreeGrid.View.GetPropertyAccessProvider().SetValue(treeNo
de.Item, AssociatedObject.sfTreeGrid.ChildPropertyName, list);
sourceCollection = list;
}
}
itemIndex = sourceCollection.Count;
}
sourceCollection.Insert(itemIndex, newItem);
AssociatedObject.sfTreeGrid.SelectionController.ResumeUpdates();
(AssociatedObject.sfTreeGrid.SelectionController as
TreeGridRowSelectionController).RefreshSelection();
e.Handled = true;
}
}
(AssociatedObject.listView.ItemsSource as
ObservableCollection<EmployeeInfo>).Remove(record as EmployeeInfo);
}
}

```

In ListView, wire the PreviewMouseMove and Drop events.

C#

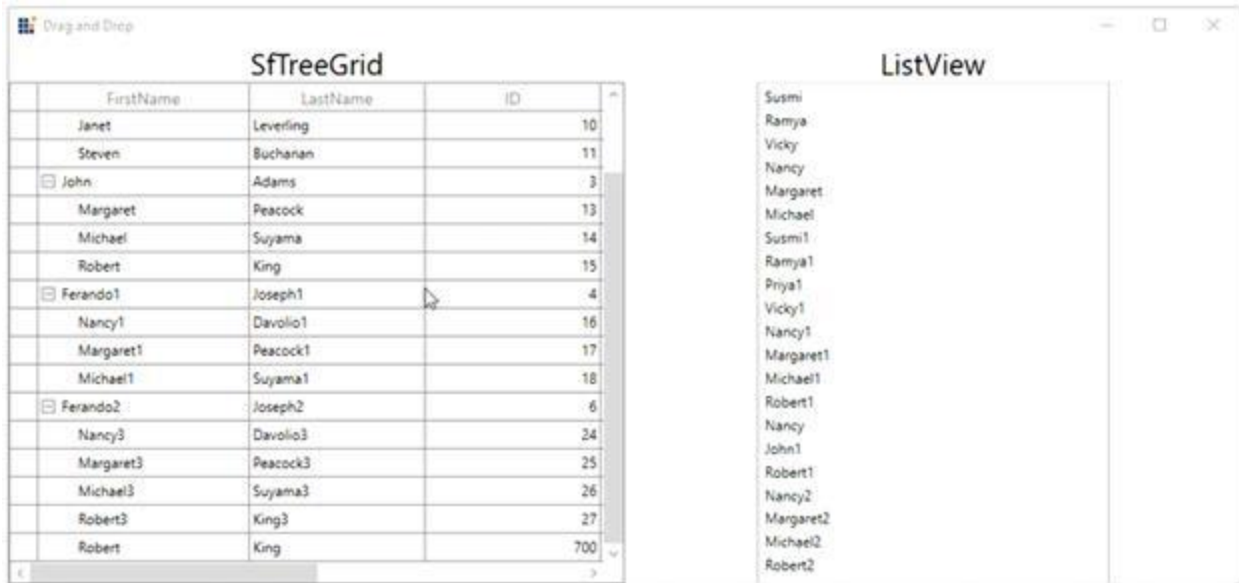
```

this.AssociatedObject.listView.PreviewMouseMove +=
ListView_PreviewMouseMove;
this.AssociatedObject.listView.Drop += ListView_Drop;
private void ListView_Drop(object sender, DragEventArgs e)
{
    ObservableCollection<TreeNode> treeNodes = new
ObservableCollection<TreeNode>();
if (e.Data.GetDataPresent("Nodes"))
treeNodes= e.Data.GetData("Nodes") as ObservableCollection<TreeNode>;
EmployeeInfo item = new EmployeeInfo();
if (treeNodes.Count == 0 || treeNodes==null)
return;
foreach (var node in treeNodes)
{

```

```
(AssociatedObject.sfTreeGrid.ItemsSource as
ObservableCollection<EmployeeInfo>).Remove(node.Item as EmployeeInfo);
if (node.HasChildNodes)
{
list.Add(node.Item as EmployeeInfo);
GetChildNodes(node);
}
else
{
list.Add(node.Item as EmployeeInfo);
}
}
foreach (var listItem in list)
{
(this.AssociatedObject.DataContext as ViewModel).Employee.Add(listItem);
}
list.Clear();
}
private void ListView_PreviewMouseMove(object sender,
System.Windows.Input.MouseEventArgs e)
{
if (e.LeftButton == MouseButtonState.Pressed)
{
ListBox dragSource = null;
var records = new ObservableCollection<object>();
ListBox parent = (ListBox)sender;
dragSource = parent;
object data = GetDataFromListBox(dragSource, e.GetPosition(parent));
records.Add(data);
var dataObject = new DataObject();
dataObject.SetData("ListViewRecords", records);
dataObject.SetData("ListView", this.AssociatedObject.listView);
if (data != null)
{
DragDrop.DoDragDrop(parent, dataObject, DragDropEffects.Move);
}
}
e.Handled = true;
}
```

Sample for dragging and dropping the items between list view and treegrid: [Sample](#).



Drag and drop between TreeViewAdv and SfTreeGrid

You can drag and drop the items between tree view and treegrid by wiring the [Drop](#) and [DragStart](#) events from the [TreeGridRowDragDropController](#) class.

C#

```
private void RowDragDropController_DragStart(object sender,
TreeGridRowDragStartEventArgs e)
{
    e.Handled = true;
    var dataObject = new DataObject();
    dataObject.SetData("SourceTreeGrid", this.AssociatedObject.sfTreeGrid);
    dataObject.SetData("Nodes", e.DraggingNodes);
    foreach (var node in e.DraggingNodes)
    {
        if (node.HasChildNodes)
        {
            records.Add(node.Item as EmployeeInfo);
            GetChildNodes(node);
        }
        else
        {
            records.Add(node.Item as EmployeeInfo);
        }
    }
    dataObject.SetData(records);
    if(records!=null)
    DragDrop.DoDragDrop(this.AssociatedObject.sfTreeGrid, dataObject,
    DragDropEffects.Copy | DragDropEffects.Move);
    records.Clear();
}

private void RowDragDropController_Drop(object sender,
TreeGridRowDropEventArgs e)
{
    if (e.IsFromOutSideSource)
    {
```

```

ObservableCollection<object> item =
e.Data.GetData(typeof(ObservableCollection<object>)) as
ObservableCollection<object>;
var record = item[0] as EmployeeInfo;
var dropPosition = e.DropPosition.ToString();
var newItem = new EmployeeInfo();
var rowIndex =
AssociatedObject.sfTreeGrid.ResolveToRowIndex(e.TargetNode.Item);
int nodeIndex = (int)rowIndex;
if (dropPosition != "None" && rowIndex != -1)
{
if (AssociatedObject.sfTreeGrid.View is TreeGridSelfRelationalView)
{
var treeNode = AssociatedObject.sfTreeGrid.GetNodeAtRowIndex(rowIndex);
if (treeNode == null)
return;
var data = treeNode.Item;
AssociatedObject.sfTreeGrid.SelectionController.SuspendUpdates();
var itemIndex = -1;
TreeNode parentNode = null;
if (dropPosition == "DropBelow" || dropPosition == "DropAbove")
{
parentNode = treeNode.ParentNode;
if (parentNode == null)
newItem = new EmployeeInfo() { FirstName = record.FirstName, LastName =
record.LastName, ID = record.ID, Salary = record.Salary, Title =
record.Title, ReportsTo = -1 };
else
{
var parent = parentNode.Item as EmployeeInfo;
newItem = new EmployeeInfo() { FirstName = record.FirstName, LastName =
record.LastName, ID = record.ID, Salary = record.Salary, Title =
record.Title, ReportsTo = parent.ID };
}
}
else if (dropPosition == "DropAsChild")
{
if (!treeNode.IsExpanded)
AssociatedObject.sfTreeGrid.ExpandNode(treeNode);
parentNode = treeNode;
var parent = parentNode.Item as EmployeeInfo;
newItem = new EmployeeInfo() { FirstName = record.FirstName, LastName =
record.LastName, ID = record.ID, Salary = record.Salary, Title =
record.Title, ReportsTo = parent.ID };
}
IList sourceCollection = null;
if (dropPosition == "DropBelow" || dropPosition == "DropAbove")
{
if (treeNode.ParentNode != null)
{
var collection =
AssociatedObject.sfTreeGrid.View.GetPropertyAccessProvider().GetValue(treeNo
de.ParentNode.Item, AssociatedObject.sfTreeGrid.ChildPropertyName) as
IEnumerable;
sourceCollection = GetSourceListCollection(collection);
}
else

```

```

{
    sourceCollection =
    GetSourceListCollection(AssociatedObject.sfTreeGrid.View.SourceCollection);
}
itemIndex = sourceCollection.IndexOf(data);
if (dropPosition == "DropBelow")
{
    {
        itemIndex += 1;
    }
}
else if (dropPosition == "DropAsChild")
{
    var collection =
    AssociatedObject.sfTreeGrid.View.GetPropertyAccessProvider().GetValue(data,
    AssociatedObject.sfTreeGrid.ChildPropertyName) as IEnumerable;
    sourceCollection = GetSourceListCollection(collection);
    if (sourceCollection == null)
    {
        var list =
        data.GetType().GetProperty(AssociatedObject.sfTreeGrid.ChildPropertyName).Pr
        opertyType.CreateNew() as IList;
        if (list != null)
        {
            AssociatedObject.sfTreeGrid.View.GetPropertyAccessProvider().SetValue(treeNo
            de.Item, AssociatedObject.sfTreeGrid.ChildPropertyName, list);
            sourceCollection = list;
        }
    }
    itemIndex = sourceCollection.Count;
}
sourceCollection.Insert(itemIndex, newItem);
AssociatedObject.sfTreeGrid.SelectionController.ResumeUpdates();
(AssociatedObject.sfTreeGrid.SelectionController as
TreeGridRowSelectionController).RefreshSelection();
e.Handled = true;
}
}
(AssociatedObject.treeview.ItemsSource as
ObservableCollection<EmployeeInfo>).Remove(record as EmployeeInfo);
}
}

```

In TreeViewAdv, you need to wire the Drop event,

C#

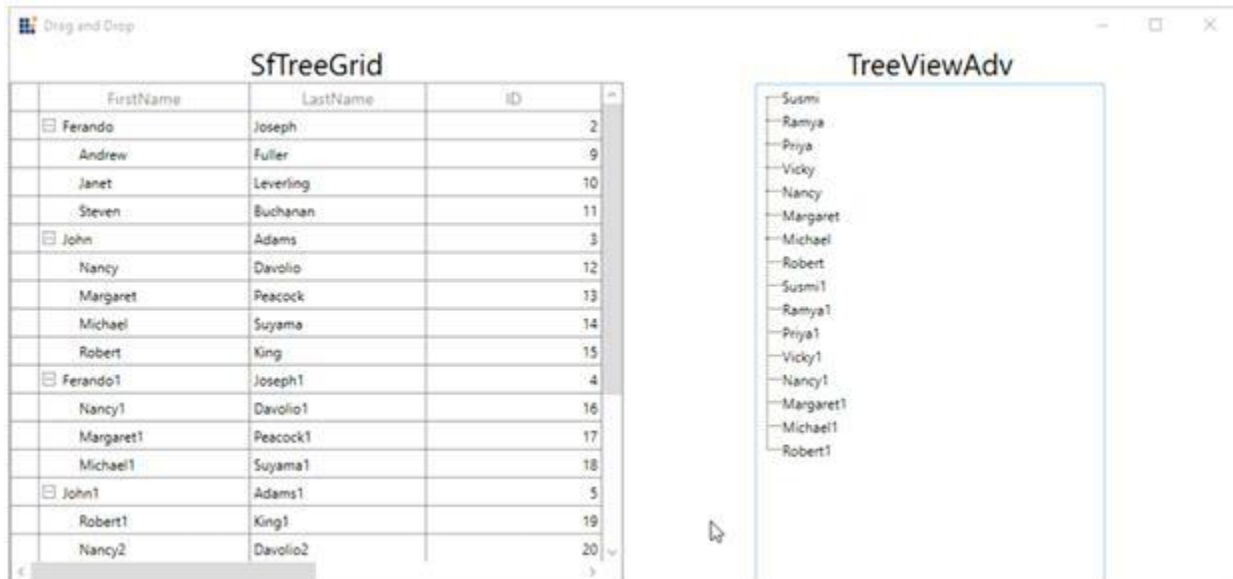
```

private void Treeview_Drop(object sender, DragEventArgs e)
{
    ObservableCollection<TreeNode> treeNodes = new
    ObservableCollection<TreeNode>();
    if (e.Data.GetDataPresent("Nodes"))
    treeNodes = e.Data.GetData("Nodes") as ObservableCollection<TreeNode>;
    EmployeeInfo item = new EmployeeInfo();
    if (treeNodes.Count == 0 || treeNodes == null)
    return;
    foreach (var node in treeNodes)

```

```
{
    (AssociatedObject.sfTreeGrid.ItemsSource as
    ObservableCollection<EmployeeInfo>).Remove(node.Item as EmployeeInfo);
}
```

Sample for dragging and dropping between TreeViewAdv and SfTreeGrid: [Sample](#).



Changing the row drop indicator

By default, the drop position will be indicated with arrows. To change the drop indicator as line, then set the [sfTreeGrid.RowDropIndicatorMode](#) as `Line`.

XML

```
<syncfusion:SfTreeGrid Margin="5"
    Name="sfTreeGrid"
    AutoExpandMode="RootNodesExpanded"
    AllowDraggingRows="True"
    AllowDrop="True"
    ChildPropertyName="ReportsTo"
    ItemsSource="{Binding Employees}"
    ParentPropertyName="ID"
    SelfRelationRootValue="-1"
    RowDropIndicatorMode="Line" />
```

C#

```
sfTreeGrid.RowDropIndicatorMode =
    Syncfusion.UI.Xaml.Grid.DropIndicatorMode.Line;
```

	First Name	Last Name	Employee ID	Title	Salary	Reports To
<input type="checkbox"/>	Ferando	Joseph	2	Management	\$2000000.00	-1
	Andrew	Fuller	9	Vice President	\$1200000.00	2
	Janet	Leverling	10	GM	\$1000000.00	2
	Steven	Buchanan	11	Manager	\$900000.00	2
<input type="checkbox"/>	John	Adams	3	Accounts	\$2000000.00	-1
	Nancy	Davolio	12	Accounts Manager	\$850000.00	3
	Margaret	Peacock	13	Accountant	\$700000.00	3
	Michael	Suyama	14	Accountant	\$700000.00	3
	Robert	King	15	Accountant	\$650000.00	3
<input type="checkbox"/>	Thomas	Jefferson	4		\$300000.00	-1
	Laura	Callahan	16		\$900000.00	4
	Anne	Dodsworth	17	Sales Representative	\$800000.00	4
	Albert	Hellstern	18	Sales Representative	\$750000.00	4
	Tim	Smith	19	Sales Representative	\$700000.00	4
	Justin	Brid	20	Sales Representative	\$700000.00	4
<input type="checkbox"/>	Andrew	Madison	5	Marketing	\$4000000.00	-1
	Caroline	Patterson	21	Receptionist	\$800000.00	5
	Xavier	Martin	22	Mail Clerk	\$700000.00	5
<input type="checkbox"/>	Ulysses	Pierce	6	HumanResource	\$1500000.00	-1

Customizing drag-and-drop rows

SfTreeGrid processes row drag and drop operations in [TreeGridRowDragDropController](#) class. You can customize the row drag-and-drop operations using the events in the [SfTreeGrid.RowDragDropController](#).

Auto expand the node on drag over

When drag over the tree node, if drop position is “Drop as child”, then you can auto expand the corresponding tree node by setting the [TreeGridRowDragDropController.CanAutoExpand](#) to **true**. It is also possible to control the delay in expanding the node when drag over using [TreeGridRowDragDropController.AutoExpandDelay](#) property. Its default value is 3 sec.

C#

```
treeGrid.RowDragDropController.CanAutoExpand = true;
treeGrid.RowDragDropController.AutoExpandDelay = new TimeSpan(0, 0, 2);
```

Customize default drag UI

SfTreeGrid provides the default UI for drag and drop. However, you can customize the drag UI using the [RowDragDropTemplate](#) property.

XML

```
<DataTemplate x:Key="dragdroptemplate">
  <Border x:Name="border" Width="250"
    Background="#ecec"
    BorderBrush="#c8c8c8" Height="60"
    BorderThickness="1.2">
    <Grid VerticalAlignment="Center"
      HorizontalAlignment="Left">
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
```



```

<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<TextBlock Padding="12,0,0,0" Text="Dragging rows count :" FontSize="14"
FontFamily="Segoe UI"
Foreground="#333333" FontWeight="Regular" Background="SkyBlue" />
<TextBlock Text="{Binding DraggingRecords.Count}" FontSize="14"
FontFamily="Segoe UI"
FontWeight="Regular"
Foreground="#333333"
Grid.Column="1" Margin="-100,0,0,0"/>
<Separator Grid.Row="1" Height="2" BorderBrush="#c8c8c8"
HorizontalAlignment="Stretch" BorderThickness="1"
VerticalAlignment="Stretch" Width="250"/>
<TextBlock Text="Drop status:"
Foreground="#333333"
Padding="12,0,0,0" Background="SkyBlue"
FontFamily="Segoe UI"
FontWeight="Regular"
FontSize="14"
Grid.Row="2"/>
<TextBlock Text="{Binding DragStatus}"
FontSize="14"
FontFamily="Segoe UI"
FontWeight="Regular"
Foreground="#333333"
Margin="-163,0,0,0"
Grid.Row="2"
Grid.Column="1"/>
</Grid>
</Border>
</DataTemplate>
<syncfusion:SfTreeGrid Name="sfTreeGrid" AutoExpandMode="RootNodesExpanded"
SelectionMode="Extended" AllowDraggingRows="True" AllowDrop="True"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID"
RowDragDropTemplate="{StaticResource DragDropTemplate}"
SelfRelationRootValue="-1" >

```

	First Name	Last Name	Employee ID	Title	Salary	ReportsTo
<input type="checkbox"/>	Ferando	Joseph	2	Management	\$2000000.00	-1
	Andrew	Fuller	9	Vice President	\$1200000.00	2
↓	Janet	Leverling	10	GM	\$1000000.00	2
↑	Steven	King	11	Manager	\$900000.00	2
<input type="checkbox"/>	John	Deity	12	Accounts	\$2000000.00	-1
	Nancy	Peacock	13	Accounts Manager	\$850000.00	3
	Margaret	Peacock	13	Accountant	\$700000.00	3
	Michael	Suyama	14	Accountant	\$700000.00	3
	Robert	King	15	Accountant	\$650000.00	3
<input type="checkbox"/>	Thomas	Jefferson	4	Sales	\$300000.00	-1
	Laura	Callahan	16	Sales Manager	\$900000.00	4
	Anne	Dodsworth	17	Sales Representative	\$800000.00	4
	Albert	Hellstern	18	Sales Representative	\$750000.00	4
	Tim	Smith	19	Sales Representative	\$700000.00	4
	Justin	Brid	20	Sales Representative	\$700000.00	4

Disable dragging of certain nodes

You can restrict the dragging for specific nodes using the DragStart event of the the [TreeGridRowDragDropController](#) class.

C#

```
private void RowDragDropController_DragStart(object sender,
Syncfusion.UI.Xaml.TreeGrid.TreeGridRowDragStartEventArgs e)
{
    var nodes = e.DraggingNodes;
    var node = nodes.FirstOrDefault(n => n.Level == 0);
    if (node != null)
        e.Handled = true;
}
```

Disable dropping of certain nodes

You can restrict the dropping for specific nodes using the Drop event of the [TreeGridRowDragDropController](#) class.

C#

```
private void RowDragDropController_Drop(object sender,
Syncfusion.UI.Xaml.TreeGrid.TreeGridRowDropEventArgs e)
{
    // Disable drop on leaf nodes.
    if (!e.TargetNode.HasChildNodes)
        e.Handled = true;
}
```

Disable default Drag UI

You can disable the draggable popup by setting the [ShowDragUI](#) as false in the Drop event of [TreeGridRowDragDropController](#) class.

C#

```
private void RowDragDropController_DragOver(object sender,
TreeGridRowDragOverEventArgs e)
{
    e.ShowDragUI = false;
}
```

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

ToolTip in WPF TreeGrid (SfTreeGrid)

ToolTip supports showing the pop-up window that displays the information when the mouse hovers over a cell of the SfTreeGrid.

Record cell tooltip

You can enable tooltip for the TreeGridCell by setting the [SfTreeGrid.ShowToolTip](#) property to `true`.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ShowToolTip="True"
ChildPropertyName="Children" ItemsSource="{Binding EmployeeDetails}">
```

C#

```
this.treeGrid.ShowToolTip = true;
```

You can enable the tooltip of a particular column by setting the [TreeGridColumn.ShowToolTip](#) property to `true`.

XML

```
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName" ShowToolTip="True" />
<syncfusion:TreeGridTextColumn HeaderText="Last Name" MappingName="LastName"
ShowToolTip="True" />
```

C#

```
this.treeGrid.Columns["FirstName"].ShowToolTip = true;
this.treeGrid.Columns["LastName"].ShowToolTip = true;
```

Note: It has higher priority than [SfTreeGrid.ShowToolTip](#).

Person ID	First Name	Last Name	DOB	City Descript	Availability
<input type="checkbox"/> 1000	Chester	Fillmore	8/20/2004	London	<input type="checkbox"/>
1000	Abraham	Tyler	11/30/1999	London	<input type="checkbox"/>
<input type="checkbox"/> 1000	Dwight	Lincoln	4/22/2004	Birmingham	<input type="checkbox"/>
1002	Benjamin	Tyler	11/29/1997	Manchester	<input type="checkbox"/>
1003	Theodore	Lincoln	6/13/2002	NewYork	<input checked="" type="checkbox"/>
1004	Herbert	Lincoln	4/13/2000	Tokyo	<input checked="" type="checkbox"/>
<input type="checkbox"/> 1001	Bill	Monroe	4/25/1999	Barcelona	<input type="checkbox"/>
1000	Richard	Monroe	9/17/1999	Durban	<input type="checkbox"/>
1001	Rutherford	Coolidge	9/1/1996	Rome	<input type="checkbox"/>
1002	George	Grant	3/18/2003	NewYork	<input type="checkbox"/>
1003	Franklin	Lincoln	10/25/2003	Cardiff	<input type="checkbox"/>
1004	John	Hayes	5/27/2002	Tokyo	<input type="checkbox"/>
<input type="checkbox"/> 1002	John	Fillmore	2/19/1999	Rome	<input type="checkbox"/>
1000	Peter	Carter	8/29/1998	Brisbane	<input checked="" type="checkbox"/>

Header tooltip

You can enable the tooltip of a header cell by setting the [TreeGridColumn.ShowHeaderToolTip](#) property to true.

XML

```
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName" ShowHeaderToolTip="True" />
</syncfusion:SfTreeGrid.Columns>
```

C#

```
this.treeGrid.Columns["FirstName"].ShowHeaderToolTip = true;
```

Person ID	First Name	Last Name	DOB	City Descript	Availability
[-] 1000	Charles	Fillmore	8/20/2004	London	<input type="checkbox"/>
1000	Abraham	Tyler	11/30/1999	London	<input type="checkbox"/>
1001	Dwight	Lincoln	4/22/2004	Birmingham	<input type="checkbox"/>
1002	Benjamin	Tyler	11/29/1997	Manchester	<input type="checkbox"/>
1003	Theodore	Lincoln	6/13/2002	NewYork	<input checked="" type="checkbox"/>
1004	Herbert	Lincoln	4/13/2000	Tokyo	<input checked="" type="checkbox"/>
[-] 1001	Bill	Monroe	4/25/1999	Barcelona	<input type="checkbox"/>
1000	Richard	Monroe	9/17/1999	Durban	<input type="checkbox"/>
1001	Rutherford	Coolidge	9/1/1996	Rome	<input type="checkbox"/>
1002	George	Grant	3/18/2003	NewYork	<input type="checkbox"/>
1003	Franklin	Lincoln	10/25/2003	Cardiff	<input type="checkbox"/>
1004	John	Hayes	5/27/2002	Tokyo	<input type="checkbox"/>
[-] 1002	John	Fillmore	2/19/1999	Rome	<input type="checkbox"/>
1000	Peter	Carter	8/29/1998	Brisbane	<input checked="" type="checkbox"/>

ToolTip customization

You can change appearance of the tooltip by customizing the style with TargetType as ToolTip.

XML

```
<Window.Resources>
<Style TargetType="ToolTip">
<Setter Property="BorderThickness" Value="1,1,1,1" />
<Setter Property="BorderBrush" Value="Red" />
<Setter Property="Background" Value="SkyBlue" />
</Style>
</Window.Resources>
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="First Name"
MappingName="FirstName" ShowToolTip="True" />
</syncfusion:SfTreeGrid.Columns>
```

Person ID	First Name	Last Name	DOB	City Description	Availability
[-] 1000	Chester	Fillmore	8/20/2004	London	<input type="checkbox"/>
1000	Abraham	Tyler	11/30/1999	London	<input type="checkbox"/>
1001	Dwight	Lincoln	4/22/2004	Birmingham	<input type="checkbox"/>
1001	Benjamin	Tyler	11/29/1997	Manchester	<input type="checkbox"/>
1003	Theodore	Lincoln	6/13/2002	NewYork	<input checked="" type="checkbox"/>
1004	Herbert	Lincoln	4/13/2000	Tokyo	<input checked="" type="checkbox"/>
[-] 1001	Bill	Monroe	4/25/1999	Barcelona	<input type="checkbox"/>
1000	Richard	Monroe	9/17/1999	Durban	<input type="checkbox"/>
1001	Rutherford	Coolidge	9/1/1996	Rome	<input type="checkbox"/>
1002	George	Grant	3/18/2003	NewYork	<input type="checkbox"/>
1003	Franklin	Lincoln	10/25/2003	Cardiff	<input type="checkbox"/>
1004	John	Hayes	5/27/2002	Tokyo	<input type="checkbox"/>
[-] 1002	John	Fillmore	2/19/1999	Rome	<input type="checkbox"/>
1000	Peter	Carter	8/29/1998	Brisbane	<input checked="" type="checkbox"/>
1001	Martin	Lincoln	5/14/1999	Tokyo	<input checked="" type="checkbox"/>

You can customize the template of the tooltip by using the [TreeGridColumn.ToolTipTemplate](#) and [TreeGridColumn.ToolTipTemplateSelector](#) properties.

Customize the tooltip using ToolTipTemplate

You can customize appearance of the tooltip of a particular column by setting the [TreeGridColumn.ToolTipTemplate](#). You can also customize appearance of the header tooltip of a particular column by using the [TreeGridColumn.HeaderToolTipTemplate](#) property.

The ToolTipTemplate receives the underlying data object as DataContext by default. You can set the [TreeGridColumn.SetCellBoundToolTip](#) to `true` to change the DataContext of the tooltip template where it sets the DataContext as DataContextHelper. The [TreeGridDataContextHelper](#) has the following properties to reuse the same template for all the columns:

 Record: Gets the underlying data record of a row which has the cell.

 Value: Gets the underlying value of a cell.

XML

```
<Window.Resources>
<local:StringToImageConverter x:Key="ImageConverter" />
<DataTemplate x:Key="TemplateToolTip">
```

```

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Image Height="100" Width="100" Source="{Binding
LastName, Converter={StaticResource ImageConverter}}" />
<TextBlock Grid.Row="1" Text="{Binding LastName}"
HorizontalAlignment="Center" />
</Grid>
</DataTemplate>
</Window.Resources>
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="Last Name" MappingName="LastName"
ToolTipTemplate="{StaticResource TemplateToolTip}" ShowToolTip="True" />
</syncfusion:SfTreeGrid.Columns>

```

C#

```

public class StringToImageConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        string imagename = value.ToString();
        return @"..\..\Assets\" + imagename + @".png";
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        return value;
    }
}

```

First Name	Last Name	Person ID	DOB	City Description	Availability
<input type="checkbox"/> Chester	Fillmore	1000	9/5/2004	London	<input type="checkbox"/>
Abraham	Tyler	1000	12/16/1999	London	<input type="checkbox"/>
Dwig		1001	5/8/2004	Birmingham	<input type="checkbox"/>
Benja		1002	12/15/1997	Manchester	<input type="checkbox"/>
Theo		1003	6/29/2002	NewYork	<input checked="" type="checkbox"/>
Herb		1004	4/29/2000	Tokyo	<input checked="" type="checkbox"/>
<input type="checkbox"/> Bill		1001	5/11/1999	Barcelona	<input type="checkbox"/>
Richard	Monroe	1000	10/3/1999	Durban	<input type="checkbox"/>
Rutherford	Coolidge	1001	9/17/1996	Rome	<input type="checkbox"/>
George	Grant	1002	4/3/2003	NewYork	<input type="checkbox"/>
Franklin	Lincoln	1003	11/10/2003	Cardiff	<input type="checkbox"/>
John	Hayes	1004	6/12/2002	Tokyo	<input type="checkbox"/>

You can get the sample [here](#).

Customize the ToolTip with ToolTipTemplateSelector

Different tooltip templates can be loaded conditionally in same column based on the data by setting the [TreeGridColumn.ToolTipTemplateSelector](#) property.

XML

```
<Window.Resources>
<DataTemplate x:Key="ToolTip1">
<Grid>
<TextBlock Text="{Binding Record.Id}" FontWeight="Bold" Foreground="Red" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="ToolTip2">
<Grid>
<TextBlock Text="{Binding Record.Id}" FontWeight="Bold" Foreground="Green"/>
</Grid>
</DataTemplate>
</Window.Resources>
<syncfusion:SfTreeGrid.Columns>
<syncfusion:TreeGridTextColumn HeaderText="Person ID" MappingName="Id"
ShowToolTip="True" >
<syncfusion:TreeGridTextColumn.ToolTipTemplateSelector>
<syncfusion:TreeGridTextColumn HeaderText="Person ID" MappingName="Id"
DisplayBinding="{Binding Path=Id, StringFormat=c}"
ShowToolTip="True" SetCellBoundToolTip="True" >
</syncfusion:TreeGridTextColumn.ToolTipTemplateSelector>
</syncfusion:TreeGridTextColumn>
</syncfusion:SfTreeGrid.Columns>
```

C#

```
public class ToolTipTemplateSelector : DataTemplateSelector
{
    private DataTemplate _defaultTemplate;
    /// <summary>
    /// Gets or sets DefaultTemplate.
    /// </summary>
    public DataTemplate DefaultTemplate
    {
        get { return _defaultTemplate; }
        set { _defaultTemplate = value; }
    }
    private DataTemplate _alternateTemplate;
    /// <summary>
    /// Gets or Sets AlternateTemplate.
    /// </summary>
    public DataTemplate AlternateTemplate
    {
        get { return _alternateTemplate; }
        set { _alternateTemplate = value; }
    }
    public override System.Windows.DataTemplate SelectTemplate(object item,
        System.Windows.DependencyObject container)
    {
        var treeGridData = item as TreeGridDataContextHelper;
        if (treeGridData == null)
```



```

return this.DefaultTemplate;
// To see what template needs to be select according to the specified
property value.
if ((treeGridData.Record as Employee).Id == (int)treeGridData.Value &&
((int)treeGridData.Value % 2) == 0)
return this.AlternateTemplate;
else
return this.DefaultTemplate;
}
}

```

The following image shows the DefaultTemplate applied through ToolTipTemplateSelector.

First Name	Last Name	Person ID	DOB	City Descriptio	Availability
<input type="checkbox"/> Chester	Buchanan	\$1,000.00	9/6/2004	London	<input type="checkbox"/>
Abraham	Fillmore	\$1,000.00	12/17/1999	London	<input type="checkbox"/>
Dwight	Grant	\$1,001.00	5/9/2004	Birmingham	<input type="checkbox"/>
Benjamin	Taylor	\$ 1001	12/16/1997	Manchester	<input type="checkbox"/>
Theodore	Grant	\$1,003.00	6/30/2002	NewYork	<input checked="" type="checkbox"/>
Herbert	Hayes	\$1,004.00	4/30/2000	Tokyo	<input checked="" type="checkbox"/>
<input type="checkbox"/> Bill	Madison	\$1,001.00	5/12/1999	Barcelona	<input type="checkbox"/>
Richard	Washington	\$1,000.00	10/4/1999	Durban	<input type="checkbox"/>
Rutherford	Clinton	\$1,001.00	9/18/1996	Rome	<input type="checkbox"/>
George	Harding	\$1,002.00	4/4/2003	NewYork	<input type="checkbox"/>
Franklin	Hayes	\$1,003.00	11/11/2003	Cardiff	<input type="checkbox"/>
John	Kennedy	\$1,004.00	6/13/2002	Tokyo	<input type="checkbox"/>

The following image shows the AlternateTemplate applied through ToolTipTemplateSelector.

First Name	Last Name	Person ID	DOB	City Descriptio	Availability
<input type="checkbox"/> Chester	Buchanan	\$1,000.00	9/6/2004	London	<input type="checkbox"/>
Abraham	Fillmore	\$1,000.00	12/17/1999	London	<input type="checkbox"/>
Dwight	Grant	\$1,001.00	5/9/2004	Birmingham	<input type="checkbox"/>
Benjamin	Taylor	\$1,002.00	12/16/1997	Manchester	<input type="checkbox"/>
Theodore	Grant	1002	6/30/2002	NewYork	<input checked="" type="checkbox"/>
Herbert	Hayes	\$1,004.00	4/30/2000	Tokyo	<input checked="" type="checkbox"/>
<input type="checkbox"/> Bill	Madison	\$1,001.00	5/12/1999	Barcelona	<input type="checkbox"/>
Richard	Washington	\$1,000.00	10/4/1999	Durban	<input type="checkbox"/>
Rutherford	Clinton	\$1,001.00	9/18/1996	Rome	<input type="checkbox"/>
George	Harding	\$1,002.00	4/4/2003	NewYork	<input type="checkbox"/>
Franklin	Hayes	\$1,003.00	11/11/2003	Cardiff	<input type="checkbox"/>
John	Kennedy	\$1,004.00	6/13/2002	Tokyo	<input type="checkbox"/>

You can get the sample [here](#).

Events

CellToolTipOpening event

The [CellToolTipOpening](#) event occurs when any tooltip of the cell is opened. The `CellToolTipOpening` event receives the [TreeGridCellToolTipOpeningEventArgs](#) as argument which has the following properties:

 Column: Gets the hovered cell column in the SfTreeGrid.

 Node: Gets the hovered cell node.

 Record: Gets the data context of hovered cell.

 RowColumnIndex: Gets the row and column index of the hovered cell.

 ToolTip: Gets the tooltip of the hovered cells.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="RootNodesExpanded"
CellToolTipOpening="TreeGrid_CellToolTipOpening"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ColumnSizer="Star"
ExpanderColumn="Id"
ItemsSource="{Binding PersonDetails}">
```

C#

```
this.treeGrid.CellToolTipOpening += TreeGrid_CellToolTipOpening;
private void TreeGrid_CellToolTipOpening(object sender,
Syncfusion.UI.Xaml.TreeGrid.TreeGridCellToolTipOpeningEventArgs e)
{
}
```

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Rows in WPF TreeGrid (SfTreeGrid)

This section explains about various row types in treegrid and its customization.





Rows in WPF TreeGrid (SfTreeGrid)

RowHeader is a special column used to indicate the status of row (current row, editing status, errors in row, etc.) which is placed as first cell of each row. You can show or hide the row header by setting [SfTreeGrid.ShowRowHeader](#) property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="AllNodesExpanded"
AutoGenerateColumns="False"
ShowRowHeader="True"
ChildPropertyName="Children"
ColumnSizer="Star"
ItemsSource="{Binding PersonDetails}"/>
```

Row indicators and its description

Row Indicator	Description
	Denotes the row which has current cell or selected item.
	Denotes row is being edited.
	Denotes the row has errors.
	Denotes that the current row which has errors.

Show row index in row header

You can display the row index value in row header by customizing the `TreeGridRowHeaderCell` style with the binding of `RowIndex` to `TextBlock.Text` property.

XML

```
<Style TargetType="syncfusion:TreeGridRowHeaderCell">
<Setter Property="Background" Value="{StaticResource
ContentBackgroundBrush}" />
<Setter Property="BorderBrush" Value="{StaticResource ContentBorderBrush}"
/>
<Setter Property="BorderThickness" Value="0,0,1,1" />
<Setter Property="Foreground" Value="#FF303030"/>
<Setter Property="Padding" Value="0,0,0,0" />
<Setter Property="FocusVisualStyle" Value="{x:Null}" />
```


```

<Setter Property="IsTabStop" Value="False" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:TreeGridRowHeaderCell">
<Border x:Name="PART_RowHeaderCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}">
<Grid>
<!--RowIndex is displayed here -->
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding RowIndex,
RelativeSource={RelativeSource TemplatedParent}}"
TextAlignment="Center" />
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

Rows in WPF TreeGrid (SfTreeGrid)

Header row is present in top of the treegrid which has column headers in it. Column header describes the caption to identify the column content.



	First Name	Last Name	ID	Availability	DOB	
<input type="checkbox"/>	Chester	Buchanan	0	<input checked="" type="checkbox"/>	3/4/2007	
<input type="checkbox"/>	Abraham	Buchanan	1	<input checked="" type="checkbox"/>	7/1/2006	
<input type="checkbox"/>	Theodore	Fillmore	2	<input checked="" type="checkbox"/>	9/24/1999	
<input type="checkbox"/>	Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/20/2004	
	George	Stogner	4	<input checked="" type="checkbox"/>	3/19/2004	
	Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/9/1998	
	Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/21/2001	
	Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/24/2004	
	Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/14/1996	
<input type="checkbox"/>	Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/12/2006	
	Herbert	Clinton	10	<input checked="" type="checkbox"/>	10/6/2002	
	Jimmy	Clinton	11	<input checked="" type="checkbox"/>	1/19/2000	
	Abraham	Clinton	12	<input checked="" type="checkbox"/>	12/7/1998	
	Thomas	Clinton	13	<input checked="" type="checkbox"/>	10/12/2004	

Hiding header row

You can hide the header row by setting [SfTreeGrid.HeaderRowHeight](#) as 0 (zero).

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="AllNodesExpanded"
AutoGenerateColumns="False"
ShowRowHeader="True"
HeaderRowHeight="0"
ChildPropertyName="Children"
ColumnSizer="Star"
ItemsSource="{Binding PersonDetails}">
```

Change the orientation of column header text to vertical

Orientation of the treegrid column header text can be changed by editing the control template of the [TreeGridHeaderCell](#) and applying `RotateTransform`.

XML

```
<Style TargetType="syncfusion:TreeGridHeaderCell">
<Setter Property="Background" Value="{StaticResource HeaderBackgroundBrush}" />
<Setter Property="Foreground" Value="{StaticResource HeaderForegroundBrush}" />
<Setter Property="BorderBrush" Value="{StaticResource HeaderBorderBrush}" />
<Setter Property="BorderThickness" Value="0,0,1,1" />
<Setter Property="HorizontalContentAlignment" Value="Center" />
<Setter Property="Padding" Value="5,3,5,3" />
<Setter Property="FontSize" Value="14" />
<Setter Property="FontWeight" Value="Normal" />
<Setter Property="IsTabStop" Value="False" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:TreeGridHeaderCell">
<Grid>
<Grid.LayoutTransform>
<RotateTransform Angle="90" />
</Grid.LayoutTransform>
<Border x:Name="PART_FooterCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}" />
<Border x:Name="PART_HeaderCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<Grid Margin="{TemplateBinding Padding}" SnapsToDevicePixels="True">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<ContentPresenter HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
VerticalAlignment="Center"
Focusable="False" />
<Grid x:Name="PART_SortButtonPresenter"
Grid.Column="1"
SnapsToDevicePixels="True">
<Grid.ColumnDefinitions>
```

```

<ColumnDefinition Width="0" MinWidth="{Binding Path=SortDirection,
Mode=OneWay, RelativeSource={RelativeSource TemplatedParent}},
Converter={StaticResource sortDirectionToWidthConverter}}" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Path Width="8.938"
Height="8.138"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="F1M753.644,-13.0589L753.736,-12.9639 753.557,-12.7816 732.137,8.63641
732.137,29.7119 756.445,5.40851 764.094,-2.24384 764.275,-2.42352
771.834,5.1286 796.137,29.4372 796.137,8.36163 774.722,-13.0589 764.181,-
23.5967 753.644,-13.0589z"
Fill="{TemplateBinding Foreground}"
SnapsToDevicePixels="True"
Stretch="Fill"
Visibility="{Binding Path=SortDirection,
RelativeSource={RelativeSource TemplatedParent}},
ConverterParameter=Ascending,
Converter={StaticResource sortDirectionToVisibilityConverter}}">
<Path.RenderTransform>
<TransformGroup>
<TransformGroup.Children>
<RotateTransform Angle="0" />
<ScaleTransform ScaleX="1" ScaleY="1" />
</TransformGroup.Children>
</TransformGroup>
</Path.RenderTransform>
</Path>
<Path Width="8.938"
Height="8.138"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="F1M181.297,177.841L181.205,177.746 181.385,177.563 202.804,156.146
202.804,135.07 178.497,159.373 170.847,167.026 170.666,167.205
163.107,159.653 138.804,135.345 138.804,156.42 160.219,177.841
170.76,188.379 181.297,177.841z"
Fill="{TemplateBinding Foreground}"
SnapsToDevicePixels="True"
Stretch="Fill"
Visibility="{Binding Path=SortDirection,
RelativeSource={RelativeSource TemplatedParent}},
ConverterParameter=Decending,
Converter={StaticResource sortDirectionToVisibilityConverter}}">
<Path.RenderTransform>
<TransformGroup>
<TransformGroup.Children>
<RotateTransform Angle="0" />
<ScaleTransform ScaleX="1" ScaleY="1" />
</TransformGroup.Children>
</TransformGroup>
</Path.RenderTransform>
</Path>
<TextBlock Grid.Column="1"
Margin="0,-4,0,0"
VerticalAlignment="Center"
FontSize="10"

```

```

Foreground="{TemplateBinding Foreground}"
SnapsToDevicePixels="True"
Text="{TemplateBinding SortNumber}"
Visibility="{TemplateBinding SortNumberVisibility}" />
</Grid>
</Grid>
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

FirstName	LastName	ID	Title
[-] Ferando	Joseph	2	Management
Andrew	Fuller	9	Vice President
Janet	Leverling	10	GM
Steven	Buchanan	11	Manager
[-] John		3	Accounts
Nancy	Davolio	12	Accounts Manager
Margaret	Peacock	13	Accountant
Michael	Suyama	14	Accountant
Robert	King	15	Accountant
[-] Ferando1	Joseph1	4	Management1
Nancy1	Davolio1	16	Accounts Manager1
Margaret1	Peacock1	17	Accountant1

You can download the sample [here](#).

Change the position of sort icon in header cell

By default, the sort icon appears at the right of the header text. You can change the default position to left of the header text by customizing the `TreeGridHeaderCell` style.

XML

```

<Style TargetType="syncfusion:TreeGridHeaderCell">
<Setter Property="Background" Value="{StaticResource HeaderBackgroundBrush}" />
<Setter Property="Foreground" Value="{StaticResource HeaderForegroundBrush}" />
<Setter Property="BorderBrush" Value="{StaticResource HeaderBorderBrush}" />

```

```

<Setter Property="BorderThickness" Value="0,0,1,1" />
<Setter Property="HorizontalContentAlignment" Value="Center" />
<Setter Property="Padding" Value="5,3,5,3" />
<Setter Property="FontSize" Value="14" />
<Setter Property="FontWeight" Value="Normal" />
<Setter Property="IsTabStop" Value="False" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:TreeGridHeaderCell">
<Grid>
<Grid.LayoutTransform>
<RotateTransform Angle="90"/>
</Grid.LayoutTransform>
<Border x:Name="PART_FooterCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}" />
<Border x:Name="PART_HeaderCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<Grid Margin="{TemplateBinding Padding}" SnapsToDevicePixels="True">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<ContentPresenter Grid.Column="1"
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
VerticalAlignment="Center"
Focusable="False" />
<Grid x:Name="PART_SortButtonPresenter"
SnapsToDevicePixels="True">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="0" MinWidth="{Binding Path=SortDirection,
Mode=OneWay, RelativeSource={RelativeSource TemplatedParent}},
Converter={StaticResource sortDirectionToWidthConverter}}" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Path Width="8.938"
Height="8.138"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="F1M753.644,-13.0589L753.736,-12.9639 753.557,-12.7816 732.137,8.63641
732.137,29.7119 756.445,5.40851 764.094,-2.24384 764.275,-2.42352
771.834,5.1286 796.137,29.4372 796.137,8.36163 774.722,-13.0589 764.181,-
23.5967 753.644,-13.0589z"
Fill="{TemplateBinding Foreground}"
SnapsToDevicePixels="True"
Stretch="Fill"
Visibility="{Binding Path=SortDirection,
RelativeSource={RelativeSource TemplatedParent},
ConverterParameter=Ascending,
Converter={StaticResource sortDirectionToVisibilityConverter}}">
<Path.RenderTransform>
<TransformGroup>
<TransformGroup.Children>
<RotateTransform Angle="0" />

```



```

<ScaleTransform ScaleX="1" ScaleY="1" />
</TransformGroup.Children>
</TransformGroup>
</Path.RenderTransform>
</Path>
<Path Width="8.938"
Height="8.138"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="F1M181.297,177.841L181.205,177.746 181.385,177.563 202.804,156.146
202.804,135.07 178.497,159.373 170.847,167.026 170.666,167.205
163.107,159.653 138.804,135.345 138.804,156.42 160.219,177.841
170.76,188.379 181.297,177.841z"
Fill="{TemplateBinding Foreground}"
SnapsToDevicePixels="True"
Stretch="Fill"
Visibility="{Binding Path=SortDirection,
RelativeSource={RelativeSource TemplatedParent},
ConverterParameter=Decending,
Converter={StaticResource sortDirectionToVisibilityConverter}}">
<Path.RenderTransform>
<TransformGroup>
<TransformGroup.Children>
<RotateTransform Angle="0" />
<ScaleTransform ScaleX="1" ScaleY="1" />
</TransformGroup.Children>
</TransformGroup>
</Path.RenderTransform>
</Path>
<TextBlock Grid.Column="1"
Margin="0,-4,0,0"
VerticalAlignment="Center"
FontSize="10"
Foreground="{TemplateBinding Foreground}"
SnapsToDevicePixels="True"
Text="{TemplateBinding SortNumber}"
Visibility="{TemplateBinding SortNumberVisibility}" />
</Grid>
</Grid>
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

First Name	Last Name	ID	Availability	DOB
☐ Millard	Johnson	3124	☑	2/22/2001
☐ Thomas	Johnson	3749	☑	11/4/2000
☐ Dwight	Adams	3874	☑	8/4/1998
☐ Dwight	Jones	3899	☑	6/13/2002
Woodrow	Roosevelt	3904	☑	11/3/2000
Ulysses	Roosevelt	3903	☑	12/7/2003
Rutherford	Roosevelt	3902	☑	11/16/1999
William	Roosevelt	3901	☑	11/23/2006
Theodore	Roosevelt	3900	☑	11/17/2000
☐ Warner	Jones	3893	☑	5/16/2007
Gerald	Johnson	3898	☑	1/8/1999
Millard	Johnson	3897	☑	3/9/1997
Herbert	Johnson	3896	☑	4/5/2005
Herbert	Johnson	3895	☑	6/2/1997
Rutherford	Johnson	3894	☑	12/15/2004
☐ Rutherford	Jones	3887	☑	11/5/2001
Woodrow	Kennedy	3802	☑	5/21/2007

Customize style of header row

You can change the header cell background and foreground for specific column or an entire grid by using [HeaderStyle](#) property.

XML

```
<syncfusion:Window.Resources>
<Style TargetType="syncfusion:TreeGridHeaderCell" x:Key="headerStyle">
<Setter Property="Background" Value="#FF7AA732"/>
<Setter Property="Foreground" Value="Red"/>
</Style>
</syncfusion:Window.Resources>
<syncfusion:SfTreeGrid Name="treeGrid"
AutoExpandMode="AllNodesExpanded"
AutoGenerateColumns="False"
HeaderStyle="{StaticResource headerStyle}"
ChildPropertyName="Children"
ColumnSizer="Star"
ItemsSource="{Binding PersonDetails}">
```

First Name	Last Name	ID	Availability	DOB
<input type="checkbox"/> Chester	Buchanan	0	<input checked="" type="checkbox"/>	3/4/2007
<input type="checkbox"/> Abraham	Buchanan	1	<input checked="" type="checkbox"/>	7/1/2006
<input type="checkbox"/> Theodore	Fillmore	2	<input checked="" type="checkbox"/>	9/24/1999
<input type="checkbox"/> Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/20/2004
George	Stogner	4	<input checked="" type="checkbox"/>	3/19/2004
Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/9/1998
Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/21/2001
Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/24/2004
Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/14/1996
<input type="checkbox"/> Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/12/2006
Herbert	Clinton	10	<input checked="" type="checkbox"/>	10/6/2002
Jimmy	Clinton	11	<input checked="" type="checkbox"/>	1/19/2000
Abraham	Clinton	12	<input checked="" type="checkbox"/>	12/7/1998
Thomas	Clinton	13	<input checked="" type="checkbox"/>	10/12/2004
Ulysses	Clinton	14	<input checked="" type="checkbox"/>	7/31/2003
<input type="checkbox"/> Barack	Harrison	15	<input checked="" type="checkbox"/>	4/20/1999

You can change the style of the particular column header by using the [HeaderStyle](#) property in column,

XML

```
<syncfusion:TreeGridTextColumn HeaderText="First Name"
HeaderStyle="{StaticResource headerStyle}" MappingName="FirstName" />
```

First Name	Last Name	ID	Availability	DOB
<input type="checkbox"/> Chester	Buchanan	0	<input checked="" type="checkbox"/>	3/4/2007
<input type="checkbox"/> Abraham	Buchanan	1	<input checked="" type="checkbox"/>	7/1/2006
<input type="checkbox"/> Theodore	Fillmore	2	<input checked="" type="checkbox"/>	9/24/1999
<input type="checkbox"/> Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/20/2004
George	Stogner	4	<input checked="" type="checkbox"/>	3/19/2004
Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/9/1998
Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/21/2001
Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/24/2004
Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/14/1996
<input type="checkbox"/> Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/12/2006
Herbert	Clinton	10	<input checked="" type="checkbox"/>	10/6/2002
Jimmy	Clinton	11	<input checked="" type="checkbox"/>	1/19/2000
Abraham	Clinton	12	<input checked="" type="checkbox"/>	12/7/1998
Thomas	Clinton	13	<input checked="" type="checkbox"/>	10/12/2004
Ulysses	Clinton	14	<input checked="" type="checkbox"/>	7/31/2003
<input type="checkbox"/> Barack	Harrison	15	<input checked="" type="checkbox"/>	4/20/1999

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Merge Cells in WPF TreeGrid (SfTreeGrid)

SfTreeGrid allows you to merge a range of adjacent cells in a row across columns using the [QueryCoveredRange](#) event.

The [QueryCoveredRange](#) event occurs when each cell is arranged. When scrolling, the merged range will be added for newly added columns through this event and will also be removed for the columns that are out of view.

[TreeGridQueryCoveredRangeEventArgs](#) of the [QueryCoveredRange](#) event provides information about the cell triggered in this event. [GridQueryCoveredRangeEventArgs.OriginalSender](#) returns the TreeGrid fired in this event using the [TreeGridQueryCoveredRangeEventArgs.Range](#) property. The adjacent cells can be merged.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
ItemsSource="{Binding EmployeeDetails}"
QueryCoveredRange="treeGrid_QueryCoveredRange"
SelectionMode="Single"
NavigationMode="Cell">
```

C#

```
treeGrid.QueryCoveredRange += TreeGrid_QueryCoveredRange;
treeGrid.SelectionMode = Syncfusion.UI.Xaml.Grid.GridSelectionMode.Single;
treeGrid.NavigationMode = Syncfusion.UI.Xaml.Grid.NavigationMode.Cell;
```

```
void TreeGrid_QueryCoveredRange(object sender,
TreeGridQueryCoveredRangeEventArgs e)
{
}
```

Column wise merging cells by fixed range

You can merge the columns by setting the range using the Left and Right properties of `TreeGridCoveredCellInfo`.

C#

```
void treeGrid_QueryCoveredRange(object sender,
Syncfusion.UI.Xaml.TreeGrid.TreeGridQueryCoveredRangeEventArgs e)
{
    if (e.RowColumnIndex.RowIndex == 1)
    {
        if (e.RowColumnIndex.ColumnIndex >= 1 && e.RowColumnIndex.ColumnIndex <= 3)
        {
            e.Range = new TreeGridCoveredCellInfo(1, 4, 1);
            e.Handled = true;
        }
    }
}
```

First Name	Last Name	Person ID	DOB	Contact Number	City
<input type="checkbox"/> Chester	Buchanan				Delhi
<input checked="" type="checkbox"/> Abraham	Fillmore	1	11-02-2002	(99)-9112	Durban
<input checked="" type="checkbox"/> Gerald	Obama	1112	11-09-1997	(99)-9116	Sydney
<input checked="" type="checkbox"/> Herbert	Harding	2223	29-10-2001	(99)-9115	Brisbane
<input checked="" type="checkbox"/> Jimmy	Hayes	3334	29-06-2001	(99)-9113	Sydney
<input checked="" type="checkbox"/> Jimmy	Jefferson	4445	07-11-2001	(99)-9115	Brisbane
<input checked="" type="checkbox"/> George	Fillmore	5556	15-03-2007	(99)-9117	Madrid
<input checked="" type="checkbox"/> Ronald	Truman	6667	19-02-2002	(99)-9118	Manchester
<input checked="" type="checkbox"/> Benjamin	Jefferson	7778	09-10-2007	(99)-9112	Barcelona
<input checked="" type="checkbox"/> Warren	Stogner	8889	28-11-2005	(99)-9117	London
<input checked="" type="checkbox"/> Dwight	Garfield	10000	02-06-2005	(99)-9113	Manchester
<input type="checkbox"/> Franklin	Washington	11111	09-11-2005	(99)-9118	Sydney

Merge all cells in an entire parent node

You can merge the entire column parent node using `TreeGridCoveredCellInfo`.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
ItemsSource="{Binding EmployeeDetails}"
QueryCoveredRange="treeGrid_QueryCoveredRange"
SelectionMode="Single"
```

```
NavigationMode="Cell">
```

C#

```
public delegate void TreeGridRequestTreeItemsHandler(object sender,
RoutedEventArgs args);
/// <summary>
/// Handles the cell merging in SfTreeGrid.
/// </summary>
public class QueryCoveredRangeBehavior : Behavior<SfTreeGrid>
{
    /// <summary>
    /// Called after the behavior is attached to an AssociatedObject.
    /// </summary>
    /// <remarks>Override this to hook up functionality to the
    AssociatedObject.</remarks>
    protected override void OnAttached()
    {
        var loader= new TreeGridRequestTreeItemsHandler(AssociatedObject_Loaded);
        loader.Invoke(null, null);
    }
    public void AssociatedObject_Loaded(object sender, RoutedEventArgs e)
    {
        this.AssociatedObject.QueryCoveredRange +=
        AssociatedObject_QueryCoveredRange;
    }
    public void AssociatedObject_QueryCoveredRange(object sender,
    TreeGridQueryCoveredRangeEventArgs e)
    {
        var treeNode =
        this.AssociatedObject.GetNodeAtRowIndex(e.RowColumnIndex.RowIndex);
        if (treeNode != null && treeNode.HasChildNodes)
        {
            if (e.RowColumnIndex.ColumnIndex >= 1 && e.RowColumnIndex.ColumnIndex <=
            this.AssociatedObject.Columns.Count)
            {
                e.Range = new TreeGridCoveredCellInfo(0,
                this.AssociatedObject.Columns.Count, e.RowColumnIndex.RowIndex);
                e.Handled = true;
            }
        }
    }
    /// <summary>
    /// Calls when the behavior is being detached from its AssociatedObject, but
    before it has actually occurred.
    /// </summary>
    /// <remarks>Override this to unhook functionality from the
    AssociatedObject.</remarks>
    protected override void OnDetaching()
    {
        base.OnDetaching();
        this.AssociatedObject.Loaded -= AssociatedObject_Loaded;
        this.AssociatedObject.QueryCoveredRange -=
        AssociatedObject_QueryCoveredRange;
    }
}
```

```
class RequestTreeItemsBehavior : Behavior<SfTreeGrid>
{
    EmployeeRepository viewModel;
    protected override void OnAttached()
    {
        base.OnAttached();
        viewModel = this.AssociatedObject.DataContext as EmployeeRepository;
        this.AssociatedObject.RequestTreeItems += AssociatedObject_RequestTreeItems;
    }
    void AssociatedObject_RequestTreeItems(object sender,
        TreeGridRequestTreeItemsEventArgs args)
    {
        if (args.ParentItem == null)
        {
            // Gets the root list - Gets all employees who have no boss.
            args.ChildItems = EmployeeRepository.GetEmployees().Where(x => x.ReportsTo
            == -1); //get all employees whose boss's id is -1 (no boss)
        }
        else //if ParentItem not null, then set args.ChildList to the child items
            for the given ParentItem.
        {
            // Gets the children of the parent object.
            Employee emp = args.ParentItem as Employee;
            if (emp != null)
            {
                // Gets all employees who report to the parent employee.
                args.ChildItems = EmployeeRepository.GetEmployees().Where(x => x.ReportsTo
                == emp.Id);
            }
        }
    }
    protected override void OnDetaching()
    {
        base.OnDetaching();
        this.AssociatedObject.RequestTreeItems -= AssociatedObject_RequestTreeItems;
    }
}
```

Title	First Name	Last Name	Employee ID	Date of Joining	Rating	Salary	Hike
Management							
Vice President	Andrew	Fuller	1001	1/9/1937	5.00	\$1,200,000.00	10.00%
GM	Janet	Leverling	1002	5/7/1939	4.00	\$1,000,000.00	8.00%
Manager	Steven	Buchanan	1003	2/5/1940	4.00	\$900,000.00	7.00%
Accounts							
Accounts Manager	Nancy	Davolio	1004	2/5/1940	5.00	\$850,000.00	7.20%
Accountant	Margaret	Peacock	1008	1/9/1945	3.00	\$700,000.00	6.90%
Accountant	Michael	Suyama	1009	2/9/1945	5.00	\$700,000.00	6.90%
Accountant	Robert	King	1010	2/9/1945	3.00	\$650,000.00	5.70%
Sales							
Sales Manager	Laura	Callahan	1005	7/1/1942	5.00	\$900,000.00	8.20%
Sales Representative	Anne	Dodsworth	1011	2/10/1945	4.00	\$800,000.00	8.00%
Sales Representative	Albert	Hellstern	1012	2/10/1945	5.00	\$750,000.00	6.90%
Sales Representative	Tim	Smith	1013	3/11/1945	5.00	\$700,000.00	5.23%
Sales Representative	Justin	Brid	1014	3/11/1945	5.00	\$700,000.00	6.00%
Back Office							
Receptionist	Caroline	Patterson	1006	7/1/1942	5.00	\$800,000.00	6.00%
Mail Clerk	Xavier	Martin	1015	1/9/1946	5.00	\$700,000.00	3.00%

Refer to [Sample](#).

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Localization in WPF TreeGrid (SfTreeGrid)

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

Below application culture changed to German.

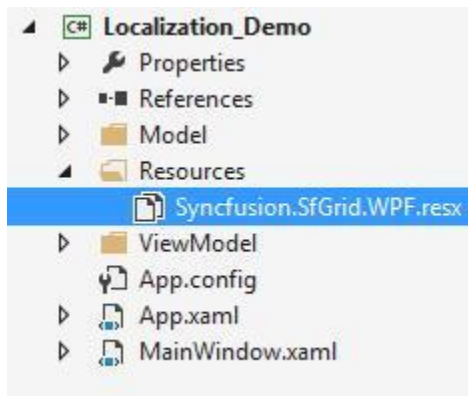
C#

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

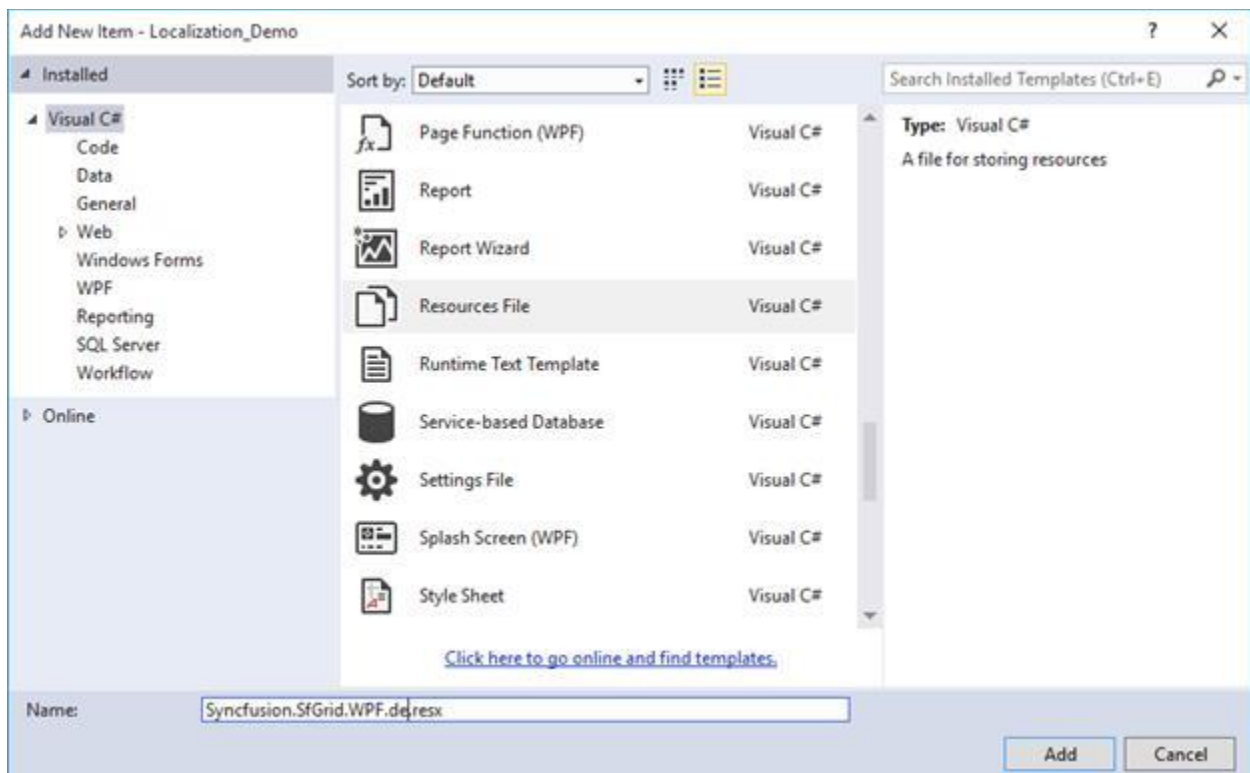
Localize the drag and drop window text in treegrid

To localize the treegrid, drag and drop window based on `CurrentUICulture` using resource files, follow the below steps.

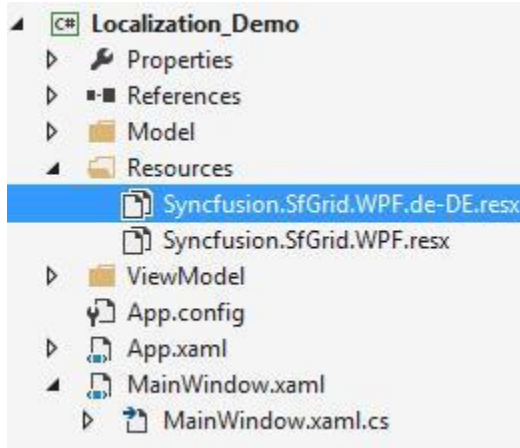
1.Create new folder and named as `Resources` in your application. 2.Add the default resource file of treegrid into `Resources` folder. You can download the Syncfusion.SfGrid.WPF.resx [here](#).



3. Right-click on the Resources folder, select **Add** and then **NewItem**. 4. In Add New Item wizard, select the **Resource File** option and name the filename as **Syncfusion.SfGrid.WPF.<culture name>.resx**. For example, you have to give name as **Syncfusion.SfGrid.WPF.de.resx** for German culture. 5. The culture name that indicates the name of language and country.



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



7. Add the Name/Value pair in Resource Designer of `Syncfusion.SfGrid.WPF.de.resx` file and change its corresponding value to corresponding culture.

	Name	Value	Comment
	Contains	Contains	Text for Contains
	CustomMargin	Custom Margin	Text for PrintDialog CustomMargin
	CustomMargins	Custom Margins	Text for PrintDialog
	CustomPageSizes	Custom Page Sizes	Text for PrintDialog
	CustomSize	Custom Size	Text for PrintDialog CustomSize
	DateFilters	Date Filters	Text for DateFilters
	Done	Done	Text for Done button
	DraggingRowCount	Zeilenanzahl ziehen	Text for Dragging rows count in DragAndDrop popup
	DropAbove	Oben fallen	Text for Drop above in RowDragAndDrop popup
▶	DropAsChild	Als Kind fallen lassen	Text for Drop as child in RowDragAndDrop popup of
	DropBelow	Unten fallen	Text for Drop below in RowDragAndDrop popup
	DropStatus	Drop status;	Text for Drop status in RowDragAndDrop popup
	Empty	Empty	Text for Empty
	EndsWith	Ends With	Text for EndsWith

	FirstName	LastName	Title	ID	Salary
[-]	Ferando	Joseph	Management	2	2000000
	Andrew	Fuller	Vice President	9	1200000
	Janet	Leverling	GM	10	1000000
[-]	Steven	Buchanan	Manager	11	900000
[+]	John	Peter	Accounts	3	2000000
	Nancy		Accounts Manager	12	850000
[>]	Ferando1		Management1	4	20000000
	Nancy1	Davolio1	Accounts Manager1	16	850000
	Margaret1	Peacock1	Accountant1	17	700000
	Michael1	Suyama1	Accountant1	18	700000
	Margaret	Peacock	Accountant	13	700000
	Michael	Suyama	Accountant	14	700000
	Robert	King	Accountant	15	650000
[-]	John1	Fernanta	Accounts1	5	20000000

You can download the sample [here](#).

Localize when the resource file is present in different assembly or different namespace

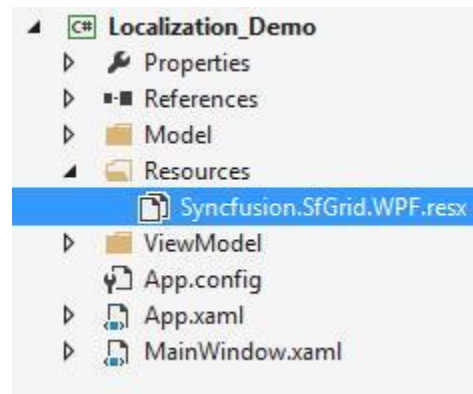
By default, the treegrid try to read the resource file from executing assembly and its default namespace by using [Assembly.GetExecuteAssembly](#) method. When the resource file is located at different assembly or namespace, then you can let treegrid know by using [GridResourceWrapper.SetResources](#) method.

C#

```
public MainWindow ()
{
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("de-DE");
    Syncfusion.UI.Xaml.Grid.GridResourceWrapper.SetResources("Assembly_name",
    "namespace_name");
    InitializeComponent();
}
```

Edit default culture resource

You can edit default resource file by adding it to **Resources** folder of your application where treegrid reads the static texts from here. You can download the default resource file from [here](#).



Now, change the Name/Value pair in Resource Designer of `Syncfusion.SfGrid.WPF.resx` file.

	Name	Value	Comment
	Contains	Contains	Text for Contains
	CustomMargin	Custom Margin	Text for PrintDialog
	CustomMargins	Custom Margins	Text for PrintDialog
	CustomPageSizes	Custom Page Sizes	Text for PrintDialog
	CustomSize	Custom Size	Text for PrintDialog CustomSize
	DateFilters	Date Filters	Text for DateFilters
	Done	Done	Text for Done button
	DraggingRowCount	Dragging rows count :	Text for Dragging rows count in
▶	DropAbove	Drop above to row	Text for Drop above in RowDragAndDrop popup
	DropAsChild	Drop as child	Text for Drop as child in
	DropBelow	Drop below	Text for Drop below in
	DropStatus	Drop status :	Text for Drop status in
	Empty	Empty	Text for Empty
	EndsWith	Ends With	Text for EndsWith
	EnterValidFilterValue	Enter Valid Filter Value	Text for EnterValidFilterValue
	Equalss	Equals	Text for Equals
	FitAllColumnsOnOne	Fit All Columns on One Page	Text for

	FirstName	LastName	Title	ID	Salary
[-]	Ferando	Joseph	Management	2	2000000
	Andrew	Fuller	Vice President	9	1200000
	Janet	Leverling	GM	10	1000000
[+]	Steven	Buchanan	Manager	11	900000
[+]	John		Accounts	3	2000000
	Nancy		Accounts Manager	12	850000
[>]	Ferando1		management1	4	20000000
	Nancy1	Davolio1	Accounts Manager1	16	850000
	Margaret1	Peacock1	Accountant1	17	700000
	Michael1	Suyama1	Accountant1	18	700000
	Margaret	Peacock	Accountant	13	700000
	Michael	Suyama	Accountant	14	700000
	Robert	King	Accountant	15	650000
[+]	John1	Fernanta	Accounts1	5	20000000
	Robert1	King1	Accountant1	19	650000
	Nancy2	Davolio2	Accounts Manager2	20	850000

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Styles and Templates in WPF TreeGrid (SfTreeGrid)

Styling Column Header

The header cell can be customized by writing style of TargetType [TreeGridHeaderCellControl](#). You can set to particular SfTreeGrid by setting [SfTreeGrid.HeaderStyle](#) property and the particular column can be styled by setting [TreeGridColumn.HeaderStyle](#) property.

Note: TreeGridColumn.HeaderStyle takes higher priority than SfTreeGrid.HeaderStyle property.

XML

```
<Window.Resources>
<Style TargetType="syncfusion:TreeGridHeaderCell" x:Key="headerStyle">
<Setter Property="FontWeight" Value="Bold"/>
<Setter Property="FontSize" Value="14"/>
<Setter Property="Background" Value="LightPink"/>
<Setter Property="Foreground" Value="DarkBlue"/>
</Style>
</Window.Resources>
<syncfusion:SfTreeGrid Name="treeGrid" Grid.Column="1" Grid.Row="1"
HeaderStyle="{StaticResource headerStyle}"
AutoExpandMode="RootNodesExpanded"
AutoGenerateColumns="False"
ChildPropertyName="Children"
ItemsSource="{Binding EmployeeDetails}"
LiveNodeUpdateMode="AllowDataShaping">
```


Order ID	Order Date	Customer ID	Customer Area	Price	Discount
[-] 1000	8/31/2000	SIMOB	London	\$32000.00	10.20%
[+] 1000	2/28/2007	FURIB	NewYork	\$33000.00	10.20%
[+] 1001	4/27/1998	VAFFE	Adelaide	\$37000.00	15.00%
[+] 1002	9/6/2002	SEVES	Birmingham	\$31000.00	11.00%
[+] 1003	7/26/1998	FOLIG	Liverpool	\$37000.00	7.70%
[+] 1004	3/4/2004	MEREP	NewYork	\$31000.00	14.00%
[+] 1005	10/30/2004	MEREP	Cardiff	\$35000.00	14.00%
[+] 1006	10/22/2001	SEVES	London	\$31000.00	5.50%
[+] 1007	9/3/2001	MEREP	NewYork	\$33000.00	10.20%
[+] 1008	10/14/2003	LINOD	Perth	\$35000.00	7.70%
[+] 1009	7/2/1999	MEREP	Canberra	\$31000.00	16.00%

Styling Stacked Headers

The appearance of stacked header can be customized by writing style of TargetType [TreeGridStackedHeaderCell](#).

XML

```
<Window.Resources>
<Style TargetType="syncfusion:TreeGridStackedHeaderCell">
<Setter Property="FontWeight" Value="Bold"/>
<Setter Property="FontSize" Value="14"/>
<Setter Property="Foreground" Value="DarkBlue"/>
</Style>
</Window.Resources>
```

Order Details		Customer Details		Price Details	
Order ID	Order Date	Customer ID	Customer Area	Price	Discount
[-] 1000	8/31/2000	SIMOB	London	\$32000.00	10.20%
[+] 1000	2/28/2007	FURIB	NewYork	\$33000.00	10.20%
[+] 1001	4/27/1998	VAFFE	Adelaide	\$37000.00	15.00%
[+] 1002	9/6/2002	SEVES	Birmingham	\$31000.00	11.00%
[+] 1003	7/26/1998	FOLIG	Liverpool	\$37000.00	7.70%
[+] 1004	3/4/2004	MEREP	NewYork	\$31000.00	14.00%
[+] 1005	10/30/2004	MEREP	Cardiff	\$35000.00	14.00%
[+] 1006	10/22/2001	SEVES	London	\$31000.00	5.50%
[+] 1007	9/3/2001	MEREP	NewYork	\$33000.00	10.20%
[+] 1008	10/14/2003	LINOD	Perth	\$35000.00	7.70%
[+] 1009	7/2/1999	MEREP	Canberra	\$31000.00	16.00%
[-] 1001	12/5/2005	FOLIG	Liverpool	\$31000.00	7.70%
[+] 1000	8/20/2002	FRANS	Durban	\$37000.00	7.70%

Setting Different Style for Each Stacked Header

You can apply the different style to stacked header by overriding the [default renderer](#) of StackedHeader.

XML

```
<Application.Resources>
<Style x:Key="style1" TargetType="syncfusion:TreeGridStackedHeaderCell">
<Setter Property="Background" Value="LightBlue" />
<Setter Property="FontFamily" Value="Segoe UI" />
<Setter Property="FontStyle" Value="Italic" />
<Setter Property="FontWeight" Value="Bold"/>
</Style>
<Style x:Key="style2" TargetType="syncfusion:TreeGridStackedHeaderCell">
<Setter Property="Background" Value="Bisque" />
<Setter Property="FontFamily" Value="Courier New" />
<Setter Property="FontStyle" Value="Oblique" />
<Setter Property="FontWeight" Value="Bold"/>
</Style>
<Style x:Key="style3" TargetType="syncfusion:TreeGridStackedHeaderCell">
<Setter Property="Background" Value="LightPink" />
<Setter Property="FontFamily" Value="Segoe UI" />
<Setter Property="FontStyle" Value="Oblique" />
<Setter Property="FontWeight" Value="Bold"/>
</Style>
</Application.Resources>
```

C#

```
//Default TreeGridStackedCellRenderer is removed.
this.treeGrid.CellRenderers.Remove("StackedHeader");
//Customized TreeGridStackedCellRenderer is added.
this.treeGrid.CellRenderers.Add("StackedHeader", new
TreeGridCustomStackedRenderer());
public class TreeGridCustomStackedRenderer :
TreeGridStackedHeaderCellRenderer
{
public TreeGridCustomStackedRenderer()
{
}
public override void OnInitializeEditElement(TreeDataColumnBase dataColumn,
TreeGridStackedHeaderCell uiElement, object dataContext)
{
if (dataColumn.ColumnIndex == 0)
uiElement.Style = App.Current.Resources["style1"] as Style;
else if (dataColumn.ColumnIndex == 2)
uiElement.Style = App.Current.Resources["style2"] as Style;
else
uiElement.Style = App.Current.Resources["style3"] as Style;
base.OnInitializeEditElement(dataColumn, uiElement, dataContext);
}
}
```

Order Details		Customer Details		Price Details	
Order ID	Order Date	Customer ID	Customer Area	Price	Discount
[-] 1000	8/31/2000	SIMOB	London	\$32000.00	10.20%
[+] 1000	2/28/2007	FURIB	NewYork	\$33000.00	10.20%
[+] 1001	4/27/1998	VAFFE	Adelaide	\$37000.00	15.00%
[+] 1002	9/6/2002	SEVES	Birmingham	\$31000.00	11.00%
[+] 1003	7/26/1998	FOLIG	Liverpool	\$37000.00	7.70%
[+] 1004	3/4/2004	MEREP	NewYork	\$31000.00	14.00%
[+] 1005	10/30/2004	MEREP	Cardiff	\$35000.00	14.00%
[+] 1006	10/22/2001	SEVES	London	\$31000.00	5.50%
[+] 1007	9/3/2001	MEREP	NewYork	\$33000.00	10.20%
[+] 1008	10/14/2003	LINOD	Perth	\$35000.00	7.70%
[+] 1009	7/2/1999	MEREP	Canberra	\$31000.00	16.00%

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Conditional Styling in WPF TreeGrid (SfTreeGrid)

You can style the treegrid and its inner elements conditionally based on data in three ways,

1. Using Converter
2. Using Data Triggers
3. Using StyleSelector

Styling ways	Performance details
Converter	Provide good performance when compared other two ways.
Trigger	When compared to converter, performance is slow while styling more number of columns or rows.
StyleSelector	It affects scrolling performance while styling more number of columns based on number of columns visible.

Conditional Styling in WPF TreeGrid (SfTreeGrid)

Style cells using converter

The record cells ([TreeGridCell](#)) can be customized conditionally by changing its property value based on cell value or data object using converter.

Here, grid cell background is changed using converter, where converter returns the value based on ID property of underlying record.

XML

```
<syncfusion:Window.Resources>
<local:StyleConverter x:Key="converter" />
```



```
</syncfusion:Window.Resources>
<syncfusion:TreeGridTextColumn HeaderText="ID"
MappingName="Id"
TextAlignment="Left" >
<syncfusion:TreeGridTextColumn.CellStyle>
<Style TargetType="syncfusion:TreeGridCell">
<Setter Property="Background" Value="{Binding
Path=Id,Converter={StaticResource converter}}"/>
</Style>
</syncfusion:TreeGridTextColumn.CellStyle>
</syncfusion:TreeGridTextColumn>
```

C#

```
internal class StyleConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        int input = (int)value;
        //custom condition is checked based on data.
        if (input < 10)
            return new SolidColorBrush(Colors.LightBlue);
        else if (input >10 && input<20)
            return new SolidColorBrush(Colors.Bisque);
        else
            return DependencyProperty.UnsetValue;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

First Name	Last Name	ID	Availability	DOB
<input type="checkbox"/> Abraham	Buchanan	1	<input checked="" type="checkbox"/>	6/28/2006
<input type="checkbox"/> Theodore	Fillmore	2	<input checked="" type="checkbox"/>	9/21/1999
<input type="checkbox"/> Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/17/2004
George	Stogner	4	<input checked="" type="checkbox"/>	3/16/2004
Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/6/1998
Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/18/2001
Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/21/2004
Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/11/1996
<input type="checkbox"/> Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/9/2006
Herbert	Clinton	10	<input checked="" type="checkbox"/>	10/3/2002
Jimmy	Clinton	11	<input checked="" type="checkbox"/>	1/16/2000
Abraham	Clinton	12	<input checked="" type="checkbox"/>	12/4/1998
Thomas	Clinton	13	<input checked="" type="checkbox"/>	10/9/2004
Ulysses	Clinton	14	<input checked="" type="checkbox"/>	7/28/2003

Style cells based on record using converter

You can also style the cells based on record instead of passing single property to converter, where converter returns the value based on underlying record. This can be assigned to GridColumn.CellStyle to style the column based on other column properties.

XML

```
<syncfusion:Window.Resources>
<local:StyleConverter x:Key="converter" />
<Style TargetType="syncfusion:TreeGridCell">
<Setter Property="Background" Value="{Binding Converter={StaticResource
converter}}"/>
</Style>
</syncfusion:Window.Resources>
```

C#

```
internal class StyleConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        var data = value as PersonInfo;
        //custom condition is checked based on data.
        if (data.Id < 10)
            return new SolidColorBrush(Colors.LightBlue);
        else if (data.Id < 20 && data.Id > 10)
            return new SolidColorBrush(Colors.Bisque);
        else
            return DependencyProperty.UnsetValue;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
```

```
{
    throw new NotImplementedException();
}
```

First Name	Last Name	ID	Availability	DOB
<input type="checkbox"/> Chester	Buchanan	0	<input checked="" type="checkbox"/>	3/1/2007
<input type="checkbox"/> Abraham	Buchanan	1	<input checked="" type="checkbox"/>	6/28/2006
<input type="checkbox"/> Theodore	Fillmore	2	<input checked="" type="checkbox"/>	9/21/1999
<input type="checkbox"/> Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/17/2004
George	Stogner	4	<input checked="" type="checkbox"/>	3/16/2004
Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/6/1998
Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/18/2001
Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/21/2004
Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/11/1996
<input type="checkbox"/> Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/9/2006
Herbert	Clinton	10	<input checked="" type="checkbox"/>	10/3/2002
Jimmy	Clinton	11	<input checked="" type="checkbox"/>	1/16/2000
Abraham	Clinton	12	<input checked="" type="checkbox"/>	12/4/1998
Thomas	Clinton	13	<input checked="" type="checkbox"/>	10/9/2004
Ulysses	Clinton	14	<input checked="" type="checkbox"/>	7/28/2003

Style cells using triggers

The record cells ([TreeGridCell](#)) can be customized by setting [Style.Triggers](#) that apply property values based on specified conditions. Multiple conditions can be specified by setting [MultiDataTrigger](#).

XML

```
<syncfusion:TreeGridTextColumn HeaderText="ID"
    MappingName="Id"
    TextAlignment="Left" >
    <syncfusion:TreeGridTextColumn.CellStyle>
    <Style TargetType="syncfusion:TreeGridCell">
    <Style.Triggers>
    <!--Background property set based on cell content-->
    <DataTrigger Binding="{Binding Path=Id}" Value="3">
    <Setter Property="Background" Value="Bisque" />
    </DataTrigger>
    <!--Background property set based on multiple conditions-->
    <MultiDataTrigger>
    <MultiDataTrigger.Conditions>
    <Condition Binding="{Binding Path=Id}" Value="11" />
    <Condition Binding="{Binding Path=FirstName}" Value="Jimmy" />
    </MultiDataTrigger.Conditions>
    <Setter Property="Background" Value="LightBlue" />
    </MultiDataTrigger>
    </Style.Triggers>
    </Style>
    </syncfusion:TreeGridTextColumn.CellStyle>
```

```
</syncfusion:TreeGridTextColumn>
```

First Name	Last Name	ID	Availability	DOB
<input type="checkbox"/> Chester	Buchanan	0	<input checked="" type="checkbox"/>	3/1/2007
<input type="checkbox"/> Abraham	Buchanan	1	<input checked="" type="checkbox"/>	6/28/2006
<input type="checkbox"/> Theodore	Fillmore	2	<input checked="" type="checkbox"/>	9/21/1999
<input type="checkbox"/> Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/17/2004
George	Stogner	4	<input checked="" type="checkbox"/>	3/16/2004
Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/6/1998
Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/18/2001
Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/21/2004
Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/11/1996
<input type="checkbox"/> Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/9/2006
Herbert	Clinton	10	<input checked="" type="checkbox"/>	10/3/2002
Jimmy	Clinton	11	<input checked="" type="checkbox"/>	1/16/2000
Abraham	Clinton	12	<input checked="" type="checkbox"/>	12/4/1998
Thomas	Clinton	13	<input checked="" type="checkbox"/>	10/9/2004
Ulysses	Clinton	14	<input checked="" type="checkbox"/>	7/28/2003

Style cells using style selector

The record cells ([TreeGridCell](#)) can be customized conditionally based on data by setting [SfTreeGrid.CellStyleSelector](#) property and the particular column record cells can be customized by setting [GridColumn.CellStyleSelector](#) property and you can get the container as [TreeGridCell](#) in the [StyleSelector](#).

XML

```
<Application.Resources>
<local:SelectorClass x:Key="styleSelector" />
<Style x:Key="redCellStyle" TargetType="syncfusion:TreeGridCell">
<Setter Property="Foreground" Value="Red" />
</Style>
<Style x:Key="blueCellStyle" TargetType="syncfusion:TreeGridCell">
<Setter Property="Foreground" Value="DarkBlue" />
</Style>
</Application.Resources>
```

C#

```
public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var data = item as PersonInfo;
        if (data != null && ((container as TreeGridCell).ColumnBase.TreeGridColumn.MappingName == "Id"))
        {

```

```
//custom condition is checked based on data.
if (data.Id < 10)
return App.Current.Resources["redCellStyle"] as Style;
return App.Current.Resources["blueCellStyle"] as Style;
}
return base.SelectStyle(item, container);
}
}
```

First Name	Last Name	ID	Availability	DOB
<input type="checkbox"/> Chester	Buchanan	0	<input checked="" type="checkbox"/>	3/1/2007
<input type="checkbox"/> Abraham	Buchanan	1	<input checked="" type="checkbox"/>	6/28/2006
<input type="checkbox"/> Theodore	Fillmore	2	<input checked="" type="checkbox"/>	9/21/1999
<input type="checkbox"/> Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/17/2004
George	Stogner	4	<input checked="" type="checkbox"/>	3/16/2004
Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/6/1998
Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/18/2001
Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/21/2004
Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/11/1996
<input type="checkbox"/> Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/9/2006
Herbert	Clinton	10	<input checked="" type="checkbox"/>	10/3/2002
Jimmy	Clinton	11	<input checked="" type="checkbox"/>	1/16/2000
Abraham	Clinton	12	<input checked="" type="checkbox"/>	12/4/1998
Thomas	Clinton	13	<input checked="" type="checkbox"/>	10/9/2004
Ulysses	Clinton	14	<input checked="" type="checkbox"/>	7/28/2003

Add image to cell

You can add the image to tree grid cell by using TreeGridTemplateColumn,

XML

```
<syncfusion:TreeGridTemplateColumn MappingName="ImageLink">
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<Image Source="{Binding Path=ImageLink,
Converter={StaticResource converter}}"/>
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.CellTemplate>
</syncfusion:TreeGridTemplateColumn>
```

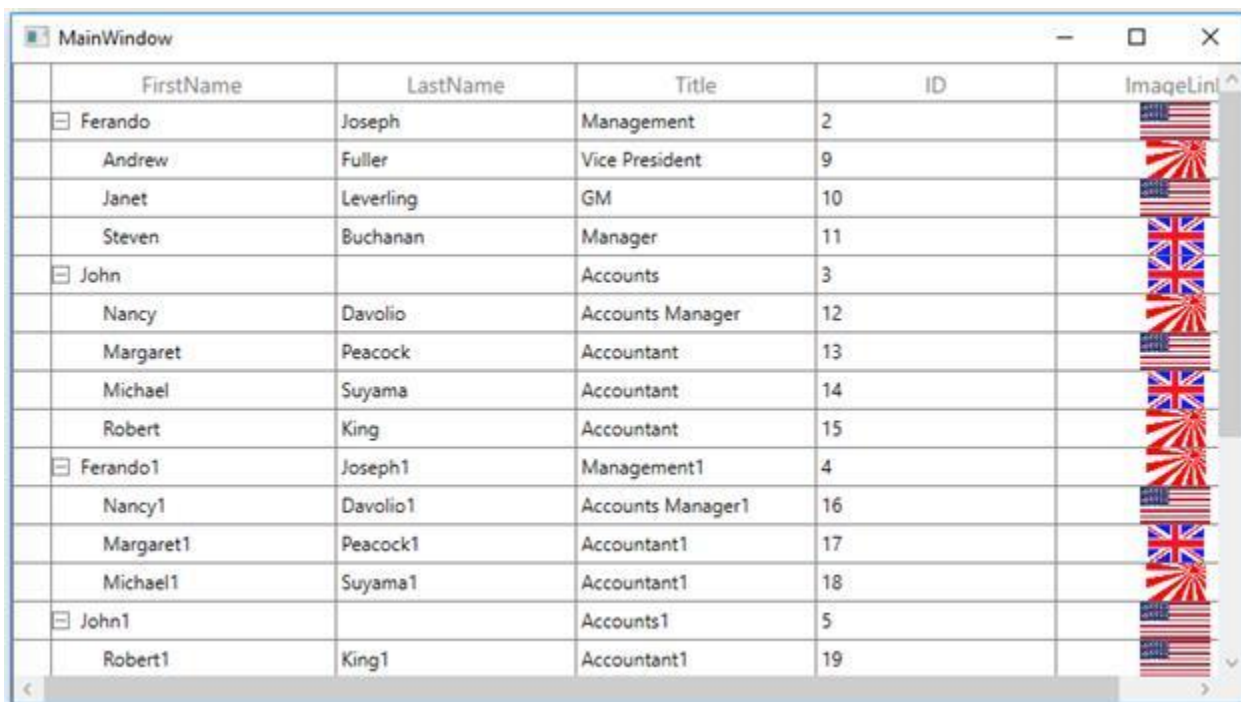
C#

```
public class StringToImageConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
string imagename = value as string;
```

```

return new BitmapImage(new Uri(string.Format(@"..\..\Images\{0}",
imagename), UriKind.Relative));
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
return null;
}
}

```



You can download the sample [here](#).

Conditional Styling in WPF TreeGrid (SfTreeGrid)

Style rows using converter

The record rows ([TreeGridRowControl](#)) can be customized conditionally by changing its property value based on 'cell value' or 'data object' by using converter, where converter returns the value based on underlying record.

XML

```

<syncfusion:Window.Resources>
<local:StyleConverter x:Key="converter"/>
<Style TargetType="syncfusion:TreeGridRowControl">
<Setter Property="Background" Value="{Binding Converter={StaticResource
converter}}"/>
</Style>
</syncfusion:Window.Resources>

```

C#

```

public class StyleConverter : IValueConverter

```



```

{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var input = (value as PersonInfo).Id;
        //custom condition is checked based on data.
        if (input < 10)
            return new SolidColorBrush(Colors.Bisque);
        else if (input < 20 && input>10)
            return new SolidColorBrush(Colors.LightBlue);
        else
            return DependencyProperty.UnsetValue;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

First Name	Last Name	ID	Availability	DOB	^
<input type="checkbox"/> Chester	Buchanan	0	<input checked="" type="checkbox"/>	3/1/2007	
<input type="checkbox"/> Abraham	Buchanan	1	<input checked="" type="checkbox"/>	6/28/2006	
<input type="checkbox"/> Theodore	Fillmore	2	<input checked="" type="checkbox"/>	9/21/1999	
<input type="checkbox"/> Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/17/2004	
George	Stogner	4	<input checked="" type="checkbox"/>	3/16/2004	
Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/6/1998	
Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/18/2001	
Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/21/2004	
Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/11/1996	
<input type="checkbox"/> Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/9/2006	
Herbert	Clinton	10	<input checked="" type="checkbox"/>	10/3/2002	
Jimmy	Clinton	11	<input checked="" type="checkbox"/>	1/16/2000	
Abraham	Clinton	12	<input checked="" type="checkbox"/>	12/4/1998	
Thomas	Clinton	13	<input checked="" type="checkbox"/>	10/9/2004	
Ulysses	Clinton	14	<input checked="" type="checkbox"/>	7/28/2003	▼

Style rows using style selector

The record rows ([TreeGridRowControl](#)) can be customized conditionally based on data by setting [SfTreeGrid.RowStyleSelector](#) property and you can get the container as [TreeGridRowControl](#) in [StyleSelector](#).

XML

```

<Application.Resources>
<local:SelectorClass x:Key="rowStyleSelector" />
<Style x:Key="rowStyle1" TargetType="syncfusion:TreeGridRowControl">
<Setter Property="Background" Value="Red" />

```

```

</Style>
<Style x:Key="rowStyle2" TargetType="syncfusion:TreeGridRowControl">
<Setter Property="Background" Value="DarkBlue" />
</Style>
</Application.Resources>

```

C#

```

public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var row = (item as TreeDataRowBase).RowData;
        var data = row as PersonInfo;
        if (data.Id < 10)
        {
            return App.Current.Resources["rowStyle1"] as Style;
        }
        return App.Current.Resources["rowStyle2"] as Style;
    }
}

```

First Name	Last Name	ID	Availability	DOB
<input checked="" type="checkbox"/> Chester	Buchanan	0	<input checked="" type="checkbox"/>	3/1/2007
<input checked="" type="checkbox"/> Abraham	Buchanan	1	<input checked="" type="checkbox"/>	6/28/2006
<input checked="" type="checkbox"/> Theodore	Filmore	2	<input checked="" type="checkbox"/>	9/21/1999
<input checked="" type="checkbox"/> Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/17/2004
George	Stogner	4	<input checked="" type="checkbox"/>	3/16/2004
Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/6/1998
Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/18/2001
Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/21/2004
Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/11/1996
<input checked="" type="checkbox"/> Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/9/2006
Henry	Simon	10	<input checked="" type="checkbox"/>	10/30/2001
Samuel	Simon	11	<input checked="" type="checkbox"/>	1/16/2000
Abraham	Simon	12	<input checked="" type="checkbox"/>	12/4/1998
Thomas	Simon	13	<input checked="" type="checkbox"/>	10/30/2004
Charles	Simon	14	<input checked="" type="checkbox"/>	1/28/2004

Conditional Styling in WPF TreeGrid (SfTreeGrid)

The appearance of row header ([GridRowHeaderCell](#)) can be customized conditionally by changing its property value based on 'cell value' or 'data object' by using converter, where converter returns the value based on Underlying record.

XML

```

<syncfusion:ChromelessWindow.Resources>
<local:StyleConverter x:Key="converter"/>
<Style TargetType="syncfusion:TreeGridRowHeaderCell">

```



```
<Setter Property="Background" Value="{Binding Converter={StaticResource
converter}}" />
</Style>
</syncfusion:ChromelessWindow.Resources>
```

C#

```
public class StyleConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var data = value as PersonInfo;
        //custom condition is checked.
        if (data.Id<10)
            return Brushes.Green;
        else
            return Brushes.Red;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

	First Name	Last Name	ID	Availability	DOB
<input type="checkbox"/>	Chester	Buchanan	0	<input checked="" type="checkbox"/>	3/1/2007
<input type="checkbox"/>	Abraham	Buchanan	1	<input checked="" type="checkbox"/>	6/28/2006
<input type="checkbox"/>	Theodore	Fillmore	2	<input checked="" type="checkbox"/>	9/21/1999
<input type="checkbox"/>	Ulysses	Harrison	3	<input checked="" type="checkbox"/>	3/17/2004
	George	Stogner	4	<input checked="" type="checkbox"/>	3/16/2004
	Franklin	Stogner	5	<input checked="" type="checkbox"/>	10/6/1998
	Andrew	Stogner	6	<input checked="" type="checkbox"/>	3/18/2001
	Theodore	Stogner	7	<input checked="" type="checkbox"/>	4/21/2004
	Rutherford	Stogner	8	<input checked="" type="checkbox"/>	12/11/1996
<input type="checkbox"/>	Calvin	Harrison	9	<input checked="" type="checkbox"/>	6/9/2006
	Herbert	Clinton	10	<input checked="" type="checkbox"/>	10/3/2002
	Jimmy	Clinton	11	<input checked="" type="checkbox"/>	1/16/2000
	Abraham	Clinton	12	<input checked="" type="checkbox"/>	12/4/1998
	Thomas	Clinton	13	<input checked="" type="checkbox"/>	10/9/2004
	Ulysses	Clinton	14	<input checked="" type="checkbox"/>	7/28/2003

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Grid Lines customization in WPF TreeGrid (SfTreeGrid)

SfTreeGrid allows you to customize the grid lines visibility to vertical, horizontal, both or none. To achieve this, use the following properties.

[SfTreeGrid.GridLinesVisibility](#): To set the border lines for the cells other than header and stacked header cells.

[SfTreeGrid.HeaderLinesVisibility](#): To set the border lines only for header and stacked header cells.

The following are the list of options available to customize grid lines visibility,

- Both
- Vertical
- Horizontal
- None

Record rows

Both

The [GridLinesVisibility.Both](#) displays the TreeGrid with both horizontal and vertical grid lines. By default GridLinesVisibility value set as Both.

XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfTreeGrid x:Name="sfTreeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="ReportsTo"
ParentPropertyName="ID"
ItemsSource="{Binding Employees}"
GridLinesVisibility="Both"
SelfRelationRootValue="-1"/>
```

C#

```
this.sfTreeGrid.GridLinesVisibility = GridLinesVisibility.Both;
```

First Name	Last Name	Employee ID	Title	Salary	Reports To
<input type="checkbox"/> Ferando	Joseph	2	Management	\$2000000.00	-1
Andrew	Fuller	9	Vice President	\$1200000.00	2
Janet	Leverling	10	GM	\$1000000.00	2
Steven	Buchanan	11	Manager	\$900000.00	2
<input type="checkbox"/> John	Adams	3	Accounts	\$2000000.00	-1
Nancy	Davolio	12	Accounts Manager	\$850000.00	3
Margaret	Peacock	13	Accountant	\$700000.00	3
Michael	Suyama	14	Accountant	\$700000.00	3
Robert	King	15	Accountant	\$650000.00	3
<input type="checkbox"/> Thomas	Jefferson	4	Sales	\$300000.00	-1
Laura	Callahan	16	Sales Manager	\$900000.00	4
Anne	Dodsworth	17	Sales Representative	\$800000.00	4
Albert	Hellstern	18	Sales Representative	\$750000.00	4
Tim	Smith	19	Sales Representative	\$700000.00	4
Justin	Brid	20	Sales Representative	\$700000.00	4

Horizontal

The [GridLinesVisibility.Horizontal](#) displays the TreeGrid with horizontal grid lines only.

XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfTreeGrid x:Name="sfTreeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="ReportsTo"
ParentPropertyName="ID"
ItemsSource="{Binding Employees}"
GridLinesVisibility="Horizontal"
SelfRelationRootValue="-1"/>
```

C#

```
this.sfTreeGrid.GridLinesVisibility = GridLinesVisibility.Horizontal;
```

First Name	Last Name	Employee ID	Title	Salary	Reports To
<input type="checkbox"/> Ferando	Joseph	2	Management	\$2000000.00	-1
Andrew	Fuller	9	Vice President	\$1200000.00	2
Janet	Leverling	10	GM	\$1000000.00	2
Steven	Buchanan	11	Manager	\$900000.00	2
<input type="checkbox"/> John	Adams	3	Accounts	\$2000000.00	-1
Nancy	Davolio	12	Accounts Manager	\$850000.00	3
Margaret	Peacock	13	Accountant	\$700000.00	3
Michael	Suyama	14	Accountant	\$700000.00	3
Robert	King	15	Accountant	\$650000.00	3
<input type="checkbox"/> Thomas	Jefferson	4	Sales	\$300000.00	-1
Laura	Callahan	16	Sales Manager	\$900000.00	4
Anne	Dodsworth	17	Sales Representative	\$800000.00	4
Albert	Hellstern	18	Sales Representative	\$750000.00	4
Tim	Smith	19	Sales Representative	\$700000.00	4
Justin	Brid	20	Sales Representative	\$700000.00	4

Vertical

The [GridLinesVisibility.Vertical](#) displays the TreeGrid with vertical grid lines only.

XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfTreeGrid x:Name="sfTreeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
GridLinesVisibility="Vertical"
ParentPropertyName="ID"
SelfRelationRootValue="-1"/>
```

C#

```
this.sfTreeGrid.GridLinesVisibility = GridLinesVisibility.Vertical;
```

First Name	Last Name	Employee ID	Title	Salary	Reports To
[-] Ferando	Joseph	2	Management	\$2000000.00	-1
Andrew	Fuller	9	Vice President	\$1200000.00	2
Janet	Leverling	10	GM	\$1000000.00	2
Steven	Buchanan	11	Manager	\$900000.00	2
[-] John	Adams	3	Accounts	\$2000000.00	-1
Nancy	Davolio	12	Accounts Manager	\$850000.00	3
Margaret	Peacock	13	Accountant	\$700000.00	3
Michael	Suyama	14	Accountant	\$700000.00	3
Robert	King	15	Accountant	\$650000.00	3
[-] Thomas	Jefferson	4	Sales	\$300000.00	-1
Laura	Callahan	16	Sales Manager	\$900000.00	4
Anne	Dodsworth	17	Sales Representative	\$800000.00	4
Albert	Hellstern	18	Sales Representative	\$750000.00	4
Tim	Smith	19	Sales Representative	\$700000.00	4
Justin	Brid	20	Sales Representative	\$700000.00	4

None

[GridLinesVisibility.None](#) displays the TreeGrid without grid lines.

XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfTreeGrid x:Name="sfTreeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="ReportsTo"
ItemsSource="{Binding Employees}"
GridLinesVisibility="None"
ParentPropertyName="ID"
SelfRelationRootValue="-1"/>
```

C#

```
this.sfTreeGrid.GridLinesVisibility = GridLinesVisibility.None;
```

First Name	Last Name	Employee ID	Title	Salary	Reports To
[-] Ferando	Joseph	2	Management	\$2000000.00	-1
Andrew	Fuller	9	Vice President	\$1200000.00	2
Janet	Leverling	10	GM	\$1000000.00	2
Steven	Buchanan	11	Manager	\$900000.00	2
[-] John	Adams	3	Accounts	\$2000000.00	-1
Nancy	Davolio	12	Accounts Manager	\$850000.00	3
Margaret	Peacock	13	Accountant	\$700000.00	3
Michael	Suyama	14	Accountant	\$700000.00	3
Robert	King	15	Accountant	\$650000.00	3
[-] Thomas	Jefferson	4	Sales	\$300000.00	-1
Laura	Callahan	16	Sales Manager	\$900000.00	4
Anne	Dodsworth	17	Sales Representative	\$800000.00	4
Albert	Hellstern	18	Sales Representative	\$750000.00	4
Tim	Smith	19	Sales Representative	\$700000.00	4
Justin	Brid	20	Sales Representative	\$700000.00	4

Header rows

You can customize the TreeGrid header lines visibility by using the [SfTreeGrid.HeaderLinesVisibility](#) property. You can also customize the header lines visibility to horizontal, vertical, none or both. By default HeaderLinesVisibility value set as Both.

XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfTreeGrid x:Name="sfTreeGrid"
AutoExpandMode="RootNodesExpanded"
ChildPropertyName="ReportsTo"
ParentPropertyName="ID"
ItemsSource="{Binding Employees}"
HeaderLinesVisibility="Horizontal"
SelfRelationRootValue="-1"/>
```

C#

```
this.sfTreeGrid.HeaderLinesVisibility = GridLinesVisibility.Horizontal;
```

First Name	Last Name	Employee ID	Title	Salary	Reports To
<input type="checkbox"/> Ferando	Joseph	2	Management	\$2000000.00	-1
Andrew	Fuller	9	Vice President	\$1200000.00	2
Janet	Leverling	10	GM	\$1000000.00	2
Steven	Buchanan	11	Manager	\$900000.00	2
<input type="checkbox"/> John	Adams	3	Accounts	\$2000000.00	-1
Nancy	Davolio	12	Accounts Manager	\$850000.00	3
Margaret	Peacock	13	Accountant	\$700000.00	3
Michael	Suyama	14	Accountant	\$700000.00	3
Robert	King	15	Accountant	\$650000.00	3
<input type="checkbox"/> Thomas	Jefferson	4	Sales	\$300000.00	-1
Laura	Callahan	16	Sales Manager	\$900000.00	4
Anne	Dodsworth	17	Sales Representative	\$800000.00	4
Albert	Hellstern	18	Sales Representative	\$750000.00	4
Tim	Smith	19	Sales Representative	\$700000.00	4
Justin	Brid	20	Sales Representative	\$700000.00	4

Limitations

- Grid lines customization are not supported for RowHeader.

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Themes in WPF TreeGrid (SfTreeGrid)

SfTreeGrid provides built-in themes which can be applied using [SfSkinManager](#) and also provides support to create custom theme using [theme studio](#).

Built-in Themes

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

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

FirstName	LastName	ID	Title	Salary	ReportsTo
▼ Ferando	Joseph	2	Management	2000000.00	-1
Andrew	Fuller	9	Vice President	1200000.00	2
Janet	Leverling	10	GM	1000000.00	2
Steven	Buchanan	11	Manager	900000.00	2
▼ John	Adams	3	Accounts	2000000.00	-1
Nancy	Davolio	12	Accounts Manager	850000.00	3
Margaret	Peacock	13	Accountant	700000.00	3
Michael	Suyama	14	Accountant	700000.00	3
Robert	King	15	Accountant	650000.00	3
▼ Thomas	Jefferson	4	Sales	300000.00	-1
Laura	Callahan	16	Sales Manager	900000.00	4
Anne	Dodsworth	17	Sales Representative	800000.00	4
Albert	Hellstern	18	Sales Representative	750000.00	4
Tim	Smith	19	Sales Representative	700000.00	4
Justin	Brid	20	Sales Representative	700000.00	4
▼ Andrew	Madison	5	Marketing	4000000.00	-1
Caroline	Patterson	21	Receptionist	800000.00	5
Xavier	Martin	22	Mail Clerk	700000.00	5
▼ Ulysses	Pierce	6	HumanResource	1500000.00	-1
Laurent	Pereira	23	HR Manager	900000.00	6
Syed	Abbas	24	HR Assistant	650000.00	6
Amy	Alberts	25	HR Assistant	650000.00	6
▼ Jimmy	Harrison	7	Purchasing	200000.00	-1
Pamela	Ansman-Wolfe	26	Purchase Manager	600000.00	7
Michael	Blythe	27	Store Keeper	550000.00	7
David	Campbell	28	Store Keeper	450000.00	7
▼ Ronald	Fillmore	8	Production	2800000.00	-1
Jillian	Carson	29	Production Manager	600000.00	8
Shu	Ito	30	Production Engineer	550000.00	8
Stephen	Jiang	31	Production Engineer	450000.00	8

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Printing in WPF TreeGrid (SfTreeGrid)

The printing feature can be achieved by exporting the tree grid to PDF and printing the exported PDF using the [PdfViewerControl](#).

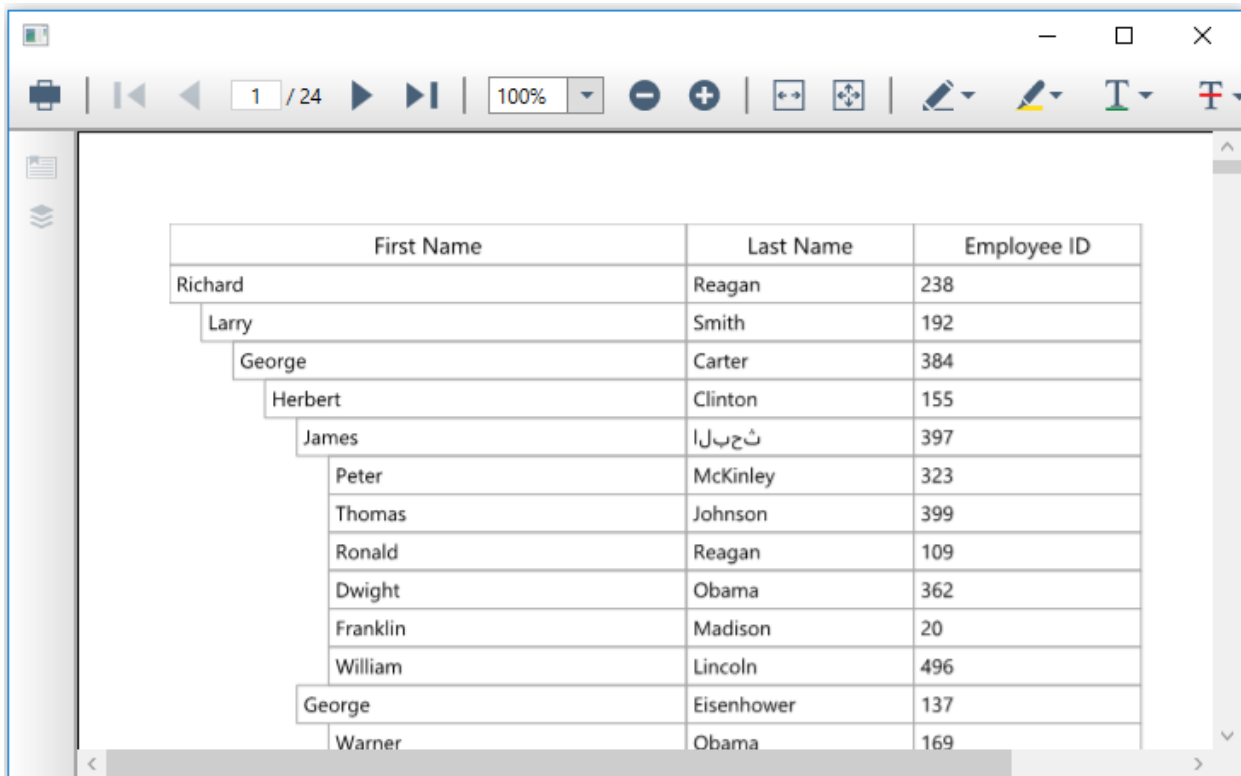
C#

```
var options = new TreeGridPdfExportingOptions();
FileStream fileStream = new FileStream("Sample.pdf", FileMode.Create);
var document = treeGrid.ExportToPdf(options);
MemoryStream stream = new MemoryStream();
document.Save(stream);
PdfViewerControl pdfViewer = new PdfViewerControl();
pdfViewer.Load(stream);
Window window = new Window();
window.Content = pdfViewer;
```

```

window.Loaded += Window_Loaded;
window.Show();
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    var toolbar = pdfViewer.Template.FindName("PART_Toolbar", pdfViewer) as
    DocumentToolbar;
    // Get an instance of the open and save file buttons using its template
    name.
    Button openButton = (Button)toolbar.Template.FindName("PART_ButtonOpen",
    toolbar);
    Button saveButton = (Button)toolbar.Template.FindName("PART_ButtonSave",
    toolbar);
    // Set visibility of the button to collapsed.
    openButton.Visibility = System.Windows.Visibility.Collapsed;
    saveButton.Visibility = Visibility.Collapsed;
}

```



You can download the [sample](#).

Print parent and expanded child nodes

You can print only the parent and expanded child nodes by overriding the [ExportNodesToPdf](#) method of the [TreeGridToPdfConverter](#) class.

C#

```

var options = new TreeGridPdfExportingOptions();
options.AllowIndentColumn = true;
options.FitAllColumnsInOnePage = true;
var document = treeGrid.ExportToPdf(options, true);
PdfViewerControl pdfViewer = new PdfViewerControl();

```

```

MemoryStream stream = new MemoryStream();
document.Save(stream);
PdfLoadedDocument ldoc = new PdfLoadedDocument(stream);
pdfViewer.Load(ldoc);
// If you want to show the pdf viewer window, enable the following line.
MainWindow pdfPage = new MainWindow();
pdfPage.Content = pdfViewer;
pdfPage.Show();
pdfViewer.Print(true);
public class TreeGridCustomPdfConverter : TreeGridToPdfConverter
{
    internal bool _excludeNonExpandedNodes;
    public TreeGridCustomPdfConverter(bool excludeNonExpandedNodes) :base()
    {
        _excludeNonExpandedNodes = excludeNonExpandedNodes;
    }
    /// <summary>
    /// ExportNodes to PDF
    /// </summary>
    /// <param name="treeGrid"></param>
    /// <param name="nodes"></param>
    /// <param name="pdfGrid"></param>
    /// <param name="pdfExportingOptions"></param>
    protected override void ExportNodesToPdf(SfTreeGrid treeGrid, TreeNodes
nodes, PdfGrid pdfGrid, TreeGridPdfExportingOptions pdfExportingOptions)
    {
        if (!_excludeNonExpandedNodes)
        {
            base.ExportNodesToPdf(treeGrid, nodes, pdfGrid, pdfExportingOptions);
        }
        else
        {
            for (int i = 0; i < nodes.Count; i++)
            {
                TreeNode node = nodes[i];
                ExportNodeToPdf(treeGrid, node, pdfGrid, pdfExportingOptions);
                if (node.IsExpanded && node.HasChildNodes)
                {
                    node.PopulateChildNodes();
                    ExportNodesToPdf(treeGrid, node.ChildNodes, pdfGrid, pdfExportingOptions);
                }
            }
        }
    }
}

```

You can download the [sample](#).

Print customization

The print page can be customized when exporting by passing the [TreeGridPdfExportingOptions](#) instance as an argument to the [ExportToPdf](#) and [ExportToPdfGrid](#) methods. Refer to this [documentation](#) to customize the export options.

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Export To Excel in WPF TreeGrid (SfTreeGrid)

SfTreeGrid provides support to export the data to Excel file. This also provides support to export the headers and stacked headers. This maintains the sorting and filtering processes when exporting.

The following assemblies should be added for exporting a tree grid to Excel:-

- Syncfusion.SfGridConverter.WPF
- Syncfusion.XlsIO.Base
- Syncfusion.XlsIO.Wpf

You can export the tree grid to Excel by using the [ExportToExcel](#) extension method present in [Syncfusion.UI.Xaml.TreeGrid.Converter](#) namespace.

C#

```
using Syncfusion.UI.Xaml.TreeGrid.Converter;
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

Note: SfTreeGrid exports the data to Excel using [XlsIO](#). You can refer to the [XlsIO documentation](#) for manipulating the exported work sheets.

Export options

The exporting operation can be customized by passing [TreeGridExcelExportingOptions](#) instance as an argument to the [ExportToExcel](#) method.

Change export mode

By default, the actual value will only be exported to Excel. To export the display text, set the [TreeGridExportMode](#) property as `Text`.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.TreeGridExportMode = TreeGridExportMode.Text;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

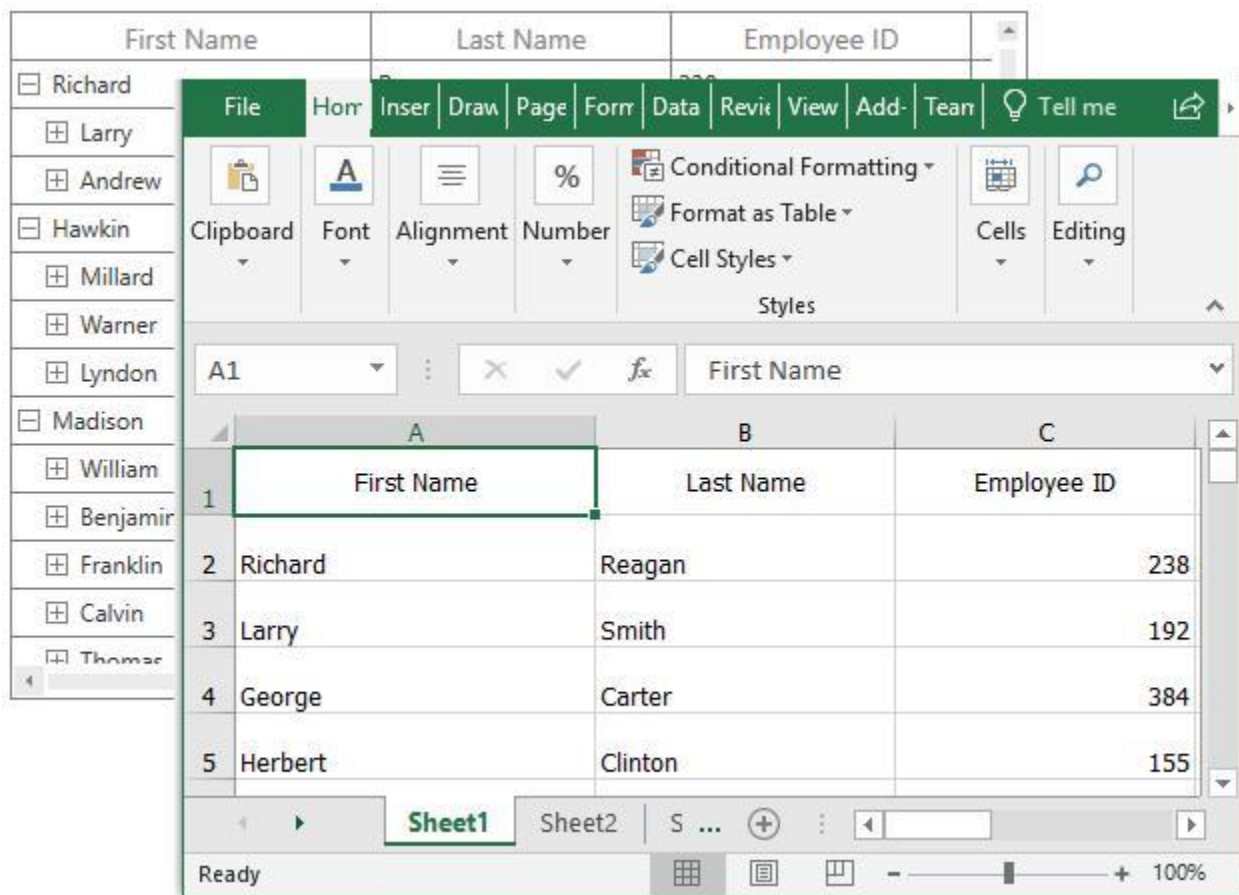
Export without outlines

By default, all the children in the tree grid will be exported in expanded state. You can disable the outlines in Excel by setting the [AllowOutliningGroups](#) property to false in [TreeGridExcelExportingOptions](#) class.

C#

```
var options = new TreeGridExcelExportingOptions();
```

```
options.ExcelVersion = ExcelVersion.Excel2013;
options.AllowOutliningGroups = false;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```



Exclude columns when exporting

By default, all the columns (including hidden columns) in the tree grid will be exported to Excel. To exclude some columns, use [ExcludeColumns](#) field in [TreeGridExcelExportingOptions](#) class.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.ExcludeColumns.Add("FirstName");
options.ExcludeColumns.Add("LastName");
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

Here, the columns having FirstName and LastName as MappingName are excluded when exporting.

Excel version

When exporting to Excel, you can specify the Excel version by using the [ExcelVersion](#) property.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

Export stacked headers to Excel

You can export the stacked headers to Excel by setting the [ExportStackedHeaders](#) property to **true**.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.ExportStackedHeaders = true;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

Export hyper-link to Excel

You can export the hyper-link to Excel by setting the [CanExportHyperLink](#) property to **true**.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.CanExportHyperLink = true;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

The default value of the [CanExportHyperLink](#) property is **true**.

Export column width to Excel

You can export the columns with its actual width by setting the [ExportColumnWidth](#) property to **true**.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.ExportColumnWidth = true;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

The default value of the [ExportColumnWidth](#) property is **true**.

Export with customized row height

You can export the tree grid to Excel with customized row height by using the [DefaultRowHeight](#) property.

C#

```
var options = new TreeGridExcelExportingOptions();
```

```
options.ExcelVersion = ExcelVersion.Excel2013;
options.DefaultRowHeight = 60;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample1.xlsx");
```

Export without grid lines

You can export the tree grid to Excel without grid lines by setting the [IsGridLinesVisible](#) property to false.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.IsGridLinesVisible = true;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

The default value of the [IsGridLinesVisible](#) property is true.

Export with indent column

You can export the tree grid to Excel with indent column to denote the nodes level by setting the [AllowIndentColumn](#) property to true.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.AllowIndentColumn = true;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample1.xlsx");
```

The default value of the [AllowIndentColumn](#) property is false.

Change the node expand state in Excel

You can change the node expanding state in Excel by using the [NodeExpandMode](#) property.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.NodeExpandMode = NodeExpandMode.CollapseAll;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

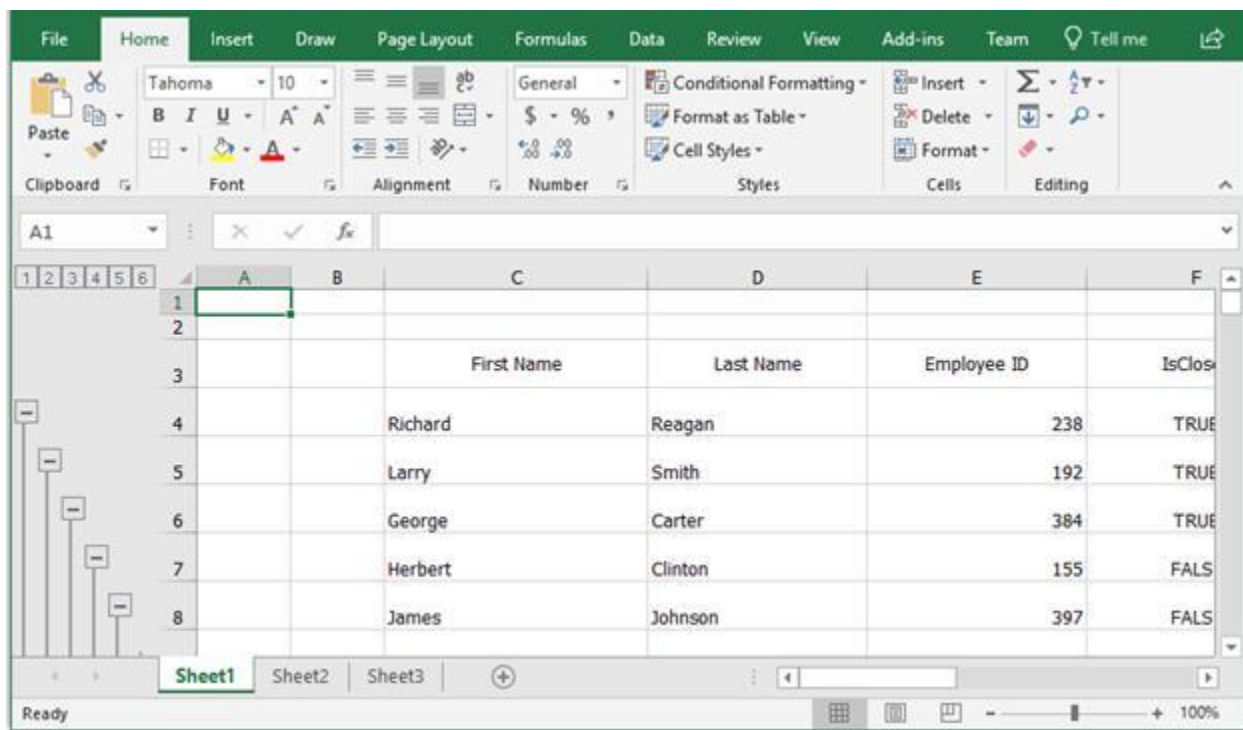
Change start row and column index while exporting

You can export the data to a specified row and column index in the worksheet, by setting the [ExcelStartRowIndex](#) and [ExcelStartColumnIndex](#) properties.

C#

```
var options = new TreeGridExcelExportingOptions();
```

```
options.ExcelVersion = ExcelVersion.Excel2013;
options.ExcelStartColumnIndex = 3;
options.ExcelStartRowIndex = 3;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```



Save options

Save Excel directory to file

After exporting to Excel, you can save the exported workbook directly to the file system using the [SaveAs](#) method.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

Save Excel to stream

After exporting to Excel, you can save the exported workbook to stream using the [SaveAs](#) method.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
FileStream fileStream = new FileStream("Output.xlsx", FileMode.Create);
```

```
workBook.SaveAs(fileStream);
```

You can refer to the [XlsIO documentation](#).

Save Excel using File dialog

After exporting to Excel, you can save the exported workbook by opening the [FileDialog](#).

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
SaveFileDialog sfd = new SaveFileDialog
{
    FilterIndex = 2,
    Filter = "Excel 97 to 2003 Files(*.xls)|*.xls|Excel 2007 to 2010 Files(*.xlsx)|*.xlsx|Excel 2013 File(*.xlsx)|*.xlsx"
};
if (sfd.ShowDialog() == true)
{
    using (Stream stream = sfd.OpenFile())
    {
        if (sfd.FilterIndex == 1)
            workBook.Version = ExcelVersion.Excel97to2003;
        else if (sfd.FilterIndex == 2)
            workBook.Version = ExcelVersion.Excel2010;
        else
            workBook.Version = ExcelVersion.Excel2013;
        workBook.SaveAs(stream);
    }
    //Message box confirmation to view the created workbook.
    if (MessageBox.Show("Do you want to view the workbook?", "Workbook has been created",
        MessageBoxButton.YesNo, MessageBoxImage.Information) ==
        MessageBoxResult.Yes)
    {
        //Launching the Excel file using the default Application.[MS Excel Or Free ExcelViewer]
        System.Diagnostics.Process.Start(sfd.FileName);
    }
}
}
```

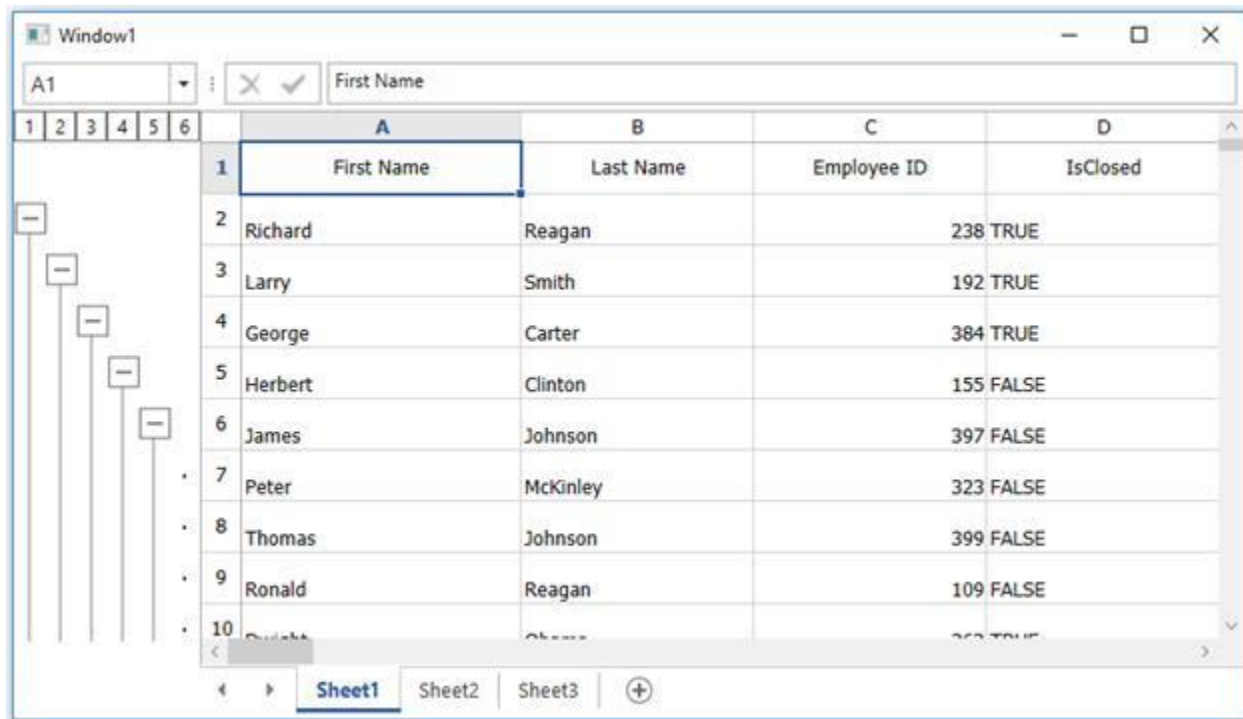
Open exported Excel without saving in disk

You can open the exported workbook without saving by using the [SfSpreadsheet](#) control.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
Window1 window1 = new Window1();
SfSpreadsheet spreadsheet = new SfSpreadsheet();
spreadsheet.Open(workBook);
```

```
window1.Content = spreadsheet;
window1.Show();
```



Export to HTML

You can save the exported workbook as HTML by using the [SaveAsHtml](#) method.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.AllowOutliningGroups = false;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAsHtml("Sample.html", HtmlSaveOptions.Default);
```

It is also possible to save the worksheet as HTML by using the [SaveAsHtml](#) method. You can refer to [XlsIO documentation](#).

Export to mail

You can export the tree grid to mail by converting to Excel and save the exported worksheet as HTML. Then the exported HTML contents should be embedded in the mail body.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.Worksheets[0].UsedRange.BorderInside(ExcelLineStyle.Thick,
ExcelKnownColors.Black);
```

```
workBook.Worksheets[0].UsedRange.BorderAround(ExcelLineStyle.Thick,
ExcelKnownColors.Black);
workBook.Worksheets[0].SaveAsHtml("test.htm",
Syncfusion.XlsIO.Implementation.HtmlSaveOptions.Default);
System.Net.Mail.MailMessage myMessage = new System.Net.Mail.MailMessage();
myMessage.To.Add("Support@syncfusion.com");
myMessage.From = new MailAddress("Support@syncfusion.com");
myMessage.Priority = MailPriority.High;
myMessage.Subject = "Order Details";
myMessage.IsBodyHtml = true;
myMessage.Body = new StreamReader("test.htm").ReadToEnd();
SmtpClient client = new SmtpClient("smtp.office365.com", 587);
client.EnableSsl = true;
client.UseDefaultCredentials = false;
client.Credentials = new NetworkCredential("Support@syncfusion.com",
"test");
int count = 0;
while (count < 3)
{
    try
    {
        client.Send(myMessage);
        count = 3;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(String.Format("Sending Mail Attemp - {0}",
count.ToString()));
        Thread.Sleep(60000);
        count++;
    }
}
Console.WriteLine("Mail has been sent...");
}
```

Export to XML

You can save the exported workbook as Xml file using the [SaveAsXml](#) method.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAsXml("Sample.xml", ExcelXmlSaveType.MSExcel);
```

Export to CSV

You can save the exported workbook as CSV using the [SaveAs](#) method.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
```



```
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.csv", ",", "");
```

Similarly, you can also save the exported worksheet to CSV. Refer to the [XlsIO documentation](#).

Customize row height and column width

After exporting the data to Excel, you can set different row heights and column widths for the columns. You can refer to [here](#) for more information.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.Worksheets[0].SetRowHeight(2, 50);
workBook.Worksheets[0].SetColumnWidth(2, 50);
workBook.SaveAs("Sample.xlsx");
```

Customize Cell appearance when exporting

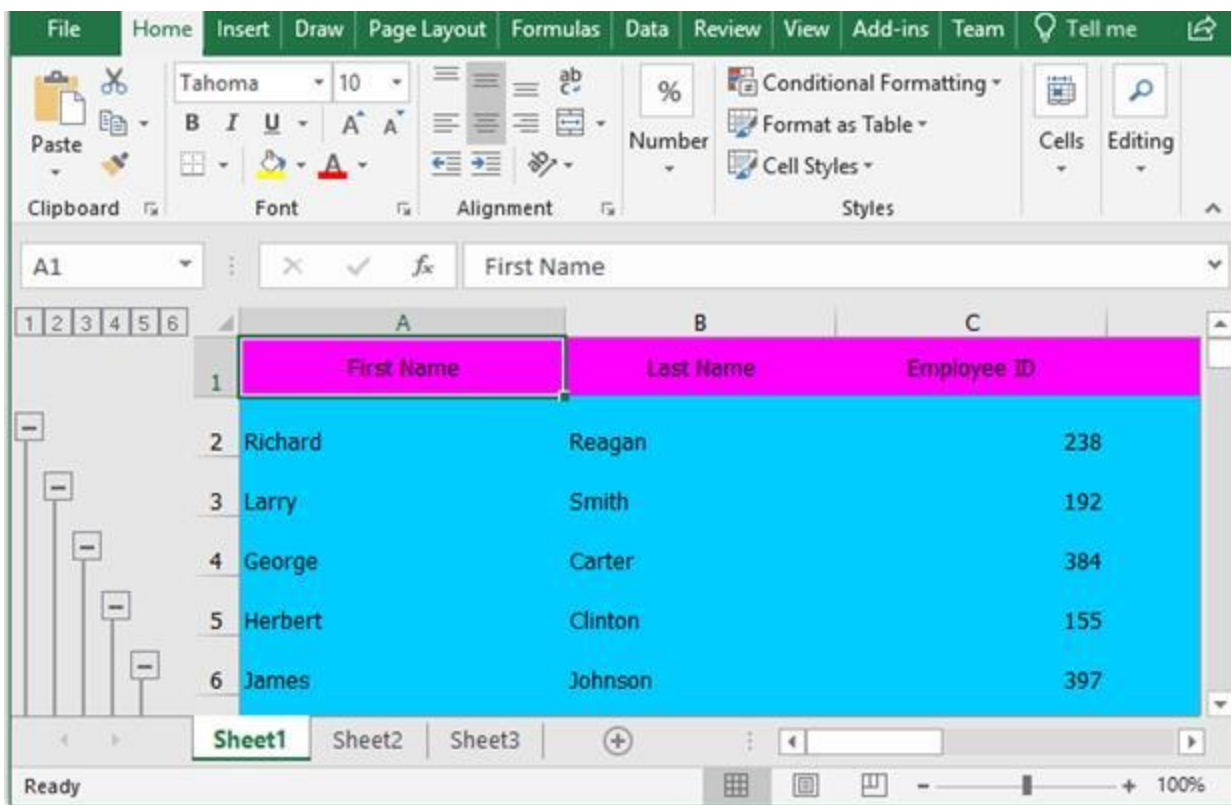
You can customize the cells by setting the [ExportingEventHandler](#) in [TreeGridExcelExportingOptions](#).

Cell styling cells based on cell type in Excel

You can customize the cell styles based on the cell type by using the [ExportingEventHandler](#).

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.ExportingEventHandler = ExportingHandler;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
private static void ExportingHandler(object sender,
TreeGridExcelExportingEventArgs e)
{
    if (e.CellType == TreeGridCellType.HeaderCell)
    {
        e.Style.ColorIndex = ExcelKnownColors.Pink;
        e.Handled = true;
    }
    else if (e.CellType == TreeGridCellType.RecordCell)
    {
        e.Style.ColorIndex = ExcelKnownColors.Sky_blue;
        e.Handled = true;
    }
}
```

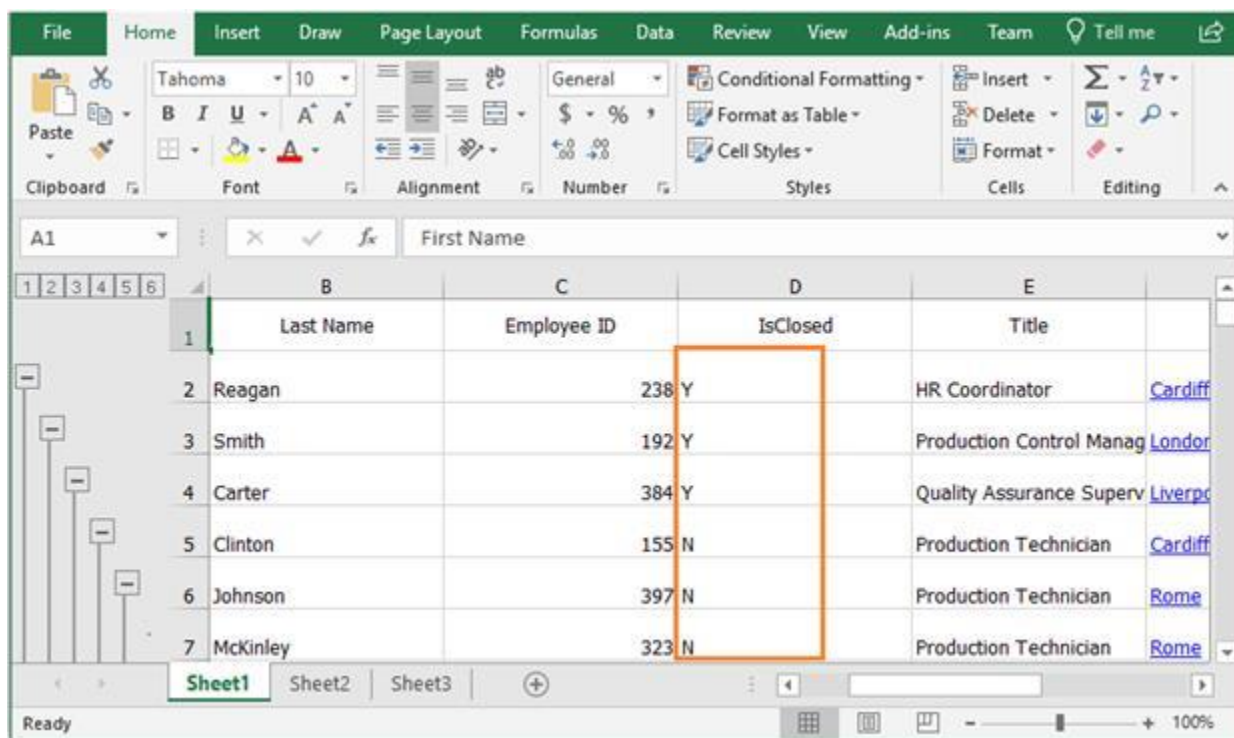


Customize the exporting content to Excel

You can customize the cell values when exporting to Excel by using the [CellsExportingEventHandler](#) in [TreeGridExcelExportingOptions](#).

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.CellsExportingEventHandler = CellExportingHandler;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
private static void CellExportingHandler(object sender,
TreeGridCellExcelExportingEventArgs e)
{
    // Based on the column mapping name and the cell type, we can change the
    cell
    //values while exporting to excel.
    if (e.CellType == TreeGridCellType.RecordCell && e.ColumnName == "IsClosed")
    {
        //if the cell value is True, "Y" will be displayed else "N" will be
        displayed.
        if (e.CellValue.Equals(true))
        e.Range.Cells[0].Value = "Y";
        else
        e.Range.Cells[0].Value = "N";
        e.Handled = true;
    }
}
```



Here, the cell values changed for the IsClosed column are based on the custom condition.

Change row style in Excel based on data

You can customize the rows based on the record values by using the [CellsExportingEventHandler](#).

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.CellsExportingEventHandler = CellExportingHandler;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
private static void CellExportingHandler(object sender,
TreeGridCellExcelExportingEventArgs e)
{
    if ((e.Node == null))
        return;
    var record = e.Node as EmployeeInfo;
    if (record.City == "NewYork")
    {
        e.Range.CellStyle.ColorIndex = ExcelKnownColors.Green;
        e.Range.CellStyle.Font.Color = ExcelKnownColors.White;
    }
}
```

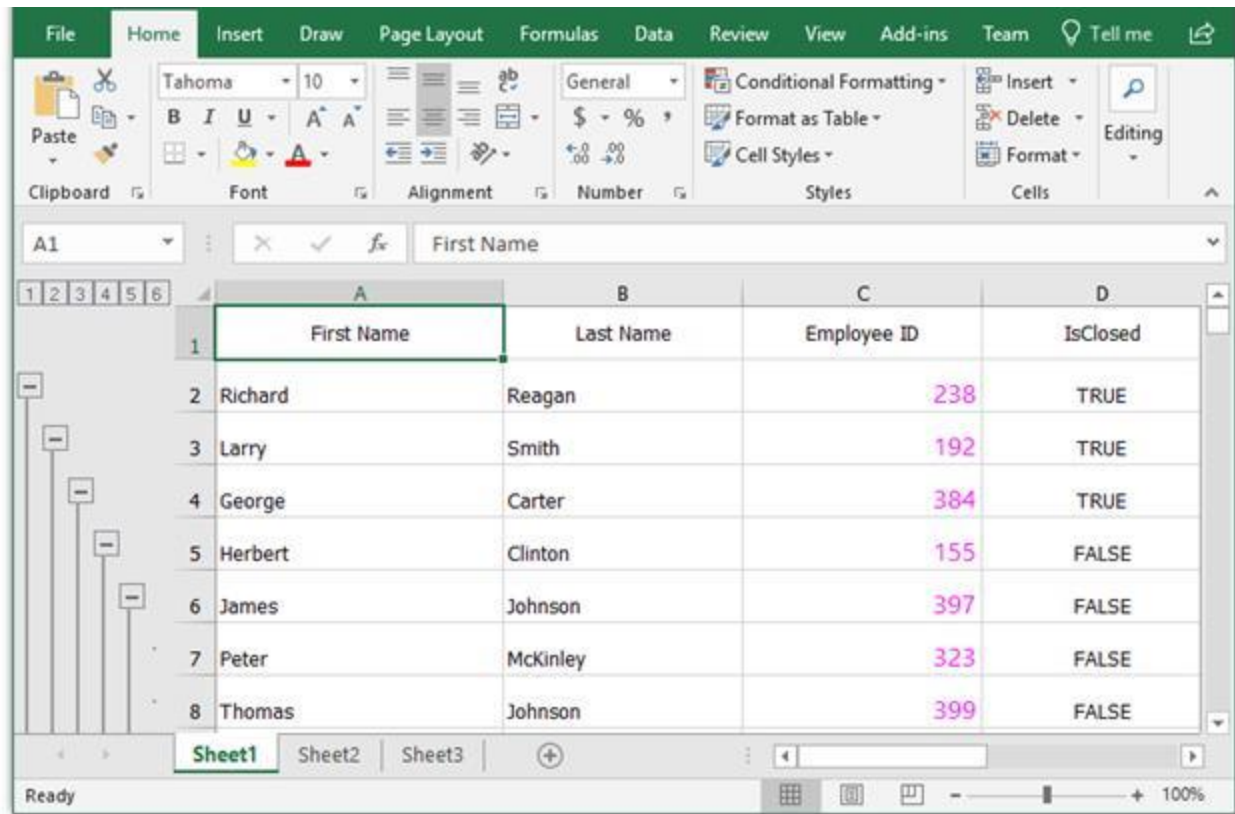
	A	B	C	D	E
12	William	Lincoln	496	TRUE	Design Engineer
13	George	Eisenhower	137	FALSE	Marketing Specialist
14	Warner	Obama	169	FALSE	Shipping and Receiving Cl
15	Chester	Pierce	264	TRUE	Maintenance Supervisor
16	James	Obama	257	FALSE	Marketing Assistant
17	Benjamin	Fillmore	146	TRUE	Production Control Manag
18	Zachary	Grant	2	TRUE	Tool Designer
19	Theodore	Truman	267	FALSE	Maintenance Supervisor

Customize the cells based on column name

You can customize the cells based on the [GridCellExcelExportingEventArgs.ColumnName](#) property in [CellsExportingEventHandler](#).

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.AllowOutliningGroups = false;
options.CellsExportingEventHandler = CellExportingHandler;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
private static void CellExportingHandler(object sender,
TreeGridCellExcelExportingEventArgs e)
{
    if (e.ColumnName != "ID")
        return;
    e.Range.CellStyle.Font.Size = 12;
    e.Range.CellStyle.Font.Color = ExcelKnownColors.Pink;
    e.Range.CellStyle.Font.FontName = "Segoe UI";
}
```



	A	B	C	D
1	First Name	Last Name	Employee ID	IsClosed
2	Richard	Reagan	238	TRUE
3	Larry	Smith	192	TRUE
4	George	Carter	384	TRUE
5	Herbert	Clinton	155	FALSE
6	James	Johnson	397	FALSE
7	Peter	McKinley	323	FALSE
8	Thomas	Johnson	399	FALSE

Customize exported workbook and worksheet

The tree grid can be exported to Excel using [XlsIO](#). You can refer to the [XlsIO documentation](#) for manipulating the workbooks and sheets after exporting.

Set borders

You can set the borders to Excel cells by directly accessing worksheet after exporting the data.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.Worksheets[0].UsedRange.BorderInside(ExcelLineStyle.Dash_dot,
ExcelKnownColors.Black);
workBook.Worksheets[0].UsedRange.BorderAround(ExcelLineStyle.Dash_dot,
ExcelKnownColors.Black);
workBook.SaveAs("Sample.xlsx");
```

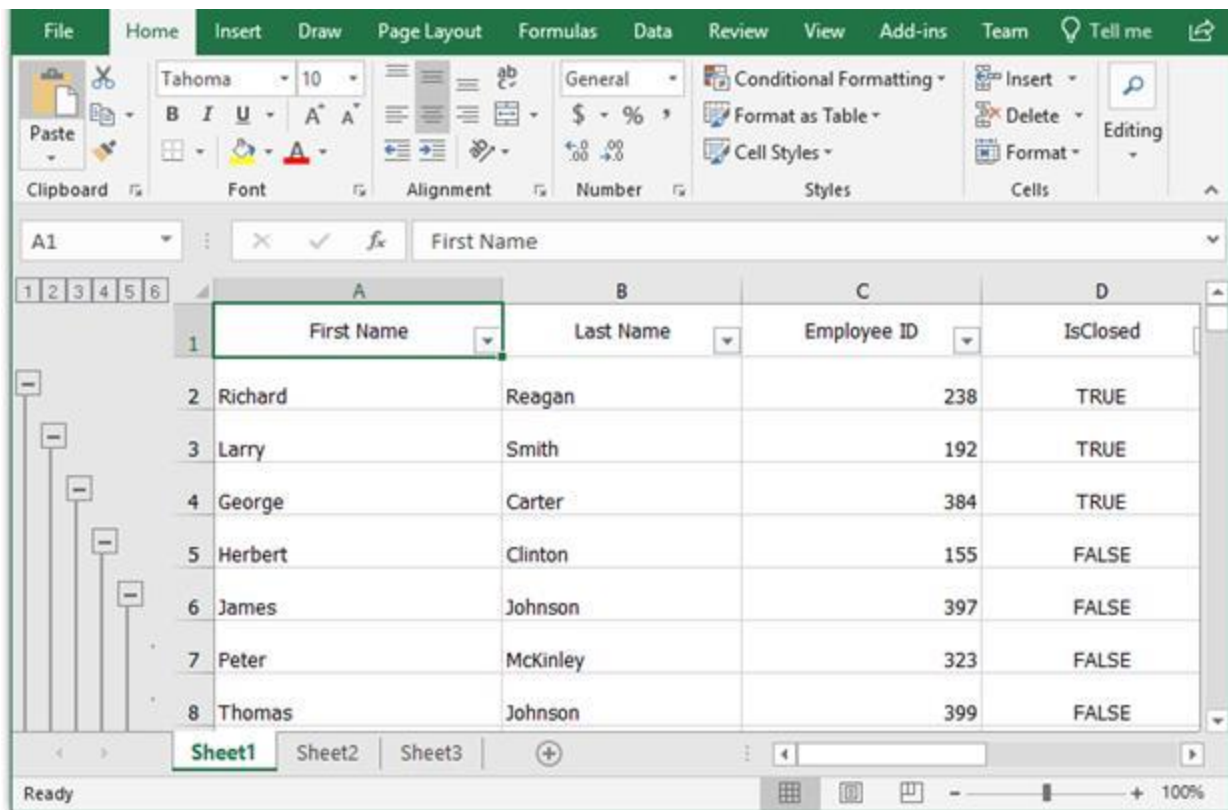
	A	B	C	D
1	First Name	Last Name	Employee ID	IsClosed
2	Richard	Reagan	238	TRUE
3	Larry	Smith	192	TRUE
4	George	Carter	384	TRUE
5	Herbert	Clinton	155	FALSE
6	James	Johnson	397	FALSE
7	Peter	McKinley	323	FALSE
8	Thomas	Johnson	399	FALSE

Enable filters

You can show filters in the exported worksheet by enabling filter for the exported range.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.Worksheets[0].AutoFilters.FilterRange =
workBook.Worksheets[0].UsedRange;
workBook.SaveAs("Sample.xlsx");
```

When using the stacked headers, you can specify the range based on the stacked headers count.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
var range = "A" + (treeGrid.StackedHeaderRows.Count + 1).ToString() + ":" +
workBook.Worksheets[0].UsedRange.End.AddressLocal;
excelEngine.Excel.Workbooks[0].Worksheets[0].AutoFilters.FilterRange =
workBook.Worksheets[0].Range[range];
workBook.SaveAs("Sample.xlsx");
```

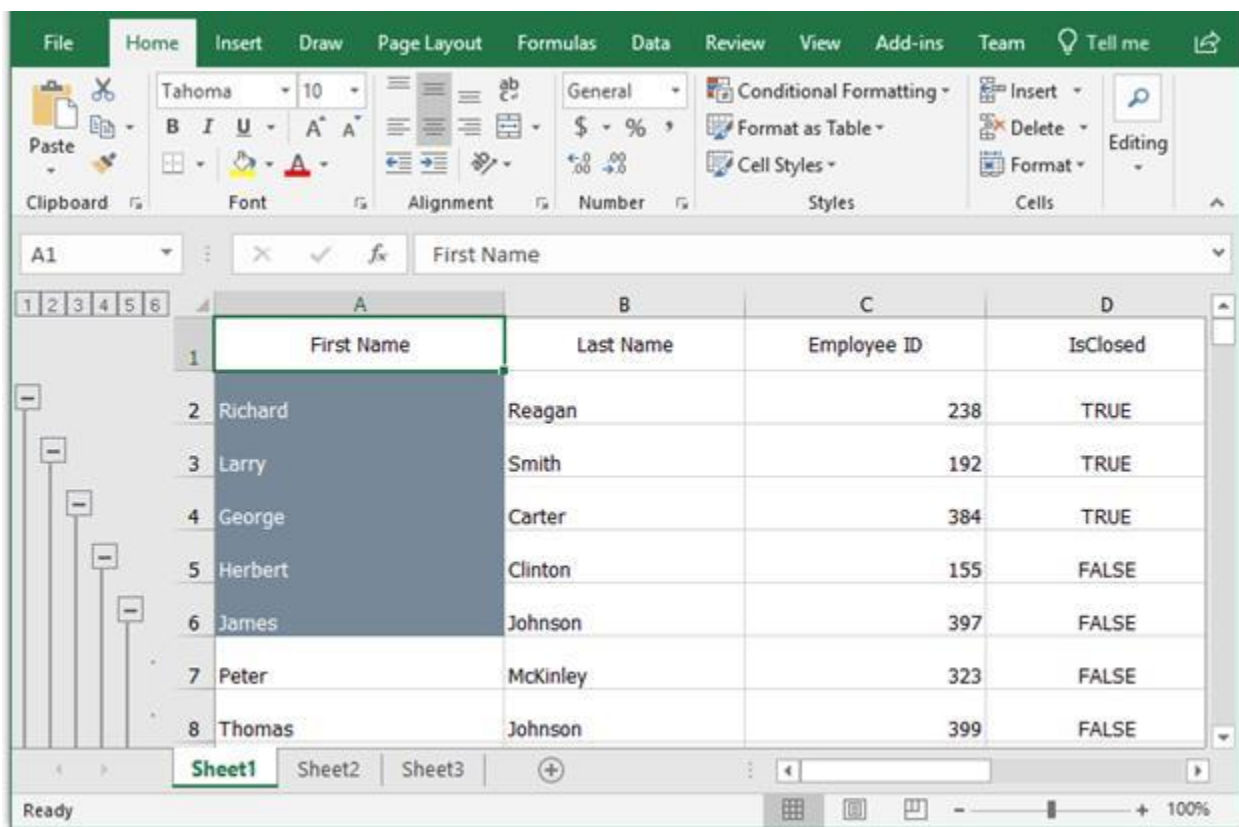
You can refer to the [XlsIO documentation](#).

Customize the range of cells

You can customize the range of cells after exporting to Excel by directly manipulating the worksheet.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.Worksheets[0].Range["A2:A6"].CellStyle.Color =
System.Drawing.Color.LightSlateGray;
workBook.Worksheets[0].Range["A2:A6"].CellStyle.Font.Color =
ExcelKnownColors.White;
workBook.SaveAs("Sample.xlsx");
```



Performance

Using the [ExcelExportingOptions.CellsExportingEventHandler](#) and changing the settings for each cell will consume more memory and time. So, avoid using the [CellsExportingEventHandler](#) and instead of this, you can do the required settings in the exported sheet.

Format column without using CellsExportingEventHandler

You can perform cell level customizations such as row-level styling, formatting a particular column in the exported worksheet.

In the following code snippet, NumberFormat for **Employee ID** column is changed in the exported sheet after exporting without using the [CellsExportingEventHandler](#).

Reference:

<http://help.syncfusion.com/file-formats/xlsio/working-with-cell-or-range-formatting>

C#

```
var options = new TreeGridExcelExportingOptions();
options.TreeGridExportMode = TreeGridExportMode.Value;
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
IWorkbook workbook = excelEngine.Excel.Workbooks[0];
workBook.ActiveSheet.Columns[2].NumberFormat = "0.0";
workBook.SaveAs("Sample.xlsx");
```


First Name	Last Name	Employee ID
Richard	Reagan	238
Larry	Smith	192
Andrew	Tyler	301
Hawkin	Stogner	244
Millard	Jones	400
Warner	Monroe	359
Lyndon	Adams	...
Madison	Smith	
William	Pierce	
Benjamin	Washington	
Franklin	Kennedy	
Calvin	McKinley	

First Name	Last Name	Employee ID
Richard	Reagan	238.0
Larry	Smith	192.0
George	Carter	384.0
Herbert	Clinton	155.0
James	Johnson	397.0

Alternate row styling without using CellsExportingEventHandler

In the following code snippet, the background color of the rows in Excel is changed based on the row index using conditional formatting for better performance.

Reference:

<http://help.syncfusion.com/file-formats/xlsio/working-with-conditional-formatting>

C#

```
var options = new TreeGridExcelExportingOptions();
options.TreeGridExportMode = TreeGridExportMode.Value;
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options);
IWorkbook workBook = excelEngine.Excel.Workbooks[0];
IConditionalFormats condition = workBook.ActiveSheet.Range[2, 1,
this.treeGrid.View.Nodes.Count + 1,
this.treeGrid.Columns.Count].ConditionalFormats;
IConditionalFormat condition1 = condition.AddCondition();
condition1.FormatType = ExcelCfType.Formula;
condition1.FirstFormula = "MOD(ROW(), 2)=0";
condition1.BackColorRGB = System.Drawing.Color.Pink;
IConditionalFormat condition2 = condition.AddCondition();
condition2.FormatType = ExcelCfType.Formula;
condition2.FirstFormula = "MOD(ROW(), 2)=1";
condition2.BackColorRGB = System.Drawing.Color.LightGray;
workBook.SaveAs("Sample.xlsx");
```

First Name	Last Name	Employee ID
Richard	Reagan	238
Larry	Smith	192
Andrew	Tyler	301

	A	B	C
	First Name	Last Name	Employee ID
Richard	Richard	Reagan	238
Larry	Larry	Smith	192
George	George	Carter	384
Herbert	Herbert	Clinton	155
James	James	Johnson	397
Peter	Peter	McKinley	323
Thomas	Thomas	Johnson	399
Ronald	Ronald	Reagan	109

How to

Export multiple tree grids to single Excel sheet

You can export multiple tree grids to single Excel sheet by merging one tree grid worksheet into another using the `Worksheet.UsedRange.CopyTo` method.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2010;
var excelEngine = treeGrid.ExportToExcel(options);
var workBook1 = excelEngine.Excel.Workbooks[0];
var worksheet1 = workBook1.Worksheets[0];
excelEngine = treeGrid1.ExportToExcel(options);
var workBook2 = excelEngine.Excel.Workbooks[0];
var worksheet2 = workBook2.Worksheets[0];
var columnCount = this.treeGrid1.Columns.Count;
//Merge the One TreeGrid Worksheet into the other TreeGrid Worksheet
worksheet2.UsedRange.CopyTo(worksheet1[1, columnCount + 1]);
workBook1.SaveAs("sample.xlsx");
```

Export the tree grid that is not loaded in view

You can export the tree grid that is not loaded in view by calling the `ApplyTemplate()` method before exporting.

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2010;
treeGrid1.ApplyTemplate();
var excelEngine = treeGrid1.ExportToExcel(options);
var workBook1 = excelEngine.Excel.Workbooks[0];
var worksheet1 = workBook1.Worksheets[0];
workBook1.SaveAs("sample.xlsx");
```

Export parent and expanded child nodes

By default, all the tree grid nodes will be exported when calling the `ExportToExcel` method. You can export only the parent and expanded child nodes by overriding the `ExportNodesToExcel` method of `TreeGridToExcelConverter` class,

C#

```
var options = new TreeGridExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = treeGrid.ExportToExcel(options, true);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
protected override void ExportNodesToExcel(SfTreeGrid treeGrid, TreeNodes
nodes, IWorksheet sheet, TreeGridExcelExportingOptions
excelExportingOptions)
{
    if (!_excludeNonExpandedNodes)
    {
        base.ExportNodesToExcel(treeGrid, nodes, sheet, excelExportingOptions);
    }
    else
    {
        for (int i = 0; i < nodes.Count; i++)
        {
            TreeNode node = nodes[i];
            ExportNodeToExcel(treeGrid, node, sheet, excelExportingOptions);
            if (node.IsExpanded && node.HasChildNodes)
            {
                node.PopulateChildNodes();
                ExportNodesToExcel(treeGrid, node.ChildNodes, sheet, excelExportingOptions);
            }
        }
    }
}
```

You can download the sample [here](#).

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Export To PDF in WPF TreeGrid (SfTreeGrid)

SfTreeGrid provides support to export the data to PDF file. This also provides support to export the headers and stacked headers. This maintains the sorting and filtering processes when exporting.

The following assemblies should be added for exporting a tree grid to PDF file:-

- Syncfusion.SfGridConverter.WPF
- Syncfusion.Pdf.Base

You can export the tree grid to PDF by using the following extension methods present in the [Syncfusion.UI.Xaml.TreeGrid.Converter](#) namespace:-

- [ExportToPdf](#)
- [ExportToPdfGrid](#)

C#

```
var options = new TreeGridPdfExportingOptions();  
var document = treeGrid.ExportToPdf(options);  
document.Save("Sample.pdf");
```

Note: SfTreeGrid exports the data to PDF file using [Essential PDF](#). You can refer to the [PDF documentation](#) for manipulation.

Export options

The exporting operation can be customized by passing [TreeGridPdfExportingOptions](#) instance as an argument to the [ExportToPdf](#) and [ExportToPdfGrid](#) methods.

Auto size column width in PDF

You can export a tree grid to PDF by fitting the column width based on its content by setting the [AutoColumnWidth](#) property to true.

C#

```
var options = new TreeGridPdfExportingOptions();  
options.AutoColumnWidth = true;  
var document = treeGrid.ExportToPdf(options);  
document.Save("Sample.pdf");
```

Auto size row height in PDF

You can export a tree grid to PDF by fitting the row height based on its content by setting the [AutoRowHeight](#) property to true.

C#

```
var options = new TreeGridPdfExportingOptions();  
options.AutoRowHeight = true;  
var document = treeGrid.ExportToPdf(options);  
document.Save("Sample.pdf");
```

Exclude columns when exporting

By default, all the columns (including hidden columns) in the tree grid will be exported to PDF. To exclude some columns when exporting to PDF, use the [ExcludeColumns](#) property in [TreeGridPdfExportingOptions](#) class.

C#

```
var options = new TreeGridPdfExportingOptions();  
options.ExcludeColumns.Add("CustomerName");  
options.ExcludeColumns.Add("Country");  
var document = treeGrid.ExportToPdf(options);  
document.Save("Sample.pdf");
```

Export format

By default, the display text will be exported to PDF. To export an actual value, set the [ExportFormat](#) property to `false`.

C#

```
var options = new TreeGridPdfExportingOptions();
options.ExportFormat = false;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

Repeat column headers on each page

The column headers can be exported on each page by setting the [RepeatHeaders](#) property.

C#

```
var options = new TreeGridPdfExportingOptions();
options.RepeatHeaders = true;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

Export all the columns in one page

When exporting to PDF, you can fit all the columns in one page by setting the [FitAllColumnsInOnePage](#) property to `true`.

C#

```
var options = new TreeGridPdfExportingOptions();
options.FitAllColumnsInOnePage = true;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

Export stacked headers to PDF

You can export the stacked headers to PDF by setting the [ExportStackedHeaders](#) property to `true`.

C#

```
var options = new TreeGridPdfExportingOptions();
options.ExportStackedHeaders = true;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

Define header and footer for PDF page

TreeGrid provides a way to display additional content at the top (header) or bottom (footer) of a page when exporting to PDF. This can be performed by setting the [PageHeaderFooterEventHandler](#) property in [TreeGridPdfExportingOptions](#) class.

You can insert any string, image or drawing in the header and footer using the `PdfHeaderFooterEventHandler` property.

Setting [PdfPageTemplateElement](#) class to the [PdfHeaderFooterEventArgs.PdfDocumentTemplate.Top](#) property loads the content at the top of a page. Setting [PdfPageTemplateElement](#) class to

the [PdfHeaderFooterEventArgs.PdfDocumentTemplate.Bottom](#) property loads the content at the bottom of a page.

C#

```
PdfFont font = new PdfStandardFont(PdfFontFamily.TimesRoman, 20f,
PdfFontStyle.Bold);
var width = e.PdfPage.GetClientSize().Width;
PdfPageTemplateElement header = new PdfPageTemplateElement(width, 38);
header.Graphics.DrawString("Order Details", font, PdfPens.Black, 70, 3);
e.PdfDocumentTemplate.Top = header;
```

Order Details

FirstName	LastName	Department	Title	Pass/Fail
BIGBOSS1	last174	department3	title1	True
first431_0	last431	department0	title0	False
first335_0	last335	department2	title1	False
first138_0	last138	department3	title2	True
first233_0	last233	department0	title4	False
first378_0	last378	department0	title4	True
first49_0	last49	department3	title4	False
first454_0	last454	department0	title3	True
first4_0	last4	department2	title1	True
first427_0	last427	department1	title4	False
first197_0	last197	department2	title0	False
first407_0	last407	department1	title2	False
first104_0	last104	department0	title0	True
first66_0	last66	department0	title0	True
first193_0	last193	department2	title3	False
first449_0	last449	department0	title0	False
first139_0	last139	department2	title3	False
first236_0	last236	department3	title0	True
first68_0	last68	department2	title1	True
first352_0	last352	department2	title1	True
first203_0	last203	department2	title3	False
first300_0	last300	department0	title4	True
first219_0	last219	department3	title4	False
first211_0	last211	department2	title1	False
first52_0	last52	department3	title0	True

Here, `string` is inserted in the header of the exported PDF file using the [DrawString](#) method. Similarly, you can insert any image, line, etc., using [DrawImage](#), [DrawLine](#), etc., methods respectively.

PDF page orientation

You can change the page orientation of PDF when exporting. The default page orientation is Portrait.

To change the page orientation, get the exported [PdfGrid](#) by using the [ExportToPdfGrid](#) method. Then, draw that PdfGrid into a [PdfDocument](#) by changing the [PageSettings.Orientation](#) property.

C#

```
var options = new TreeGridPdfExportingOptions();
var document = new PdfDocument();
document.PageSettings.Orientation = PdfPageOrientation.Landscape;
var page = document.Pages.Add();
var PDFGrid = treeGrid.ExportToPdfGrid(options);
var format = new PdfGridLayoutFormat()
{
    Layout = PdfLayoutType.Paginate,
    Break = PdfLayoutBreakType.FitPage
};
PDFGrid.Draw(page, new PointF(), format);
document.Save("Sample.pdf");
```

Save options

Save PDF directly to file

After exporting to PDF, you can save the exported PDF file directly to file system using the [Save](#) method.

C#

```
var options = new TreeGridPdfExportingOptions();
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

You can refer to the [PDF documentation](#).

Save PDF as stream

After exporting to PDF, you can save the exported PDF file to stream using the [Save](#) method.

C#

```
var options = new TreeGridPdfExportingOptions();
FileStream fileStream = new FileStream("Sample.pdf", FileMode.Create);
var document = treeGrid.ExportToPdf(options);
document.Save(fileStream);
fileStream.Close();
```

Save PDF using file dialog

After exporting to PDF, you can save the exported PDF file by opening [FileDialog](#).

C#

```
var options = new TreeGridPdfExportingOptions();
FileStream fileStream = new FileStream("Sample.pdf", FileMode.Create);
var document = treeGrid.ExportToPdf(options);
SaveFileDialog sfd = new SaveFileDialog
{
    Filter = "PDF Files (*.pdf) | *.pdf"
};
if (sfd.ShowDialog() == true)
{
    document.Save(fileStream);
    fileStream.Close();
}
```

```
using (Stream stream = sfd.OpenFile())
{
    document.Save(stream);
}
//Message box confirmation to view the created Pdf file.
if (MessageBox.Show("Do you want to view the Pdf file?", "Pdf file has been
created",
    MessageBoxButton.YesNo, MessageBoxImage.Information) ==
    MessageBoxResult.Yes)
{
    //Launching the Pdf file using the default Application.
    System.Diagnostics.Process.Start(sfd.FileName);
}
}
```

Open exported PDF without saving in disk

You can view the exported PDF document without saving in a disk by using the [PDFViewerControl](#).

C#

```
var options = new TreeGridPdfExportingOptions();
FileStream fileStream = new FileStream("Sample.pdf", FileMode.Create);
var document = treeGrid.ExportToPdf(options);
MemoryStream stream = new MemoryStream();
document.Save(stream);
PdfViewerControl pdfViewer = new PdfViewerControl();
pdfViewer.Load(stream);
Window window = new Window();
window.Content = pdfViewer;
window.Show();
```


First Name	Last Name	Employee ID
Richard	Reagan	238
Larry	Smith	192
George	Carter	384
Herbert	Clinton	155
James	ثحبلا	397
Peter	McKinley	323
Thomas	Johnson	399
Ronald	Reagan	109
Dwight	Obama	362
Franklin	Madison	20
William	Lincoln	496
George	Eisenhower	137
Warner	Obama	169
Chester	ثحبلا	264
James	Obama	257
Benjamin	Fillmore	146
Zachary	Grant	2
Theodore	Truman	267

Cell appearance customization when exporting

When exporting, you can customize the cells in a PDF document by setting the [CellsExportingEventHandler](#) property in [TreeGridPdfExportingOptions](#) class.

Cell styling based on cell type in PDF

You can customize the cell styles based on the [CellType](#) using the [CellsExportingEventHandler](#) property.

C#

```
var options = new TreeGridPdfExportingOptions();
options.CellsExportingEventHandler = GridPdfExportingEventHandler;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
void GridPdfExportingEventHandler(object sender,
TreeGridCellPdfExportingEventArgs e)
{
    if (e.CellType == TreeGridCellType.HeaderCell)
        e.PdfGridCell.Style.BackgroundBrush = PdfBrushes.LightSteelBlue;
    else if (e.CellType == TreeGridCellType.RecordCell)
        e.PdfGridCell.Style.BackgroundBrush = PdfBrushes.Wheat;
}
```

FirstName	LastName	Department	Title	Pass/Fail
BIGBOSS1	last1174	department3	title1	True
first431_0	last431	department0	title0	False
first335_0	last335	department2	title1	False
first138_0	last138	department3	title2	True
first233_0	last233	department0	title4	False
first378_0	last378	department0	title4	True
first49_0	last49	department3	title4	False
first454_0	last454	department0	title3	True
first4_0	last4	department2	title1	True
first427_0	last427	department1	title4	False
first197_0	last197	department2	title0	False
first407_0	last407	department1	title2	False
first104_0	last104	department0	title0	True
first66_0	last66	department0	title0	True
first193_0	last193	department2	title3	False
first449_0	last449	department0	title0	False
first139_0	last139	department2	title3	False
first236_0	last236	department3	title0	True
first68_0	last68	department2	title1	True
first352_0	last352	department2	title1	True
first203_0	last203	department2	title3	False
first300_0	last300	department0	title4	True
first219_0	last219	department3	title4	False
first211_0	last211	department2	title1	False
first52_0	last52	department3	title0	True
first250_0	last250	department0	title3	True
first297_0	last297	department0	title2	False
first196_0	last196	department2	title2	True

Cell styling based on the TreeGridPdfExportingOptions in PDF

You can style the stacked header, header and record cells in PDF by using the [StackedHeaderCellStyle](#), [HeaderCellStyle](#) and [RecordCellStyle](#) respectively in [TreeGridPdfExportingOptions](#) class.

C#

```
var options = new TreeGridPdfExportingOptions();
//Style for Stacked Headers
var stackedHeaderCellStyle = new PdfGridCellStyle();
stackedHeaderCellStyle.BackgroundBrush = PdfBrushes.LightPink;
stackedHeaderCellStyle.Borders.All = new PdfPen(PdfBrushes.DarkGray, 0.2f);
options.StackedHeaderCellStyle = stackedHeaderCellStyle;
//Style for Header
var headerCellStyle = new PdfGridCellStyle();
headerCellStyle.BackgroundBrush = PdfBrushes.LightPink;
headerCellStyle.Borders.All = new PdfPen(PdfBrushes.DarkGray, 0.2f);
options.HeaderCellStyle = headerCellStyle;
//Style for RecordCell
var recordCell = new PdfGridCellStyle();
recordCell.BackgroundBrush = PdfBrushes.LightBlue;
recordCell.Borders.All = new PdfPen(PdfBrushes.DarkGray, 0.2f);
options.RecordCellStyle = recordCell;
options.ExportStackedHeaders = true;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

Customize exporting content

You can customize the cell values when exporting to PDF using [CellsExportingEventHandler](#) and [TreeGridPdfExportingOptions](#).

C#

```
var options = new TreeGridPdfExportingOptions();
options.CellsExportingEventHandler = GridPdfExportingEventHandler;
options.FitAllColumnsInOnePage = true;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
void GridPdfExportingEventHandler(object sender,
TreeGridCellPdfExportingEventArgs e)
{
    // Based on the column mapping name and the cell type, you can change the
    cell values while exporting to PDF.
    if (e.CellType == TreeGridCellType.RecordCell && e.ColumnName == "IsClosed")
    {
        //if the cell value is True, "Y" will be displayed else "N" will be
        displayed.
        if (e.CellValue.Equals("True"))
            e.CellValue = "Y";
        else
            e.CellValue = "N";
    }
}
```

FirstName	LastName	Department	Title	Pass/Fail
BIGBOSS1	last174	department3	title1	Y
first431_0	last431	department0	title0	N
frst335_0	last335	department2	title1	N
first138_0	last138	department3	title2	Y
first233_0	last233	department0	title4	N
first378_0	last378	department0	title4	Y
first49_0	last49	department3	title4	N
first454_0	last454	department0	title3	Y
first4_0	last4	department2	title1	Y
first427_0	last427	department1	title4	N
first197_0	last197	department2	title0	N
first407_0	last407	department1	title2	N
first104_0	last104	department0	title0	Y
first66_0	last66	department0	title0	Y
first193_0	last193	department2	title3	N
first449_0	last449	department0	title0	N
first139_0	last139	department2	title3	N
first236_0	last236	department3	title0	Y
first68_0	last68	department2	title1	Y
first352_0	last352	department2	title1	Y
first203_0	last203	department2	title3	N
first300_0	last300	department0	title4	Y
frst219_0	last219	department3	title4	N
first211_0	last211	department2	title1	N
first52_0	last52	department3	title0	Y
first250_0	last250	department0	title3	Y
first297_0	last297	department0	title2	N
first196_0	last196	department2	title2	Y

Export images to PDF

By default, images loaded in the [GridTemplateColumn](#) will not be exported to PDF. To export it, use [CellsExportingEventHandler](#) property in [TreeGridPdfExportingOptions](#) class.

In [CellsExportingEventHandler](#), an image will be loaded in [PdfGridCell](#) class.





























C#

```
var options = new TreeGridPdfExportingOptions();
options.CellsExportingEventHandler = GridPdfExportingEventHandler;
options.FitAllColumnsInOnePage = true;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
void GridPdfExportingEventHandler(object sender,
TreeGridCellPdfExportingEventArgs e)
{
if (e.CellType == TreeGridCellType.RecordCell && e.ColumnName == "IsClosed")
```

```

{
var style = new PdfGridCellStyle();
PdfPen normalBorder = new PdfPen(PdfBrushes.DarkGray, 0.2f);
System.Drawing.Image image = null;
//Images are exported based on the CellValue
if (e.CellValue.Equals("True"))
{
//Access the image from the specified path
image = System.Drawing.Image.FromFile(@"..\..\Images\pass.png");
}
else
image = System.Drawing.Image.FromFile(@"..\..\Images\fail.png");
//Create the PDFImage for the specified image and assigned to
BackgroundImage of the PdfGridCellStyle
style.BackgroundImage = PdfImage.FromImage(image);
e.PdfGridCell.ImagePosition = PdfGridImagePosition.Fit;
e.PdfGridCell.Style = style;
//customize the Border color of PdfGridCell
e.PdfGridCell.Style.Borders.All = normalBorder;
e.CellValue = null;
}
}

```

FirstName	LastName	Department	Title	Pass/Fail
BIGBOSS1	last174	department3	title1	
first431_0	last431	department0	title0	
first335_0	last335	department2	title1	
first138_0	last138	department3	title2	
first233_0	last233	department0	title4	
first378_0	last378	department0	title4	
first49_0	last49	department3	title4	
first454_0	last454	department0	title3	
first4_0	last4	department2	title1	
first427_0	last427	department1	title4	
first197_0	last197	department2	title0	
first407_0	last407	department1	title2	
first104_0	last104	department0	title0	
first66_0	last66	department0	title0	
first193_0	last193	department2	title3	
first449_0	last449	department0	title0	
first139_0	last139	department2	title3	
first236_0	last236	department3	title0	
first68_0	last68	department2	title1	
first352_0	last352	department2	title1	
first203_0	last203	department2	title3	
first300_0	last300	department0	title4	
first219_0	last219	department3	title4	
first211_0	last211	department2	title1	
first52_0	last52	department3	title0	
first250_0	last250	department0	title3	
first297_0	last297	department0	title2	
first196_0	last196	department2	title2	

You can download the sample [here](#).

Embed fonts in PDF file

By default, some fonts (such as Unicode font) are not supported in PDF. In this case, it is possible to embed the font in PDF document with the help of [PdfTrueTypeFont](#) class.

C#

```
var options = new TreeGridPdfExportingOptions();
options.CellsExportingEventHandler = GridPdfExportingEventHandler;
options.FitAllColumnsInOnePage = true;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
void GridPdfExportingEventHandler(object sender,
TreeGridCellPdfExportingEventArgs e)
{
    if (e.CellType == TreeGridCellType.RecordCell)
    {
        PdfFont font = new PdfTrueTypeFont(new Font("Arial", 14), true);
        e.PdfGridCell.Style.Font = font;
    }
}
```

You can download the sample [here](#).

Export parent and expanded child nodes

By default, all the tree grid nodes will be exported when exporting. You can export the parent and expanded child nodes alone by overriding the [ExportNodesToPdf](#) method of [TreeGridToPdfConverter](#) class,

C#

```
var options = new TreeGridPdfExportingOptions();
FileStream fileStream = new FileStream("Sample1.pdf", FileMode.Create);
var document = treeGrid.ExportToPdf(options, true);
document.Save(fileStream);
fileStream.Close();
public class TreeGridCustomPdfConverter : TreeGridToPdfConverter
{
    internal bool _excludeNonExpandedNodes;
    public TreeGridCustomPdfConverter(bool excludeNonExpandedNodes) :base()
    {
        _excludeNonExpandedNodes = excludeNonExpandedNodes;
    }
    protected override void ExportNodesToPdf(SfTreeGrid treeGrid, TreeNodes
nodes, PdfGrid pdfGrid, TreeGridPdfExportingOptions pdfExportingOptions)
    {
        if (!_excludeNonExpandedNodes)
        {
            base.ExportNodesToPdf(treeGrid, nodes, pdfGrid, pdfExportingOptions);
        }
        else
        {
            for (int i = 0; i < nodes.Count; i++)
            {
                TreeNode node = nodes[i];
                ExportNodeToPdf(treeGrid, node, pdfGrid, pdfExportingOptions);
                if (node.IsExpanded && node.HasChildNodes)
            }
        }
    }
}
```

```
{
node.PopulateChildNodes();
ExportNodesToPdf(treeGrid, node.ChildNodes, pdfGrid, pdfExportingOptions);
}
}
}
```

You can download the sample [here](#).

Export Middle Eastern languages (Arabic and Hebrew) content to PDF

By default, [Middle Eastern languages](#) (Arabic and Hebrew) in the tree grid will be exported from left to right in PDF. You can export them as displayed in the tree grid (export from right to left) by enabling the [RightToLeft](#) property in [PdfStringFormat](#) class and apply the format to [PdfGridCell](#) class using the [CellsExportingEventHandler](#) property.

C#

```
var options = new TreeGridPdfExportingOptions();
options.CellsExportingEventHandler = GridPdfExportingEventHandler;
var document = treeGrid.ExportToPdf(options);
document.Save("Sample.pdf");
void GridPdfExportingEventHandler(object sender,
TreeGridCellPdfExportingEventArgs e)
{
if (e.CellType != TreeGridCellType.RecordCell)
return;
PdfStringFormat format = new PdfStringFormat();
//format the string from right to left.
format.RightToLeft = true;
e.PdfGridCell.StringFormat = format;
}
```

First Name	Last Name	Employee ID
<input type="checkbox"/> Richard	Reagan	238
<input type="checkbox"/> Larry	Smith	192
<input type="checkbox"/> George	Carter	384
<input type="checkbox"/> Herbert	Clinton	155
<input type="checkbox"/> James	البحث	307

Arabic text displayed to Right to Left

First Name	Last Name	Employee ID
Richard	Reagan	238
Larry	Smith	192
George	Carter	384
Herbert	Clinton	155
James	البحث	307
Peter	McKinley	323
Thomas	Johnson	399
Ronald	Reagan	109
Dwight	Obama	362
Franklin	Madison	20
William	Lincoln	496
George	Eisenhower	137

Arabic text Export to PDF

You can download the sample [here](#).

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

MVVM in WPF TreeGrid (SfTreeGrid)

This section explains various MVVM requirements using SfTreeGrid.

Bind the SelectedItem property of treegrid

You can bind the [SelectedItem](#) property directly to treegrid by setting the SfTreeGrid.SelectedItem property.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
Grid.Row="1"
ChildPropertyName="ReportsTo"
AutoExpandMode="AllNodesExpanded"
ShowRowHeader="True"
SelectedItem="{Binding SelectedItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
AutoGenerateColumns="False"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID"
SelfRelationRootValue="-1"/>
```

Whenever the SelectedItem is changed, the ViewModel property will get notified.

C#

```
public class ViewModel: NotificationObject
```



```
{
private object selectedItem;
public object SelectedItem
{
get
{
return selectedItem;
}
set
{
selectedItem = value;
RaisePropertyChanged("SelectedItem");
}
}
}
```

You can download the sample [here](#).

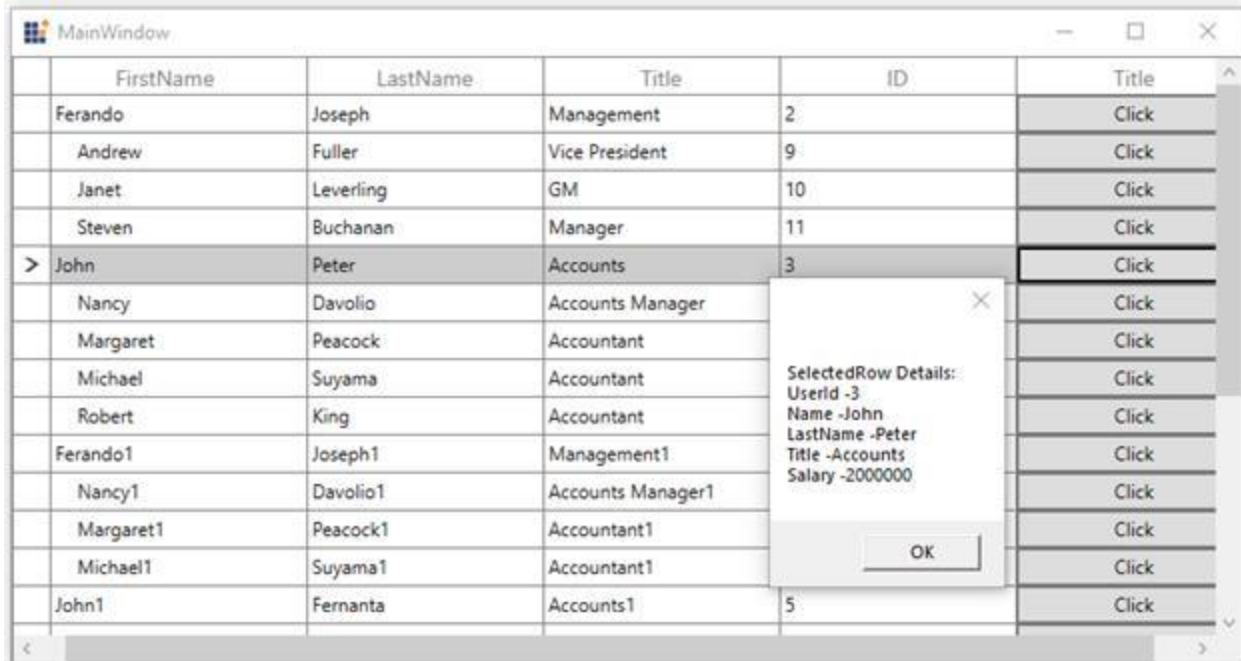
Bind button command to view model

You can load a button for the columns in treegrid using [TreeGridTemplateColumn](#). When loading the buttons, you can bind a command in ViewModel using [ElementName](#) binding.

In the following example, ViewModel command receives the underlying data object as command parameter, since the DataContext is bound as command parameter.

XML

```
<syncfusion:TreeGridTemplateColumn MappingName="Title"
syncfusion:FocusManagerHelper.WantsKeyInput="True">
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<Button Content="Click" syncfusion:FocusManagerHelper.FocusedElement="True"
Command="{Binding Path=DataContext.RowDataCommand, ElementName=treeGrid}"
CommandParameter="{Binding}"/>
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.CellTemplate>
</syncfusion:TreeGridTemplateColumn>
```



You can download the sample [here](#).

Bind combobox column ItemsSource from view model

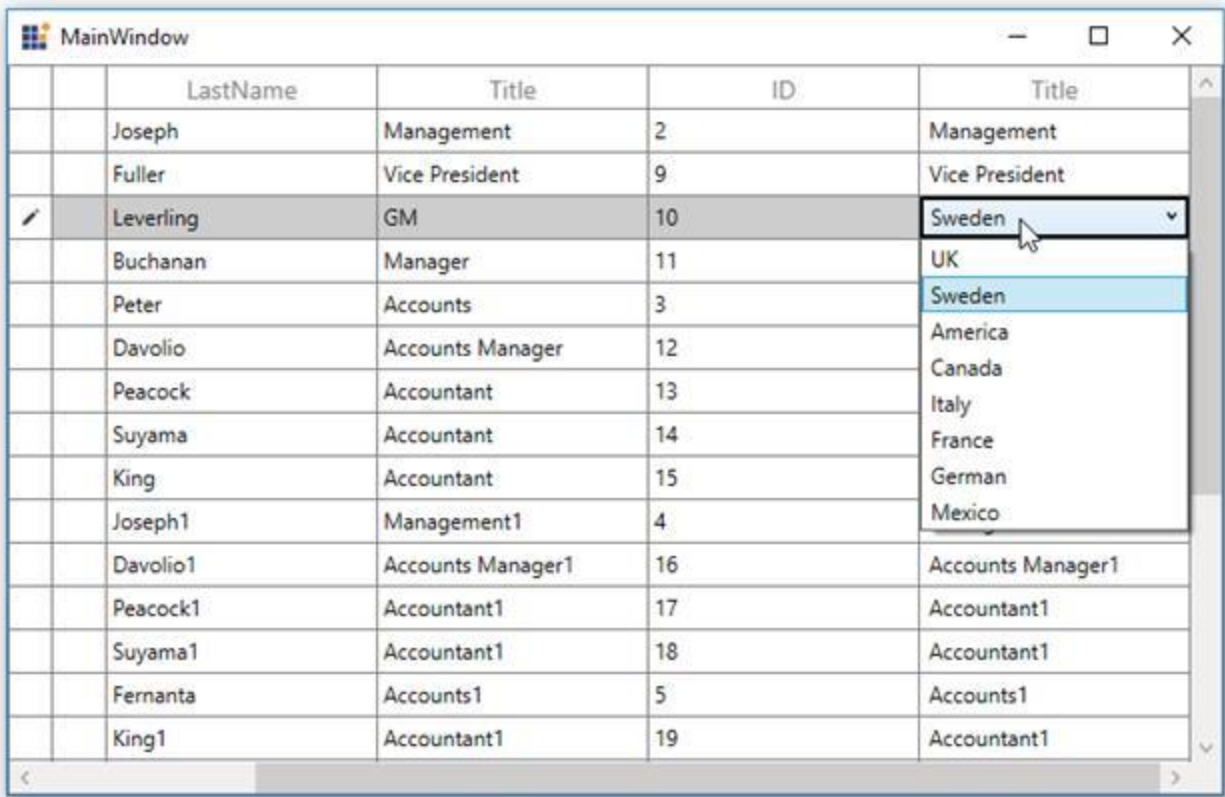
You can bind the ItemsSource from ViewModel to [TreeGridComboBoxColumn](#) or using `ElementName` binding.

XML

```
<syncfusion:TreeGridComboBoxColumn AllowEditing="True"
MappingName="Title"
HeaderText="Title"
ItemsSource="{Binding DataContext.TitleList,
ElementName=treeGrid}" />
```

C#

```
private ObservableCollection<string> titleList;
public ObservableCollection<string> TitleList
{
    get { return titleList; }
    set { titleList = value; }
}
```



You can download the sample [here](#).

Bind view model ItemsSource to ComboBox inside template

You can load a ComboBox inside [TreeGridTemplateColumn](#) and bind the ItemsSource from ViewModel to ComboBox using `ElementName` binding.

XML

```
<syncfusion:TreeGridTemplateColumn MappingName="Title"
syncfusion:FocusManagerHelper.WantsKeyInput="True">
<syncfusion:TreeGridTemplateColumn.CellTemplate>
<DataTemplate>
<TextBlock Text="{Binding Title}"/>
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.CellTemplate>
<syncfusion:TreeGridTemplateColumn.EditTemplate>
DataTemplate>
<ComboBox ItemsSource="{Binding Path=DataContext.TitleList,
ElementName=treeGrid}" />
</DataTemplate>
</syncfusion:TreeGridTemplateColumn.EditTemplate>
</syncfusion:TreeGridTemplateColumn>
```

MainWindow

	FirstName	LastName	Title	ID	Title
[-] Ferando	Joseph	Management	2	Management	
[-] Andrew	Fuller	Vice President	9	Vice President	
/ Janet	Leverling	GM	10	Sweden	
[-] Steven	Buchanan	Manager	11	Manager	
[-] John	Peter	Accounts	3	Accounts	
[-] Nancy	Davolio	Accounts Manager	12	Accounts Manager	
[-] Margaret	Peacock	Accountant	13	Accountant	
[-] Michael	Suyama	Accountant	14	Accountant	
[-] Robert	King	Accountant	15	Accountant	
[-] Ferando1	Joseph1	Management1	4	Management1	
[-] Nancy1	Davolio1	Accounts Manager1	16	Accounts Manager1	
[-] Margaret1	Peacock1	Accountant1	17	Accountant1	
[-] Michael1	Suyama1	Accountant1	18	Accountant1	
[-] John1	Fernanta	Accounts1	5	Accounts1	

You can download the sample [here](#).

Bind columns from view model

You can bind the [SfTreeGrid.Columns](#) property in ViewModel by having the binding property of Syncfusion.SfGrid.UI.Xaml.TreeGrid.Columns type. Thus, you can set binding to the SfTreeGrid.Columns property that provides DataContext of treegrid in ViewModel.

XML

```
<syncfusion:SfTreeGrid Name="treeGrid"
Grid.Row="1"
ChildPropertyName="ReportsTo"
AutoExpandMode="AllNodesExpanded"
ShowRowHeader="True"
Columns="{Binding SfGridColumns, Mode=TwoWay}"
AutoGenerateColumns="False"
ItemsSource="{Binding Employees}"
ParentPropertyName="ID"
SelfRelationRootValue="-1">
</syncfusion:SfTreeGrid>
```

Refer to the following code example in which the treegrid column is populated with some [TreeGridTextColumn](#) when creating the ViewModel instance.

C#

```
public class ViewModel: NotificationObject
{
    #region Private Variables
    private ObservableCollection<EmployeeInfo> _employees;
    #endregion
    #region ctor
    public ViewModel()
    {
        this.Employees = GetEmployeesDetails();
    }
}
```

```

rowDataCommand = new RelayCommand(ChangeCanExecute);
this.sfGridColumn = new TreeGridColumn();
sfGridColumn.Add(new TreeGridTextColumn() { MappingName = "FirstName" });
sfGridColumn.Add(new TreeGridTextColumn() { MappingName = "LastName" });
sfGridColumn.Add(new TreeGridTextColumn() { MappingName = "Title" });
sfGridColumn.Add(new TreeGridTextColumn() { MappingName = "Salary" });
}
}
#endregion
}

```

You can download the sample [here](#).

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Load On Demand in WPF TreeGrid (SfTreeGrid)

SfTreeGrid supports to load the data in on-demand. It helps to load the child items from services when end-user expands the node. This can be achieved by using the [SfTreeGrid.LoadOnDemandCommand](#) command or [SfTreeGrid.RequestTreeItems](#) event as follows.

Load on demand using command

SfTreeGrid allows you to load child items only when they are requested to load on-demand. Initially populate the root Nodes by assigning `SfTreeGrid.ItemsSource` and then when any node is expanded, child items can be loaded using [SfTreeGrid.LoadOnDemandCommand](#).

XML

```

<syncfusion:ChromelessWindow.DataContext>
<local:ViewModel />
</syncfusion:ChromelessWindow.DataContext>
<syncfusion:SfTreeGrid Name="treeGrid"
AutoGenerateColumns="False"
LoadOnDemandCommand="{Binding CommandLoad}"
ItemsSource="{Binding EmployeeDetails}" />

```

C#

```

public class ViewModel : NotificationObject
{
    public ICommand CommandLoad
    { get; set; }
    /// <summary>
    /// Initializes a new instance of the <see cref="ViewModel"/> class.
    /// </summary>
    public ViewModel()
    {
        CommandLoad = new LoadCommand();
        EmployeeDetails = new List<EmployeeInfo>();
        EmployeeDetails.Add(new EmployeeInfo() { FirstName = "Hwakin", ID = 65,
        LastName = "Grant", Title = "Business Manager", Salary = 200000, ReportsTo =
        -1 });
    }
}

```

```

EmployeeDetails.Add(new EmployeeInfo() { FirstName = "Madison", ID = 34,
LastName = "Bush", Title = "Vice President", Salary = 200000, ReportsTo = -1
});
EmployeeDetails.Add(new EmployeeInfo() { FirstName = "Richard", ID = 36,
LastName = "Monroe", Title = "HR Coordinator", Salary = 200000, ReportsTo =
-1 });
EmployeeDetails.Add(new EmployeeInfo() { FirstName = "Jack", ID = 43,
LastName = "Madison", Title = "Supervisor", Salary = 25000, ReportsTo = 55
});
}
/// <summary>
/// Gets or sets the employee details.
/// </summary>
/// <value>The employee details.</value>
public List<EmployeeInfo> EmployeeDetails
{
    get;
    set;
}
/// <summary>
/// Gets the reportees.
/// </summary>
/// <param name="bossID">The boss ID.</param>
/// <returns></returns>
public IEnumerable<EmployeeInfo> GetReportees(int bossID)
{
    List<EmployeeInfo> list = new List<EmployeeInfo>();
    if (bossID == 65)
    {
        list.Add(new EmployeeInfo() { FirstName = "John", ID = 5, LastName = "Polk",
Title = "Marketing Specialist", Salary = 100000, ReportsTo = 65 });
        list.Add(new EmployeeInfo() { FirstName = "Bill", ID = 26, LastName =
"Wilson", Title = "Senior Tool Designer", Salary = 100000, ReportsTo = 65
});
        list.Add(new EmployeeInfo() { FirstName = "Jimmy", ID = 23, LastName =
"Harry", Title = "Stocker", Salary = 50000, ReportsTo = 65 });
        list.Add(new EmployeeInfo() { FirstName = "Harry", ID = 68, LastName =
"Coolidge", Title = "Maintainence Supervisor", Salary = 50000, ReportsTo =
65 });
    }
    if (bossID == 34)
    {
        list.Add(new EmployeeInfo() { FirstName = "Franklin", ID = 24, LastName =
"Madison", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo
= 34 });
        list.Add(new EmployeeInfo() { FirstName = "William", ID = 66, LastName =
"Nixon", Title = "Production Technician", Salary = 40000, ReportsTo = 34 });
        list.Add(new EmployeeInfo() { FirstName = "Grover", ID = 35, LastName =
"Taft", Title = "Stocker", Salary = 50000, ReportsTo = 34 });
        list.Add(new EmployeeInfo() { FirstName = "Bill", ID = 18, LastName =
"Nixon", Title = "Maintainence Supervisor", Salary = 50000, ReportsTo = 34
});
    }
    if (bossID == 36)
    {

```

```

list.Add(new EmployeeInfo() { FirstName = "Madison", ID = 64, LastName =
"Franklin", Title = "Quality Assurance Supervisor", Salary = 40000,
ReportsTo = 36 });
list.Add(new EmployeeInfo() { FirstName = "Wilson", ID = 67, LastName =
"Harry", Title = "Production Technician", Salary = 40000, ReportsTo = 36 });
list.Add(new EmployeeInfo() { FirstName = "Harry", ID = 59, LastName =
"Taft", Title = "Stocker", Salary = 50000, ReportsTo = 36 });
list.Add(new EmployeeInfo() { FirstName = "Jimmy", ID = 08, LastName =
"Grover", Title = "Maintainence Supervisor", Salary = 50000, ReportsTo = 36
});
}
if (bossID == 5)
{
list.Add(new EmployeeInfo() { FirstName = "Franklin", ID = 47, LastName =
"William", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo
= 5 });
list.Add(new EmployeeInfo() { FirstName = "Taft", ID = 52, LastName =
"Peter", Title = "Production Technician", Salary = 40000, ReportsTo = 5 });
list.Add(new EmployeeInfo() { FirstName = "Harry", ID = 34, LastName =
"Wilson", Title = "Stocker", Salary = 50000, ReportsTo = 5 });
list.Add(new EmployeeInfo() { FirstName = "Grover", ID = 41, LastName =
"John", Title = "Maintainence Supervisor", Salary = 50000, ReportsTo = 5 });
}
if (bossID == 26)
{
list.Add(new EmployeeInfo() { FirstName = "Andrew", ID = 11, LastName =
"Wilson", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo
= 26 });
list.Add(new EmployeeInfo() { FirstName = "George", ID = 24, LastName =
"Madison", Title = "Production Technician", Salary = 40000, ReportsTo = 26
});
list.Add(new EmployeeInfo() { FirstName = "Peter", ID = 33, LastName =
"Taft", Title = "Stocker", Salary = 50000, ReportsTo = 26 });
list.Add(new EmployeeInfo() { FirstName = "Jimmy", ID = 54, LastName =
"Harry", Title = "Maintainence Supervisor", Salary = 50000, ReportsTo = 26
});
}
if (bossID == 23)
{
list.Add(new EmployeeInfo() { FirstName = "Bush", ID = 98, LastName =
"Bill", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo =
23 });
list.Add(new EmployeeInfo() { FirstName = "Hayes", ID = 32, LastName =
"William", Title = "Production Technician", Salary = 40000, ReportsTo = 23
});
list.Add(new EmployeeInfo() { FirstName = "Madison", ID = 12, LastName =
"Franklin", Title = "Stocker", Salary = 50000, ReportsTo = 23 });
list.Add(new EmployeeInfo() { FirstName = "Arthur", ID = 96, LastName =
"Grover", Title = "Maintainence Supervisor", Salary = 50000, ReportsTo = 23
});
}
if (bossID == 18)
{
list.Add(new EmployeeInfo() { FirstName = "Adams", ID = 71, LastName =
"Reagan", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo
= 18 });
}

```

```
list.Add(new EmployeeInfo() { FirstName = "Nixon", ID = 92, LastName = "Tyler", Title = "Production Technician", Salary = 40000, ReportsTo = 18 });
list.Add(new EmployeeInfo() { FirstName = "Coolidge", ID = 43, LastName = "Polk", Title = "Design Engineer", Salary = 50000, ReportsTo = 18 });
list.Add(new EmployeeInfo() { FirstName = "George", ID = 84, LastName = "Harrison", Title = "Maintainence Supervisor", Salary = 50000, ReportsTo = 18 });
}
if (bossID == 68)
{
list.Add(new EmployeeInfo() { FirstName = "Madison", ID = 125, LastName = "Bill", Title = "Marketing Specialist", Salary = 100000, ReportsTo = 68 });
list.Add(new EmployeeInfo() { FirstName = "Nixon", ID = 226, LastName = "Wilson", Title = "Senior Tool Designer", Salary = 100000, ReportsTo = 68 });
};
list.Add(new EmployeeInfo() { FirstName = "Taft", ID = 253, LastName = "Franklin", Title = "Stocker", Salary = 50000, ReportsTo = 68 });
list.Add(new EmployeeInfo() { FirstName = "Harry", ID = 618, LastName = "Coolidge", Title = "Maintainence Supervisor", Salary = 50000, ReportsTo = 68 });
}
if (bossID == 24)
{
list.Add(new EmployeeInfo() { FirstName = "Franklin", ID = 244, LastName = "Madison", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo = 24 });
list.Add(new EmployeeInfo() { FirstName = "William", ID = 166, LastName = "Nixon", Title = "Production Technician", Salary = 40000, ReportsTo = 24 });
list.Add(new EmployeeInfo() { FirstName = "Grover", ID = 355, LastName = "Taft", Title = "Stocker", Salary = 50000, ReportsTo = 24 });
list.Add(new EmployeeInfo() { FirstName = "Bill", ID = 148, LastName = "Harry", Title = "Production Supervisor", Salary = 50000, ReportsTo = 24 });
}
if (bossID == 66)
{
list.Add(new EmployeeInfo() { FirstName = "Stogner", ID = 64, LastName = "Martin", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo = 66 });
list.Add(new EmployeeInfo() { FirstName = "Wilson", ID = 67, LastName = "Harry", Title = "Production Technician", Salary = 40000, ReportsTo = 66 });
list.Add(new EmployeeInfo() { FirstName = "Harrison", ID = 59, LastName = "Will", Title = "Stocker", Salary = 50000, ReportsTo = 66 });
list.Add(new EmployeeInfo() { FirstName = "Jimmy", ID = 08, LastName = "Grover", Title = "Maintainence Supervisor", Salary = 50000, ReportsTo = 66 });
};
}
if (bossID == 35)
{
list.Add(new EmployeeInfo() { FirstName = "Franklin", ID = 417, LastName = "William", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo = 35 });
list.Add(new EmployeeInfo() { FirstName = "Taft", ID = 522, LastName = "Peter", Title = "Production Technician", Salary = 40000, ReportsTo = 35 });
list.Add(new EmployeeInfo() { FirstName = "Harry", ID = 341, LastName = "Wilson", Title = "Stocker", Salary = 50000, ReportsTo = 35 });
};
}
```



```
list.Add(new EmployeeInfo() { FirstName = "Grover", ID = 141, LastName = "John", Title = "Maintenance Supervisor", Salary = 50000, ReportsTo = 35 });
}
if (bossID == 64)
{
list.Add(new EmployeeInfo() { FirstName = "Andrew", ID = 141, LastName = "Wilson", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo = 64 });
list.Add(new EmployeeInfo() { FirstName = "Will", ID = 244, LastName = "Madison", Title = "Production Technician", Salary = 40000, ReportsTo = 64 });
list.Add(new EmployeeInfo() { FirstName = "Harrison", ID = 343, LastName = "Taft", Title = "Stocker", Salary = 50000, ReportsTo = 64 });
list.Add(new EmployeeInfo() { FirstName = "Jimmy", ID = 544, LastName = "Harry", Title = "Maintenance Supervisor", Salary = 50000, ReportsTo = 64 });
}
if (bossID == 67)
{
list.Add(new EmployeeInfo() { FirstName = "Bush", ID = 98, LastName = "Bill", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo = 67 });
list.Add(new EmployeeInfo() { FirstName = "Hayes", ID = 32, LastName = "William", Title = "Production Technician", Salary = 40000, ReportsTo = 67 });
list.Add(new EmployeeInfo() { FirstName = "Madison", ID = 12, LastName = "Franklin", Title = "Stocker", Salary = 50000, ReportsTo = 67 });
list.Add(new EmployeeInfo() { FirstName = "Arthur", ID = 96, LastName = "Grover", Title = "Maintenance Supervisor", Salary = 50000, ReportsTo = 67 });
}
if (bossID == 59)
{
list.Add(new EmployeeInfo() { FirstName = "Adams", ID = 711, LastName = "Reagan", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo = 59 });
list.Add(new EmployeeInfo() { FirstName = "Nixon", ID = 192, LastName = "Tyler", Title = "Production Technician", Salary = 40000, ReportsTo = 59 });
list.Add(new EmployeeInfo() { FirstName = "Coolidge", ID = 413, LastName = "Polk", Title = "Design Engineer", Salary = 50000, ReportsTo = 59 });
list.Add(new EmployeeInfo() { FirstName = "George", ID = 841, LastName = "Harrison", Title = "Maintenance Supervisor", Salary = 50000, ReportsTo = 59 });
}
if (bossID == 8)
{
list.Add(new EmployeeInfo() { FirstName = "William", ID = 71, LastName = "Bush", Title = "Quality Assurance Supervisor", Salary = 40000, ReportsTo = 8 });
list.Add(new EmployeeInfo() { FirstName = "Harry", ID = 92, LastName = "Tyler", Title = "Production Technician", Salary = 40000, ReportsTo = 8 });
list.Add(new EmployeeInfo() { FirstName = "Arthur", ID = 43, LastName = "Polk", Title = "Design Engineer", Salary = 50000, ReportsTo = 8 });
list.Add(new EmployeeInfo() { FirstName = "George", ID = 84, LastName = "Harrison", Title = "Maintenance Supervisor", Salary = 50000, ReportsTo = 8 });
};
```

```
}  
return list;  
}  
}
```

C#

```
class LoadCommand : ICommand  
{  
    public event EventHandler CanExecuteChanged;  
    public bool CanExecute(object parameter)  
    {  
        TreeNode node = (parameter as TreeNode);  
        EmployeeInfo emp = node.Item as EmployeeInfo;  
        if (emp != null)  
            if (emp.ReportsTo == -1 || emp.ReportsTo == 34 || emp.ReportsTo == 36 ||  
                emp.ReportsTo == 65)  
                return true;  
            return false;  
        }  
    public void Execute(object parameter)  
    {  
        TreeNode node = (parameter as TreeNode);  
        EmployeeInfo emp = node.Item as EmployeeInfo;  
        if (emp != null)  
        {  
            node.PopulateChildNodes((App.Current.MainWindow.DataContext as  
                ViewModel).GetReportees(emp.ID));  
        }  
    }  
}
```

Handling expander visibility

SfTreeGrid shows the expander for a particular node based on return value of [CanExecute](#) method of [LoadOnDemandCommand](#). If [CanExecute](#) returns [true](#), then expander icon is displayed for that node. If [CanExecute](#) returns [false](#), then expander icon will not displayed for that node. [CanExecute](#) method gets called to decide the visibility of expander icon and before executing [LoadOnDemandCommand](#).

C#

```
public bool CanExecute(object parameter)  
{  
    TreeNode node = (parameter as TreeNode);  
    EmployeeInfo emp = node.Item as EmployeeInfo;  
    if (emp != null)  
        if (emp.ReportsTo == -1 || emp.ReportsTo == 34 || emp.ReportsTo == 36 ||  
            emp.ReportsTo == 65)  
            return true;  
        return false;  
    }  
}
```

On-demand loading of child items

You can load child items for the node in [Execute](#) method of `LoadOnDemandCommand`. Execute method will get called when user expands the tree node. In execute method, you can populate the child nodes by calling [TreeNode.PopulateChildNodes](#) method by passing the child items collection.

C#

```
public void Execute(object parameter)
{
    TreeNode node = (parameter as TreeNode);
    EmployeeInfo emp = node.Item as EmployeeInfo;
    if (emp != null)
    {
        node.PopulateChildNodes((App.Current.MainWindow.DataContext as
        ViewModel).GetReportees(emp.ID));
    }
}
```

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

Load on demand using event

SfTreeGrid supports to load the data in on-demand through [SfTreeGrid.RequestTreeItems](#) event. RequestTreeItems event is triggered at the time of loading and when user expand any node at runtime. SfTreeGrid gets the root and leaf nodes through this event handler.

`TreeGridRequestTreeItemsEventArgs.ParentItem` denotes the data object looking for its child nodes. If it is null, it denotes SfTreeGrid requesting root nodes.

In the below example SfTreeGrid is populated through `SfTreeGrid.RequestTreeItems` instead of setting [SfTreeGrid.ItemsSource](#).

C#

```
ViewModel viewModel = new ViewModel();
treeGrid.RequestTreeItems += TreeGrid_RequestTreeItems;
private void TreeGrid_RequestTreeItems(object sender,
TreeGridRequestTreeItemsEventArgs args)
{
    if (args.ParentItem == null)
    {
        args.ChildItems = viewModel.Employees.Where(x => x.ReportsTo == -1);
    }
    else
    {
        EmployeeInfo employee = args.ParentItem as EmployeeInfo;
        if (employee != null)
        {
            args.ChildItems = viewModel.GetEmployees().Where(x => x.ReportsTo ==
            employee.ID);
        }
    }
}
```

C#

```
public class EmployeeInfo
{
    int _id;
    string _firstName;
    string _lastName;
    private string _title;
    double? _salary;
    int _reportsTo;
    public string FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }
    public string LastName
    {
        get { return _lastName; }
        set { _lastName = value; }
    }
    public int ID
    {
        get { return _id; }
        set { _id = value; }
    }
    public string Title
    {
        get { return _title; }
        set { _title = value; }
    }
    public double? Salary
    {
        get { return _salary; }
        set { _salary = value; }
    }
    public int ReportsTo
    {
        get { return _reportsTo; }
        set { _reportsTo = value; }
    }
}
```

C#

```
public class ViewModel
{
    public ViewModel()
    {
        this.Employees = this.GetEmployees();
    }
    private ObservableCollection<EmployeeInfo> _employees;
    public ObservableCollection<EmployeeInfo> Employees
    {
        get { return _employees; }
        set { _employees = value; }
    }
    public ObservableCollection<EmployeeInfo> GetEmployees()
    {

```

```

ObservableCollection<EmployeeInfo> employeeDetails = new
ObservableCollection<EmployeeInfo>();
employeeDetails.Add(new EmployeeInfo() { FirstName = "Ferando", LastName =
"Joseph", Title = "Management", Salary = 2000000, ReportsTo = -1, ID = 2 });
employeeDetails.Add(new EmployeeInfo() { FirstName = "John", LastName =
"Adams", Title = "Accounts", Salary = 2000000, ReportsTo = -1, ID = 3 });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Thomas", LastName =
"Jefferson", Title = "Sales", Salary = 300000, ReportsTo = -1, ID = 4 });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Andrew", LastName =
"Madison", Title = "Marketing", Salary = 4000000, ReportsTo = -1, ID = 5 });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Ulysses", LastName =
"Pierce", Title = "HumanResource", Salary = 1500000, ReportsTo = -1, ID = 6
});
employeeDetails.Add(new EmployeeInfo() { FirstName = "Jimmy", LastName =
"Harrison", Title = "Purchasing", Salary = 200000, ReportsTo = -1, ID = 7
});
employeeDetails.Add(new EmployeeInfo() { FirstName = "Ronald", LastName =
"Fillmore", Title = "Production", Salary = 2800000, ReportsTo = -1, ID = 8
});
//Management
employeeDetails.Add(new EmployeeInfo() { FirstName = "Andrew", LastName =
"Fuller", ID = 9, Salary = 1200000, ReportsTo = 2, Title = "Vice President"
});
employeeDetails.Add(new EmployeeInfo() { FirstName = "Janet", LastName =
"Leverling", ID = 10, Salary = 1000000, ReportsTo = 2, Title = "GM" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Steven", LastName =
"Buchanan", ID = 11, Salary = 900000, ReportsTo = 2, Title = "Manager" });
//Accounts
employeeDetails.Add(new EmployeeInfo() { FirstName = "Nancy", LastName =
"Davolio", ID = 12, Salary = 850000, ReportsTo = 3, Title = "Accounts
Manager" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Margaret", LastName =
"Peacock", ID = 13, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Michael", LastName =
"Suyama", ID = 14, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Robert", LastName =
"King", ID = 15, Salary = 650000, ReportsTo = 3, Title = "Accountant" });
//Sales
employeeDetails.Add(new EmployeeInfo() { FirstName = "Laura", LastName =
"Callahan", ID = 16, Salary = 900000, ReportsTo = 4, Title = "Sales Manager"
});
employeeDetails.Add(new EmployeeInfo() { FirstName = "Anne", LastName =
"Dodsworth", ID = 17, Salary = 800000, ReportsTo = 4, Title = "Sales
Representative" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Albert", LastName =
"Hellstern", ID = 18, Salary = 750000, ReportsTo = 4, Title = "Sales
Representative" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Tim", LastName =
"Smith", ID = 19, Salary = 700000, ReportsTo = 4, Title = "Sales
Representative" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Justin", LastName =
"Brid", ID = 20, Salary = 700000, ReportsTo = 4, Title = "Sales
Representative" });
//Back Office
employeeDetails.Add(new EmployeeInfo() { FirstName = "Caroline", LastName =
"Patterson", ID = 21, Salary = 800000, ReportsTo = 5, Title = "Receptionist"
});

```

```
employeeDetails.Add(new EmployeeInfo() { FirstName = "Xavier", LastName =
"Martin", ID = 22, Salary = 700000, ReportsTo = 5, Title = "Mail Clerk" });
//HR
employeeDetails.Add(new EmployeeInfo() { FirstName = "Laurent", LastName =
"Pereira", ID = 23, Salary = 900000, ReportsTo = 6, Title = "HR Manager" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Syed", LastName =
"Abbas", ID = 24, Salary = 650000, ReportsTo = 6, Title = "HR Assistant" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Amy", LastName =
"Alberts", ID = 25, Salary = 650000, ReportsTo = 6, Title = "HR Assistant"
});
//Purchasing
employeeDetails.Add(new EmployeeInfo() { FirstName = "Pamela", LastName =
"Ansman-Wolfe", ID = 26, Salary = 600000, ReportsTo = 7, Title = "Purchase
Manager" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Michael", LastName =
"Blythe", ID = 27, Salary = 550000, ReportsTo = 7, Title = "Store Keeper"
});
employeeDetails.Add(new EmployeeInfo() { FirstName = "David", LastName =
"Campbell", ID = 28, Salary = 450000, ReportsTo = 7, Title = "Store Keeper"
});
//Production
employeeDetails.Add(new EmployeeInfo() { FirstName = "Jillian", LastName =
"Carson", ID = 29, Salary = 600000, ReportsTo = 8, Title = "Production
Manager" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Shu", LastName =
"Ito", ID = 30, Salary = 550000, ReportsTo = 8, Title = "Production
Engineer" });
employeeDetails.Add(new EmployeeInfo() { FirstName = "Stephen", LastName =
"Jiang", ID = 31, Salary = 450000, ReportsTo = 8, Title = "Production
Engineer" });
return employeeDetails;
}
}
```

You can let SfTreeGrid to populate the data at runtime by calling [SfTreeGrid.RepopulateTree](#) method.

First Name	Employee ID	Last Name	Title	Salary	Reports To
<input type="checkbox"/> Richard	238	Adams	HR Coordinator	\$200000.00	-1
<input type="checkbox"/> Hawkin	244	Bush	Business Manager	\$200000.00	-1
<input type="checkbox"/> Madison	174	Hayes	Vice President	\$200000.00	-1

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

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

UI Automation in WPF TreeGrid (SfTreeGrid)

Requirements and configuration

To test SfTreeGrid control with CUITs, build the extension project and place it in the mentioned location. You can get the extension project from the following location.

SfTreeGrid	https://github.com/SyncfusionExamples/coded-ui-extension-project-for-wpf-treegrid
-------------------	---

1. Open the extension project and build it.
2. Get the following tabulated assembly from the bin folder.

SfGrid.WPF	Syncfusion.VisualStudio.TestTools.UITest.SfGridExtension.dll
-------------------	--

The SfGridExtension assembly must be placed into the following directory based on your Visual Studio version.

For Visual Studio 2010:

C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\10.0\UITestExtensionPackages

For Visual Studio 2012:

C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\11.0\UITestExtensionPackages

For Visual Studio 2013:

C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\13.0\UITestExtensionPackages

For Visual Studio 2015:

C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\14.0\UITestExtensionPackages

For Visual Studio 2017:

C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\15.0\UITestExtensionPackages

Note: Syncfusion.VisualStudio.TestTools.UITest.SfGridExtension.dll needs to be installed in GAC location. Please refer to the MSDN link for [GAC](#) installation.

Steps to work with Coded UI

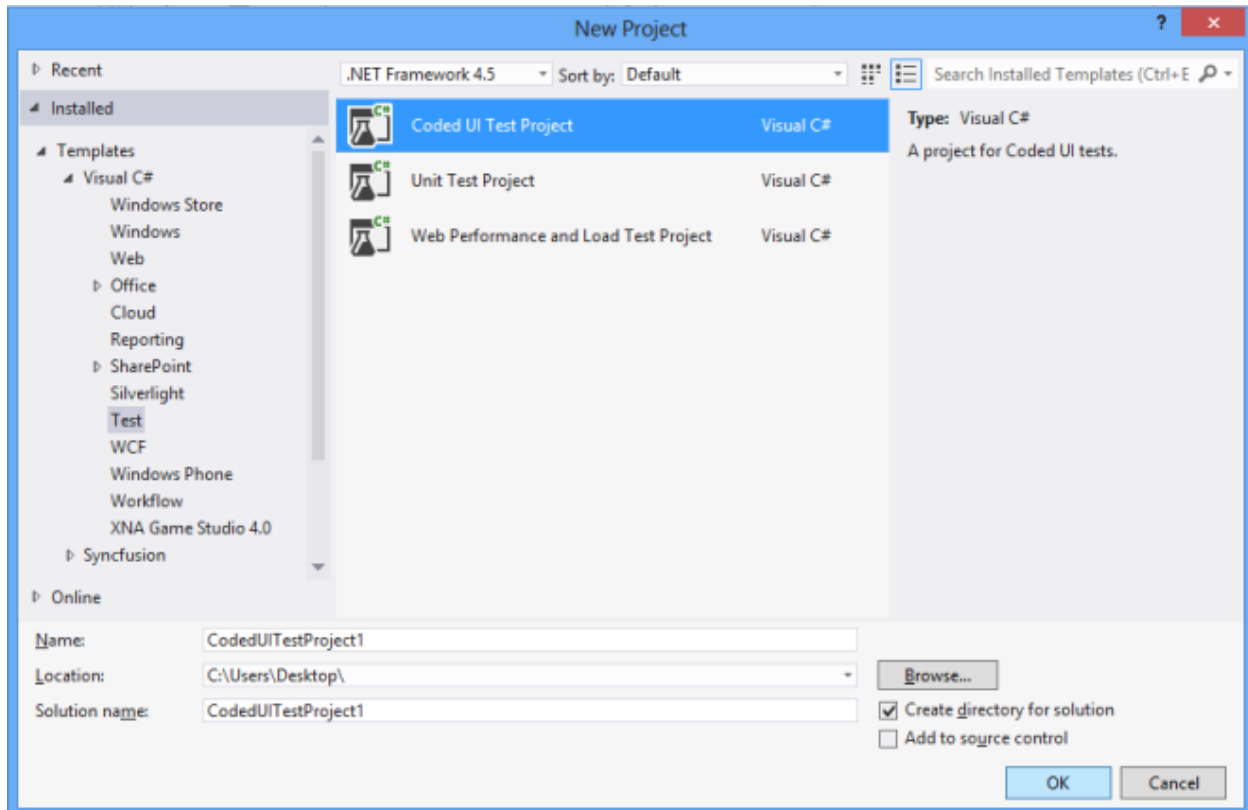
This section explains how to create a CodedUITest project and test the tree grid application.

1. Create a new WPF application or open an existing WPF application with tree grid and enable the Coded UI test in tree grid. To enable CUITs, set AutomationPeerHelper.EnableCodedUI to true and access the AutomationPeerHelper class from Syncfusion.UI.Xaml.Grid namespace as demonstrated in the following code sample.

C#

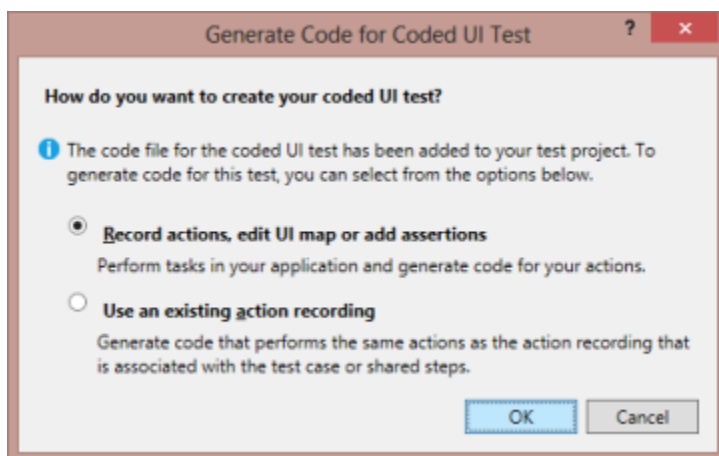
```
using Syncfusion.UI.Xaml.Grid;
public MainWindow()
{
    InitializeComponent();
    AutomationPeerHelper.EnableCodedUI = true;
}
```

2. Build the application and launch the .exe file from the bin folder.
3. Create a Coded UI test project as shown in the following screenshot.



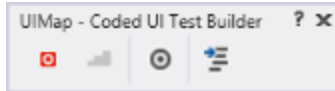
Add New Project

4. After created a new Coded UI project, a CUIT file is added automatically, and the Generate Code dialog box appears. In this, choose Record actions, edit UI map, or add assertions.



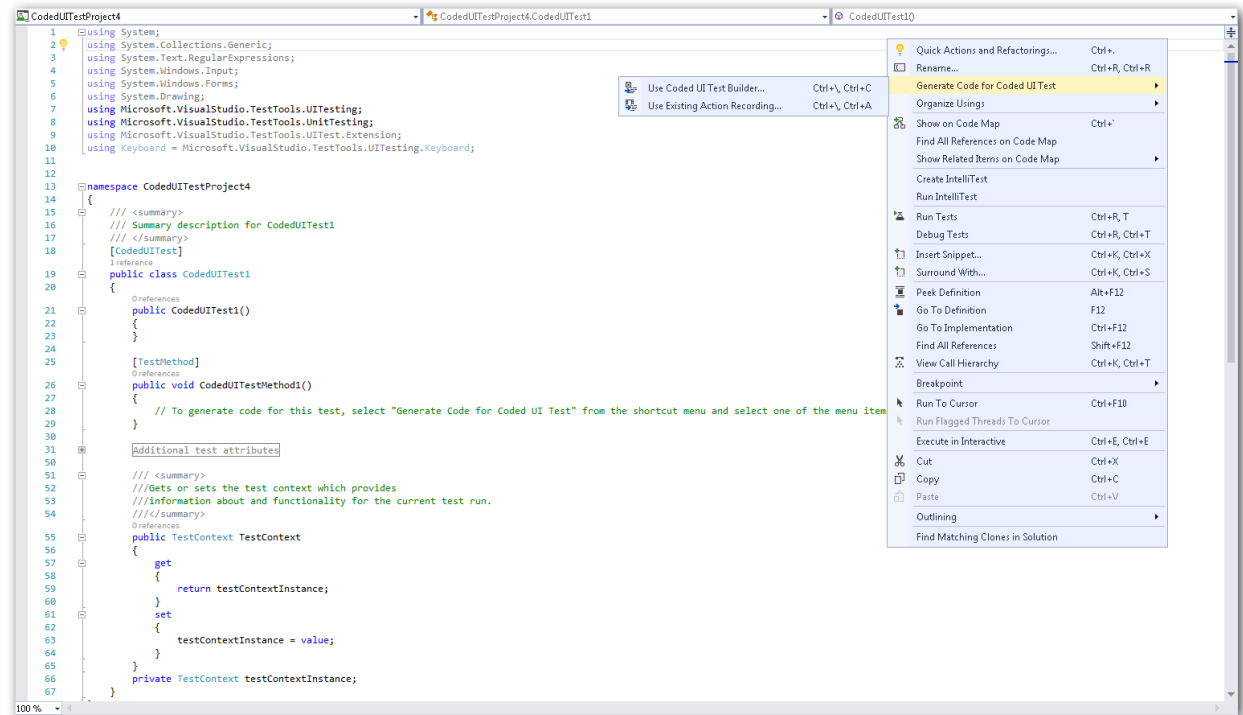
Generate Code for Coded UI Test

5. Now, minimize the Coded UI project Visual Studio, and CodedUITestBuilder appears in the bottom-right corner of your window. You can record the actions by clicking the Start Recording in CodedUITestBuilder.



CodedUITestBuilder

- Open the CodedUITestBuilder from the existing Coded UI project by right-clicking the CodedUITestMethod in the CUIT file and clicking the Generate Code for Coded UI test as shown in the following screenshot. You can see the same CodedUITestBuilder in the bottom-right corner of the window.



CodedUITestMethod

- Now, drag the Crosshairs to the UI elements of your WPF SfTreeGrid application, and it shows the available properties of the inner UI elements in SfTreeGrid.
- Record the actions made on UI elements by clicking the Record button on the CodedUITest builder. For example you can record the action of changing the cell value in SfTreeGrid. Click the Pause button to finish the record.

	Name		Other Details	
	FirstName	LastName	Id	Hike
	Chester	Buchanan	0.00	15.00 %
	Abraham	Fillmore	1.00	7.70 %
	Gerald	Obama	1112.00	5.50 %
	Herbert	Harding	2223.00	6.00 %
	Jimmy	sync	3334.00	16.00 %
	Jimmy	Jefferson	4445.00	10.20 %
	George	Fillmore	5556.00	14.00 %
	Ronald	Truman	6667.00	6.80 %
	Benjamin	Jefferson	7778.00	7.70 %
	Warren	Stogner	8889.00	5.50 %
	Dwight	Garfield	10000.00	15.00 %
	Franklin	Washington		
	Andrew	Buchanan		
	Benjamin	Cleveland		

CodedUITest

- After the record has been completed, click the GenerateCode icon in CodedUITestBuilder for generate a test method, and then close the CodedUITestBuilder. You can see the generated code for cell value changed action as follows.

C#

```

public void RecordedMethod1 ()
{
    public void RecordedMethod2 ()
    {
        #region Variable Declarations
        WpfText uIHardingText =
            this.UIWpfWindow.UITreeGridRowControlCustom.UITreeGridCellCustom.UIHardingText;
        WpfEdit uIItemEdit =
            this.UIWpfWindow.UITreeGridRowControlCustom.UITreeGridCellCustom.UIItemEdit;
        WpfText uIJeffersonText =
            this.UIWpfWindow.UITreeGridRowControlCustom1.UITreeGridCellCustom.UIJeffersonText;
        #endregion
        // Double-Click 'Harding' label
        Mouse.DoubleClick(uIHardingText, new Point(75, 5));
        // Type 'sync' in text box
        uIItemEdit.Text = this.RecordedMethod2Params.UIItemEditText;
        // Click 'Jefferson' label
        Mouse.Click(uIJeffersonText, new Point(140, 6));
    }
}

```

10. Create an assertion to check the modified cell value. Drag the crosshair to the modified cell, and the Assertion window appears. The properties for control (Cell) is now listed in the Assertion dialog box. You can add assertion by clicking the Generate Code button in CodedUITestBuilder.

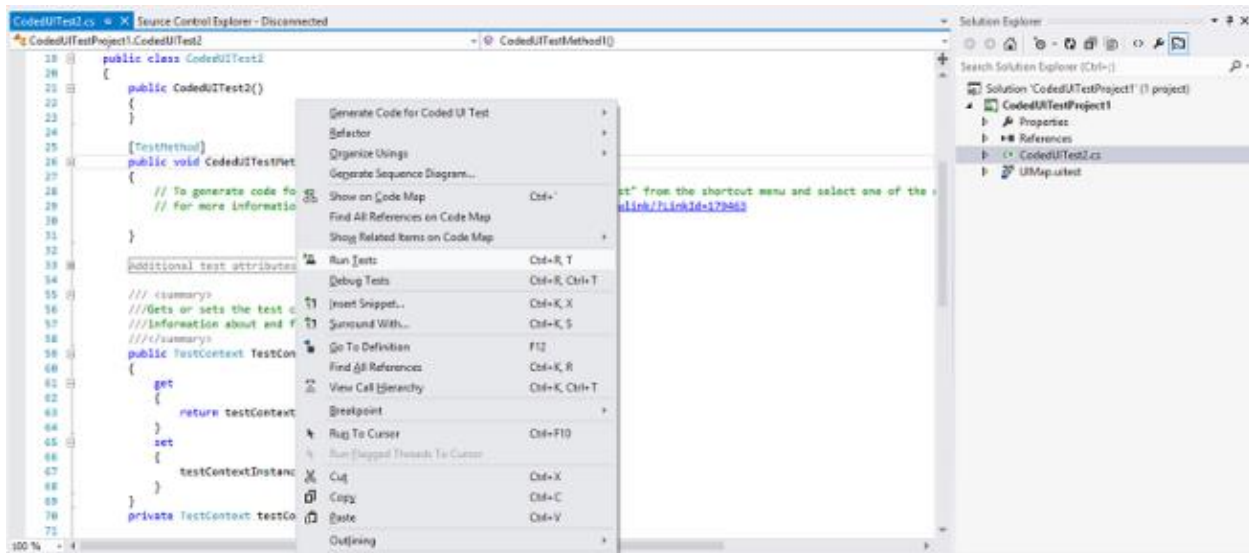
The screenshot shows the Coded UI Test Builder interface. On the left, a table lists names in two columns: 'FirstName' and 'LastName'. The cell containing 'Fillmore' in the 'LastName' column is highlighted with a blue border. On the right, the 'Other Details' pane is open, showing the 'Add Assertion' button and a table of properties for the selected 'TreeGridCell'. The properties include 'ClassName', 'ControlType', 'TechnologyName', 'Control Specific' (CellValue, FormattedValue, RowIndex, ColumnIndex, ColumnName, HeaderText), and 'Value'.

FirstName	LastName
Chester	Buchanan
Abraham	Fillmore
Gerald	Obama
Herbert	Harding
Jimmy	Hayes
Jimmy	Jefferson
George	Fillmore
Ronald	Truman
Benjamin	Jefferson
Warren	Stogner
Dwight	Garfield
Franklin	Washington
Andrew	Buchanan
Benjamin	Cleveland
Warner	Carter

Other Details
 Coded UI Test Builder - Add Assertions: UITre...
 Add Assertion
 Property Value
 Search
 ClassName TreeGridCell
 ControlType Custom
 TechnologyName UIA
 Control Specific
 CellValue Fillmore
 FormattedValue Fillmore
 RowIndex 3
 ColumnIndex 2
 ColumnName LastName
 HeaderText LastName

Assertion window

11. After all the tests and assertion have been created, right-click the Test method, and click Run Tests to run the test as follows.



Run Test

Properties

UI Elements	Properties
TreeGrid	RowCountColumnCountSelectionModeSelectionUnitSelectionIndexSelectedItemCount
TreeGridCell	CellValueFormattedValueRowIndexColumnIndexColumnName
TreeGridHeaderCell	MappingNameSortDirectionSortNumberVisibility
TreeGridRowHeaderCell	RowErrorMessageRowIndexState
TreeGridStackedHeaderCell	Default Properties
TreeGridExpanderCell	Default properties

Automation peer class name	Control name in code generation	Property provider class name
SfTreeGridAutomationPeer	WpfSfTreeGrid	SfTreeGridPropertyProvider
TreeGridCellAutomationPeer	WpfSfTreeGridCell	SfTreeGridCellPropertyProvider
TreeGridHeaderCellAutomationPeer	WpfSfTreeGridHeaderCell	SfTreeGridHeaderCellPropertyProvider
TreeGridRowHeaderCellAutomationPeer	WpfSfTreeGridRowHeaderCell	SfTreeGridRowHeaderCellPropertyProvider
TreeGridStackedHeaderCellAutomationPeer	WpfSfTreeGridStackedHeaderCell	SfTreeGridStackedHeaderCellPropertyProvider
TreeGridExpanderCellAutomationPeer	WpfTreeGridExpanderCell	SfTreeGridExpanderCellPropertyProvider

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

Helpers in WPF TreeGrid (SfTreeGrid)

IndexResolver

SfTreeGrid has [TreeGridIndexResolver](#) static class present in the Syncfusion.UI.Xaml.TreeGrid namespace has some extension methods to resolve from row or column index to node or visible column index and vice-versa.

For example, you can get a node from its row index using the [GetNodeAtRowIndex](#) method.

C#

```
treeGrid.GetNodeAtRowIndex(3);
```

Prototype table

Prototype	Description
ResolveToNodeIndex(int rowindex)	Resolves the node index from the row index. When row index does not find any record, it returns -1. The NodeIndex denotes the index of node in SfTreeGrid.View.Nodes.
ResolveToRowIndex(int nodeIndex)	Resolves the row index from the node index associated with SfTreeGrid.View.Nodes. When node index is lesser than 0 it returns -1.
ResolveToRowIndex(object data)	Resolves the row index from the data associated with SfTreeGrid.View.Nodes. When the given node is not available in the collection, it returns -1.
ResolveToRowIndex(TreeNode node)	Resolves the row index from the node associated with SfTreeGrid.View.Nodes. When the given node is not available in the collection, it returns -1.
ResolveToGridVisibleColumnIndex(int columnIndex)	Resolves the TreeGridColumn index from the visible column index. It excludes row header and indent column.
ResolveToScrollColumnIndex(int gridColumnIndex)	Resolves column index from the grid column index associated with SfTreeGrid.Columns. It also includes the indent column and row header also.
ResolveToStartColumnIndex()	Returns the start column index of the VisibleColumn.
GetHeaderIndex()	Returns the header row index.
GetNodeAtRowIndex(int rowIndex)	Gets the tree node based on the row index.

Dispose

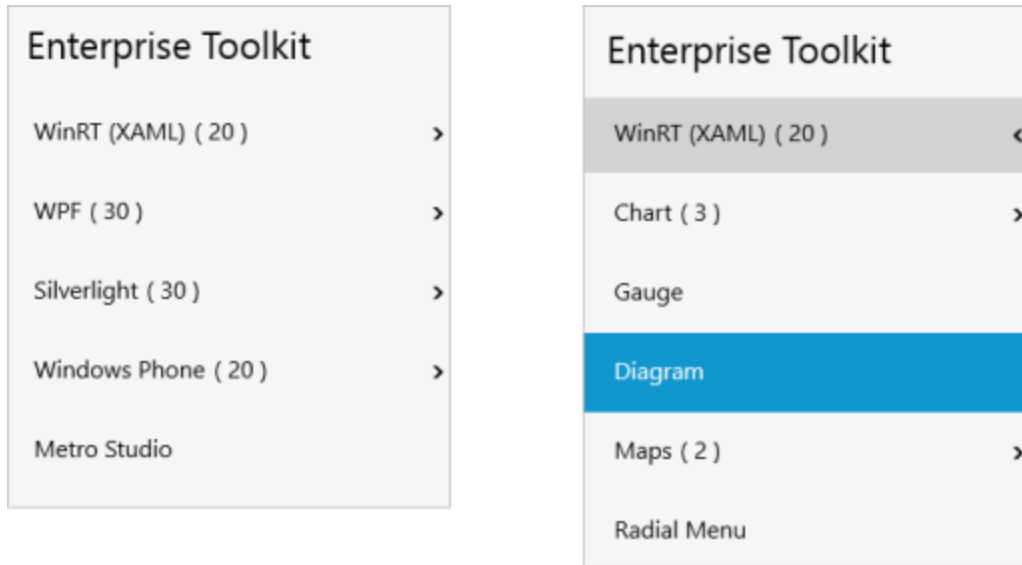
This method is associated with relinquishes memory and clears all references associated with treegrid. When you call this method, it releases all references for treegrid. So, the memory occupied using treegrid is reclaimed. You should call the [SfTreeGrid.Dispose](#) method to release the memory.

Note: You can refer to our [WPF TreeGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeGrid example](#) to know how to render and configure the treegrid.

SfTreeNavigator

WPF Tree Navigator (SfTreeNavigator) Overview

The tree navigator control provides a unique interface that can expand a tree structure in-place without taking up more space on the screen.



Key features

- Items Source – Any business object collection can be bound to control.
- Hierarchical Data Template – Business objects displayed in the hierarchy can be customized with single template.
- Navigation Mode – Two types of navigation mode (Default and Extended) support.

Getting Started with WPF Tree Navigator (SfTreeNavigator)

Namespace : Syncfusion.Windows.Controls.Navigation

Assembly : Syncfusion.SfTreeNavigator.WPF (in Syncfusion.SfTreeNavigator.WPF.dll)

The following code sample shows how to create the Tree Navigator from code-behind and XAML,

XAML

```
<navigation:SfTreeNavigator Header="Enterprise Toolkit" >
<navigation:SfTreeNavigatorItem Header="WinRT (XAML)">
<navigation:SfTreeNavigatorItem Header="Chart"/>
<navigation:SfTreeNavigatorItem Header="Tools"/>
</navigation:SfTreeNavigatorItem>
<navigation:SfTreeNavigatorItem Header="Metro Studio"/>
</navigation:SfTreeNavigator>
```

C#

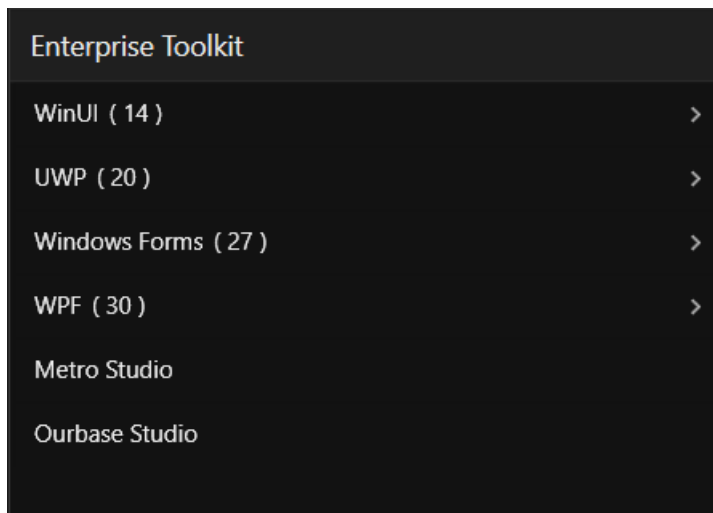
```
SfTreeNavigator sfToolkit = new SfTreeNavigator();
SfTreeNavigatorItem winrt = new SfTreeNavigatorItem() {Header = "WinRT
(XAML)"};
SfTreeNavigatorItem metroStudio = new SfTreeNavigatorItem() {Header = "Metro
Studio"};
SfTreeNavigatorItem winrt_chart = new SfTreeNavigatorItem() {Header =
"Chart"};
SfTreeNavigatorItem winrt_tools = new SfTreeNavigatorItem() {Header =
"Tools"};
winrt.Items.Add(winrt_chart);
```

```
winrt.Items.Add(winrt_tools);  
sfToolkit.Items.Add(winrt);  
sfToolkit.Items.Add(metroStudio);
```

Theme

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

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



Populating Items in WPF Tree Navigator (SfTreeNavigator)

Items source

[Tree Navigator items](#) can be populated with the business object collection. Let us create a [Tree Navigator](#) which will show the list of Syncfusion Enterprise Toolkit products.

Create a Model class with the necessary properties.

C#

```
public class TreeModel : NotificationObject  
{  
    public TreeModel()  
    {  
        Models = new ObservableCollection<TreeModel>();  
    }  
    private string header;  
    public string Header  
    {  
        get { return header; }  
        set  
        {  
            header = value;  
            RaisePropertyChanged("Header");  
        }  
    }  
    private ObservableCollection<TreeModel> models;
```



```
public ObservableCollection<TreeModel> Models
{
    get { return models; }
    set { models = value; }
}
```

Note: NotificationObject is a class which implements INotifyPropertyChanged interface.

Create a View Model class with the hierarchical items as follows.

C#

```
public class TreeViewModel
{
    private List<TreeModel> models;
    public List<TreeModel> Models
    {
        get { return models; }
        set { models = value; }
    }
    public TreeViewModel()
    {
        Models = new List<TreeModel>();
        TreeModel winrt = new TreeModel() {Header = "WinRT (XAML)"};
        TreeModel metroStudio = new TreeModel() {Header = "Metro Studio"};
        TreeModel winrt_chart = new TreeModel() {Header = "Chart"};
        TreeModel winrt_tools = new TreeModel() {Header = "Tools"};
        winrt.Models.Add(winrt_chart);
        winrt.Models.Add(winrt_tools);
        Models.Add(winrt);
        Models.Add(metroStudio);
    }
}
```

Bind the Models collection to the ItemsSource property of the [Tree Navigator](#) control as follows.

XML

```
<navigation:SfTreeNavigator ItemsSource="{Binding Models}"
Header="Enterprise Toolkit"
Width="300" Height="400"
HorizontalAlignment="Center"
VerticalAlignment="Center" />
```

This will populate the [Tree Navigator](#) as shown below.



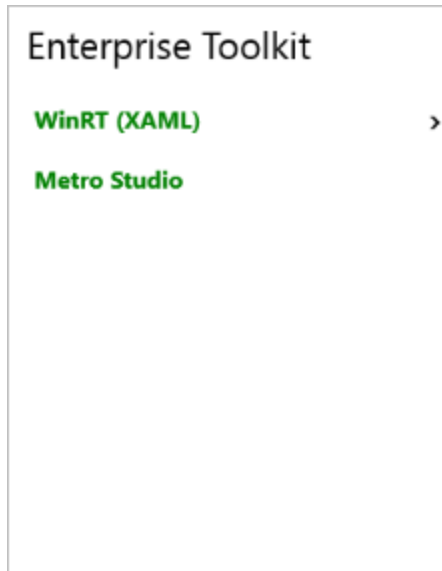
Item template

ItemTemplate property of the [Tree Navigator](#) can be used to customize the display of business objects.

XML

```
<navigation:SfTreeNavigator ItemsSource="{Binding Models}"
Header="Enterprise Toolkit"
Width="300" Height="400"
HorizontalAlignment="Center"
VerticalAlignment="Center"
>
  <navigation:SfTreeNavigator.ItemTemplate>
    <HierarchicalDataTemplate ItemsSource="{Binding Models}">
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding Header}"
          Foreground="Green" FontWeight="Bold"
          VerticalAlignment="Center" Margin="18 0 0 0"/>
      </StackPanel>
    </HierarchicalDataTemplate>
  </navigation:SfTreeNavigator.ItemTemplate>
</navigation:SfTreeNavigator>
```

This will populate the [Tree Navigator](#) as follows.



See Also

[How to enable/disable menu items in WPF SfTreeNavigator using MVVM?](#)

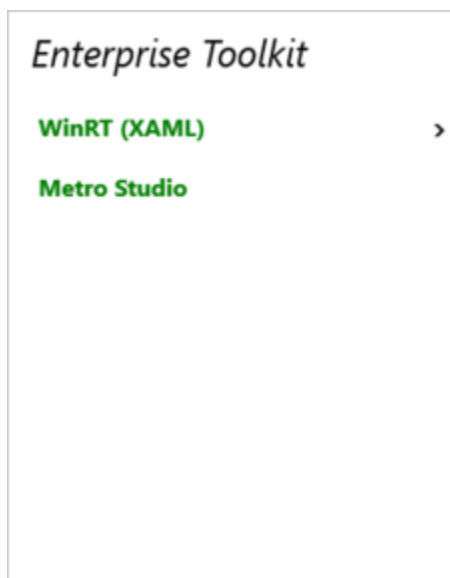
[Header Template in WPF Tree Navigator \(SfTreeNavigator\)](#)

The HeaderTemplate property of Tree Navigator can be used to customize the Tree Navigator header.

XML

```
<navigation:SfTreeNavigator.HeaderTemplate>
<DataTemplate>
<TextBlock Text="{Binding}" FontStyle="Italic"/>
</DataTemplate>
</navigation:SfTreeNavigator.HeaderTemplate>
```

Tree Navigator now displayed as shown below.



Navigation Mode in WPF Tree Navigator (SfTreeNavigator)

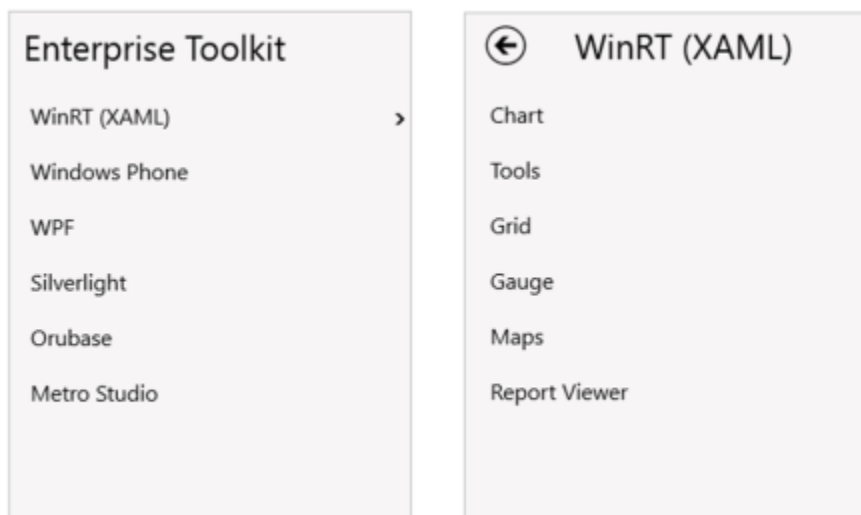
Two types of Navigation mode are supported by Tree Navigator to navigate between hierarchy levels.

Default

In this navigation mode, the header of current hierarchy level item can be displayed in the top of the Tree Navigator with the back button. This back button is used to navigate towards the root from the current level.

XML

```
<navigation:SfTreeNavigator ItemsSource="{Binding Models}"  
Header="Enterprise Toolkit"  
NavigationMode="Default"  
Width="300" Height="400"  
HorizontalAlignment="Center"  
VerticalAlignment="Center" />
```

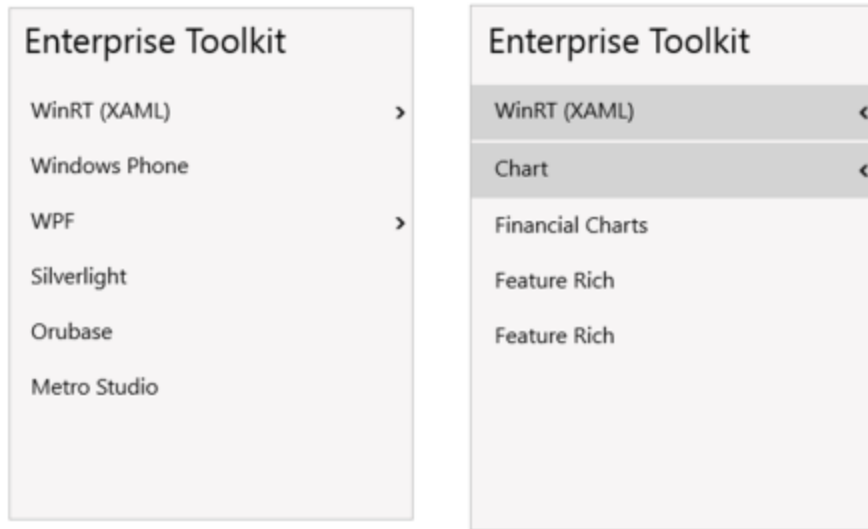


Extended

In this navigation mode, header of each level from the root to current level stacked one by one in the top of the Tree Navigator. When click on any of that header will take us to the corresponding level.

XML

```
<navigation:SfTreeNavigator ItemsSource="{Binding Models}"  
Header="Enterprise Toolkit"  
NavigationMode="Extended"  
Width="300" Height="400"  
HorizontalAlignment="Center"  
VerticalAlignment="Center" />
```



Note: Header of the Extended mode can be styled using `TreeNavigatorHeaderItem` available in the same namespace.

Selected Item in WPF Tree Navigator (SfTreeNavigator)

The `SelectedItem` property of `SfTreeNavigator` can be used to get or set the `SelectedItem` in `SfTreeNavigator`.

Please find the code example for the same from below:

XML

```
<Grid>
<!--Binding the selecteditem for TreeNavigator-->
<navigation:SfTreeNavigator Header="MailBox" x:Name="TreeNavigator"
Width="500" Height="300" SelectedItem="{Binding SelectedItem, Mode=TwoWay}">
<navigation:SfTreeNavigatorItem Header="Mail"/>
<navigation:SfTreeNavigatorItem Header="Favorite Folders"/>
<navigation:SfTreeNavigatorItem Header="Contacts">
<navigation:SfTreeNavigatorItem Header="Task"/>
</navigation:SfTreeNavigatorItem>
<navigation:SfTreeNavigatorItem Header="Notes"/>
</navigation:SfTreeNavigator>
</Grid>
```

C#

```
public partial class MainWindow : ChromelessWindow
{
    public MainWindow()
    {
        InitializeComponent();
        this.DataContext = new ViewModel();
        //Set the selecteditem
        (this.DataContext as ViewModel).SelectedItem = TreeNavigator.Items[1];
    }
}
```

```
//Initiate the viewmodel class
public class ViewModel
{
    private object selecteditem;
    public object SelectedItem
    {
        // Get the selecteditem
        get
        {
            return selecteditem;
        }
        set
        {
            //Set the selecteditem
            selecteditem = value;
        }
    }
}
```

VB.NET

```
'Initiate the viewmodel class
Partial Public Class MainWindow
    Inherits ChromelessWindow
    Public Sub New()
        InitializeComponent()
        Me.DataContext = New ViewModel()
        (TryCast(Me.DataContext, ViewModel)).SelectedItem = TreeNavigator.Items(1)
    End Sub
    Public Class ViewModel
        Private selecteditem As Object
        'Get the selecteditem
        Public Property SelectedItem As Object
        Get
            Return SelectedItem
        End Get
        'Set the selecteditem
        Set(ByVal value As Object)
            selecteditem = value
        End Set
    End Property
End Class
```

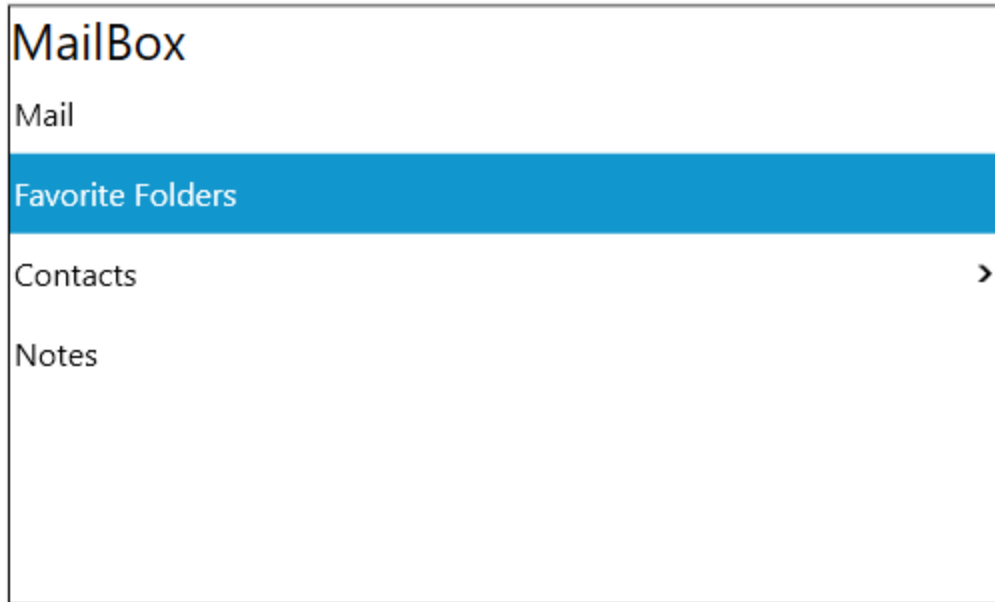


Fig i: Shows the Item has been selected in SfTreeView

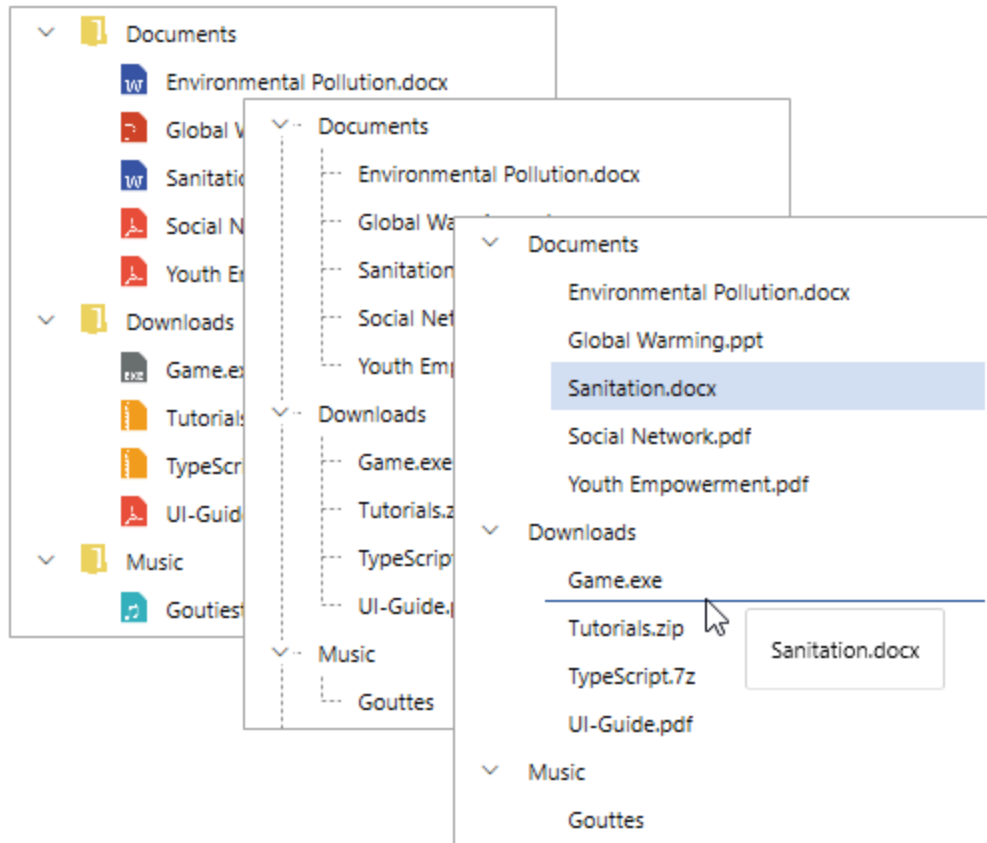
SfTreeView

WPF TreeView (SfTreeView) Overview

The Syncfusion [WPF TreeView](#) is a data-oriented control that displays data in a hierarchical structure with expanding and collapsing nodes. It is commonly used to illustrate a folder structure, or nested relationships in an application.

Key features

- Enhanced performance: Optimized view reuse strategy and flat rendering architecture for enhanced performance.
- Bound and unbound modes: Support to bind hierarchical data or add unbound tree nodes.
- On demand loading: Support to load the nodes on demand when the end user expands the node.
- Selection: Support to select the nodes with different selection modes and keyboard navigation.
- Drag and drop: Support to reorder the nodes by dragging and dropping them.
- Templating: Provides complete UI customization using the template and template selectors.
- Root lines: Support to show the lines between tree nodes.



Note: You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Getting Started with WPF TreeView (SfTreeView)

This section provides a quick overview for getting started with the **SfTreeView** for WPF. Walk through the entire process of creating a real world of this control.

Assembly Deployment

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

Refer to this documentation to find more details about installing nuget packages in a WPF application.

Creating simple application with SfTreeView

- Creating the project
- Add SfTreeView to Project
- Populating Nodes without data source - Unbound Mode
- Bind to a hierarchical data source - Bound Mode
- Bind to a Hierarchy Property Descriptors data source - Bound Mode

Creating the project

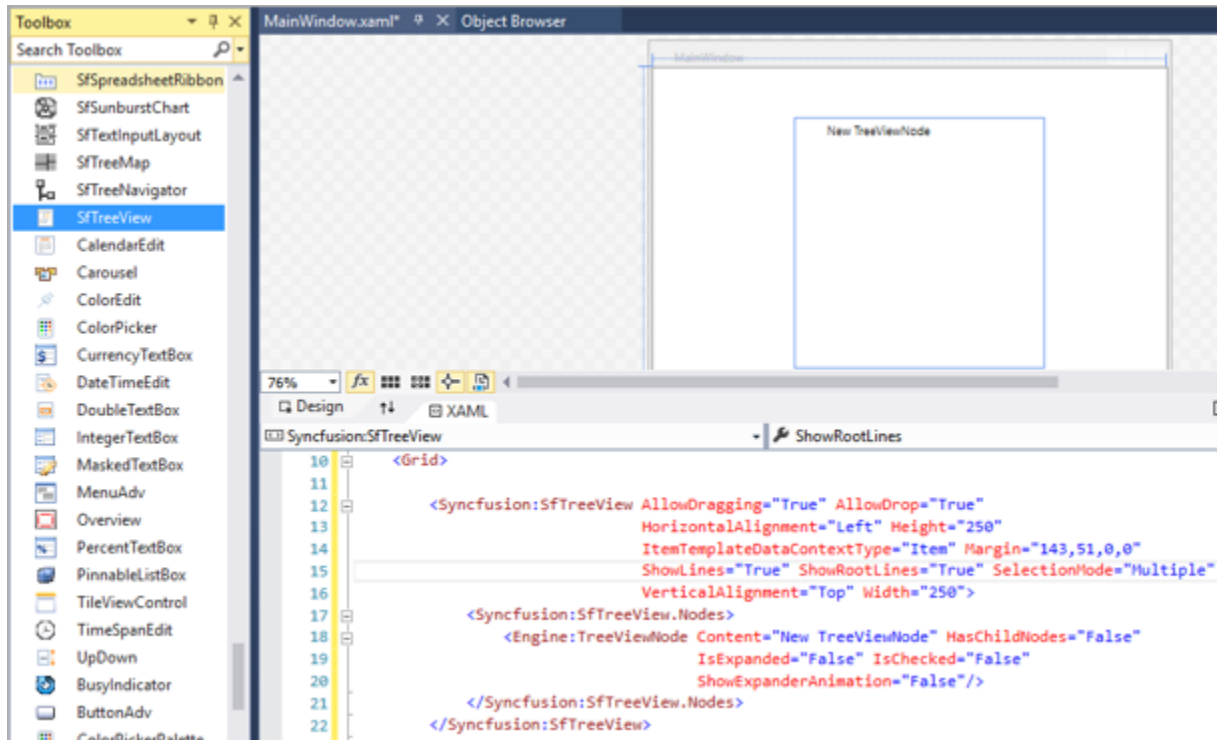
Create new WPF Project in Visual Studio to display [SfTreeView](#) with data objects.

Add SfTreeView to Project

The WPF TreeView (SfTreeView) control can be added to project by the following ways.

Adding SfTreeView by designer.

WPF TreeView (SfTreeView) control can be added to the application by dragging it from Toolbox and dropping it in Designer view. The required assembly references will be added automatically.



Adding SfTreeView by XAML.

In order to add control manually in XAML, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.SfBusyIndicator.WPF
 - Syncfusion.SfTreeView.WPF
 - Syncfusion.SfGridCommon.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page.
3. Declare SfTreeView control in XAML page.

XML

```
<Window x:Class="GettingStarted.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:local="clr-namespace:GettingStarted"
  xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
  mc:Ignorable="d"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <syncfusion:SfTreeView x:Name="treeView" />
  </Grid>
```

```
</Grid>
</Window>
```

Adding SfTreeView by C#.

In order to add control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.SfBusyIndicator.WPF
 - Syncfusion.SfTreeView.WPF
 - Syncfusion.SfGridCommon.WPF
2. Import SfTreeView namespace **using Syncfusion.UI.Xaml.TreeView** .
3. Create SfTreeView control instance and add it to the Page.

C#

```
using Syncfusion.UI.Xaml.TreeView;
using System.Windows;
namespace GettingStarted
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfTreeView treeView = new SfTreeView();
            Root_Grid.Children.Add(treeView);
        }
    }
}
```

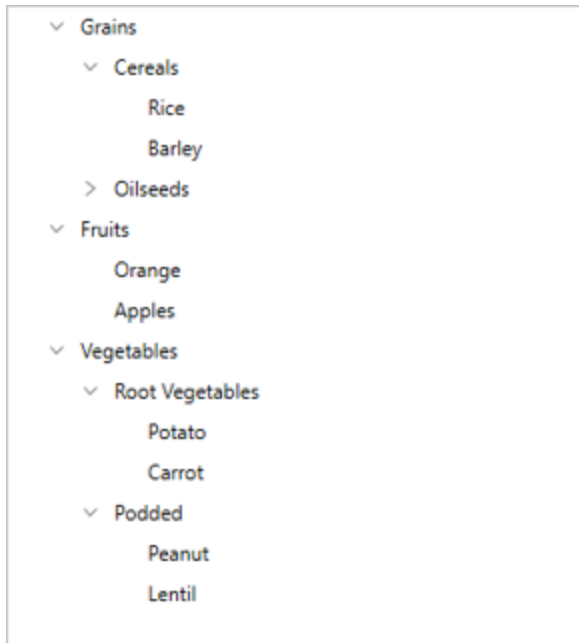
Populating Nodes without data source - Unbound Mode

You can create and manage the [TreeViewNode](#) objects by yourself to display the data in a hierarchical view. To create a tree view, you use a [SfTreeView](#) control and a hierarchy of [TreeViewNode](#) objects. You create the node hierarchy by adding one or more root nodes to the [SfTreeView.Nodes](#) collection. Each [TreeViewNode](#) can have more nodes added to its Children collection. You can nest tree view nodes to whatever depth you require.

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:GettingStarted"
xmlns:Syncfusion="http://schemas.syncfusion.com/wpf" xmlns:Engine="clr-namespace:Syncfusion.UI.Xaml.TreeView.Engine;assembly=Syncfusion.SfTreeView.WPF" x:Class="GettingStarted.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
```

```
<Grid>
<Syncfusion:SfTreeView HorizontalAlignment="Left" Height="414"
Margin="318,0,0,0" VerticalAlignment="Center" Width="250">
<Syncfusion:SfTreeView.Nodes>
<Engine:TreeNode Content="Grains" IsExpanded="True">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Cereals" IsExpanded="True">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Rice"/>
<Engine:TreeNode Content="Barley"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
<Engine:TreeNode Content="Oilseeds">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Safflower"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
<Engine:TreeNode Content="Fruits" IsExpanded="true">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Orange"/>
<Engine:TreeNode Content="Apples" IsExpanded="true"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
<Engine:TreeNode Content="Vegetables" IsExpanded="true">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Root Vegetables" IsExpanded="true">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Potato"/>
<Engine:TreeNode Content="Carrot"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
<Engine:TreeNode Content="Podded">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Peanut"/>
<Engine:TreeNode Content="Lentil"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
</Syncfusion:SfTreeView.Nodes>
</Syncfusion:SfTreeView>
</Grid>
</Window>
```



Bind to a hierarchical data source - Bound Mode

You can create a tree view by binding the `ItemsSource` to a hierarchical data source. To create a tree view using data binding, set a hierarchical collection to the [ItemsSource](#) property. Then in the [ItemTemplate](#) and [ExpanderTemplate](#), set the child items collection to the `ItemsSource` property.

XML

```
<Window x:Class="NodeWithImageDemo.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:NodeWithImageDemo"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:Syncfusion="http://schemas.syncfusion.com/wpf" xmlns:Engine="clr-
namespace:Syncfusion.UI.Xaml.TreeView.Engine;assembly=Syncfusion.SfTreeView.
WPF"
mc:Ignorable="d">
<Window.DataContext>
<local:FileManagerViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:SfTreeView x:Name="sfTreeView"
ChildPropertyName="SubFiles"
ItemsSource="{Binding ImageNodeInfo}">
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<Grid x:Name="grid">
<Grid Grid.Row="0">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="20" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Grid>
<Image Source="{Binding ImageIcon}"
```

```
VerticalAlignment="Center"
HorizontalAlignment="Center"
Height="16"
Width="16"/>
</Grid>
<Grid Grid.Column="1"
Margin="1,0,0,0"
VerticalAlignment="Center">
<TextBlock Text="{Binding ItemName}"
Foreground="Black"
FontSize="14"
VerticalAlignment="Center"
</Grid>
</Grid>
</Grid>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>
</Grid>
</Window>
```

C#

```
public class FileManager : INotifyPropertyChanged
{
    private string fileName;
    private ImageSource imageIcon;
    private ObservableCollection<FileManager> subFiles;
    public ObservableCollection<FileManager> SubFiles
    {
        get { return subFiles; }
        set
        {
            subFiles = value;
            RaisedOnPropertyChanged("SubFiles");
        }
    }
    public string ItemName
    {
        get { return fileName; }
        set
        {
            fileName = value;
            RaisedOnPropertyChanged("FolderName");
        }
    }
    public ImageSource ImageIcon
    {
        get { return imageIcon; }
        set
        {
            imageIcon = value;
            RaisedOnPropertyChanged("ImageIcon");
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
```

```
public void RaisedOnPropertyChanged(string _PropertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(_PropertyName));
    }
}
}
```

C#

```
public class FileManagerViewModel
{
    private ObservableCollection<FileManager> imageNodeInfo;
    public FileManagerViewModel()
    {
        GenerateSource();
    }
    public ObservableCollection<FileManager> ImageNodeInfo
    {
        get { return imageNodeInfo; }
        set { this.imageNodeInfo = value; }
    }
    private void GenerateSource()
    {
        var nodeImageInfo = new ObservableCollection<FileManager>();
        Assembly assembly = typeof(GettingStated).GetTypeInfo().Assembly;
        var doc = new FileManager() { ItemName = "Documents", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var download = new FileManager() { ItemName = "Downloads", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var mp3 = new FileManager() { ItemName = "Music", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var pictures = new FileManager() { ItemName = "Pictures", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var video = new FileManager() { ItemName = "Videos", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var pollution = new FileManager() { ItemName = "Environmental
            Pollution.docx", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_word.png",
            assembly) };
        var globalWarming = new FileManager() { ItemName = "Global Warming.ppt",
            ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_ppt.png",
            assembly) };
        var sanitation = new FileManager() { ItemName = "Sanitation.docx", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_word.png",
            assembly) };
        var socialNetwork = new FileManager() { ItemName = "Social Network.pdf",
            ImageIcon =
```

```
ImageSource.FromResource("GettingStartedBound.Icons.treeview_pdf.png",
assembly) };
var youthEmpower = new FileManager() { ItemName = "Youth Empowerment.pdf",
ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_pdf.png",
assembly) };
var games = new FileManager() { ItemName = "Game.exe", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_exe.png",
assembly) };
var tutorials = new FileManager() { ItemName = "Tutorials.zip", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_zip.png",
assembly) };
var TypeScript = new FileManager() { ItemName = "TypeScript.7z", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_zip.png",
assembly) };
var uiGuide = new FileManager() { ItemName = "UI-Guide.pdf", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_pdf.png",
assembly) };
var song = new FileManager() { ItemName = "Goutiest", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_mp3.png",
assembly) };
var camera = new FileManager() { ItemName = "Camera Roll", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
assembly) };
var stone = new FileManager() { ItemName = "Stone.jpg", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_png.png",
assembly) };
var wind = new FileManager() { ItemName = "Wind.jpg", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_png.png",
assembly) };
var img0 = new FileManager() { ItemName = "WIN_20160726_094117.JPG",
ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_img0.png",
assembly) };
var img1 = new FileManager() { ItemName = "WIN_20160726_094118.JPG",
ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_img1.png",
assembly) };
var video1 = new FileManager() { ItemName = "Naturals.mp4", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_video.png",
assembly) };
var video2 = new FileManager() { ItemName = "Wild.mpg", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_video.png",
assembly) };
doc.SubFiles = new ObservableCollection<FileManager>
{
pollution,
globalWarming,
sanitation,
socialNetwork,
youthEmpower
};
download.SubFiles = new ObservableCollection<FileManager>
{
games,
tutorials,
TypeScript,
```

```

uiGuide
};
mp3.SubFiles = new ObservableCollection<FileManager>
{
    song
};
pictures.SubFiles = new ObservableCollection<FileManager>
{
    camera,
    stone,
    wind
};
camera.SubFiles = new ObservableCollection<FileManager>
{
    img0,
    img1
};
video.SubFiles = new ObservableCollection<FileManager>
{
    video1,
    video2
};
nodeImageInfo.Add(doc);
nodeImageInfo.Add(download);
nodeImageInfo.Add(mp3);
nodeImageInfo.Add(pictures);
nodeImageInfo.Add(video);
imageNodeInfo = nodeImageInfo;
}
}

```

Bind to a Hierarchy Property Descriptors data source - Bound Mode

You can create a tree view by binding the `ItemsSource` to a hierarchy property descriptors data source. To create a tree view using hierarchical data binding, set a hierarchical collection to the [ItemsSource](#) property, and then set the `TargetType` and `ChildPropertyName` property values in [HierarchyPropertyDescriptors](#).

XML

```

<Window x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:treeViewEngine="clr-namespace:Syncfusion.UI.Xaml.TreeView.Engine;assembly=Syncfusion.SfTreeView.WPF"
mc:Ignorable="d"
Title="MainWindow" >
<Window.DataContext>
<local:FileManagerViewModel/>
</Window.DataContext>
<Grid>

```



```
<syncfusion:SfTreeView x:Name="treeView" ItemsSource="{Binding
ImageNodeInfo}">
<syncfusion:SfTreeView.HierarchyPropertyDescriptors>
<treeViewEngine:HierarchyPropertyDescriptor TargetType="{x:Type
local:Folder}" ChildPropertyName="Files"/>
<treeViewEngine:HierarchyPropertyDescriptor TargetType="{x:Type local:File}"
ChildPropertyName="SubFiles"/>
<treeViewEngine:HierarchyPropertyDescriptor TargetType="{x:Type
local:SubFile}" ChildPropertyName="Items"/>
</syncfusion:SfTreeView.HierarchyPropertyDescriptors>
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding FileName}" VerticalAlignment="Center" />
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>
</Grid>
</Window>
```

C#

```
public class Folder : INotifyPropertyChanged
{
    private string fileName;
    private ImageSource imageIcon;
    private ObservableCollection<File> files;
    public Folder()
    {
    }
    public ObservableCollection<File> Files
    {
        get { return files; }
        set
        {
            files = value;
            RaisedOnPropertyChanged("SubFiles");
        }
    }
    public string FileName
    {
        get { return fileName; }
        set
        {
            fileName = value;
            RaisedOnPropertyChanged("FileName");
        }
    }
    public ImageSource ImageIcon
    {
        get { return imageIcon; }
        set
        {
            imageIcon = value;
            RaisedOnPropertyChanged("ImageIcon");
        }
    }
}
```

```
public event PropertyChangedEventHandler PropertyChanged;
public void RaisedOnPropertyChanged(string _PropertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(_PropertyName));
    }
}

public class File : INotifyPropertyChanged
{
    private string fileName;
    private ImageSource imageIcon;
    private ObservableCollection<SubFile> subFiles;
    public File()
    {
    }
    public ObservableCollection<SubFile> SubFiles
    {
        get { return subFiles; }
        set
        {
            subFiles = value;
            RaisedOnPropertyChanged("SubFiles");
        }
    }
    public string FileName
    {
        get { return fileName; }
        set
        {
            fileName = value;
            RaisedOnPropertyChanged("FileName");
        }
    }
    public ImageSource ImageIcon
    {
        get { return imageIcon; }
        set
        {
            imageIcon = value;
            RaisedOnPropertyChanged("ImageIcon");
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    public void RaisedOnPropertyChanged(string _PropertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(_PropertyName));
        }
    }
}

public class SubFile : INotifyPropertyChanged
{
    private string fileName;
    private ImageSource imageIcon;
```

```

public SubFile()
{
}
public string FileName
{
    get { return fileName; }
    set
    {
        fileName = value;
        RaisedOnPropertyChanged("FolderName");
    }
}
public ImageSource ImageIcon
{
    get { return imageIcon; }
    set
    {
        imageIcon = value;
        RaisedOnPropertyChanged("ImageIcon");
    }
}
public event PropertyChangedEventHandler PropertyChanged;
public void RaisedOnPropertyChanged(string _PropertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(_PropertyName));
    }
}
}

```

C#

```

public class FileManagerViewModel
{
    public ObservableCollection<Folder> Folders { get; set; }
    public ObservableCollection<File> Files { get; set; }
    public ObservableCollection<SubFile> SubFiles { get; set; }
    public FileManagerViewModel()
    {
        this.Folders = GetFiles();
    }
    private ObservableCollection<Folder> GetFiles()
    {
        var nodeImageInfo = new ObservableCollection<Folder>();
        Assembly assembly = typeof(NodeWithImage).GetTypeInfo().Assembly;
        var doc = new Folder() { FileName = "Documents", ImageIcon =
        ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
        iew_folder.png", assembly) };
        var download = new Folder() { FileName = "Downloads", ImageIcon =
        ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
        iew_folder.png", assembly) };
        var mp3 = new Folder() { FileName = "Music", ImageIcon =
        ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
        iew_folder.png", assembly) };
    }
}

```

```
var pictures = new Folder() { FileName = "Pictures", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_folder.png", assembly) };
var video = new Folder() { FileName = "Videos", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_folder.png", assembly) };
var pollution = new File() { FileName = "Environmental Pollution.docx",
    ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_word.png", assembly) };
var globalWarming = new File() { FileName = "Global Warming.ppt", ImageIcon
    =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_ppt.png", assembly) };
var sanitation = new File() { FileName = "Sanitation.docx", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_word.png", assembly) };
var socialNetwork = new File() { FileName = "Social Network.pdf", ImageIcon
    =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_pdf.png", assembly) };
var youthEmpower = new File() { FileName = "Youth Empowerment.pdf",
    ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_pdf.png", assembly) };
var games = new File() { FileName = "Game.exe", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_exe.png", assembly) };
var tutorials = new File() { FileName = "Tutorials.zip", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_zip.png", assembly) };
var typeScript = new File() { FileName = "TypeScript.7z", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_zip.png", assembly) };
var uiGuide = new File() { FileName = "UI-Guide.pdf", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_pdf.png", assembly) };
var song = new File() { FileName = "Gouttes", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_mp3.png", assembly) };
var camera = new File() { FileName = "Camera Roll", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_folder.png", assembly) };
var stone = new File() { FileName = "Stone.jpg", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_png.png", assembly) };
var wind = new File() { FileName = "Wind.jpg", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_png.png", assembly) };
var img0 = new SubFile() { FileName = "WIN_20160726_094117.JPG", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_img0.png", assembly) };
var img1 = new SubFile() { FileName = "WIN_20160726_094118.JPG", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
    iew_img1.png", assembly) };
```

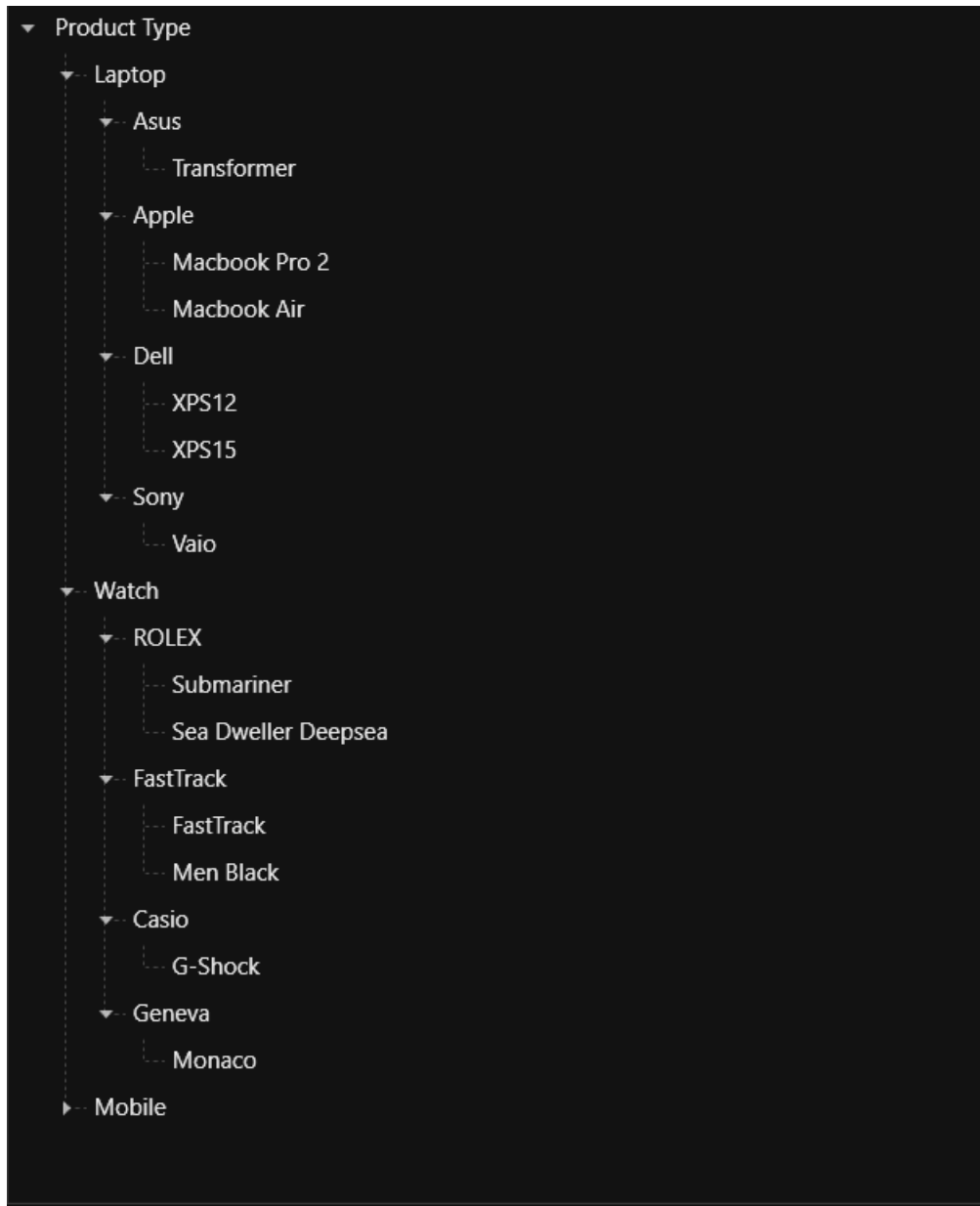
```
var video1 = new File() { FileName = "Naturals.mp4", ImageIcon =
ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
iew_video.png", assembly) };
var video2 = new File() { FileName = "Wild.mpeg", ImageIcon =
ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treev
iew_video.png", assembly) };
doc.Files = new ObservableCollection<File>
{
    pollution,
    globalWarming,
    sanitation,
    socialNetwork,
    youthEmpower
};
download.Files = new ObservableCollection<File>
{
    games,
    tutorials,
    typeScript,
    uiGuide
};
mp3.Files = new ObservableCollection<File>
{
    song
};
pictures.Files = new ObservableCollection<File>
{
    camera,
    stone,
    wind
};
camera.SubFiles = new ObservableCollection<SubFile>
{
    img0,
    img1
};
video.Files = new ObservableCollection<File>
{
    video1,
    video2
};
nodeImageInfo.Add(doc);
nodeImageInfo.Add(download);
nodeImageInfo.Add(mp3);
nodeImageInfo.Add(pictures);
nodeImageInfo.Add(video);
return nodeImageInfo;
}
```

Theme

WPF TreeView (SfTreeView) supports various built-in themes. Refer to the below links to apply themes for the SfTreeView,

- [Apply theme using SfSkinManager](#)

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



Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Data Population in WPF TreeView (SfTreeView)

TreeView can be populated either with the data source by using a [ItemsSource](#) property or by creating and adding the [TreeViewNode](#) in hierarchical structure to [Nodes](#) property.

Populating Nodes by data binding - Bound Mode

[Nodes](#) can be populated in bound mode includes following steps.

- Create hierarchical data model
- Bind data model to treeview

To update the collection changes in UI, it is necessary to define [NotificationSubscriptionMode](#) to Treeview as CollectionChanged /PropertyChanged.

NotificationSubscriptionMode enum has following members:

- CollectionChange - Updates its tree structure when child items collection gets changed.
- PropertyChange - Updates its ChildItems when associated collection property gets changed.
- None - It is a default mode and it doesn't reflect collection/property changes in UI.

To decide how to populate the nodes, it is necessary to set this **NodePopulationMode** API to Treeview.

The [NodePopulationMode](#) API has following enum values:

- OnDemand - Populate the child nodes only when parent nodes is expanded. It is the default value.
- Instant - Populates all the child nodes when Treeview control is initially loaded.

Create Data Model for treeview

Create a simple data source as shown in the following code example in a new class file, and save it as FileManager.cs file:

C#

```
//FileManager.cs
public class FileManager : INotifyPropertyChanged
{
    private string fileName;
    private ImageSource imageIcon;
    private ObservableCollection<FileManager> subFiles;
    public ObservableCollection<FileManager> SubFiles
    {
        get { return subFiles; }
        set
        {
            subFiles = value;
            RaisedOnPropertyChanged("SubFiles");
        }
    }
    public string ItemName
    {
        get { return fileName; }
        set
        {
            fileName = value;
            RaisedOnPropertyChanged("FolderName");
        }
    }
    public ImageSource ImageIcon
    {
        get { return imageIcon; }
        set
```

```
{
    imageIcon = value;
    RaisedOnPropertyChanged("ImageIcon");
}
}
public event PropertyChangedEventHandler PropertyChanged;
public void RaisedOnPropertyChanged(string _PropertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(_PropertyName));
    }
}
}
```

Create a model repository class with ImageNodeInfo collection property initialized with required number of data objects in a new class file as shown in the following code example, and save it as FileManagerViewModel.cs file:

C#

```
public class FileManagerViewModel
{
    private ObservableCollection<FileManager> imageNodeInfo;
    public FileManagerViewModel()
    {
        GenerateSource();
    }
    public ObservableCollection<FileManager> ImageNodeInfo
    {
        get { return imageNodeInfo; }
        set { this.imageNodeInfo = value; }
    }
    private void GenerateSource()
    {
        var nodeImageInfo = new ObservableCollection<FileManager>();
        Assembly assembly = typeof(GettingStated).GetTypeInfo().Assembly;
        var doc = new FileManager() { ItemName = "Documents", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var download = new FileManager() { ItemName = "Downloads", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var mp3 = new FileManager() { ItemName = "Music", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var pictures = new FileManager() { ItemName = "Pictures", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var video = new FileManager() { ItemName = "Videos", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
            assembly) };
        var pollution = new FileManager() { ItemName = "Environmental
            Pollution.docx", ImageIcon =
            ImageSource.FromResource("GettingStartedBound.Icons.treeview_word.png",
            assembly) };
    }
}
```



```
var globalWarming = new FileManager() { ItemName = "Global Warming.ppt",
ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_ppt.png",
assembly) };
var sanitation = new FileManager() { ItemName = "Sanitation.docx", ImageIcon
= ImageSource.FromResource("GettingStartedBound.Icons.treeview_word.png",
assembly) };
var socialNetwork = new FileManager() { ItemName = "Social Network.pdf",
ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_pdf.png",
assembly) };
var youthEmpower = new FileManager() { ItemName = "Youth Empowerment.pdf",
ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_pdf.png",
assembly) };
var games = new FileManager() { ItemName = "Game.exe", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_exe.png",
assembly) };
var tutorials = new FileManager() { ItemName = "Tutorials.zip", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_zip.png",
assembly) };
var TypeScript = new FileManager() { ItemName = "TypeScript.7z", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_zip.png",
assembly) };
var uiGuide = new FileManager() { ItemName = "UI-Guide.pdf", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_pdf.png",
assembly) };
var song = new FileManager() { ItemName = "Goutiest", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_mp3.png",
assembly) };
var camera = new FileManager() { ItemName = "Camera Roll", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_folder.png",
assembly) };
var stone = new FileManager() { ItemName = "Stone.jpg", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_png.png",
assembly) };
var wind = new FileManager() { ItemName = "Wind.jpg", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_png.png",
assembly) };
var img0 = new FileManager() { ItemName = "WIN_20160726_094117.JPG",
ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_img0.png",
assembly) };
var img1 = new FileManager() { ItemName = "WIN_20160726_094118.JPG",
ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_img1.png",
assembly) };
var video1 = new FileManager() { ItemName = "Naturals.mp4", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_video.png",
assembly) };
var video2 = new FileManager() { ItemName = "Wild.mpg", ImageIcon =
ImageSource.FromResource("GettingStartedBound.Icons.treeview_video.png",
assembly) };
doc.SubFiles = new ObservableCollection<FileManager>
{
pollution,
globalWarming,
```

```

sanitation,
socialNetwork,
youthEmpower
};
download.SubFiles = new ObservableCollection<FileManager>
{
games,
tutorials,
TypeScript,
uiGuide
};
mp3.SubFiles = new ObservableCollection<FileManager>
{
song
};
pictures.SubFiles = new ObservableCollection<FileManager>
{
camera,
stone,
wind
};
camera.SubFiles = new ObservableCollection<FileManager>
{
img0,
img1
};
video.SubFiles = new ObservableCollection<FileManager>
{
video1,
video2
};
nodeImageInfo.Add(doc);
nodeImageInfo.Add(download);
nodeImageInfo.Add(mp3);
nodeImageInfo.Add(pictures);
nodeImageInfo.Add(video);
imageNodeInfo = nodeImageInfo;
}
}

```

Bind to hierarchical datasource

To create a tree view using data binding, set a hierarchical data collection to the [ItemsSource](#) property. And set the child object name to the [ChildPropertyName](#) property.

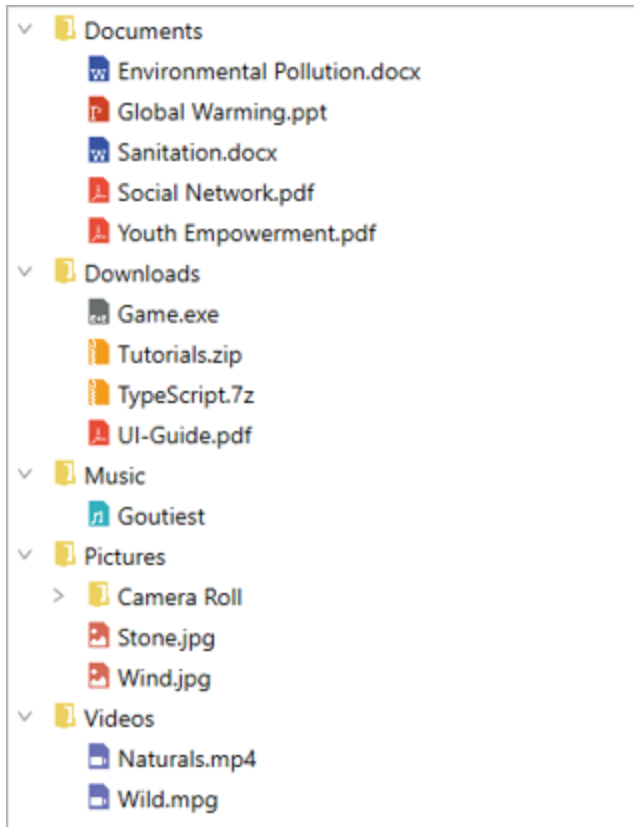
XML

```

<Window x:Class="NodeWithImageDemo.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:NodeWithImageDemo"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:Syncfusion="http://schemas.syncfusion.com/wpf" xmlns:Engine="clr-
namespace:Syncfusion.UI.Xaml.TreeView.Engine;assembly=Syncfusion.SfTreeView.
WPF"

```

```
mc:Ignorable="d">
<Window.DataContext>
<local:FileManagerViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:SfTreeView x:Name="sfTreeView"
ChildPropertyName="SubFiles"
ItemsSource="{Binding ImageNodeInfo}">
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<Grid x:Name="grid">
<Grid Grid.Row="0">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="20" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Grid>
<Image Source="{Binding ImageIcon}"
VerticalAlignment="Center"
HorizontalAlignment="Center"
Height="16"
Width="16"/>
</Grid>
<Grid Grid.Column="1"
Margin="1,0,0,0"
VerticalAlignment="Center">
<TextBlock Text="{Binding ItemName}"
Foreground="Black"
FontSize="14"
VerticalAlignment="Center"
</Grid>
</Grid>
</Grid>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>
</Grid>
</Window>
```



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

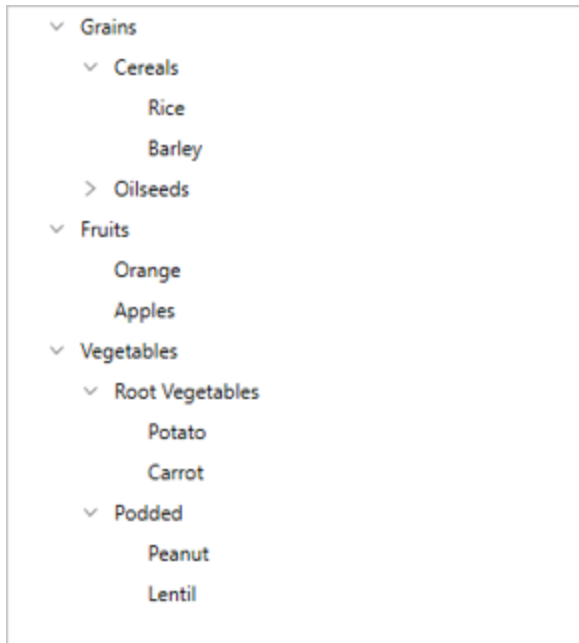
Populating Nodes without data source - Unbound Mode

You can create and manage the [TreeViewNode](#) objects by yourself to display the data in a hierarchical view. To create a tree view, you use a [SfTreeView](#) control and a hierarchy of [TreeViewNode](#) objects. You create the node hierarchy by adding one or more root nodes to the [SfTreeView.Nodes](#) collection. Each [TreeViewNode](#) can have more nodes added to its Children collection. You can nest tree view nodes to whatever depth you require.

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:GettingStarted"
xmlns:Syncfusion="http://schemas.syncfusion.com/wpf" xmlns:Engine="clr-
namespace:Syncfusion.UI.Xaml.TreeView.Engine;assembly=Syncfusion.SfTreeView.
WPF" x:Class="GettingStarted.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<Syncfusion:SfTreeView HorizontalAlignment="Left" Height="414"
Margin="318,0,0,0" VerticalAlignment="Center" Width="250">
<Syncfusion:SfTreeView.Nodes>
<Engine:TreeViewNode Content="Grains" IsExpanded="True">
<Engine:TreeViewNode.ChildNodes>
```

```
<Engine:TreeViewNode Content="Cereals" IsExpanded="True">
  <Engine:TreeViewNode.ChildNodes>
    <Engine:TreeViewNode Content="Rice"/>
    <Engine:TreeViewNode Content="Barley"/>
  </Engine:TreeViewNode.ChildNodes>
</Engine:TreeViewNode>
<Engine:TreeViewNode Content="Oilseeds">
  <Engine:TreeViewNode.ChildNodes>
    <Engine:TreeViewNode Content="Safflower"/>
  </Engine:TreeViewNode.ChildNodes>
</Engine:TreeViewNode>
<Engine:TreeViewNode Content="Fruits" IsExpanded="true">
  <Engine:TreeViewNode.ChildNodes>
    <Engine:TreeViewNode Content="Orange"/>
    <Engine:TreeViewNode Content="Apples" IsExpanded="true"/>
  </Engine:TreeViewNode.ChildNodes>
</Engine:TreeViewNode>
<Engine:TreeViewNode Content="Vegetables" IsExpanded="true">
  <Engine:TreeViewNode.ChildNodes>
    <Engine:TreeViewNode Content="Root Vegetables" IsExpanded="true">
      <Engine:TreeViewNode.ChildNodes>
        <Engine:TreeViewNode Content="Potato"/>
        <Engine:TreeViewNode Content="Carrot"/>
      </Engine:TreeViewNode.ChildNodes>
    </Engine:TreeViewNode>
    <Engine:TreeViewNode Content="Podded">
      <Engine:TreeViewNode.ChildNodes>
        <Engine:TreeViewNode Content="Peanut"/>
        <Engine:TreeViewNode Content="Lentil"/>
      </Engine:TreeViewNode.ChildNodes>
    </Engine:TreeViewNode>
  </Engine:TreeViewNode.ChildNodes>
</Engine:TreeViewNode>
</Syncfusion:SfTreeView.Nodes>
</Syncfusion:SfTreeView>
</Grid>
</Window>
```



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

Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Appearance in WPF TreeView (SfTreeView)

The TreeView allows customizing appearance of the underlying data, and provides different functionalities to the end-user.

ItemTemplate

The [WPF TreeView](#) allows you to customize the appearance of content view and expander view by setting the [ItemTemplate](#) and [ExpanderTemplate](#) properties.

XML

```
<Window x:Class="NodeWithImageDemo.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:NodeWithImageDemo"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d">
<Window.DataContext>
<local:FileManagerViewModel/>
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition/>
</Grid.RowDefinitions>
<syncfusion:SfTreeView x:Name="sfTreeView" Grid.Row="1"
ChildPropertyName="SubFiles"
FullRowSelect="True"
```

```

ItemsSource="{Binding ImageNodeInfo}" >
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<Grid x:Name="grid">
<Grid Grid.Row="0">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="20" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Grid>
<Image Source="{Binding ImageIcon}"
VerticalAlignment="Center"
HorizontalAlignment="Center"
Height="16"
Width="16"/>
</Grid>
<Grid Grid.Column="1"
Margin="1,0,0,0"
VerticalAlignment="Center">
<Label Content="{Binding ItemName}"
Foreground="Black"
FontSize="11"
VerticalContentAlignment="Center" />
</Grid>
</Grid>
</Grid>
</Grid>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>
</Grid>
</Window>

```

BindingContext for ItemTemplate

By default, the binding context of tree view item will be the data model object for Bound Mode and [TreeNode](#) for Unbound Mode.

For Bound Mode, you can change the binding context of the treeview items by using [ItemTemplateContextType](#) property.

XML

```

<Window x:Class="NodeWithImageDemo.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:NodeWithImageDemo"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d">
<Window.DataContext>
<local:FileManagerViewModel/>
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition/>

```

```

</Grid.RowDefinitions>
<TextBlock VerticalAlignment="Center"
TextWrapping="Wrap"
Padding="5,0,0,0"
FontSize="14"
Margin="5,5,5,20">
<Run Text="This sample demonstrates the default functionalities to include
images in SfTreeView."/>
</TextBlock>
<syncfusion:SfTreeView x:Name="sfTreeView" Grid.Row="1"
ChildPropertyName="SubFiles"
FullRowSelect="True"
ItemTemplateDataContextType="Node"
ItemsSource="{Binding ImageNodeInfo}" >
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<Grid x:Name="grid">
<Grid Grid.Row="0">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="20" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Grid>
<Image Source="{Binding Content.ImageIcon}"
VerticalAlignment="Center"
HorizontalAlignment="Center"
Height="16"
Width="16"/>
</Grid>
<Grid Grid.Column="1"
Margin="1,0,0,0"
VerticalAlignment="Center">
<Label Content="{Binding Content.ItemName}"
Foreground="Black"
FontSize="11"
VerticalContentAlignment="Center" />
</Grid>
</Grid>
</Grid>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>
</Grid>
</Window>

```

ItemTemplate Selector

The TreeView allows you to customize the appearance of each item with different templates based on specific constraints by using the [ItemTemplateSelector](#). You can choose a [DataTemplate](#) for each item at runtime based on the value of data-bound property using `ItemTemplateSelector`.

XML

```

<Window x:Class="NodeWithImageDemo.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"

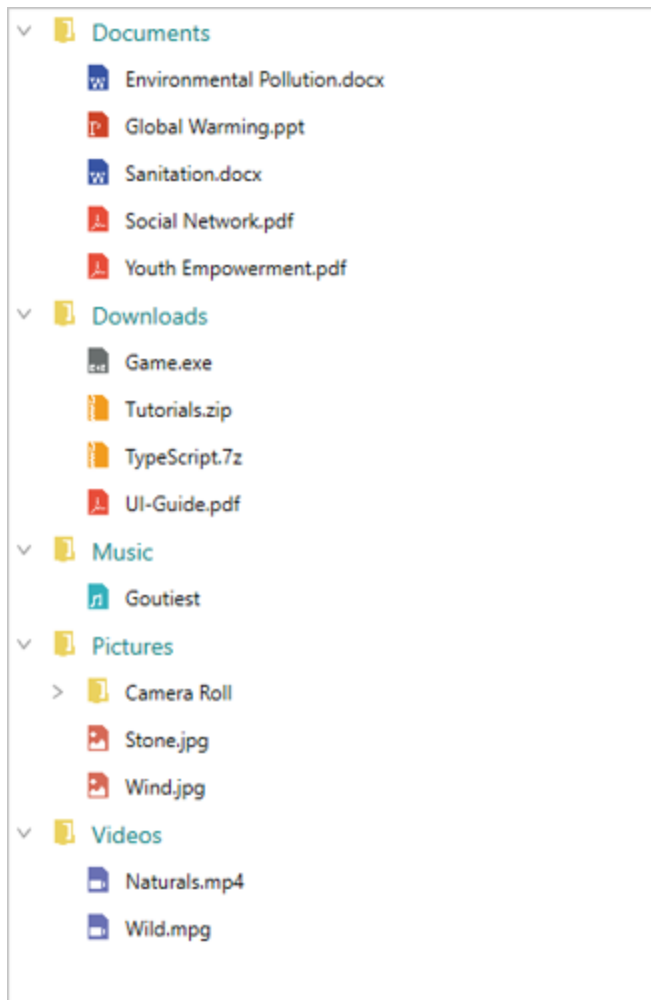
```



```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:NodeWithImageDemo"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d">
<Window.DataContext>
<local:FileManagerViewModel/>
</Window.DataContext>
<Window.Resources>
<local:ItemTemplateSelector x:Key="itemTemplateSelector"/>
</Window.Resources>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition/>
</Grid.RowDefinitions>
<syncfusion:SfTreeView x:Name="sfTreeView"
Grid.Row="1"
ChildPropertyName="SubFiles"
FullRowSelect="True"
ItemTemplateDataContextType="Node"
ItemsSource="{Binding ImageNodeInfo}"
ItemTemplateSelector="{StaticResource itemTemplateSelector}" >
</syncfusion:SfTreeView>
</Grid>
</Window>
```

C#

```
class ItemTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        {
            var treeviewNode = item as TreeViewNode;
            if (treeviewNode == null)
                return null;
            if (treeviewNode.Level == 0)
                return Application.Current.MainWindow.FindResource("RootTemplate") as DataTemplate;
            else
                return Application.Current.MainWindow.FindResource("ChildTemplate") as DataTemplate;
        }
    }
}
```



Note: View sample in [GitHub](#)

Indentation

The TreeView allows customizing the indent spacing of items by setting the [Indentation](#) property. The default value of this property is 20. This property can be customized at runtime.

XML

```
<Syncfusion:SfTreeView x:Name="sfTreeView" Indentation="40" />
```

C#

```
SfTreeView sfTreeView = new SfTreeView();  
sfTreeView.Indentation = 40;
```

ExpanderWidth

The TreeView allows customizing the width of expander view by setting the [ExpanderWidth](#) property. The default value of this property is 20. This property can be customized at runtime.

XML

```
<Syncfusion:SfTreeView x:Name="sfTreeView" ExpanderWidth="40" />
```

C#

```
SfTreeView sfTreeView = new SfTreeView();
sfTreeView.ExpanderWidth = "40";
```

ExpanderPosition

The TreeView allows you change the position of expander view by setting the [ExpanderPosition](#) property. The default value of this property is **Start**. This property has following two positions:

Start: Allows displaying the expander view at the start position.

End: Allows displaying the expander view at the end position.

XML

```
<Syncfusion:SfTreeView x:Name="sfTreeView" ExpanderPosition="End">
```

C#

```
SfTreeView sfTreeView = new SfTreeView();
sfTreeView.ExpanderPosition = ExpanderPosition.End;
```

Level based styling

The TreeView allows you to customize the style of [TreeViewItem](#) based on different levels by using [IValueConverter](#).

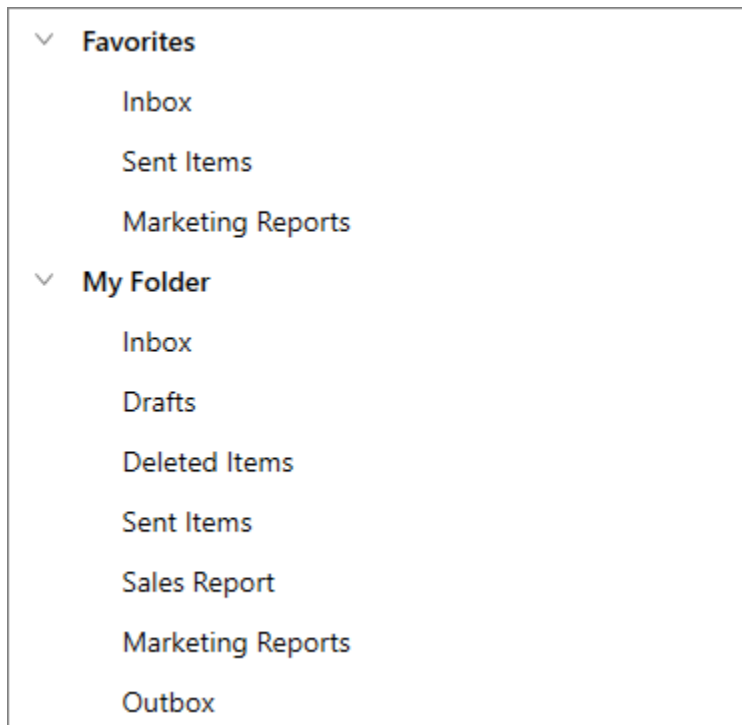
XML

```
<Window.Resources>
<local:FontAttributeConverter x:Key="FontAttributeConverter"/>
</Window.Resources>
<Window.DataContext>
<local:MailFolderViewModel x:Name="viewModel"/>
</Window.DataContext>
<Grid>
<Syncfusion:SfTreeView HorizontalAlignment="Left"
ItemTemplateDataContextType="Node" ItemsSource="{Binding Folders}"
ChildPropertyName="SubFolder" >
<Syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<Label Content="{Binding Content.FolderName}" FontWeight="{Binding Level,
Converter={StaticResource FontAttributeConverter}}"
FontSize="14"/>
</DataTemplate>
</Syncfusion:SfTreeView.ItemTemplate>
</Syncfusion:SfTreeView>
</Grid>
```

C#

```
public class FontAttributeConverter : IValueConverter
{
```

```
public object Convert(object value, Type targetType, object parameter,
    CultureInfo culture)
{
    var level = (int)value;
    return level == 0 ? FontWeights.Bold : FontWeights.Regular;
}
public object ConvertBack(object value, Type targetType, object parameter,
    CultureInfo culture)
{
    throw new NotImplementedException();
}
}
```



Note: View sample in [GitHub](#)

Animation

The SfTreeView supports to animate expanding or collapsing the [TreeViewNode](#). To enable/disable the animation use [IsAnimationEnabled](#) property of SfTreeView.

XML

```
<Syncfusion:SfTreeView x:Name="sfTreeView" IsAnimationEnabled="true">
```

C#

```
SfTreeView sfTreeView = new SfTreeView();
sfTreeView.IsAnimationEnabled = true;
```



Note: You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

[Expand and Collapse in WPF TreeView \(SfTreeView\)](#)

The [WPF TreeView](#) allows you to expand and collapse the nodes either by user interaction on the nodes or by programmatically.

[Expand Action Trigger](#)

Expanding and Collapsing of nodes can be performed either by tapping the expander view or in both expander view and content view by setting the [ExpandActionTrigger](#) property.

XML

```
<syncfusion:SfTreeView x:Name="sfTreeView" ExpandActionTrigger="Node" />
```

C#

```
// Expands by tapping both expander view and content view.
sfTreeView.ExpandActionTrigger = ExpandActionTrigger.Node;
```

Auto Expand Mode

By default, the treeview items will be in collapsed state. You can define how the nodes to be expanded while loading the TreeView by using [AutoExpandMode](#) property.

The [AutoExpandMode](#) property is only applicable for bound mode. For Unbound mode you need to set [IsExpanded](#) property to [true](#) while creating the nodes, to be in expanded state while loading the TreeView.

- None : All items are collapsed when loaded.
- RootNodes : Expands only the root item when loaded.
- AllNodes : Expands all the items when loaded.

Expand or collapse the nodes based on property of underlying data object

You can bind expand state of node to the bool property in underlying data object by using [IsExpandedPropertyName](#) property. TreeView updates the expanded of node when underlying data object property gets changed and vice versa.

XML

```
xmlns:treeviewengine="clr-
namespace:Syncfusion.UI.Xaml.TreeView.Engine;assembly=Syncfusion.SfTreeView.
WPF"
<syncfusion:SfTreeView x:Name="treeView"
ItemsSource="{Binding Folders}">
<syncfusion:SfTreeView.HierarchyPropertyDescriptors>
<treeviewengine:HierarchyPropertyDescriptor
IsExpandedPropertyName="IsExpanded" ChildPropertyName="SubFiles"
TargetType="{x:Type local:FileManager}" />
</syncfusion:SfTreeView.HierarchyPropertyDescriptors>
</syncfusion:SfTreeView>
```

C#

```
public class FileManager : INotifyPropertyChanged
{
    private string fileName;
    private ObservableCollection<FileManager> subFiles;
    private bool isExpanded;
    public ObservableCollection<FileManager> SubFiles
    {
        get { return subFiles; }
        set
        {
            subFiles = value;
            RaisedOnPropertyChanged("SubFiles");
        }
    }
    public string ItemName
```

```

{
    get { return fileName; }
    set
    {
        fileName = value;
        RaisedOnPropertyChanged("ItemName");
    }
}

public bool IsExpanded
{
    get { return isExpanded; }
    set
    {
        isExpanded = value;
        RaisedOnPropertyChanged("IsExpanded");
    }
}

public event PropertyChangedEventHandler PropertyChanged;
public void RaisedOnPropertyChanged(string _PropertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(_PropertyName));
    }
}

public class FileManagerViewModel
{
    private ObservableCollection<FileManager> folders;
    public FileManagerViewModel()
    {
        GenerateSource();
    }
    public ObservableCollection<FileManager> Folders
    {
        get { return folders; }
        set { this.folders = value; }
    }
    private void GenerateSource()
    {
        var fileManager = new ObservableCollection<FileManager>();
        var doc = new FileManager() { ItemName = "Documents", IsExpanded = true };
        var download = new FileManager() { ItemName = "Downloads", IsExpanded = false };
        var pollution = new FileManager() { ItemName = "Environmental Pollution.docx" };
        var globalWarming = new FileManager() { ItemName = "Global Warming.ppt" };
        var sanitation = new FileManager() { ItemName = "Sanitation.docx" };
        var socialNetwork = new FileManager() { ItemName = "Social Network.pdf" };
        var youthEmpower = new FileManager() { ItemName = "Youth Empowerment.pdf" };
        var games = new FileManager() { ItemName = "Game.exe" };
        var tutorials = new FileManager() { ItemName = "Tutorials.zip" };
        var TypeScript = new FileManager() { ItemName = "TypeScript.7z" };
        var uiGuide = new FileManager() { ItemName = "UI-Guide.pdf" };
        doc.SubFiles = new ObservableCollection<FileManager>
        {
            pollution,

```

```
globalWarming,
sanitation,
socialNetwork,
youthEmpower
};
download.SubFiles = new ObservableCollection<FileManager>
{
games,
tutorials,
TypeScript,
uiGuide
};
fileManager.Add(doc);
fileManager.Add(download);
folders = fileManager;
}
}
```

Note: `IsExpandedPropertyName` property is not supported for unbound mode and it accepts only boolean type property.

Programmatic Expand and Collapse

TreeView allows programmatic expand and collapse based on the [TreeViewNode](#) and level by using following methods.

- [ExpandNode\(TreeViewNode item\)](#) - Method to expand the particular `TreeViewNode` passed to it.
- [CollapseNode\(TreeViewNode item\)](#) - Method to collapse the particular `TreeViewNode` passed to it.
- [ExpandNodes\(int level\)](#) - Method to expand the all items of level passed to it.
- [CollapseNodes\(int level\)](#) - Method to collapse the all items of level passed to it.

C#

```
// Expands all the nodes of root level '0'
sfTreeView.ExpandNodes(0);
// Collapses all the nodes of root level '0'
sfTreeView.CollapseNodes(0);
// Expand a particular node.
sfTreeView.ExpandNode(node);
// Collapse a particular node.
sfTreeView.CollapseNode(node);
```

Expand and Collapse all the nodes

Expand and Collapse all the [TreeViewNode](#) programmatically at runtime by using the `SfTreeView.ExpandAll` method and `SfTreeView.CollapseAll` method.

C#

```
//Expands all the nodes
sfTreeView.ExpandAll();
//Collapses all the nodes
sfTreeView.CollapseAll();
```


Expand and Collapse using Keyboard

TreeView allows to expand and collapse the nodes by using right and left arrows keys. To expand a node, press the right arrow key and to collapse a node, press the left arrow key on the focused item.

Events

TreeView exposes following events to handle expanding and collapsing of items.

- [NodeCollapsing](#) - It occurs when a node is being collapsed.
- [NodeExpanding](#) - It occurs when a node is being expanded.
- [NodeCollapsed](#) - It occurs when a node is collapsed.
- [NodeExpanded](#) - It occurs when a node is expanded.

The expanding and collapsing interactions can be handled with the help of `NodeCollapsing` and `NodeExpanding` events and expanded and collapsed interactions can be handled with help of `NodeCollapsed` and `NodeExpanded` events.

Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Interactivity in WPF TreeView (SfTreeView)

This section explains about how to interact with `TreeView` and its items.

Interacting with TreeView items

ItemTapped event

The [ItemTapped](#) event will be triggered whenever tapping the item. [ItemTappedEventArgs](#) has the following members which provides the information for `ItemTapped` event:

- **Node:** Gets the [TreeViewNode](#) and data associated with the tapped item as its arguments.
- **Position:** Gets the touch position in the tapped item.
- **Handled:** Gets or sets whether the event is handled or not.

C#

```
sfTreeView.ItemTapped += SfTreeView_ItemTapped;
private void SfTreeView_ItemTapped(object sender, ItemTappedEventArgs e)
{
    MessageBox.Show("Tapped Item: " + (e.Node.Content as Model).State);
}
```

ItemDoubleTapped event

The [ItemDoubleTapped](#) event will be triggered whenever double tapping the item. The [ItemDoubleTappedEventArgs](#) has the following members providing information for the `ItemDoubleTapped` event:

- **Node:** Gets the [TreeViewNode](#) and data associated with the double tapped item as its arguments.

- **Position**: Gets the touch position in the double tapped item.
- **Handled**: Gets or sets whether the event is handled or not.

C#

```
sfTreeView.ItemDoubleTapped += SfTreeView_ItemDoubleTapped;
private void SfTreeView_ItemDoubleTapped(object sender,
ItemDoubleTappedEventArgs e)
{
    MessageBox.Show("DoubleTapped Item: " + (e.Node.Content as Model).State);
}
```

ItemHolding event

The [ItemHolding](#) event will be triggered whenever the item is long pressed.

[ItemHoldingEventArgs](#) has the following members which provides the information for **ItemHolding** event:

- **Node**: Gets the [TreeViewNode](#) and data associated with the hold item as its arguments.
- **Position**: Gets the touch position in the hold item.

C#

```
sfTreeView.ItemHolding += SfTreeView_ItemHolding;
private void SfTreeView_ItemHolding(object sender, ItemHoldingEventArgs e)
{
    MessageBox.Show("HoldItem: " + (e.Node.Content as Model).State);
}
```

Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Scrolling in WPF TreeView (SfTreeView)

The TreeView provides various options to achieve programmatic scrolling. Please walk through the below section in detail to achieve the same.

Bring Into View

The TreeView allows programmatic scrolling based on the data model and [TreeViewNode](#) by using the [BringIntoView](#) method.

C#

```
private void BringIntoView_Click(object sender, RoutedEventArgs e)
{
    var count = viewModel.Items.Count;
    var data = viewModel.Items[count - 1];
    sfTreeView.BringIntoView(data);
}
```

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

The `BringIntoView` method comprises of other optional parameters to decide on the way in which the child item should come into view.

Scroll to the child item with animation

The second optional parameter `disableAnimation` in `BringIntoView` method decides whether the scrolling animation should be enabled or disabled when the child item comes into view. By default, the scrolling will be animated.

- If the parameter value is `true`, scrolling animation will be disabled.
- If the parameter value is `false`, scrolling animation will be enabled.

C#

```
private void BringIntoView_Click(object sender, RoutedEventArgs e)
{
    var count = viewModel.Items.Count;
    var data = viewModel.Items[count - 1];
    // Here, the second optional parameter has been passed as true hence it will
    // disable the animation
    sfTreeView.BringIntoView(data, true);
}
```

Scroll to the collapsed child item

The third optional parameter `canExpand` in `BringIntoView` method decides whether we need to expand and show the collapsed node or not when item passed for `BringIntoView` method which is in collapsed state. By default, this parameter value will be `false`.

- If the parameter value is `true`, TreeView expands the collapsed node if it is collapsed and scroll to the specified item.
- If the parameter value is `false`, TreeView does not expand the collapsed node and only scroll for item which is not in collapsed state.

C#

```
private void BringIntoView_Click(object sender, RoutedEventArgs e)
{
    var count = viewModel.Items.Count;
    var data = viewModel.Items[count - 1];
    sfTreeView.BringIntoView(data, false, true);
}
```

Note: We need to set the [NodePopulationMode](#) API value as `TreeNodePopulationMode.Instant` for scrolling to the collapsed item in addition to the additional parameter passed to the `BringIntoView` method.

Scroll the item into specified position

The fourth optional parameter `scrollToPosition` in `BringIntoView` method allows to position the scrolled item in the view. The scrolled item can take either of the four positions as explained below. The default position is `Start`.

- **Start:** Scroll to make the node positioned at the start of the view.
- **MakeVisible:** Scroll to make a specified node visible in the view. If the specified node is already in view, scrolling will not occur.
- **Center:** Scroll to make the node positioned at the center of the view.
- **End:** Scroll to make the node positioned at the end of the view.

C#

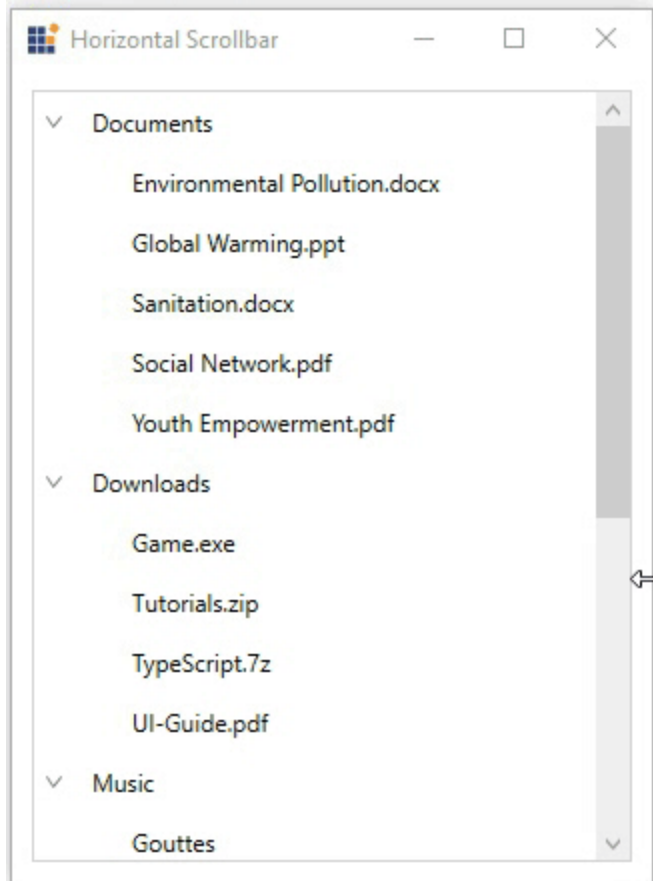
```
private void BringIntoView_Click(object sender, RoutedEventArgs e)
{
    var count = viewModel.Items.Count;
    var data = viewModel.Items[count - 1];
    // Scrolls to the data item to make visible in the view.
    sfTreeView.BringIntoView(data, false, true, ScrollToPosition.MakeVisible);
}
```

Horizontal scrolling

By default, horizontal scrollbar is not enabled in the TreeView. If you want to enable horizontal scrolling based on the content, you should set [ScrollViewer.HorizontalScrollBarVisibility](#) as **Auto**.

XML

```
<syncfusion:SfTreeView
x:Name="treeView"
ScrollViewer.HorizontalScrollBarVisibility="Auto"
AutoExpandMode="AllNodes"
IsAnimationEnabled="True"
ItemsSource="{Binding Countries}"/>
```



Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Editing in WPF TreeView (SfTreeView)

The TreeView provides support for editing and it can be enabled or disabled by using [SfTreeView.AllowEditing](#) property. You can enter edit mode in a node by pressing F2 key and also by single click or double click by setting [EditTrigger](#) property. The editing changes in a node will be committed only when user move to next node or pressing Enter key.

It is necessary to define [EditTemplate](#) / [EditTemplateSelector](#) for bound mode, to enable editing. For UnboundMode, textbox will be loaded in edit mode by default.

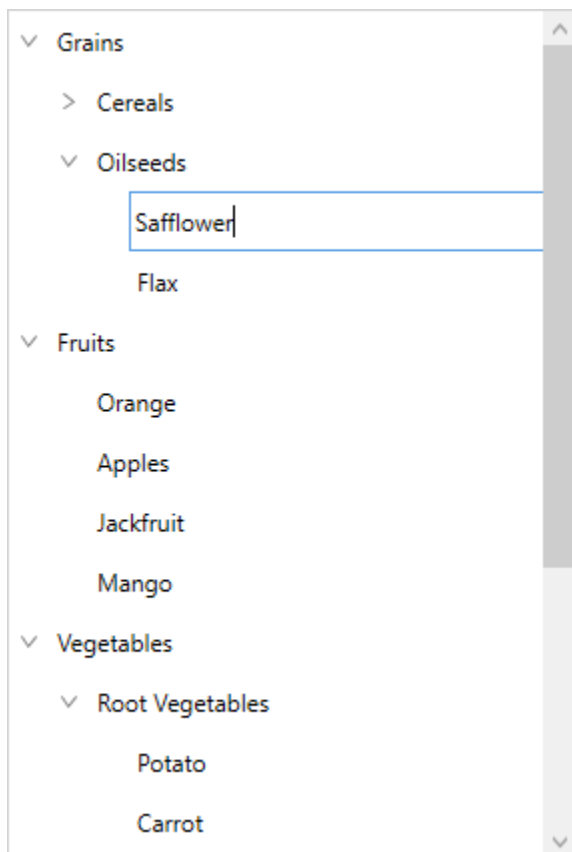
XML

```
<syncfusion:SfTreeView x:Name="sfTreeView"
    ItemsSource="{Binding Items}"
    ChildPropertyName="Files"
    AutoExpandMode="RootNodes"
    AllowEditing="True"
>
    <syncfusion:SfTreeView.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Name}" VerticalAlignment="Center"/>
        </DataTemplate>
    </syncfusion:SfTreeView.ItemTemplate>
```

```
<syncfusion:SfTreeView.EditTemplate>
<DataTemplate>
<TextBox Text="{Binding Name}"
VerticalContentAlignment="Center"
Margin="-4,0,-4,0"
Height="{Binding ItemHeight,ElementName=sfTreeView}" />
</DataTemplate>
</syncfusion:SfTreeView.EditTemplate>
</syncfusion:SfTreeView>
```

C#

```
sfTreeView.AllowEditing = true;
```



Edit mode

By default, you can move to edit mode by pressing F2 key. TreeView allows you to edit the node in single click(Tap) or double click(DoubleTap) by setting [EditTrigger](#) property.

XML

```
<syncfusion:SfTreeView x:Name="sfTreeView"
ItemsSource="{Binding Countries}"
AutoExpandMode="RootNodes"
EditTrigger="DoubleTap"
AllowEditing="True"/>
```

C#

```
this.sfTreeView.EditTrigger =  
Syncfusion.UI.Xaml.TreeView.TreeViewEditTrigger.DoubleTap;
```

Programmatic Editing

Begin the editing

The TreeView allows you to edit the node programmatically by calling the [BeginEdit](#) method.

C#

```
this.sfTreeView.Loaded += TreeView_Loaded;  
private void TreeView_Loaded(object sender, RoutedEventArgs e)  
{  
    this.sfTreeView.BeginEdit(this.sfTreeView.Nodes[0]);  
}
```

Note: [CurrentItem](#) is set to the node when the BeginEdit is called.

End the editing

You can call [EndEdit](#) method to programmatically end the editing for specific node.

C#

```
this.sfTreeView.Loaded += TreeView_Loaded;  
private void TreeView_Loaded(object sender, RoutedEventArgs e)  
{  
    this.sfTreeView.EndEdit(this.sfTreeView.Nodes[0]);  
}
```

Revert the edited changes while pressing Escape key

By default, TreeView does not have support for rollback the changes when pressing the ESC key while editing the TreeView node. But it supports to rollback the changes when an underlying data object implements the [IEditableObject](#) interface.

The user can take a backup of existing data of a node in the [BeginEdit](#) method and can change the existing data to the current data in the [CancelEdit](#) method to rollback the changes.

The below code snippet explains the simple implementation of IEditableObject interface to rollback the changes.

C#

```
public class Country : INotifyPropertyChanged, IEditableObject  
{  
    private bool isSelected;  
    internal string name;  
    private ObservableCollection<State> states;  
    internal Country backUpData;  
    private Country currentData;  
    public Country()  
    {  
    }  
    public Country(string name):base()  
    {  
    }  
}
```

```
this.currentData = new Country();
this.currentData.name = name;
this.currentData.isSelected = false;
}
public ObservableCollection<State> States
{
    get
    {
        return states;
    }
    set
    {
        states = value;
        RaisedOnPropertyChanged("States");
    }
}
public string Name
{
    get
    {
        return this.currentData.name;
    }
    set
    {
        this.currentData.name = value;
        RaisedOnPropertyChanged("Name");
    }
}
public bool IsSelected
{
    get
    {
        return this.currentData.isSelected;
    }
    set
    {
        this.currentData.isSelected = value;
        RaisedOnPropertyChanged("IsSelected");
    }
}
public event PropertyChangedEventHandler PropertyChanged;
public void RaisedOnPropertyChanged(string _PropertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(_PropertyName));
    }
}
public void BeginEdit()
{
    Debug.WriteLine("BeginEdit is Called.");
    backUpData = new Country();
    backUpData.name = this.currentData.name;
    backUpData.isSelected = this.currentData.isSelected;
}
public void CancelEdit()
{

```



```
Debug.WriteLine("CancelEdit is Called.");  
this.currentData = backUpData;  
}  
public void EndEdit()  
{  
    Debug.WriteLine("EndEdit is Called.");  
}  
}
```

Note: View sample in [GitHub](#)

Events

ItemBeginEdit Event

The [ItemBeginEdit](#) event occurs when the node enters edit mode. The [TreeViewItemBeginEditEventArgs](#) has the following members which provides information about the [ItemBeginEdit](#) event.

- [Node](#) : Gets the [TreeViewNode](#) which is being edited.

You can cancel the editing of certain nodes using custom logic within this event by setting [TreeViewItemBeginEditEventArgs.Cancel](#) as true.

C#

```
sfTreeView.ItemBeginEdit += TreeView_ItemBeginEdit;  
private void TreeView_ItemBeginEdit(object sender,  
TreeViewItemBeginEditEventArgs e)  
{  
    if (e.Node.Content == "Grains")  
        e.Cancel = true;  
}
```

ItemEndEdit Event

The [ItemEndEdit](#) event occurs when the node leaves the edit mode. The [TreeViewItemEndEditEventArgs](#) has the following members which provides information about the [ItemEndEdit](#) event.

- [Node](#) : Gets the [TreeViewNode](#) which is being edited.

You can cancel the editing from being ended for certain nodes using custom logic within this event by setting [TreeViewItemEndEditEventArgs.Cancel](#) as true.

C#

```
sfTreeView.ItemEndEdit += TreeView_ItemEndEdit;  
private void TreeView_ItemEndEdit(object sender,  
TreeViewItemEndEditEventArgs e)  
{  
    if (e.Node.Content == "Cereals")  
        e.Cancel = true;  
}
```

Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

CRUD Operations in WPF TreeView (SfTreeView)

TreeView listens and responds to the CRUD operations such as add, delete and data update (property change) at runtime. Also, it supports [editing](#), delete by pressing Delete key.

Add nodes

TreeView allows user to add new node directly by adding new data object to underlying collection in bound mode and by adding [TreeViewNode](#) to [Nodes](#) collection in unbound mode.

C#

```
// For bound mode
(sfTreeView.DataContext as ViewModel).Countries.Add(new Country() { Name =
"Germany" });
// For Unbound mode
sfTreeView.Nodes.Add(new TreeViewNode() { Content = "Germany" });
```

Delete nodes

TreeView provides built-in support to delete the selected nodes in user interface (UI) by pressing Delete key. You can enable the deleting support by setting the [SfTreeView.AllowDeleting](#) property to `true`.

XML

```
<syncfusion:SfTreeView
x:Name="sfTreeView"
AllowDeleting="True"
ChildPropertyName="States"
ItemsSource="{Binding Countries}"/>
```

C#

```
sfTreeView.AllowDeleting = true;
```

You can delete node directly in underlying collection also using `Remove ()` or `RemoveAt (int index)`.

C#

```
//For Bound mode
(sfTreeView.DataContext as
ViewModel).Countries.Remove(sfTreeView.SelectedItem as Country);
// OR
(sfTreeView.DataContext as ViewModel).Orders.RemoveAt(2);
// For Unbound mode
sfTreeView.Nodes.Remove(sfTreeView.Nodes[0]);
//OR
sfTreeView.Nodes.RemoveAt(2);
```

Event customization

Delete selected nodes conditionally

You can cancel the node deletion by using the `ItemDeletingEventArgs.Cancel` of [ItemDeleting](#) event. This event occurs when the node is being deleted using Delete key. You can skip certain nodes when deleting more than one node by removing items from [ItemDeletingEventArgs.Nodes](#).

C#

```
sfTreeView.ItemDeleting += TreeView_ItemDeleting;
private void TreeView_ItemDeleting (object sender, ItemDeletingEventArgs e)
{
    var nodeCollection = e.Nodes.ToList();
    foreach (var node in nodeCollection)
    {
        var country = node.Content as Country;
        if (country != null)
        {
            if (country.Name == "Brazil")
            {
                e.Cancel = true;
            }
            else if (country.Name == "India")
            {
                e.Nodes.Remove(node);
            }
        }
    }
}
```

Reset selection after deleting the selected node

You can handle the selection after remove the nodes through [SfTreeView.SelectedItem](#) property in [ItemDeleted](#) event. This event occurs after the node is deleted using Delete key.

C#

```
sfTreeView.ItemDeleted += TreeView_ItemDeleted;
private void TreeView_ItemDeleted (object sender, ItemDeletedEventArgs e)
{
    if (sfTreeView.Nodes.Count > 0)
    {
        sfTreeView.SelectedItem = sfTreeView.Nodes[0].Content;
    }
}
```

Modify nodes

Treeview allows user to modify the data in a node by [editing](#).

Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Selection in WPF TreeView (SfTreeView)

This section explains how to perform selection and its related operations in the TreeView.

UI Selection

The TreeView allows selecting the items either programmatically or touch interactions by setting the [SelectionMode](#) property value to other than **None**. The control has different selection modes to perform selection operations as listed as follows.

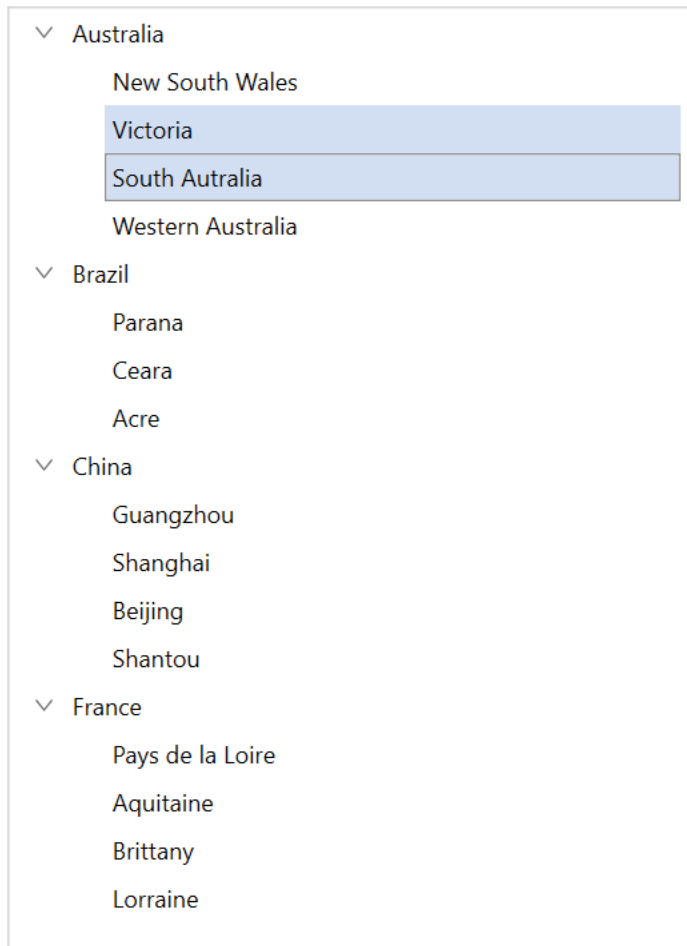
- **None**: Allows disabling the selection.
- **Single**: Allows selecting the single item only. When clicking on the selected item, selection will not be cleared. This is the default value for **SelectionMode**.
- **SingleDeselect**: Allows selecting the single item only. When clicking on the selected item, selection gets cleared.
- **Multiple**: Allows selecting more than one item. Selection is not cleared when selecting more than one items. When clicking on the selected item, selection gets cleared.
- **Extended**: Allows to select the multiple items using the common key modifiers.

XML

```
<syncfusion:SfTreeView x:Name="sfTreeView" SelectionMode="Multiple"/>
```

C#

```
sfTreeView.SelectionMode = SelectionMode.Multiple;
```



Programmatic Selection

When the [SelectionMode](#) is other than **None**, the item or items in the TreeView can be selected from the code by setting the [SelectedItem](#), or adding items to the [SelectedItems](#) property based on the [SelectionMode](#).

When the selection mode is **Single** or **SingleDeselect**, programmatically select an item by setting the underlying object to the [SelectedItem](#) property.

C#

```
sfTreeView.SelectedItem = viewModel.Items[0];
```

When the selection mode is **Multiple**, programmatically select more than one item by adding the underlying object to the [SelectedItems](#) property.

C#

```
sfTreeView.SelectedItems.Add(viewModel.Items[2]);  
sfTreeView.SelectedItems.Add(viewModel.Items[3]);
```

Warning: If an item is selected programmatically when `SelectionMode` is `None` and if multiple items are programmatically selected when `SelectionMode` is `Single` or `SingleDeselect`, then exception will be thrown internally.

Select the nodes based on property of underlying data object

You can bind selection state of node to the bool property in underlying data object by using `IsSelectedPropertyName` property. TreeView updates the selection of node when underlying data object property gets changed and vice versa.

XML

```
<treeView:SfTreeView x:Name="treeView"
ItemsSource="{Binding Folders}">
<treeView:SfTreeView.HierarchyPropertyDescriptors>
<treeView:HierarchyPropertyDescriptor IsSelectedPropertyName="IsSelected"
ChildPropertyName="SubFiles" TargetType="local:FileManager" />
</treeView:SfTreeView.HierarchyPropertyDescriptors>
</treeView:SfTreeView>
```

C#

```
public class FileManager : INotifyPropertyChanged
{
    private string fileName;
    private ObservableCollection<FileManager> subFiles;
    private bool isSelected;
    public ObservableCollection<FileManager> SubFiles
    {
        get { return subFiles; }
        set
        {
            subFiles = value;
            RaisedOnPropertyChanged("SubFiles");
        }
    }
    public string ItemName
    {
        get { return fileName; }
        set
        {
            fileName = value;
            RaisedOnPropertyChanged("ItemName");
        }
    }
    public bool IsSelected
    {
        get { return isSelected; }
        set
        {
            isSelected = value;
            RaisedOnPropertyChanged("IsSelected");
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    public void RaisedOnPropertyChanged(string _PropertyName)
    {
    }
```

```
if (PropertyChanged != null)
{
    PropertyChanged(this, new PropertyChangedEventArgs(_PropertyName));
}
}
}

public class FileManagerViewModel
{
    private ObservableCollection<FileManager> folders;
    public FileManagerViewModel()
    {
        GenerateSource();
    }
    public ObservableCollection<FileManager> Folders
    {
        get { return folders; }
        set { this.folders = value; }
    }
    private void GenerateSource()
    {
        var fileManager = new ObservableCollection<FileManager>();
        var doc = new FileManager() { ItemName = "Documents", IsSelected = true };
        var download = new FileManager() { ItemName = "Downloads", IsSelected = false };
        var pollution = new FileManager() { ItemName = "Environmental Pollution.docx" };
        var globalWarming = new FileManager() { ItemName = "Global Warming.ppt" };
        var sanitation = new FileManager() { ItemName = "Sanitation.docx" };
        var socialNetwork = new FileManager() { ItemName = "Social Network.pdf", IsSelected = true };
        var youthEmpower = new FileManager() { ItemName = "Youth Empowerment.pdf" };
        var games = new FileManager() { ItemName = "Game.exe" };
        var tutorials = new FileManager() { ItemName = "Tutorials.zip" };
        var TypeScript = new FileManager() { ItemName = "TypeScript.7z" };
        var uiGuide = new FileManager() { ItemName = "UI-Guide.pdf" };
        doc.SubFiles = new ObservableCollection<FileManager>
        {
            pollution,
            globalWarming,
            sanitation,
            socialNetwork,
            youthEmpower
        };
        download.SubFiles = new ObservableCollection<FileManager>
        {
            games,
            tutorials,
            TypeScript,
            uiGuide
        };
        fileManager.Add(doc);
        fileManager.Add(download);
        folders = fileManager;
    }
}
```

Note: `IsSelectedPropertyName` property is not supported for unbound mode and it accepts only boolean type property.

Selected items

Gets selected Items

The TreeView gets all the selected items through the `SelectedItems` property and gets the single item by using the `SelectedItem` property.

Clear selected items

The selected items can be cleared by calling the `SelectedItems.Clear()` method.

C#

```
sfTreeView.SelectedItems.Clear();
```

CurrentItem vs SelectedItem

The TreeView gets the selected item by using the `SelectedItem` and `CurrentItem` properties. Both `SelectedItem` and `CurrentItem` returns the same data object when selecting single item. When selecting more than one item, the `SelectedItem` property returns the first selected item, and the `CurrentItem` property returns the last selected item.

Selected item style

Selection background

The TreeView allows changing the selection background color for the selected items by using the `SelectionBackgroundColor` property. You can also change the selection background color at runtime.

Selection foreground

The TreeView allows changing the selection foreground color for the selected items by using the `SelectionForegroundColor` property. You can also change the selection foreground color at runtime.

Note: `SelectionForegroundColor` is applicable only for unbound mode.

Events

SelectionChanging Event

The `SelectionChanging` event is raised while selecting an item at the execution time. The `ItemSelectionChangingEventArgs` has the following members which provides the information for `SelectionChanging` event:

- `AddedItems`: Gets collection of the underlying data objects where the selection is going to process.
- `RemovedItems`: Gets collection of the underlying data objects where the selection is going to remove.

You can cancel the selection process within this event by setting the `ItemSelectionChangingEventArgs.Cancel` property to true.

C#

```
sfTreeView.SelectionChanging += TreeView_SelectionChanging;  
private void TreeView_SelectionChanging(object sender,  
Syncfusion.UI.Xaml.TreeView.ItemSelectionChangingEventArgs e)  
{
```



```

if (e.AddedItems.Count > 0 && e.AddedItems[0] == viewModel.Items[0])
{
    e.Cancel = true;
}
}

```

SelectionChanged event

The [SelectionChanged](#) event will occur once selection process has been completed for the selected item in the TreeView. The [ItemSelectionChangedEventArgs](#) has the following members which provides information for [SelectionChanged](#) event:

- [AddedItems](#): Gets collection of the underlying data objects where the selection has been processed.
- [RemovedItems](#): Gets collection of the underlying data objects where the selection has been removed.

C#

```

sfTreeView.SelectionChanged += TreeView_SelectionChanged;
private void TreeView_SelectionChanged(object sender,
Syncfusion.UI.Xaml.TreeView.ItemSelectionChangedEventArgs e)
{
    sfTreeView.SelectedItems.Clear();
}

```

Note: [SelectionChanging](#) and [SelectionChanged](#) events will be triggered only on UI interactions.

Key Navigation

The TreeView allows to select or navigate the items through keyboard interactions. When the [SelectionMode](#) is [Multiple](#) or [Extended](#), the [FocusBorderColor](#) will set to the CurrentItem.

FocusBorderColor

The [FocusBorderColor](#) property is used to set the border color for the current focused item. The default color is [LightSlateGray](#).

FocusBorderThickness

The [FocusBorderThickness](#) property is used to set the border thickness for the current focused item. The default thickness is [1](#).

How to add selection on right click

By default, TreeView doesn't allow selection on right click. However, selection can be added in application level by adding the tree node content to [TreeView.SelectedItems](#) collection, for this we retrieve the node at the specified mouse point using [GetNodeAt](#) method.

Below is the code example, which adds the node content to selected items upon right click on the tree node by checking the exact behavior of [FullRowSelect](#) support.

XML

```

<treeView:SfTreeView x:Name="treeView"
AutoExpandMode="RootNodes"
ItemsSource="{Binding Countries}"

```

```
SelectionMode="Multiple"
MouseRightButtonDown="treeView_MouseRightButtonDown">
</treeView:SfTreeView>
```

C#

```
private bool IsMouseOverOnExpander(Syncfusion.UI.Xaml.TreeView.TreeViewItem
treeViewItem, Point point)
{
    if (treeViewItem.TreeViewItemInfo.TreeView.ExpanderPosition ==
    ExpanderPosition.Start)
    return point.X < treeViewItem.IndentationWidth + treeViewItem.ExpanderWidth;
    else
    return point.X > (treeViewItem.ActualWidth - treeViewItem.ExpanderWidth);
}
private void treeView_MouseRightButtonDown(object sender,
MouseButtonEventArgs e)
{
    var treeViewNode = this.treeView.GetNodeAt(e.GetPosition(this.treeView));
    var itemInfo = treeView.GetItemInfo(treeViewNode.Content);
    var itemPoint = e.GetPosition(itemInfo.Element);
    if (!this.treeView.FullRowSelect && IsMouseOverOnExpander(itemInfo.Element,
    itemPoint))
    return;
    if (this.treeView.SelectedItems == null)
    this.treeView.SelectedItems = new
    System.Collections.ObjectModel.ObservableCollection<object>();
    this.treeView.SelectedItems.Add(treeViewNode.Content);
}
```

Limitation

- When a grid is loaded inside the [ItemTemplate](#) with background color, the [SelectionBackgroundColor](#) will not display. Because, it overlaps the [SelectionBackgroundColor](#). In this case, set the background color for the TreeView instead of grid in the ItemTemplate.
- When the [TreeView](#) contains duplicated items in the collection, only the first item whose instance was created initially will be selected or deselected.

Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

[Checkbox in WPF TreeView \(SfTreeView\)](#)

SfTreeView provides support for loading [CheckBox](#) in each node, and allows users to check/uncheck the corresponding node. So, you should add checkbox in the [ItemTemplate](#) of the [SfTreeView](#) and bind the [IsChecked](#) property of the [TreeViewNode](#).

[Working with Checkbox in BoundMode](#)

When you are populating treeview nodes from [ItemsSource](#), then you can get or set the checked items by using [CheckedItems](#) property.

SfTreeView supports to check multiple items through binding the [CheckedItems](#) property from view model with `ObservableCollection<object>` type.

Note: Set [ItemTemplateDataContextType](#) as `Node` to bind the `TreeNode.IsChecked` property to `CheckBox` in `ItemTemplate`.

Note: TreeView process and sets [TreeNode.IsChecked](#) based on `CheckedItems` only when you are binding `ItemsSource`.

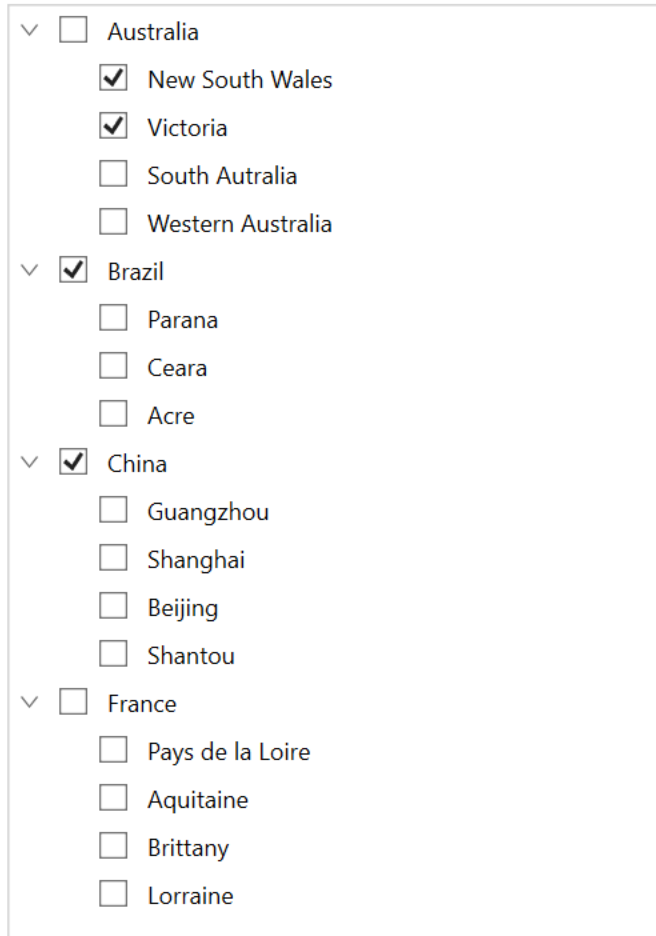
XML

```
<syncfusion:SfTreeView
x:Name="sfTreeView"
Margin="10"
BorderThickness="1"
AutoExpandMode="AllNodes"
BorderBrush="LightGray"
AllowDragging="True"
SelectionMode="Multiple"
CheckBoxMode="Recursive"
CheckedItems="{Binding CheckedStates}"
ChildPropertyName="Models"
ExpandActionTrigger="Node"
ItemTemplateDataContextType="Node"
FocusVisualStyle="{x:Null}"
IsAnimationEnabled="True"
ItemsSource="{Binding Items}">
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<Grid>
<CheckBox x:Name="CheckBox" FocusVisualStyle="{x:Null}"
IsChecked="{Binding IsChecked, Mode=TwoWay}"/>
<TextBlock FontSize="12" VerticalAlignment="Center" Text="{Binding
Content.State}" Margin="25,0,0,0"/>
</Grid>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>
```

C#

```
public class ViewModel : NotificationObject
{
    public ObservableCollection<Model> Items { get; set; }
    private ObservableCollection<object> checkedStates;
    public ObservableCollection<object> CheckedStates
    {
        get { return checkedStates; }
        set { checkedStates = value; }
    }
    public ViewModel()
    {
        Items = new ObservableCollection<Model>();
        checkedStates = new ObservableCollection<object>();
        var country1 = new Model { State = "Australia" };
        var country2 = new Model { State = "Brazil" };
    }
}
```

```
var country3 = new Model { State = "China" };
var country4 = new Model { State = "France" };
var aus_state1 = new Model { State = "New South Wales" };
var aus_state2 = new Model { State = "Victoria" };
var aus_state3 = new Model { State = "South Australia" };
var aus_state4 = new Model { State = "Western Australia" };
var brazil_state1 = new Model { State = "Parana" };
var brazil_state2 = new Model { State = "Ceara" };
var brazil_state3 = new Model { State = "Acre" };
var china_state1 = new Model { State = "Guangzhou" };
var china_state2 = new Model { State = "Shanghai" };
var china_state3 = new Model { State = "Beijing" };
var china_state4 = new Model { State = "Shantou" };
var france_state1 = new Model { State = "Pays de la Loire" };
var france_state2 = new Model { State = "Aquitaine" };
var france_state3 = new Model { State = "Brittany" };
var france_state4 = new Model { State = "Lorraine" };
country1.Models.Add(aus_state1);
country1.Models.Add(aus_state2);
country1.Models.Add(aus_state3);
country1.Models.Add(aus_state4);
country2.Models.Add(brazil_state1);
country2.Models.Add(brazil_state2);
country2.Models.Add(brazil_state3);
country3.Models.Add(china_state1);
country3.Models.Add(china_state2);
country3.Models.Add(china_state3);
country3.Models.Add(china_state4);
country4.Models.Add(france_state1);
country4.Models.Add(france_state2);
country4.Models.Add(france_state3);
country4.Models.Add(france_state4);
Items.Add(country1);
Items.Add(country2);
Items.Add(country3);
Items.Add(country4);
checkedStates.Add(aus_state1);
checkedStates.Add(aus_state2);
checkedStates.Add(country2);
checkedStates.Add(country3);
}
}
```



Note: View sample in [GitHub](#)

Working with Checkbox in UnboundMode

You can directly set the checkbox state by setting the [TreeViewNode.IsChecked](#) property value while creating nodes.

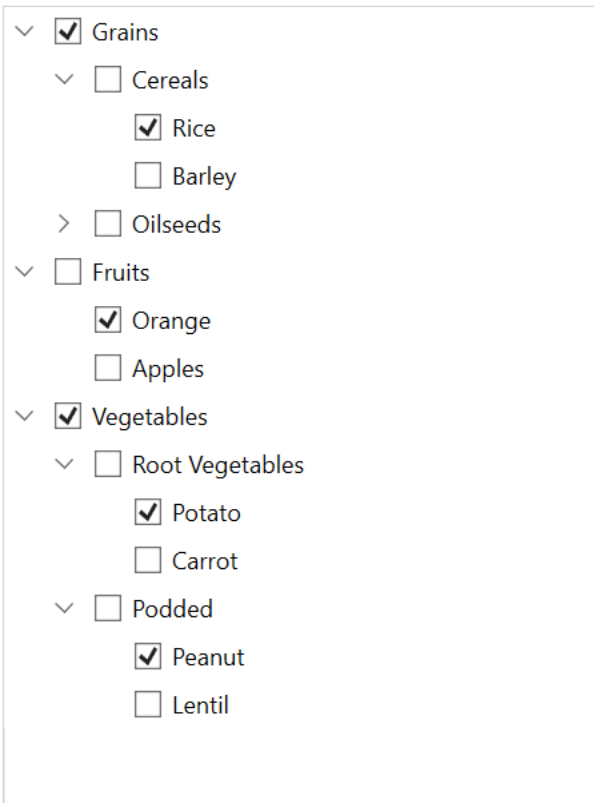
XML

```
<Window
x:Class="TreeNodeWithCheckBoxDemo.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:TreeNodeWithCheckBoxDemo"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:Engine="clr-namespace:Syncfusion.UI.Xaml.TreeView.Engine;assembly=Syncfusion.SfTreeView.WPF"
Title="Node With CheckBox"
Width="450"
Height="500"
Icon="App.ico"
WindowStartupLocation="CenterScreen">
<Grid>
<syncfusion:SfTreeView x:Name="sfTreeView"
Width="400"/>
```

```

Margin="10,10,10,10"
BorderThickness="1"
BorderBrush="LightGray"
IsAnimationEnabled="True"
ItemTemplateDataContextType="Node"
CheckBoxMode="Recursive" >
<syncfusion:SfTreeView.Nodes>
<Engine:TreeNode Content="Grains" IsExpanded="True" IsChecked="True">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Cereals" IsExpanded="True">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Rice" IsChecked="True"/>
<Engine:TreeNode Content="Barley"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
<Engine:TreeNode Content="Oilseeds">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Safflower" IsChecked="True"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
<Engine:TreeNode Content="Fruits" IsExpanded="true">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Orange" IsChecked="True"/>
<Engine:TreeNode Content="Apples" IsExpanded="true"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
<Engine:TreeNode Content="Vegetables" IsExpanded="true"
IsChecked="True">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Root Vegetables" IsExpanded="true">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Potato" IsChecked="True"/>
<Engine:TreeNode Content="Carrot"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
<Engine:TreeNode Content="Podded">
<Engine:TreeNode.ChildNodes>
<Engine:TreeNode Content="Peanut" IsChecked="True"/>
<Engine:TreeNode Content="Lentil"/>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
</Engine:TreeNode.ChildNodes>
</Engine:TreeNode>
</syncfusion:SfTreeView.Nodes>
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<Grid>
<CheckBox x:Name="CheckBox" FocusVisualStyle="{x:Null}"
IsChecked="{Binding IsChecked, Mode=TwoWay}" Content="{Binding Content}"/>
</Grid>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>
</Grid>
</Window>

```



Note: View sample in [GitHub](#)

CheckBox State

SfTreeView process [IsChecked](#) property (checkbox state) of [TreeNode](#) based on [CheckBoxMode](#) property. [CheckBoxMode](#) defines how parent and child node's checkbox state updates when user check or un-check the node. By default, its value is [None](#). Checkbox contains the following three states:

- **None:** Check and uncheck are updates only in the view, but it will not affect the [CheckedItems](#) collection.
- **Individual:** Checkbox state affect individual node only, and it does not affect the parent node or child nodes checkbox state or [IsChecked](#) property value.
- **Recursive:** Check and uncheck the node value affects the parent and child nodes checkbox state. For example, If parent nodes checkbox state is check/uncheck then the all of its child nodes checkbox state is check/uncheck. If all the child nodes are check/uncheck within the parent node, then parent node will be check/uncheck. If any of the child node is check, then the parent node will be in intermediate state.

XML

```
<syncfusion:SfTreeView x:Name="sfTreeView" CheckBoxMode="Recursive" />
```

C#

```
sfTreeView.CheckBoxMode = CheckBoxMode.Recursive;
```

Note: In recursive mode, the parent nodes checkbox state or `IsChecked` property value is updated only in UI interaction.

Get or Set Checked Items

Get or Set Checked Items in Bound Mode

You can get or set list of items to be checked or un-checked by using [CheckedItems](#) property.

When the [CheckBoxMode](#) is other than `None`, the individual `TreeViewNode` or collection of `TreeViewNode` can be checked from the code by setting the `CheckedItems`, or adding items to the `CheckedItems` property based on the `CheckBoxMode`.

Note: Programmatically adding or removing the node value not affects their parent and child nodes checkbox state.

C#

```
sfTreeView.CheckedItems.Add(viewModel.Items[2]);  
sfTreeView.CheckedItems.Add(viewModel.Items[3]);
```

Events

NodeChecked event

The [NodeChecked](#) event raised when checking and unchecking the checkbox at run time. The [NodeCheckedEventArgs](#) has the following members, which provide information for the `NodeChecked` event.

- **Node:** Gets the `TreeViewNode` and data associated with the checked item as its arguments.

C#

```
sfTreeView.NodeChecked += SfTreeView_NodeChecked;  
private void SfTreeView_NodeChecked(object sender, NodeCheckedEventArgs e)  
{  
}
```

Note: `NodeChecked` event occurs only in UI interactions. You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Drag and drop in WPF TreeView (SfTreeView)

TreeView allows drag and drop the items within the treeview control by setting the [AllowDragging](#) property as `true`. It is also possible to drag and drop the items between treeview and other controls such as `ListView` and `SfDataGrid`.

XML

```
<treeview:SfTreeView  
Name="sfTreeView"  
AllowDragging="True"
```

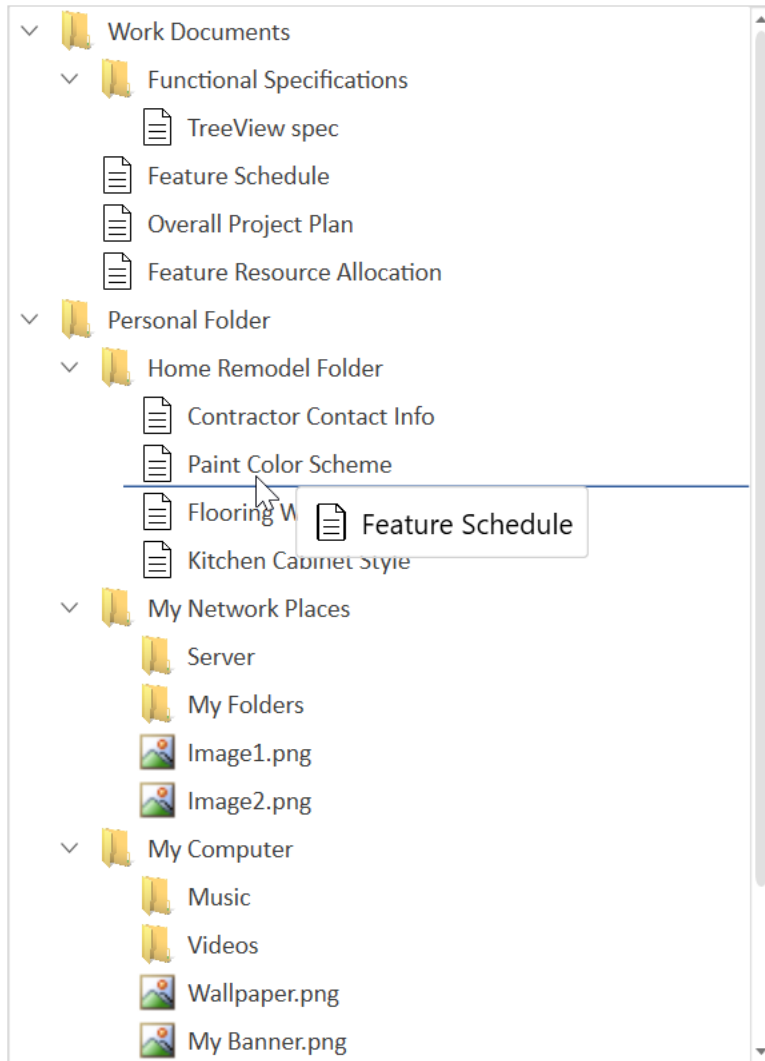


```
ChildPropertyName="Childs"  
ItemsSource="{Binding Nodes1}" />
```

C#

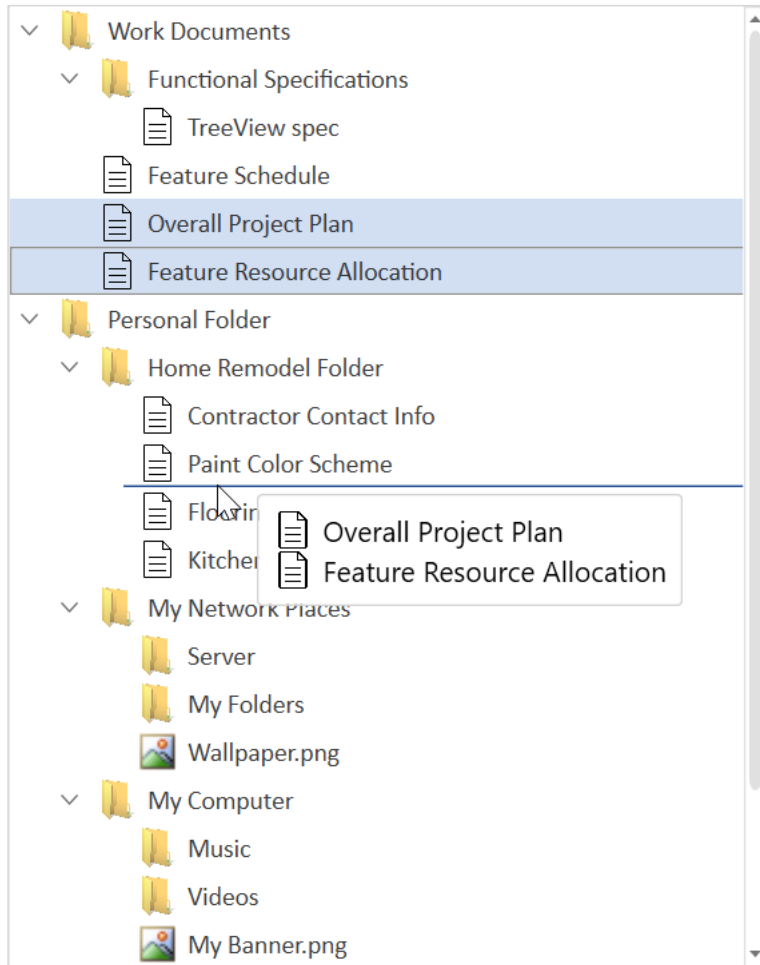
```
sfTreeView.AllowDragging = true;
```

While dropping, the dragged items can be added above or below to the target item based on drag indicator position.



Dragging multiple items

SfTreeView allows to drag multiple selected items. To enable multiple selection, set the [SfTreeView.SelectionMode](#) as **Multiple** or **Extended**.



Drag and drop events

SfTreeView triggers the following events when drag and drop:

ItemDragStarting event

[ItemDragStarting](#) event occurs when you starting to drag the items in treeview. The [TreeViewItemDragStartingEventArgs](#) has the following member, which provides information for the ItemDragStarting event.

- [Data](#) : Gets or Sets a data object that contains the data associated while dragging the items.
- [DraggingNodes](#) : Gets the collection of [TreeViewNode](#) which are dragged.If you set the Data property, the value of DraggingNodes property will be null.
- [Cancel](#) : Gets or sets a value indicating whether dragging should be canceled.

C#

```
sfTreeView.ItemDragStarting += SfTreeView_ItemDragStarting;  
private void SfTreeView_ItemDragStarting(object sender,  
Syncfusion.UI.Xaml.TreeView.TreeViewItemDragStartingEventArgs e)  
{  
}
```

ItemDragStarted event

[ItemDragStarted](#) event occurs after started the dragging, in treeview. The [TreeViewItemDragStartedEventArgs](#) has the following member, which provides information for the [ItemDragStarted](#) event.

- [Data](#) : Gets a data object that contains the data associated while dragging the items.
- [DraggingNodes](#) : Gets the collection of [TreeViewNode](#) which are dragged.

C#

```
sfTreeView.ItemDragStarted += SfTreeView_ItemDragStarted;  
private void SfTreeView_ItemDragStarted(object sender,  
Syncfusion.UI.Xaml.TreeView.TreeViewItemDragStartedEventArgs e)  
{  
}
```

ItemDragOver event

[ItemDragOver](#) event occurs continuously while item is dragged within the targeted SfTreeView. The [TreeViewItemDragOverEventArgs](#) has the following members, which provide information for the [ItemDragOver](#) event.

- [Data](#) : Gets a data object that contains the data associated while dragging the items.
- [DraggingNodes](#) : Gets the collection of [TreeViewNode](#) which are dragged.
- [DragSource](#) : Gets the source of the transferred data.
- [DropPosition](#) : Gets or sets the position where dragged nodes are going to be dropped.
- [TargetNode](#) : Gets the node where the dragged nodes are going to be dropped.

C#

```
sfTreeView.ItemDragOver += SfTreeView_ItemDragOver;  
private void SfTreeView_ItemDragOver(object sender,  
Syncfusion.UI.Xaml.TreeView.TreeViewItemDragOverEventArgs e)  
{  
}
```

ItemDropping event

[ItemDropping](#) event occurs when item is dropping within the targeted SfTreeView. The [TreeViewItemDroppingEventArgs](#) has the following members, which provide information for the [ItemDropping](#) event.

- [Data](#) : Gets a data object that contains the data associated while dragging the items.
- [DraggingNodes](#) : Gets the collection of [TreeViewNode](#) which are dragged.
- [DragSource](#) : Gets the source of the transferred data.
- [DropPosition](#) : Gets or sets the position where dragged nodes are going to be dropped.
- [Handled](#) : Gets or sets a value indicating whether the event is handled. If this event is handled, dragged nodes will not be dropped to TreeView.
- [TargetNode](#) : Gets the node where the dragged nodes are going to be dropped.

C#

```
sfTreeView.ItemDropping += SfTreeView_ItemDropping;
private void SfTreeView_ItemDropping(object sender,
Syncfusion.UI.Xaml.TreeView.TreeViewItemDroppingEventArgs e)
{
}
```

ItemDropped event

[ItemDropped](#) event occurs when item is dropped within the targeted SfTreeView. The [TreeViewItemDroppedEventArgs](#) has the following members, which provide information for the **Drop** event.

- [Data](#) : Gets a data object that contains the data associated while dragging the items.
- [DraggingNodes](#) : Gets the collection of [TreeViewNode](#) which are dragged.
- [DragSource](#) : Gets the source of the transferred data.
- [DropPosition](#) : Gets the position where dragged nodes are dropped.
- [TargetNode](#) : Gets the node where the dragged nodes are dropped.

C#

```
sfTreeView.ItemDropped += SfTreeView_ItemDropped;
private void SfTreeView_ItemDropped(object sender,
Syncfusion.UI.Xaml.TreeView.TreeViewItemDroppedEventArgs e)
{
}
```

Customizing the drag and drop operation

Disable dragging of certain items in WPF TreeView

You can restrict the dragging of certain nodes in SfTreeView by using the [SfTreeView.ItemDragStarting](#) event.

C#

```
sfTreeView.ItemDragStarting += SfTreeView_ItemDragStarting;
private void SfTreeView_ItemDragStarting(object sender,
Syncfusion.UI.Xaml.TreeView.TreeViewItemDragStartingEventArgs e)
{
    var record = e.DraggingNodes[0].Content as Model;
    if (record.Header == "Feature Schedule")
        e.Cancel = true;
}
```

Disable dropping on certain items in WPF TreeView

You can restrict the dropping the items on certain nodes in SfTreeView by using the [SfTreeView.ItemDropping](#) event.

C#

```
sfTreeView.ItemDropping += SfTreeView_ItemDropping;
private void SfTreeView_ItemDropping(object sender,
Syncfusion.UI.Xaml.TreeView.TreeViewItemDroppingEventArgs e)
{
    var record = e.TargetNode.Content as Model;
```

```
if (record.Header == "Home Remodel Folder")
    e.Handled = true;
}
```

Customize the drop position

You can customize the drop position of dragging nodes in SfTreeView by using the [SfTreeView.ItemDropping](#) event.

C#

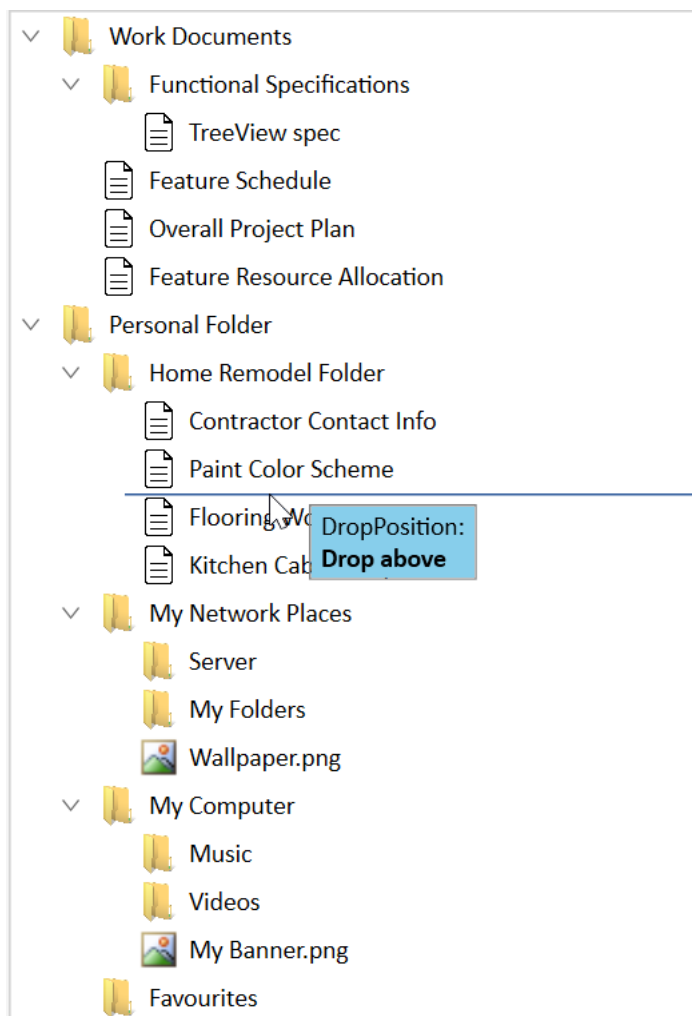
```
sfTreeView.ItemDropping += SfTreeView_ItemDropping;
private void SfTreeView_ItemDropping(object sender,
Syncfusion.UI.Xaml.TreeView.TreeViewItemDroppingEventArgs e)
{
    var record = e.TargetNode.Content as Model;
    if (record.Header == "Home Remodel Folder")
        e.DropPosition = Syncfusion.UI.Xaml.TreeView.DropPosition.DropAsChild;
}
```

Customizing drag Popup

To customize the draggable popup, use the [DragPreviewTemplate](#) property in the SfTreeView. The DataContext of DragPreviewTemplate is [TreeViewDragInfo](#).

XML

```
<syncfusion:ChromelessWindow.Resources>
<DataTemplate x:Key="dragPreviewTemplate">
<Border BorderBrush="Gray" BorderThickness="1">
<Grid Background="SkyBlue">
<Grid.RowDefinitions>
<RowDefinition />
<RowDefinition/>
</Grid.RowDefinitions>
<TextBlock Grid.Row="0" Margin="5,2,5,0"
VerticalAlignment="Center" Text="DropPosition:"/>
<TextBlock Grid.Row="1"
Margin="5,0,5,2"
VerticalAlignment="Center"
Text="{Binding DragCaption}" FontWeight="Bold"/>
</Grid>
</Border>
</DataTemplate>
</syncfusion:ChromelessWindow.Resources>
<treeview:SfTreeView
Name="sfTreeView"
AllowDragging="True"
ChildPropertyName="Childs"
DragPreviewTemplate="{StaticResource dragPreviewTemplate}"
ItemsSource="{Binding Nodes1}" />
```



Drag and drop between two TreeView's

You can customize the dragging operation between two treeview by using the [SfTreeView.ItemDragStarting](#) , [SfTreeView.ItemDropping](#) and [SfTreeView.ItemDropped](#) events.

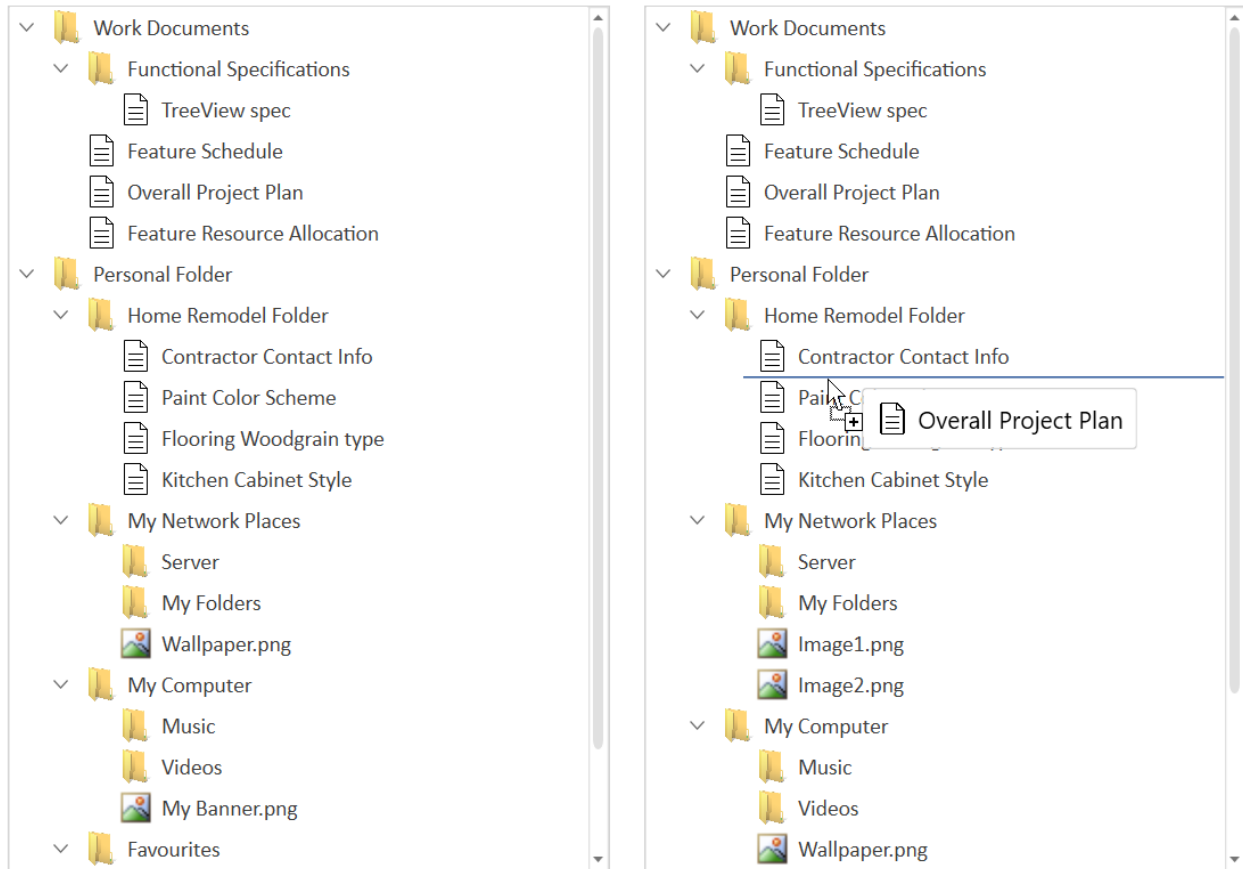
C#

```
AssociatedObject.sfTreeView1.ItemDragStarting +=
SfTreeView1_ItemDragStarting;
AssociatedObject.sfTreeView1.ItemDropping += SfTreeView1_ItemDropping;
AssociatedObject.sfTreeView2.ItemDropping += SfTreeView2_ItemDropping;
AssociatedObject.sfTreeView2.ItemDropped += SfTreeView1_ItemDropped;
/// <summary>
/// Customizing the ItemStarting event
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SfTreeView1_ItemDragStarting(object sender,
Syncfusion.UI.Xaml.TreeView.TreeViewItemDragStartingEventArgs e)
{
    //Restrict the dragging for certain node
    var record = e.DraggingNodes[0].Content as Model;
    if (record.Header == "Feature Schedule")
        e.Cancel = true;
```

```

}
/// <summary>
/// Customizing the ItemDropping event
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SfTreeView1_ItemDropping(object sender,
Syncfusion.UI.Xaml.TreeView.TreeViewItemDroppingEventArgs e)
{
    //Restrict the dropping in first treeview
    e.Handled = true;
}
/// <summary>
/// Customizing the ItemDropping event
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SfTreeView2_ItemDropping(object sender,
Syncfusion.UI.Xaml.TreeView.TreeViewItemDroppingEventArgs e)
{
    //Restrict the dropping for drop position as above
    if (e.DropPosition == Syncfusion.UI.Xaml.TreeView.DropPosition.DropAbove)
    e.Handled = true;
    //Restrict the dropping on certain nodes
    var record = e.TargetNode.Content as Model;
    if (record.Header == "My Folders")
    e.Handled = true;
}
/// <summary>
/// Customize the ItemDropped event
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SfTreeView1_ItemDropped(object sender,
Syncfusion.UI.Xaml.TreeView.TreeViewItemDroppedEventArgs e)
{
    var parentNode = e.TargetNode.ParentNode;
    var collection = parentNode.ChildNodes;
    var record = e.DraggingNodes[0].Content as Model;
    int count = 0;
    foreach (var child in parentNode.ChildNodes)
    {
        var childNode = child.Content as Model;
        if (childNode.Header == record.Header)
        {
            count++;
            if (count > 1)
            {
                // Remove dropped node if the parent has the same node in it
                collection.Remove(child);
            }
        }
    }
    return;
}
}
}
}

```



Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Tree lines in WPF TreeView (SfTreeView)

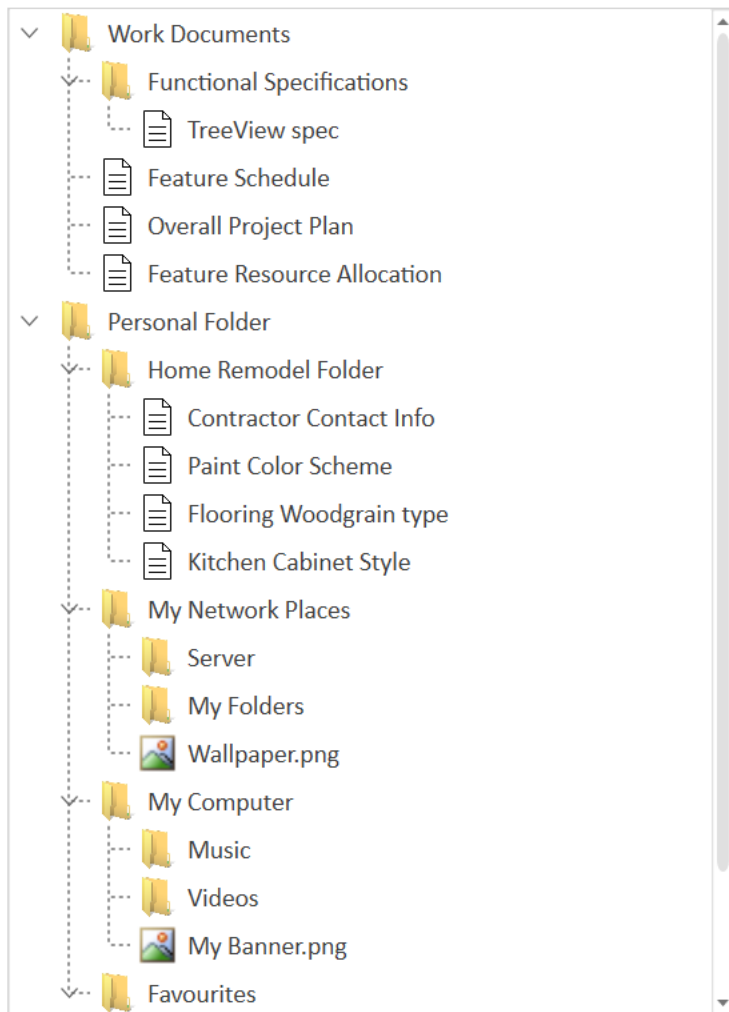
TreeView allows to show the tree lines for treeview nodes by enabling the [ShowLines](#) property as `true`. The default value is `false`.

XML

```
<treeview:SfTreeView
Name="sfTreeView"
ShowLines="True"
ChildPropertyName="Childs"
ItemTemplate="{StaticResource SfTreeView_ItemTemplate}"
ItemsSource="{Binding Nodes1}" />
```

C#

```
sfTreeView.ShowLines = true;
```

Enable tree line for root nodes

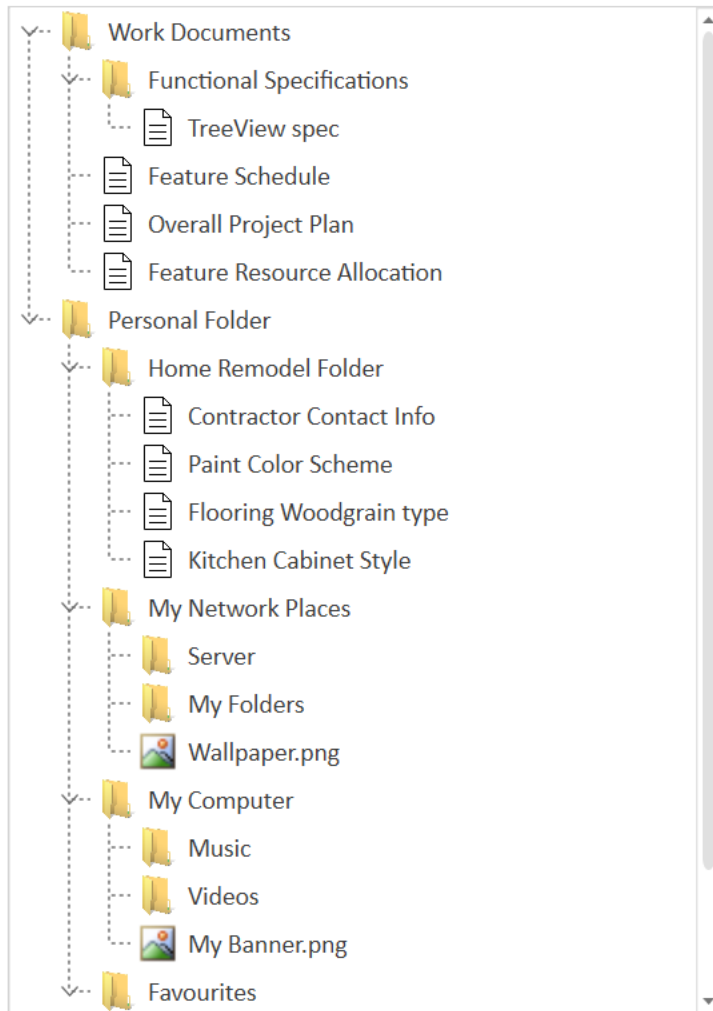
TreeView also supports to show the tree lines for root nodes by enabling the [ShowRootLines](#) property as `true`. The default value is `false`.

XML

```
<treeview:SfTreeView
Name="sfTreeView"
ShowLines="True"
ShowRootLines="True"
ChildPropertyName="Childs"
ItemTemplate="{StaticResource SfTreeView_ItemTemplate}"
ItemsSource="{Binding Nodes1}" />
```

C#

```
sfTreeView.ShowLines = true;
sfTreeView.ShowRootLines = true;
```



Customizing the tree lines

Customizing the line color

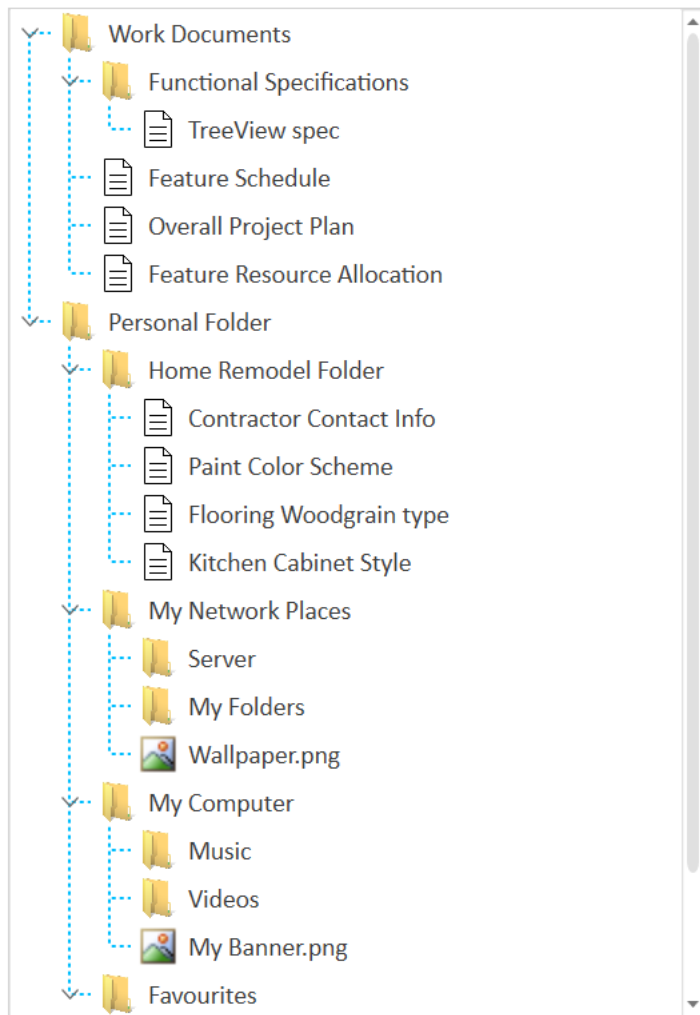
TreeView allows to change the color of tree lines by using the [LineStroke](#) property. The default value is `System.Windows.Media.Colors.LightSlateGray`.

XML

```
<treeview:SfTreeView
Name="sfTreeView"
ShowLines="True"
ShowRootLines="True"
LineStroke="DeepSkyBlue"
ChildPropertyName="Childs"
ItemTemplate="{StaticResource SfTreeView_ItemTemplate}"
ItemsSource="{Binding Nodes1}" />
```

C#

```
sfTreeView.ShowLines = true;
sfTreeView.ShowRootLines = true;
sfTreeView.LineStroke = new SolidColorBrush(Colors.DeepSkyBlue);
```



Customizing the line thickness

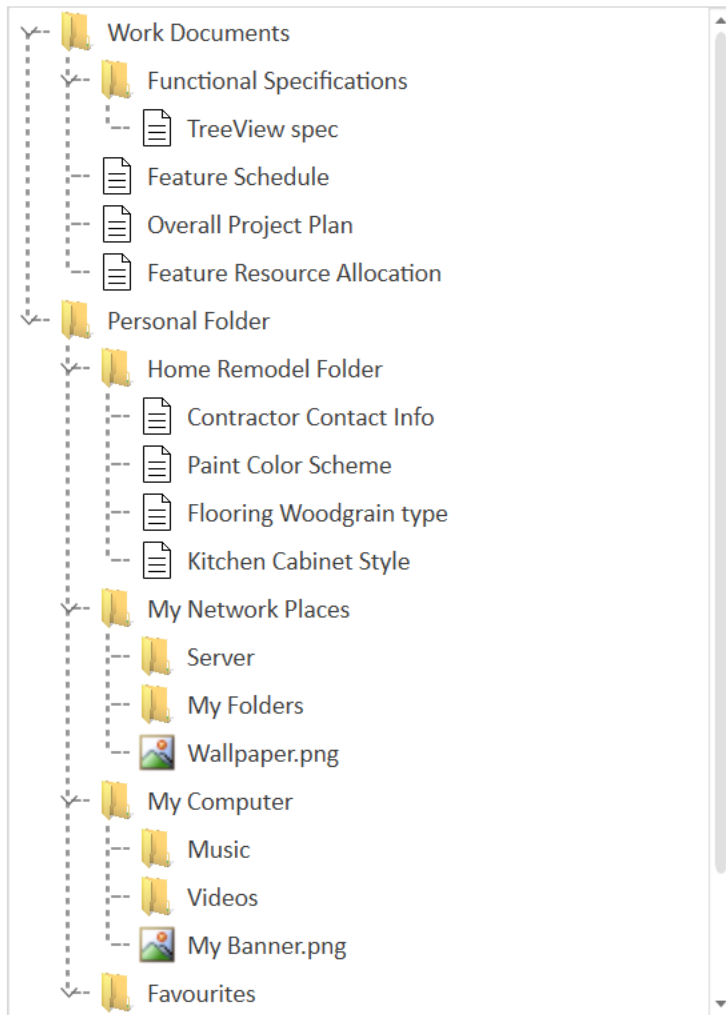
TreeView allows to change the thickness of tree lines by using the [LineStrokeThickness](#) property. The default value is `1`.

XML

```
<treeview:SfTreeView
Name="sfTreeView"
ShowLines="True"
ShowRootLines="True"
LineStrokeThickness="1.5"
ChildPropertyName="Childs"
ItemTemplate="{StaticResource SfTreeView_ItemTemplate}"
ItemsSource="{Binding Nodes1}" />
```

C#

```
sfTreeView.ShowLines = true;
sfTreeView.ShowRootLines = true;
sfTreeView.LineStrokeThickness = 1.5;
```



Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Context menu in WPF TreeView (SfTreeView)

This section explains how to show ContextMenu and using built-in commands in the TreeView.

ContextMenu for Nodes

The TreeView provides an entirely customizable context menu to expose the functionality on user interface. You can create context menu for nodes in an efficient manner.

You can set context menu for the nodes by using [SfTreeView.ItemContextMenu](#) property.

Built-in Commands

The TreeView provides support for the following built-in commands

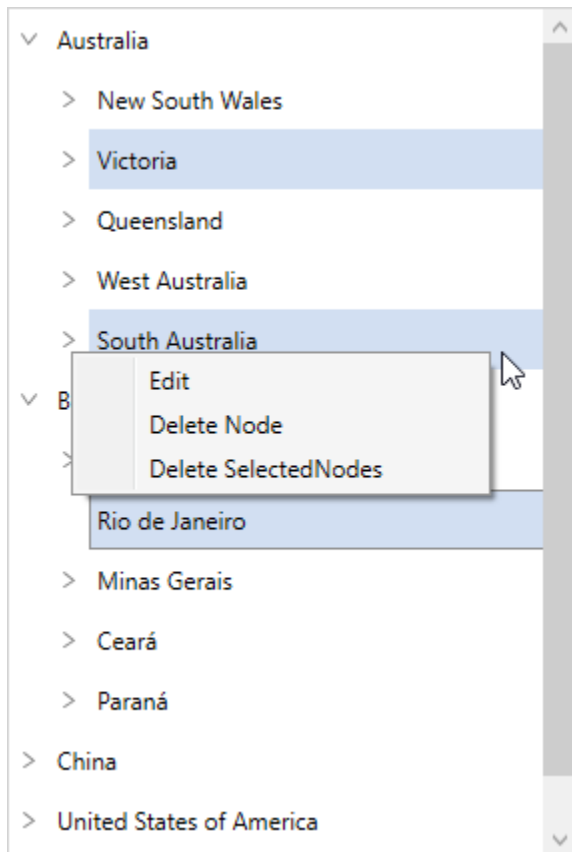
- [Edit](#) - Command to start the editing of the node.
- [DeleteNode](#) - Command to delete the node.
- [DeleteSelectedNodes](#) - Command to delete all the selected nodes.

XML

```

<syncfusion:SfTreeView x:Name="sfTreeView"
ItemsSource="{Binding Folders}"
ChildPropertyName="SubFiles"
AllowEditing="True"
SelectionMode="Multiple">
<syncfusion:SfTreeView.ItemContextMenu>
<ContextMenu>
<MenuItem Command="{x:Static syncfusion:TreeViewCommands.Edit}"
CommandParameter="{Binding }"></MenuItem>
<MenuItem Command="{x:Static syncfusion:TreeViewCommands.DeleteNode}"
CommandParameter="{Binding }"></MenuItem>
<MenuItem Command="{x:Static
syncfusion:TreeViewCommands.DeleteSelectedNodes}" CommandParameter="{Binding
}"></MenuItem>
</ContextMenu>
</syncfusion:SfTreeView.ItemContextMenu>
</syncfusion:SfTreeView>

```

**Custom Commands**

The TreeView allows to show ContextMenu using custom commands when built-in commands does not meet your requirement.

For an example, custom command is used to expand the nodes using context menu in the following example

XML

```

<syncfusion:SfTreeView x:Name="sfTreeView"
ItemsSource="{Binding Folders}"
ChildPropertyName="SubFiles"
AllowEditing="True"
SelectionMode="Multiple">
<syncfusion:SfTreeView.ItemContextMenu>
<ContextMenu>
<MenuItem Command="{Binding Path=TreeView.DataContext.ExpandCommand}"
CommandParameter="{Binding }" Header="Expand"/>
</ContextMenu>
</syncfusion:SfTreeView.ItemContextMenu>
</syncfusion:SfTreeView>

```

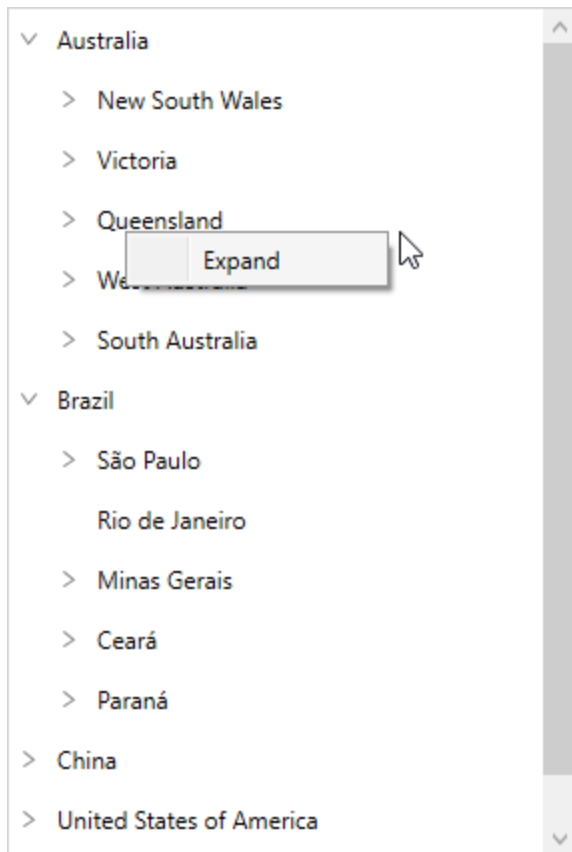
C#

```

public class BaseCommand : ICommand
{
    #region Fields
    readonly Action<object> _execute;
    readonly Predicate<object> _canExecute;
    #endregion
    #region Constructors
    /// <summary>
    /// Creates a new command that can always execute.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    public BaseCommand(Action<object> execute)
    : this(execute, null)
    {
    }
    /// <summary>
    /// Creates a new command.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    /// <param name="canExecute">The execution status logic.</param>
    public BaseCommand(Action<object> execute, Predicate<object> canExecute)
    {
        if (execute == null)
            throw new ArgumentNullException("execute");
        _execute = execute;
        _canExecute = canExecute;
    }
    #endregion
    #region ICommand Members
    public bool CanExecute(object parameter)
    {
        return _canExecute == null ? true : _canExecute(parameter);
    }
    public event EventHandler CanExecuteChanged
    {
        add { CommandManager.RequerySuggested += value; }
        remove { CommandManager.RequerySuggested -= value; }
    }
    public void Execute(object parameter)

```

```
{
    _execute(parameter);
}
#endregion
}
public class FileManagerViewModel
{
    public FileManagerViewModel()
    {
        ExpandCommand = new BaseCommand(ExecuteExpandCommand,
        CanExecuteExpandCommand);
    }
    public BaseCommand ExpandCommand { get; set; }
    private bool CanExecuteExpandCommand(object obj)
    {
        if (obj == null)
            return false;
        var treeViewContextMenuInfo = obj as TreeViewItemContextMenuInfo;
        if (treeViewContextMenuInfo.Node.HasChildNodes)
            return true;
        return false;
    }
    private static void ExecuteExpandCommand(object obj)
    {
        if (obj == null)
            return;
        var treeViewContextMenuInfo = obj as TreeViewItemContextMenuInfo;
        treeViewContextMenuInfo.TreeView.ExpandNode(treeViewContextMenuInfo.Node);
    }
}
```



Events

ItemContextMenuOpening Event

[ItemContextMenuOpening](#) event occurs while opening the context menu in the TreeView.

[ItemContextMenuOpeningEventArgs](#) has the following members which provides the information about [ItemContextMenuOpening](#) event

- [MenuInfo](#) - Gets an instance of [TreeViewContextMenuInfo](#) that contains information about treeview and treeview node.
- [ContextMenu](#) - Gets or sets the context menu getting opened.

You can cancel showing of [ItemContextMenu](#) for certain nodes using custom logic within this event by setting [ItemContextMenuOpeningEventArgs.Cancel](#) as true.

C#

```
sfTreeView.ItemContextMenuOpening += TreeView_ItemContextMenuOpening;  
private void TreeView_ItemContextMenuOpening(object sender,  
Syncfusion.UI.Xaml.TreeView.ItemContextMenuOpeningEventArgs e)  
{  
    if (e.MenuInfo.Node.Level == 2)  
        e.Cancel = true;  
}
```


Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

MVVM in WPF TreeView (SfTreeView)

This section explains about how to work with MVVM pattern in TreeView.

Binding properties in MVVM pattern

Binding SelectedItem

TreeView support to select the items through binding the [SelectedItem](#) property from view model by implementing the `INotifyPropertyChanged` interface that gives the call back notification to UI.

XML

```
<syncfusion:SfTreeView
x:Name="sfTreeView"
SelectedItem="{Binding SelectedNode}"
ChildPropertyName="Models"
ItemTemplateDataContextType="Node"
ItemsSource="{Binding Items}" >
</syncfusion:SfTreeView>
```

C#

```
sfTreeView.SetBinding(SfTreeView.SelectedItemProperty, new
Binding("SelectedNode"));
```

C#

```
public class ViewModel : NotificationObject
{
    public ObservableCollection<Model> Items { get; set; }
    public object SelectedNode { get; set; }
    public ViewModel()
    {
        Items = new ObservableCollection<Model>();
        var country1 = new Model { State = "Australia" };
        var country2 = new Model { State = "Brazil" };
        var country3 = new Model { State = "China" };
        var country4 = new Model { State = "France" };
        var aus_state1 = new Model { State = "New South Wales" };
        var aus_state2 = new Model { State = "Victoria" };
        var aus_state3 = new Model { State = "South Australia" };
        var aus_state4 = new Model { State = "Western Australia" };
        var brazil_state1 = new Model { State = "Parana" };
        var brazil_state2 = new Model { State = "Ceara" };
        var brazil_state3 = new Model { State = "Acre" };
        var china_state1 = new Model { State = "Guangzhou" };
        var china_state2 = new Model { State = "Shanghai" };
        var china_state3 = new Model { State = "Beijing" };
        var china_state4 = new Model { State = "Shantou" };
        var france_state1 = new Model { State = "Pays de la Loire" };
        var france_state2 = new Model { State = "Aquitaine" };
        var france_state3 = new Model { State = "Brittany" };
        var france_state4 = new Model { State = "Lorraine" };
```

```
country1.Models.Add(aus_state1);
country1.Models.Add(aus_state2);
country1.Models.Add(aus_state3);
country1.Models.Add(aus_state4);
country2.Models.Add(brazil_state1);
country2.Models.Add(brazil_state2);
country2.Models.Add(brazil_state3);
country3.Models.Add(china_state1);
country3.Models.Add(china_state2);
country3.Models.Add(china_state3);
country3.Models.Add(china_state4);
country4.Models.Add(france_state1);
country4.Models.Add(france_state2);
country4.Models.Add(france_state3);
country4.Models.Add(france_state4);
Items.Add(country1);
Items.Add(country2);
Items.Add(country3);
Items.Add(country4);
SelectedNode = country2;
}
}
```

- ▼ Australia
 - New South Wales
 - Victoria
 - South Autralia
 - Western Australia
- ▼ Brazil
 - Parana
 - Ceara
 - Acre
- ▼ China
 - Guangzhou
 - Shanghai
 - Beijing
 - Shantou
- ▼ France
 - Pays de la Loire
 - Aquitaine
 - Brittany
 - Lorraine

Binding SelectedItems

TreeView support to select multiple items through binding the [SelectedItems](#) property from view model with `ObservableCollection<object>` type.

XML

```
<syncfusion:SfTreeView
x:Name="sfTreeView"
SelectionMode="Multiple"
SelectedItems="{Binding SelectedNodes}"
ChildPropertyName="Models"
ItemsSource="{Binding Items}" >
</syncfusion:SfTreeView>
```

C#

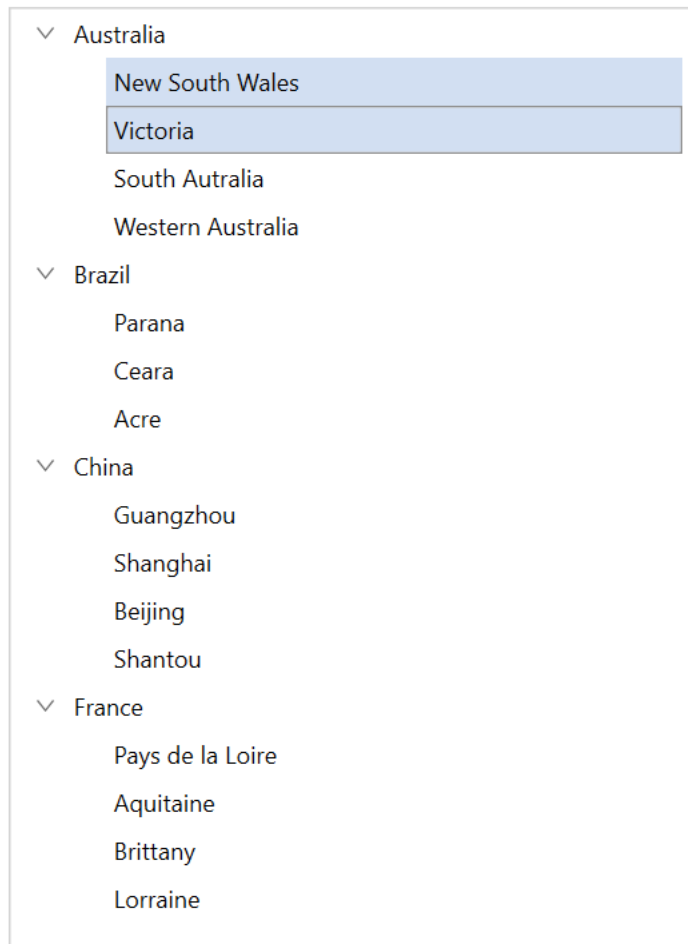
```
sfTreeView.SelectionMode = SelectionMode.Multiple;
sfTreeView.SetBinding(SfTreeView.SelectedItemsProperty, new
Binding("SelectedNodes"));
```

C#

```
public class ViewModel : NotificationObject
{
    public ObservableCollection<Model> Items { get; set; }
    public ObservableCollection<object> SelectedNodes { get; set; }
    public ViewModel()
    {
        Items = new ObservableCollection<Model>();
        SelectedNodes = new ObservableCollection<object>();
        var country1 = new Model { State = "Australia" };
        var country2 = new Model { State = "Brazil" };
        var country3 = new Model { State = "China" };
        var country4 = new Model { State = "France" };
        var aus_state1 = new Model { State = "New South Wales" };
        var aus_state2 = new Model { State = "Victoria" };
        var aus_state3 = new Model { State = "South Australia" };
        var aus_state4 = new Model { State = "Western Australia" };
        var brazil_state1 = new Model { State = "Parana" };
        var brazil_state2 = new Model { State = "Ceara" };
        var brazil_state3 = new Model { State = "Acre" };
        var china_state1 = new Model { State = "Guangzhou" };
        var china_state2 = new Model { State = "Shanghai" };
        var china_state3 = new Model { State = "Beijing" };
        var china_state4 = new Model { State = "Shantou" };
        var france_state1 = new Model { State = "Pays de la Loire" };
        var france_state2 = new Model { State = "Aquitaine" };
        var france_state3 = new Model { State = "Brittany" };
        var france_state4 = new Model { State = "Lorraine" };
        country1.Models.Add(aus_state1);
        country1.Models.Add(aus_state2);
        country1.Models.Add(aus_state3);
        country1.Models.Add(aus_state4);
        country2.Models.Add(brazil_state1);
        country2.Models.Add(brazil_state2);
        country2.Models.Add(brazil_state3);
    }
}
```

```
country3.Models.Add(china_state1);
country3.Models.Add(china_state2);
country3.Models.Add(china_state3);
country3.Models.Add(china_state4);
country4.Models.Add(france_state1);
country4.Models.Add(france_state2);
country4.Models.Add(france_state3);
country4.Models.Add(france_state4);
Items.Add(country1);
Items.Add(country2);
Items.Add(country3);
Items.Add(country4);
SelectedNodes.Add(aus_state1);
SelectedNodes.Add(aus_state2);
}
}
```

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



Event to command

The **TreeView** event can be converted into commands using [Behaviors](#). To achieve this, create a command in the ViewModel class and associate it to the TreeView event using [Behaviors](#).

XML

```

<syncfusion:SfTreeView
x:Name="sfTreeView"
Margin="20"
BorderThickness="1"
SelectionMode="Multiple"
AutoExpandMode="AllNodes"
BorderBrush="LightGray"
ChildPropertyName="Models"
ItemTemplateDataContextType="Node"
ItemsSource="{Binding Items}">
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<Grid>
<TextBlock FontSize="12" VerticalAlignment="Center" Text="{Binding
Content.State}" />
</Grid>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
<I:Interaction.Triggers>
<I:EventTrigger EventName="SelectionChanged" >
<cmd:EventToCommand Command="{Binding SelectionChangedCommand}"
PassEventArgsToCommand="True" />
</I:EventTrigger>
</I:Interaction.Triggers>
</syncfusion:SfTreeView>

```

C#

```

public class ViewModel : NotificationObject
{
    public ObservableCollection<Model> Items { get; set; }
    private ICommand selectionChangedCommand;
    public ICommand SelectionChangedCommand
    {
        get
        {
            return selectionChangedCommand;
        }
        set
        {
            selectionChangedCommand = value;
        }
    }
    public ViewModel()
    {
        Items = new ObservableCollection<Model>();
        SelectionChangedCommand = new BaseCommand(OnSelectionChanged);
        var country1 = new Model { State = "Australia" };
        var country2 = new Model { State = "Brazil" };
        var country3 = new Model { State = "China" };
        var country4 = new Model { State = "France" };
        var aus_state1 = new Model { State = "New South Wales" };
        var aus_state2 = new Model { State = "Victoria" };
        var aus_state3 = new Model { State = "South Australia" };
    }
}

```

```

var aus_state4 = new Model { State = "Western Australia" };
var brazil_state1 = new Model { State = "Parana" };
var brazil_state2 = new Model { State = "Ceara" };
var brazil_state3 = new Model { State = "Acre" };
var china_state1 = new Model { State = "Guangzhou" };
var china_state2 = new Model { State = "Shanghai" };
var china_state3 = new Model { State = "Beijing" };
var china_state4 = new Model { State = "Shantou" };
var france_state1 = new Model { State = "Pays de la Loire" };
var france_state2 = new Model { State = "Aquitaine" };
var france_state3 = new Model { State = "Brittany" };
var france_state4 = new Model { State = "Lorraine" };
country1.Models.Add(aus_state1);
country1.Models.Add(aus_state2);
country1.Models.Add(aus_state3);
country1.Models.Add(aus_state4);
country2.Models.Add(brazil_state1);
country2.Models.Add(brazil_state2);
country2.Models.Add(brazil_state3);
country3.Models.Add(china_state1);
country3.Models.Add(china_state2);
country3.Models.Add(china_state3);
country3.Models.Add(china_state4);
country4.Models.Add(france_state1);
country4.Models.Add(france_state2);
country4.Models.Add(france_state3);
country4.Models.Add(france_state4);
Items.Add(country1);
Items.Add(country2);
Items.Add(country3);
Items.Add(country4);
}
private void OnSelectionChanged(object obj)
{
var args = obj as ItemSelectionChangedEventArgs;
if (args.AddedItems.Count > 0)
MessageBox.Show("Selected State: " + (args.AddedItems[0] as
Model).State.ToString());
}
}

```

Note: View sample in [GitHub](#). You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Load On Demand in WPF TreeView (SfTreeView)

TreeView allows you to load child items only when they are requested using Load on-demand (Lazy load). It helps to load the child items from services when end-user expands the node. Initially populate the root [Nodes](#) by assigning [ItemsSource](#) and then when any node is expanded, child items can be loaded using [LoadOnDemandCommand](#). Load on-demand is applicable for bound mode only.

XML

```

<Window
x:Class="LoadOnDemandDemo.LoadOnDemand"

```

```

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:local="clr-namespace:LoadOnDemandDemo"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="Load On Demand"
mc:Ignorable="d">
<Window.DataContext>
<local:MusicInfoRepository />
</Window.DataContext>
<Grid>
<syncfusion:SfTreeView
x:Name="sfTreeView"
Margin="10"
AllowDragging="True"
BorderBrush="LightGray"
BorderThickness="1"
ExpandActionTrigger="Node"
FocusVisualStyle="{x:Null}"
IsAnimationEnabled="True"
ItemHeight="30"
LoadOnDemandCommand="{Binding TreeViewOnDemandCommand}"
ItemsSource="{Binding Menu}" >
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<Label
VerticalContentAlignment="Center"
Content="{Binding ItemName}"
FocusVisualStyle="{x:Null}"
FontSize="12" />
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>
</Grid>
</Window>

```

C#

```

/// <summary>
/// ViewModel class that implements <see cref="Command"/> for load on
demand.
/// </summary>
public class MusicInfoRepository
{
    DispatcherTimer timer;
    TreeViewNode currentNode;
    private ObservableCollection<MusicInfo> menu;
    public ObservableCollection<MusicInfo> Menu
    {
        get { return menu; }
        set { menu = value; }
    }
    private ICommand treeViewOnDemandCommand;
    public ICommand TreeViewOnDemandCommand
    {

```

```

get{ return treeViewOnDemandCommand; }
set{ treeViewOnDemandCommand = value; }
}
public MusicInfoRepository()
{
    timer = new DispatcherTimer();
    timer.Interval = new TimeSpan(0, 0, 0, 0, 2000);
    timer.Tick += Timer_Tick;
    this.Menu = this.GetMenuItems();
    TreeViewOnDemandCommand = new OnDemandCommand(ExecuteOnDemandLoading,
    CanExecuteOnDemandLoading);
}
private void Timer_Tick(object sender, EventArgs e)
{
    MusicInfo musicInfo = currentNode.Content as MusicInfo;
    //Fetching child items to add
    var items = GetSubMenu(musicInfo.ID);
    // Populating child items for the node in on-demand
    currentNode.PopulateChildNodes(items);
    if (items.Count() > 0)
    //Expand the node after child items are added.
    currentNode.IsExpanded = true;
    //Stop the animation after load on demand is executed, if animation not
    stopped, it remains still after execution of load on demand.
    currentNode.ShowExpanderAnimation = false;
    timer.Stop();
}
/// <summary>
/// CanExecute method is called before expanding and initialization of node.
/// Returns whether the node has child nodes or not.
/// Based on return value, expander visibility of the node is handled.
/// </summary>
/// <param name="sender">TreeViewNode is passed as default parameter
</param>
/// <returns>Returns true, if the specified node has child items to load on
demand and expander icon is displayed for that node, else returns false and
icon is not displayed.</returns>
private bool CanExecuteOnDemandLoading(object sender)
{
    var hasChildNodes = ((sender as TreeViewNode).Content as
    MusicInfo).HasChildNodes;
    if (hasChildNodes)
    return true;
    else
    return false;
}
/// <summary>
/// Execute method is called when any item is requested for load-on-demand
items.
/// </summary>
/// <param name="obj">TreeViewNode is passed as default parameter </param>
private void ExecuteOnDemandLoading(object obj)
{
    var node = obj as TreeViewNode;
    // Skip the repeated population of child items when every time the node
    expands.
    if (node.ChildNodes.Count > 0)

```



```

{
    node.IsExpanded = true;
    return;
}
//Animation starts for expander to show progressing of load on demand
node.ShowExpanderAnimation = true;
var treeView = Application.Current.MainWindow.FindName("sfTreeView") as
SfTreeView;
treeView.Dispatcher.BeginInvoke(DispatcherPriority.ApplicationIdle, new
Action(() =>
{
    currentNode = node;
    timer.Start();
}));
}
private ObservableCollection<MusicInfo> GetMenuItems()
{
    ObservableCollection<MusicInfo> menuItems = new
    ObservableCollection<MusicInfo>();
    menuItems.Add(new MusicInfo() { ItemName = "Discover Music", HasChildNodes =
    true, ID = 1 });
    menuItems.Add(new MusicInfo() { ItemName = "Sales and Events", HasChildNodes
    = true, ID = 2 });
    menuItems.Add(new MusicInfo() { ItemName = "Categories", HasChildNodes =
    true, ID = 3 });
    menuItems.Add(new MusicInfo() { ItemName = "MP3 Albums", HasChildNodes =
    true, ID = 4 });
    menuItems.Add(new MusicInfo() { ItemName = "Fiction Book Lists",
    HasChildNodes = true, ID = 5 });
    return menuItems;
}
public IEnumerable<MusicInfo> GetSubMenu(int id)
{
    ObservableCollection<MusicInfo> menuItems = new
    ObservableCollection<MusicInfo>();
    if (id == 1)
    {
        menuItems.Add(new MusicInfo() { ItemName = "Hot Singles", HasChildNodes =
        false, ID = 11 });
        menuItems.Add(new MusicInfo() { ItemName = "Rising Artists", HasChildNodes =
        false, ID = 12 });
        menuItems.Add(new MusicInfo() { ItemName = "Live Music", HasChildNodes =
        false, ID = 13 });
        menuItems.Add(new MusicInfo() { ItemName = "More in Music", HasChildNodes =
        true, ID = 14 });
    }
    else if (id == 2)
    {
        menuItems.Add(new MusicInfo() { ItemName = "100 Albums - $10 Each",
        HasChildNodes = false, ID = 21 });
        menuItems.Add(new MusicInfo() { ItemName = "Hip-Hop and R&B Sale",
        HasChildNodes = false, ID = 22 });
        menuItems.Add(new MusicInfo() { ItemName = "CD Deals", HasChildNodes =
        false, ID = 23 });
    }
    else if (id == 3)
    {

```

```

menuItems.Add(new MusicInfo() { ItemName = "Songs", HasChildNodes = false,
ID = 31 });
menuItems.Add(new MusicInfo() { ItemName = "Bestselling Albums",
HasChildNodes = false, ID = 32 });
menuItems.Add(new MusicInfo() { ItemName = "New Releases", HasChildNodes =
false, ID = 33 });
menuItems.Add(new MusicInfo() { ItemName = "MP3 Albums", HasChildNodes =
false, ID = 34 });
}
else if (id == 4)
{
menuItems.Add(new MusicInfo() { ItemName = "Rock Music", HasChildNodes =
false, ID = 41 });
menuItems.Add(new MusicInfo() { ItemName = "Gospel", HasChildNodes = false,
ID = 42 });
menuItems.Add(new MusicInfo() { ItemName = "Latin Music", HasChildNodes =
false, ID = 43 });
menuItems.Add(new MusicInfo() { ItemName = "Jazz", HasChildNodes = false, ID
= 44 });
}
else if (id == 5)
{
menuItems.Add(new MusicInfo() { ItemName = "Hunger Games", HasChildNodes =
false, ID = 51 });
menuItems.Add(new MusicInfo() { ItemName = "Pride and Prejudice",
HasChildNodes = false, ID = 52 });
menuItems.Add(new MusicInfo() { ItemName = "Harry Potter", HasChildNodes =
false, ID = 53 });
menuItems.Add(new MusicInfo() { ItemName = "Game Of Thrones", HasChildNodes
= false, ID = 54 });
}
else if (id == 14)
{
menuItems.Add(new MusicInfo() { ItemName = "Music Trade-In", HasChildNodes =
false, ID = 141 });
menuItems.Add(new MusicInfo() { ItemName = "Redeem a Gift card",
HasChildNodes = false, ID = 142 });
menuItems.Add(new MusicInfo() { ItemName = "Band T-Shirts", HasChildNodes =
false, ID = 143 });
}
return menuItems;
}
}

```

C#

```

/// <summary>
/// Model
/// </summary>
public class MusicInfo : NotificationObject
{
#region Fields
public string itemName;
public int id;
public bool hasChildNodes;
#endregion
}

```

```

#region Properties
public string ItemName
{
    get { return itemName; }
    set
    {
        itemName = value;
        RaisePropertyChanged("ItemName");
    }
}
public int ID
{
    get { return id; }
    set
    {
        id = value;
        RaisePropertyChanged("ID");
    }
}
public bool HasChildNodes
{
    get { return hasChildNodes; }
    set
    {
        hasChildNodes = value;
        RaisePropertyChanged("HasChildNodes");
    }
}
#endregion
}

```

Note: LoadOnDemandCommand receives [TreeViewNode](#) as command parameter by default.

Handling expander visibility

TreeView shows the expander for a particular node based on return value of [CanExecute](#) method of [LoadOnDemandCommand](#). If [CanExecute](#) returns `true`, then expander icon is displayed for that node. If [CanExecute](#) returns `false`, then expander icon will not displayed for that node. [CanExecute](#) method gets called to decide the visibility of expander icon and before executing [LoadOnDemandCommand](#).

C#

```

/// <summary>
/// CanExecute method is called before expanding and initialization of node.
/// Returns whether the node has child nodes or not.
/// Based on return value, expander visibility of the node is handled.
/// </summary>
/// <param name="sender">TreeViewNode is passed as default parameter
</param>
/// <returns>Returns true, if the specified node has child items to load on
demand and expander icon is displayed for that node, else returns false and
icon is not displayed.</returns>
private bool CanExecuteOnDemandLoading(object sender)
{
    var hasChildNodes = ((sender as TreeViewNode).Content as
    MusicInfo).HasChildNodes;
}

```

```
if (hasChildNodes)
return true;
else
return false;
}
```

On-demand loading of child items

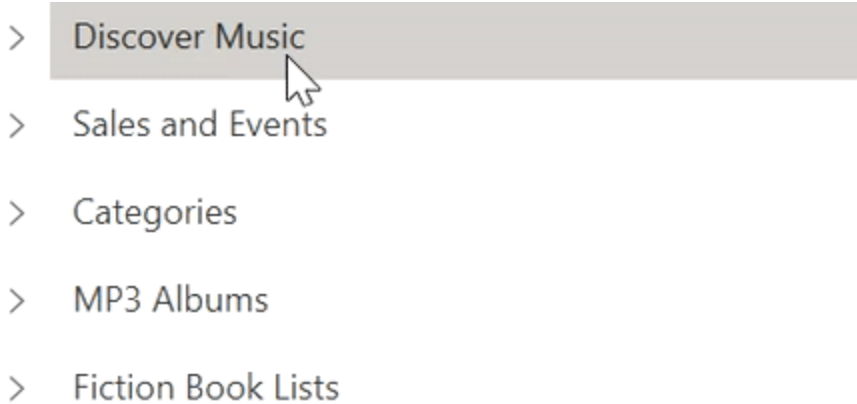
You can load child items for the node in [Execute](#) method of `LoadOnDemandCommand`. `Execute` method will get called when user expands the tree node. In `LoadOnDemand.Execute` method, you have can perform following operations,

- Show or hide busy indicator in the place of expander by setting [TreeViewNode.ShowExpanderAnimation](#) until the data fetched.
- Once data fetched, you can populate the child nodes by calling [TreeViewNode.PopulateChildNodes](#) method by passing the child items collection.
- When load on-demand command executes expanding operation will not be handled by `TreeView`. So, you have to set [TreeViewNode.IsExpanded](#) property to `true` to expand the tree node after populating child nodes.
- You can skip population of child items again and again when every time the node expands, based on [TreeViewNode.ChildNodes](#) count.

C#

```
/// <summary>
/// Execute method is called when any item is requested for load-on-demand
/// items.
/// </summary>
/// <param name="obj">TreeViewNode is passed as default parameter </param>
private void ExecuteOnDemandLoading(object obj)
{
    var node = obj as TreeViewNode;
    // Skip the repeated population of child items when every time the node
    // expands.
    if (node.ChildNodes.Count > 0)
    {
        node.IsExpanded = true;
        return;
    }
    //Animation starts for expander to show progressing of load on demand
    node.ShowExpanderAnimation = true;
    var sfTreeView = Application.Current.MainWindow.FindName("sfTreeView") as
    SfTreeView;
    sfTreeView.Dispatcher.BeginInvoke(DispatcherPriority.ApplicationIdle, new
    Action(() =>
    {
        currentNode = node;
        timer.Start();
    }));
}
```

Note: View sample in [GitHub](#)

- 
- > Discover Music
 - > Sales and Events
 - > Categories
 - > MP3 Albums
 - > Fiction Book Lists

Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Item Height Customization in WPF TreeView (SfTreeView)

The TreeView provides various options to customize the height of items. To achieve this customization, please walk through the below sections:

Customize Item Height

The TreeView allows customizing the height of items by setting the [ItemHeight](#) property. The default value of this property is 24. This property can be customized at runtime.

XML

```
<syncfusion:SfTreeView x:Name="sfTreeView" ItemHeight="30" />
```

C#

```
sfTreeView.ItemHeight = 30;
```

Customize Item height using `QueryNodeSize` event

The TreeView allows customizing the height of the items using [QueryNodeSize](#) event. This event is raised whenever the item comes into view and triggered with [QueryNodeSizeEventArgs](#).

The `SfTreeView.QueryNodeSize` event provides the following arguments:

- [Node](#): This argument contains the [TreeViewNode](#) and data associated with it.
- [Height](#): This argument contains the default item height of the queried item and can be set with desired size.
- [Handled](#): Decides whether the specified or measured height can be set to the item or not. The default value is `false`. When this property is not set, the decided height will not set to the item.

Customize specific item height using custom value

The TreeView allows customizing the height of the specific item by setting the custom value directly to the [Height](#) argument which is available in [QueryNodeSizeEventArgs](#).

XML

```
<syncfusion:SfTreeView
x:Name="sfTreeView"
Margin="10"
QueryNodeSize="SfTreeView_QueryNodeSize"
ExpandActionTrigger="Node"
ItemsSource="{Binding Menu}" />
```

C#

```
sfTreeView.QueryNodeSize += SfTreeView_QueryNodeSize;
private void SfTreeView_QueryNodeSize(object sender,
Syncfusion.UI.Xaml.TreeView.QueryNodeSizeEventArgs e)
{
    if (e.Node.Level == 0)
    {
        //Returns speified item height for items.
        e.Height = 50;
        e.Handled = true;
    }
}
```

Autofit item height based on content

The TreeView allows adjusting height of items based on the content measured size while loaded by setting the `Height` argument with value returned from [QueryNodeSizeEventArgs.GetAutoFitNodeHeight](#) method.

XML

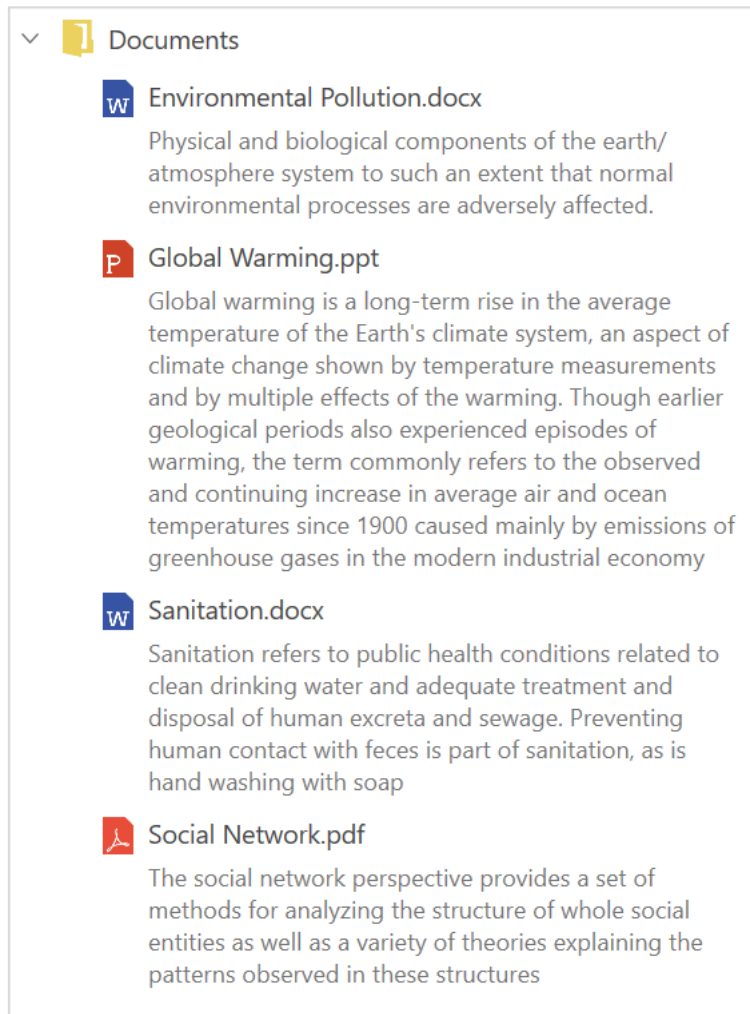
```
<syncfusion:SfTreeView
x:Name="sfTreeView"
Margin="10"
QueryNodeSize="SfTreeView_QueryNodeSize"
```

```
ExpandActionTrigger="Node"  
ItemsSource="{Binding Menu}" />
```

C#

```
sfTreeView.QueryNodeSize += SfTreeView_QueryNodeSize;  
private void SfTreeView_QueryNodeSize(object sender,  
Syncfusion.UI.Xaml.TreeView.QueryNodeSizeEventArgs e)  
{  
    if (e.Node.Level == 0)  
    {  
        //Returns specified item height for items.  
        e.Height = 30;  
        e.Handled = true;  
    }  
    else  
    {  
        // Returns item height based on the content loaded.  
        e.Height = e.GetAutoFitNodeHeight();  
        e.Handled = true;  
    }  
}
```

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



Limitations

- Define the size of the image when loading image in the [SfTreeView.ItemTemplate](#). Because, it does not return actual measured size when measuring before item layout.

Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

Right to left(RTL) in WPF TreeView (SfTreeView)

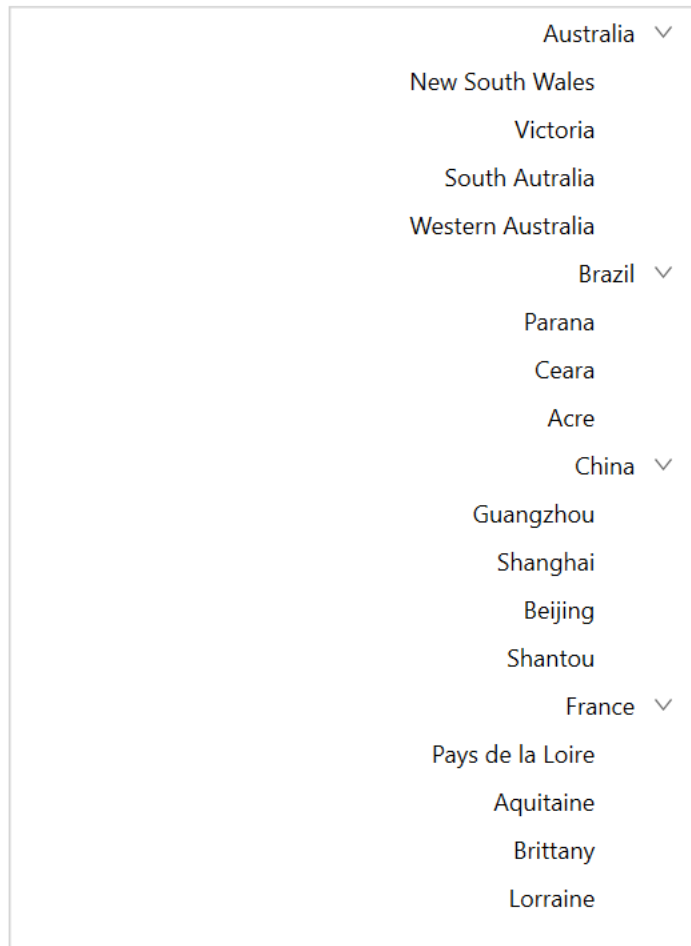
TreeView supports to change the flow of text to the right-to-left direction by setting the [FlowDirection](#) to `RightToLeft`.

XML

```
<syncfusion:SfTreeView x:Name="sfTreeView" FlowDirection="RightToLeft"/>
```

C#

```
sfTreeView.FlowDirection = FlowDirection.RightToLeft;
```

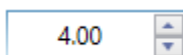



Note: You can refer to our [WPF TreeView](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF TreeView example](#) to know how to represent hierarchical data in a tree-like structure with expand and collapse node options.

UpDown

WPF NumericUpDown (UpDown) Overview

The WPF [UpDown](#) control displays numeric values. The value can be edited by scrolling the values and by using the Increment and Decrement buttons of the UpDown control. It allows to define maximum and minimum values to which the user can increment/decrement the input's value. Also can specify the interval, that will be applied to the value upon each increase/decrease.



Features

Value - Provides different set of values. The values are Minimum and Maximum value.

Null value - Provides option to set the null value.

Editing - Provides option to change the value by up and down and also edit the text part.

Culture - Provides different culture support based on NumberDecimalSeparator in UpDown.

Animation - Provides animation support for speed of the UpDown control.

Keyboard and Mouse support - Provide option to change the value by using keyboard and mouse.

Appearance - Provides support for several built-in skins and blendable support for customize the appearance.

Getting Started with WPF NumericUpDown (UpDown)

This section explains how to create [WPF NumericUpDown](#) (UpDown) control and its structure.

Structure of UpDown



The following are the elements of the UpDown control:

- **Text area** - It is the area where the numeric values are displayed.
- **Increment button** - It is a repeat button that can be clicked to increment the current value of the UpDown control.
- **Decrement button** - It is a repeat button that can be clicked to decrement the current value of the UpDown control.

Assembly deployment

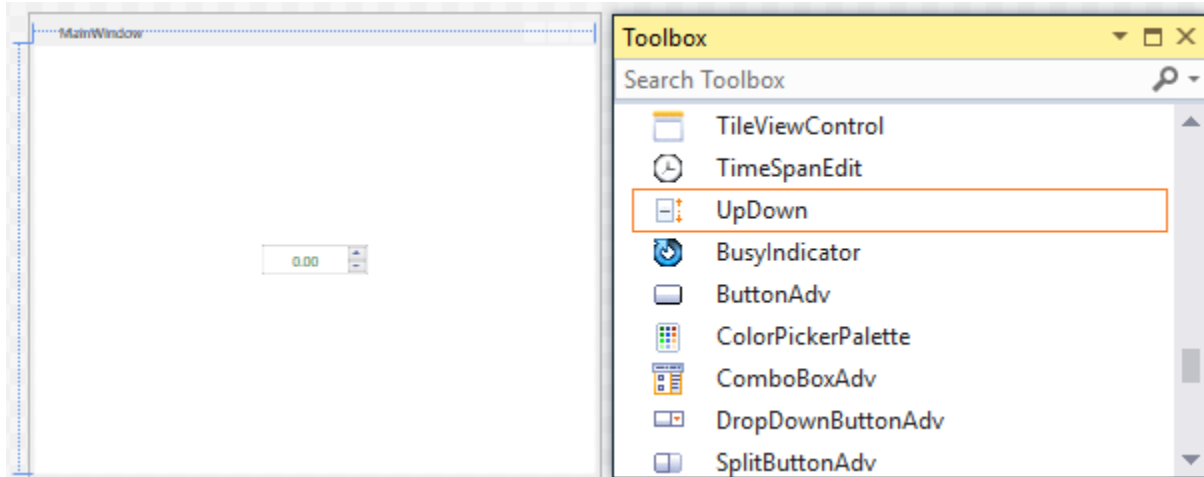
Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

[Click here](#) to find more details on how to install nuget packages in WPF application.

Adding WPF UpDown control via designer

The [UpDown](#) control can be added to an application by dragging it from the toolbox to design view. The following dependent assembly will be added automatically.

- Syncfusion.Shared.WPF



Adding WPF UpDown control via XAML

In order to add the [UpDown](#) control manually in XAML, do the below steps,

- 1) Create a new WPF project in Visual Studio. 2) Add the following required assembly reference to the project. * Syncfusion.Shared.WPF 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> and declare the UpDown control in XAML page.

XML

```
<Window x:Class="Application_New.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525">
<Grid Name="grid">
<syncfusion:UpDown Name="upDown" Width="100" Height="23"/>
</Grid>
</Window>
```



Adding WPF UpDown control via C#

In order to add the UpDown control manually in C#, do the below steps,

- 1) Create a new WPF application via Visual Studio. 2) Add the following required assembly reference to the project. * Syncfusion.Shared.WPF 3) Import UpDown namespace **Syncfusion.Windows.Shared**. 3) Create an instance of UpDown control and add it to the main window.

C#

```
UpDown updown = new UpDown();  
updown.Width = 100;  
updown.Height = 23;  
grid.Children.Add(updown);
```



Value

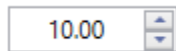
The [Value](#) property is used for set the value to UpDown control.

XML

```
<syncfusion:UpDown Name="upDown" Height="23" Value="10" Width="85"/>
```

C#

```
updown.Value = 10;
```



Step Value

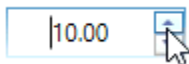
The [Step](#) property is used to specify the interval to increase or decrease the value while pressing the spin or repeat buttons in the UpDown control.

XML

```
<syncfusion:UpDown Name="upDown" Height="25" Value="10" Step="5" Width="90"/>
```

C#

```
updown.Value = 10;  
updown.Step = 5;
```



Number formatting

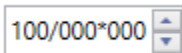
The number formatting of UpDown control can be customized by setting [UpDown.NumberFormatInfo](#) property by specifying the culture-specific group separator, decimal separator, and the number of decimal digits. You can show the group separator by enable the [GroupSeparatorEnabled](#) property.

XML

```
<Window x:Class="Application_New.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:globalization="clr-namespace:System.Globalization;assembly=mscorlib"
Title="MainWindow" Height="350" Width="525" >
<syncfusion:UpDown Name="upDown" Height="25" Width="90" Value="100000"
GroupSeparatorEnabled="True">
<syncfusion:UpDown.NumberFormatInfo>
<globalization:NumberFormatInfo NumberGroupSeparator="/"
NumberDecimalDigits="3" NumberDecimalSeparator="*" />
</ syncfusion:UpDown.NumberFormatInfo>
</ syncfusion:UpDown>
```

C#

```
//Assign a value
updown.Value = 100000;
//Initialize numberformatinfo
NumberFormatInfo numberFormatInfo = new NumberFormatInfo();
// set the format of number and group
updown.GroupSeparatorEnabled = true;
updown.NumberFormatInfo = numberFormatInfo;
updown.NumberFormatInfo.NumberGroupSeparator = "/";
updown.NumberFormatInfo.NumberDecimalDigits = 3;
updown.NumberFormatInfo.NumberDecimalSeparator = "*";
```



For more number formatting in UpDown control, you can use the [Culture](#) property. The **Culture** property is used to format the values based on the respective culture.

XML

```
<syncfusion:UpDown Name="upDown" Height="25" Width="90" Value="100000"
Culture="bs-Latn" GroupSeparatorEnabled="True">
</syncfusion:UpDown>
```

C#

```
CultureInfo cultureInfo = new CultureInfo("bs-Latn");
updown.Culture = cultureInfo;
```



Theme

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

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



Restriction in WPF NumericUpDown (UpDown)

This section explains about how to set the value and restrict the minimum and maximum value in WPF UpDown control.

Value

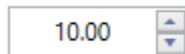
The [Value](#) property is used for set the value to UpDown control.

XML

```
<syncfusion:UpDown Name="upDown" Height="25" Width="90" Value="10" />
```

C#

```
UpDown updown = new UpDown();
updown.Height = 25;
updown.Width = 90;
updown.Value = 10;
grid.Children.Add(updown);
```



Value event

The UpDown control notifies the value changes through [ValueChanged](#) and [ValueChanging](#) events. You can use the `OldValue` and `NewValue` property to get the old and new value in `ValueChanged` event. In `ValueChanging` event, you can use the `Cancel` property in event argument to avoid the changes.

C#

```
updown.ValueChanged += Up_ValueChanged;
updown.ValueChanging += Up_ValueChanging;
private void Up_ValueChanging(object sender, ValueChangingEventArgs e)
{
    // To cancel the changing value
    e.Cancel = true;
}
private void Up_ValueChanged(DependencyObject d,
    DependencyPropertyPropertyChangedEventArgs e)
{
    // Get old and new value
    var newValue = e.NewValue;
    var oldValue = e.OldValue;
}
```

Null value

The [UpDown](#) control accepts null values. When the [Value](#) is set to null, the UpDown control will show zero value by default. You can change this to display some other numerical value using

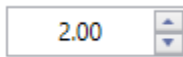
[NullValue](#) property. The [UseNullOption](#) property must be enabled to see the `NullValue` specified.

XML

```
<syncfusion:UpDown Name="upDown" Height="25" Width="90" Value="{x:Null}"
NullValue="2" UseNullOption="True" />
```

C#

```
updown.UseNullOption = true;
updown.Value = null;
updown.NullValue = 2;
```



Watermark

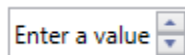
The [NullValueText](#) property enables the UpDown control to show watermark text instead of numeric value when [Value](#) is null. The [UseNullOption](#) property must be enabled to see the [NullValueText](#) specified.

XML

```
<syncfusion:UpDown Name="upDown" Height="25" Width="90" Value="{x:Null}"
NullValueText="Enter a value" UseNullOption="True" />
```

C#

```
updown.Value = null;
updown.NullValueText = "Enter a value";
```



Note: The [UseNullOption](#) property must be enabled if you want to see the [NullValue](#) or [NullValueText](#) in UpDown control. If both [NullValue](#) and [NullValueText](#) is specified, you will see only [NullValue](#) but not [NullValueText](#).

Minimum and Maximum value

The value of UpDown control can be restricted within maximum and minimum limit. The spin button helps to increase or decrease the value by using mouse interaction. Once the increase or decrease value reached the predefined maximized or minimized value, the value does not change.

Another way,

By using the keyboard, you can not enter the value above or below the predefined maximized or minimized value.

- **MaxValue** - The [MaxValue](#) property is the maximum value that can be set for the UpDown control.
- **MinValue** - The [MinValue](#) property is the minimum value that can be set for the UpDown control.

XML

```
<syncfusion:UpDown Name="upDown" Height="25" Width="90" MinValue="0"
MaxValue="100" />
```

C#

```
updown.MaxValue = 100;
updown.MinValue = 0;
```



Minimum and Minimum validation

You can choose when to validate the maximum and minimum limit, while changing the values by using [MaxValidation](#) and [MinValidation](#) property.

- **OnKeyPress** - On setting the [MaxValidation](#) or [MinValidation](#) to OnKeyPress, then the value in the UpDown control is validated soon after a key is pressed. So, it is not possible to provide any invalid input at all.
- **OnLostFocus** - On setting the [MaxValidation](#) or [MinValidation](#) to OnLostFocus, then the value in the UpDown control is validated when the UpDown control loses focus. That is, the **UpDown** will accept any value, validation will happen only after control loses its keyboard focus. After validation, when the value of the UpDown control is greater than the MaxValue or lesser than the MinValue, the value will be changed automatically is set to MaxValue or MinValue respectively.
- **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 enabled. When **MaxValueOnExceedMaxDigit** is false, 20 will be retained and last entered 0 will be ignored.
- **MinValueOnExceedMinDigit** - Similarly, When you give input lesser than specified minimum limit, **MinValueOnExceedMinDigit** property will decide either it should retain the old value or reset to minimum limit that is specified.

XML

```
<syncfusion:UpDown Name="upDown" Height="25" Width="90"
MinValueOnExceedMinDigit="True" MaxValueOnExceedMaxDigit="True"
MaxValidation="OnKeyPress" MinValidation="OnKeyPress" MinValue="0"
MaxValue="100" />
```

C#

```
updown.MaxValidation = MaxValidation.OnKeyPress;
updown.MinValidation = MinValidation.OnKeyPress;
updown.MinValueOnExceedMinDigit = true;
updown.MaxValueOnExceedMaxDigit = true;
```


AllowEdit

The [AllowEdit](#) property is used to restrict the editing in **UpDown** control by setting its value to **False**. The default value is **True**.

XML

```
<syncfusion:UpDown Name="upDown" Height="25" Width="90" AllowEdit="False"
MinValue="0" MaxValue="100" />
```

C#

```
updown.AllowEdit = false;
```

Interaction in WPF NumericUpDown (UpDown)

This section explains about how to change the value by using mouse and keyboard in WPF [UpDown](#) control.

Keyboard and Mouse support

The **UpDown** control allows to increase or decrease the value by pressing up-arrow and down-arrow keys in keyboard or mouse wheel over the control. The [Step](#) property is used to specify the interval of increment or decrement.

Increment or decrement value in mouse wheel

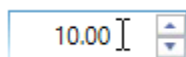
You can increase or decrease the current value by scrolling over **UpDown** control. You can enable it by setting the [IsScrollingOnCircle](#) property as **true**. You can disable the value changing on mouse scrolling by using the [IsScrollingOnCircle](#) property as **false**. The default value of [IsScrollingOnCircle](#) property is **true**.

XML

```
<syncfusion:UpDown Name="upDown" Width="100" Height="23"
IsScrollingOnCircle="True" />
```

C#

```
UpDown updown = new UpDown();
updown.Value = 10;
updown.IsScrollingOnCircle = true;
grid.Children.Add(updown);
```



Step

The [Step](#) property is used to specify the interval to increase or decrease the value while pressing the spin buttons in the UpDown control. For example, the **Step** value is set to 5 so that the **UpDown** control value increases or decreases by 5 while pressing the spin buttons.

Another way,

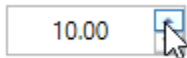
You can also increase or decrease the present value of UpDown control by using up-arrow and down-arrow keys in keyboard.

XML

```
<syncfusion:UpDown Name="upDown" Value="10" Width="100" Height="23"
Step="5"/>
```

C#

```
updown.Value = 10;
updown.Width = 100;
updown.Height = 23;
updown.Step = 5;
```



Animation speed

When a value change in the UpDown control by using the repeated buttons, the transition from the current value to the new value is animated in UpDown control. The animation speed can be controlled by using [AnimationSpeed](#) property.

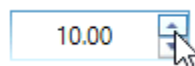
Note: Specifying whether its 0 to 1 seconds or milliseconds.

XML

```
<syncfusion:UpDown Name="upDown" Value="10" AnimationSpeed="0.5" Width="100"
Height="23"/>
```

C#

```
updown.Value = 10;
updown.Width = 100;
updown.Height = 23;
updown.AnimationSpeed = 0.5;
```



Range Adorner

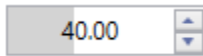
You can show the adorner over UpDown control based on the minimum and maximum values by setting [EnableRangeAdorner](#) property to `true`. The default value is `false`. You can also change the background color of the range adorner using [RangeAdornerBackground](#) property.

XML

```
<syncfusion:UpDown Name="upDown" Height="25" Width="90" Value="40"
RangeAdornerBackground="Gray" EnableRangeAdorner="True" MinValue="0"
MaxValue="100" />
```

C#

```
updown.MinValue = 0;
updown.MaxValue = 100;
updown.EnableRangeAdorner = true;
updown.RangeAdornerBackground = Brushes.Gray;
```



Number Formatting in WPF NumericUpDown (UpDown)

This section explains how to format the value in WPF UpDown control.

Decimal digit

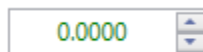
The [NumberDecimalDigits](#) property is used to specify the number of digits to be displayed after the decimal point in the [UpDown](#) control.

XML

```
<syncfusion:UpDown Name="upDown" Height="23" Width="100"
NumberDecimalDigits="4" />
```

C#

```
UpDown updown = new UpDown();
updown.Height = 23;
updown.Width = 100;
updown.NumberDecimalDigits = 4;
grid.Children.Add(updown);
```



Group separator

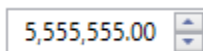
The group separator is the character used to group the values. You can show the group separator in UpDown control by enable [GroupSeparatorEnabled](#) property. The default value is `False`.

XML

```
<syncfusion:UpDown Name="upDown" Width="100" Height="23" Value="5555555"
GroupSeparatorEnabled="True" />
```

C#

```
updown.Value = 5555555;
updown.GroupSeparatorEnabled = true;
```



NumberFormatInfo

The number formatting of UpDown control can be customized by setting [UpDown.NumberFormatInfo](#) property by specifying the culture-specific group separator, decimal separator, and the number of decimal digits. You can show the group separator by enable the [GroupSeparatorEnabled](#) property.

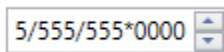
XML

```
<syncfusion:UpDown Name="upDown" Value="5555555"
GroupSeparatorEnabled="True">
<syncfusion:UpDown.NumberFormatInfo>
```

```
<globalization:NumberFormatInfo NumberGroupSeparator="/"
NumberDecimalDigits="4" NumberDecimalSeparator="*" />
</ syncfusion:UpDown.NumberFormatInfo>
</ syncfusion:UpDown>
```

C#

```
//Assign a value
updown.Value = 5555555;
//Initialize numberformatinfo
NumberFormatInfo numberFormatInfo = new NumberFormatInfo();
// set the format of number and group
updown.GroupSeparatorEnabled = true;
updown.NumberFormatInfo = numberFormatInfo;
updown.NumberFormatInfo.NumberGroupSeparator = "/";
updown.NumberFormatInfo.NumberDecimalDigits = 4;
updown.NumberFormatInfo.NumberDecimalSeparator = "*";
```

**Culture**

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

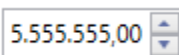
For example, the Latin culture is used into the UpDown control.

XML

```
<syncfusion:UpDown Name="upDown" Culture="bs-Latn" Value="5555555"
Width="100" Height="23" />
```

C#

```
System.Globalization.CultureInfo cultureInfo = new
System.Globalization.CultureInfo("bs-Latn");
updown.Culture = cultureInfo;
updown.Value = 5555555;
```

**Text alignment**

You can align the text by using the [TextAlignment](#) property.

XML

```
<syncfusion:UpDown Value="40" Height="23" TextAlignment="Left" Width="100"
/>
```

C#

```
updown.TextAlignment = TextAlignment.Left;
```



Styles and Templates in WPF NumericUpDown (UpDown)

The background and foreground of the [UpDown](#) control can be customized by editing its style or by using the properties exposed by the [UpDown](#) control.

Positive color

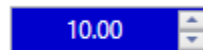
The background and foreground for the positive value can be customized using the [Background](#) and [Foreground](#) properties of the UpDown control.

XML

```
<syncfusion:UpDown Name="upDown" Background="MediumBlue" Foreground="White"
Value="10" Height="23" Width="100" />
```

C#

```
UpDown updown = new UpDown();
updown.Height = 23;
updown.Width = 100;
updown.Value = 10;
updown.Background = Brushes.MediumBlue;
updown.Foreground = Brushes.White;
grid.Children.Add(updown);
```



Negative color

The background and foreground for the negative value can be customized using the [NegativeBackground](#) and [NegativeForeground](#) properties of UpDown control. The [NegativeBackground](#) and [NegativeForeground](#) properties are enabled by setting the [EnableNegativeColors](#) property is set to [True](#).

XML

```
<syncfusion:UpDown Name="upDown" EnableNegativeColors="True"
NegativeBackground="Yellow" NegativeForeground="BlueViolet" Value="-2"
Height="23" Width="100" />
```

C#

```
updown.Value = -2;
updown.EnableNegativeColors = true;
updown.NegativeBackground = Brushes.Yellow;
updown.NegativeForeground = Brushes.BlueViolet;
```



Zero color

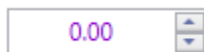
The color of zero value can be changed by using the [ZeroColor](#) property. The [ZeroColor](#) property can be enabled by setting the [ApplyZeroColor](#) property is set to **True**.

XML

```
<syncfusion:UpDown Name="upDown" ApplyZeroColor="True"
ZeroColor="DarkViolet" Value="0" Height="23" Width="100" />
```

C#

```
updown.ApplyZeroColor = true;
updown.ZeroColor = Brushes.DarkViolet;
```



Focused color

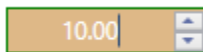
The background, foreground and border color for the UpDown control can be customized using the [FocusedBackground](#), [FocusedBorderBrush](#) and [FocusedForeground](#) properties when the control is focused. These properties will work only when value of [EnableFocusedColors](#) property is **True**. By default, the value of [EnableFocusedColors](#) property is **True**.

XML

```
<syncfusion:UpDown Name="upDown" EnableFocusedColors="True"
FocusedBackground="BurlyWood" FocusedForeground="White"
FocusedBorderBrush="Green" Value="10" Height="23" Width="100" />
```

C#

```
updown.Value = 10;
updown.EnableFocusedColors = true;
updown.FocusedBackground = Brushes.BurlyWood;
updown.FocusedForeground = Brushes.White;
updown.FocusedBorderBrush = Brushes.Green;
```

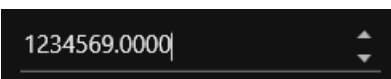


Note: The positive, negative and zero value colors get default color of the control while focusing the Updown control.

Theme

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

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



Wizard Control

WPF Wizard Control Overview

Wizard Control is similar to the wizard displayed in the installation process. Wizard control contains the Wizard Page which is used to define the wizard pages. You can switch between the pages by using the Next button, Back button, and so on. Navigation to the wizard page and look and feel is fully customizable.

Features

- Wizard controls contains the Wizard Page which is used to add wizard pages.
- Each wizard page has the Next, Close, Back, Help and Finish buttons for navigating through the wizard pages.
- Provides support to customize the look and feel.

Getting Started with WPF Wizard Control

This section gives a quick overview for working with the [WizardControl](#).

Highlighting features

You can find some important features of [WizardControl](#) below.

- WizardControl contains the [WizardPage](#), which is used to add multiple pages.
- Each wizard page has the Next, Close, Back, Help and Finish buttons for navigating between the wizard pages.
- Allows you to customize the appearance of WizardControl and WizardPage.

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 [WizardControl](#) control in any application.

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

[How to install nuget packages](#)

Creating Application with WizardControl

In this walk through, user will create a WPF application that contains [WizardControl](#) control.

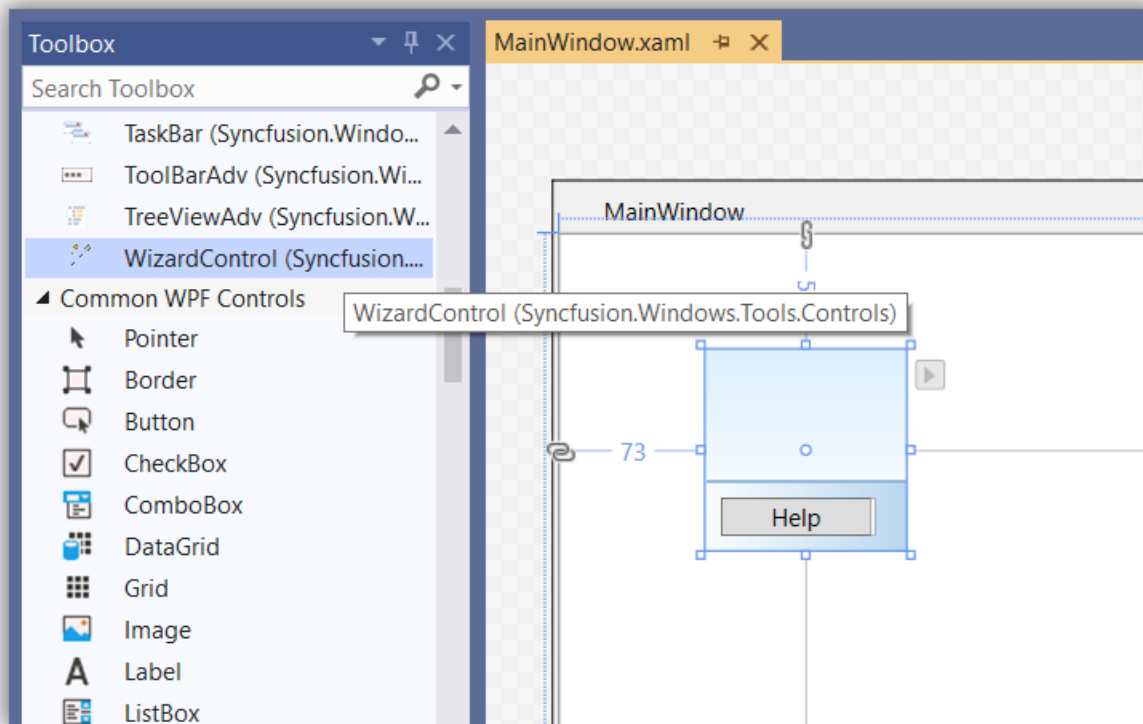
1. [Creating project](#)
2. [Adding control via designer](#)
3. [Adding control manually in XAML](#)
4. [Adding control manually in C#](#)
5. [Creating Data Model for sample application](#)
6. [Binding to Data](#)

Creating project

Below section provides detailed information to create new project in Visual Studio to display [WizardControl](#).

Adding control via designer

The [WizardControl](#) can be added to the application by dragging it from Toolbox and dropping it in designer. The required [assemblies](#) will be added automatically.



Adding control manually in XAML

In order to add [WizardControl](#) control manually in XAML, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.Shared.Wpf
 - Syncfusion.Tools.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page.
3. Declare [WizardControl](#) 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:WizardControl Name="wizardControl"/>
</Grid>
```



```
</Window>
```

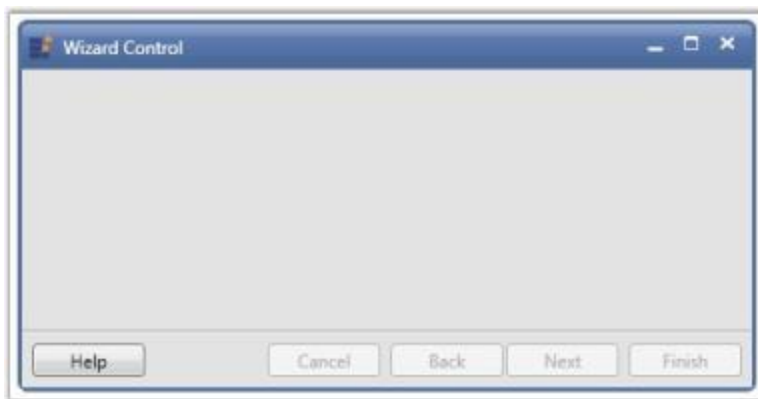
Adding control manually in C#

In order to add [WizardControl](#) control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
 - Syncfusion.Shared.Wpf
 - Syncfusion.Tools.Wpf
2. Import WizardControl namespace **Syncfusion.Windows.Tools.Controls**.
3. Create WizardControl instance and add it to the page.

C#

```
using System.Windows;
using Syncfusion.Windows.Tools.Controls;
namespace WizardControl
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            WizardControl wizardControl = new WizardControl();
            this.Content = wizardControl;
        }
    }
}
```



Adding multiple pages

You can add multiple pages in [WizardControl](#) using the [WizardPage](#) control. The Cancel, Back, Next and Finish buttons enables and disables automatically based on the current visible wizard page.

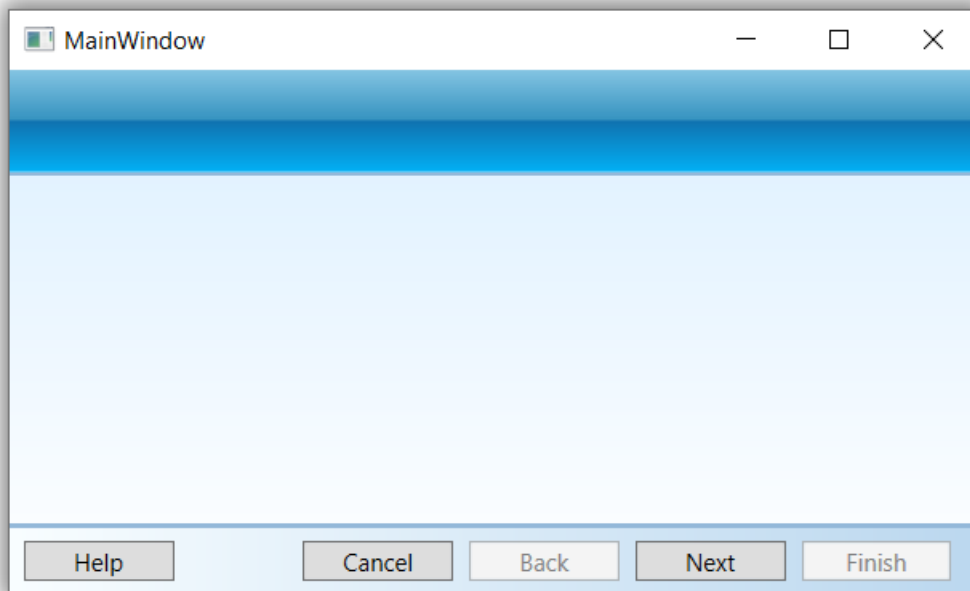
XML

```
<syncfusion:WizardControl Name="wizardControl">
<syncfusion:WizardPage Name="wizardPage1"/>
<syncfusion:WizardPage Name="wizardPage2"/>
```

```
<syncfusion:WizardPage Name="wizardPage3"/>
</syncfusion:WizardControl>
```

C#

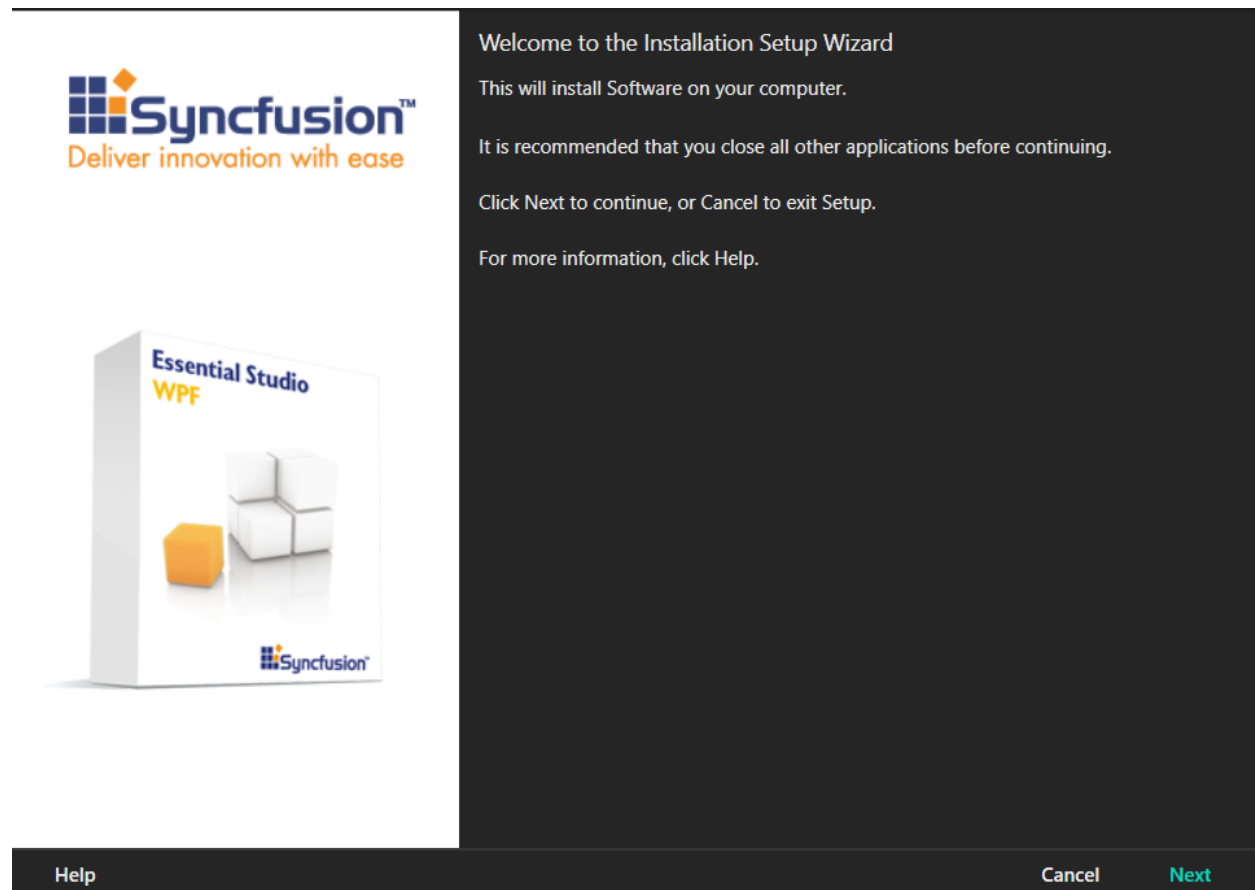
```
WizardControl wizardControl = new WizardControl();
WizardPage wizardPage1 = new WizardPage();
WizardPage wizardPage2 = new WizardPage();
WizardPage wizardPage2 = new WizardPage();
wizardControl.Items.Add(wizardPage1);
wizardControl.Items.Add(wizardPage2);
wizardControl.Items.Add(wizardPage3);
```



Theme

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

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



Interactive Features in WPF Wizard Control

This section illustrates the following interactive feature of [Wizard Control](#).

Populating pages in Wizard Control

The pages can be added to [Wizard Control](#) by following ways,

- Populating by Wizard Pages
- Populating by Data Binding

Populating by Wizard Pages

You can add any number of [wizard pages](#) to the [Wizard Control](#) and there are plenty of properties in the Wizard control that are used to customize the appearance and function of the wizard pages.

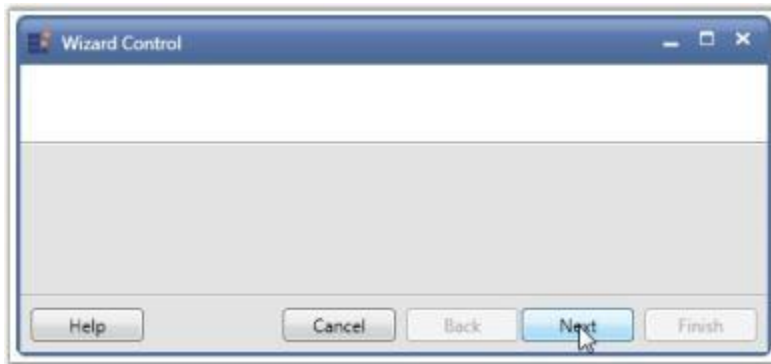
XML

```
<syncfusion:WizardControl Name="wizardControl">  
  <syncfusion:WizardPage Name="wizardPage1"/>  
  <syncfusion:WizardPage Name="wizardPage2"/>  
</syncfusion:WizardControl>
```

C#

```
WizardControl wizardControl = new WizardControl();  
WizardPage wizardPage1 = new WizardPage();  
WizardPage wizardPage2 = new WizardPage();
```

```
wizardControl.Items.Add(wizardPage1);  
wizardControl.Items.Add(wizardPage2);
```



Populating by Data Binding

Below steps will explain on how to add the Items through [ItemsSource](#) property of [Wizard Control](#).

1.Create a Model class with the necessary properties.

C#

```
public class Model : NotificationObject  
{  
    private string title;  
    public string Title  
    {  
        get  
        {  
            return title;  
        }  
        set  
        {  
            title = value;  
            RaisePropertyChanged("Title");  
        }  
    }  
    private string content;  
    public string Content  
    {  
        get  
        {  
            return content;  
        }  
        set  
        {  
            content = value;  
            RaisePropertyChanged("Description");  
        }  
    }  
}
```

2.Create collection of PageItems in ViewModel class to populate pages.

C#

```

private ObservableCollection<Model> items;
public ObservableCollection<Model> PageItems
{
    get
    {
        return items;
    }
    set
    {
        items = value;
        RaisePropertyChanged("PageItems");
    }
}

```

3. Populate the PageItems collection as follows.

C#

```

private void PopulatePageItems()
{
    items.Add(new Model { Title = "XML Developer's Guide", Content = "An in-
depth look at creating applications with XML." });
    items.Add(new Model { Title = "Midnight Rain", Content = "A former architect
battles corporate zombies, an evil sorceress, and her own childhood to
become queen of the world." });
    items.Add(new Model { Title = "Oberon's Legacy", Content = "In post
apocalypse England, the mysterious agent known only as Oberon helps to
create a new life for the inhabitants of London." });
    items.Add(new Model { Title = "Lover Birds", Content = "When Carla meets
Paul at an ornithology conference, tempers fly as feathers get ruffled." });
    items.Add(new Model { Title = "Science Fiction", Content = "After an
inadvertent trip through a Heisenberg Uncertainty Device, James discovers
the problems of being quantum." });
}

```

4. Create a ViewModel instance and use it as DataContext for the Root Window.

XML

```

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

```

5. Bind the PageItems collection to the [ItemsSource](#) property of the WizardControl. Content of the WizardPage can be displayed using the [ItemTemplate](#) property.

XML

```

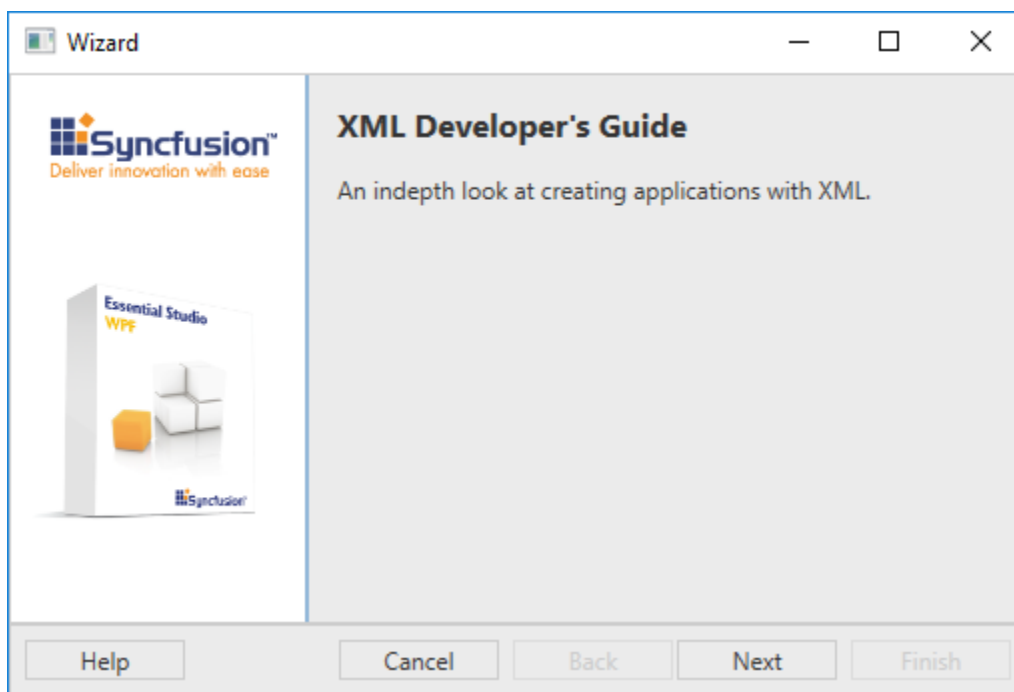
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Window.Resources>
<Style x:Key="WizardPageStyle" TargetType="syncfusion:WizardPage">
<Setter Property="Title" Value="{Binding Title}"/>
<Setter Property="PageType" Value="Exterior"/>
<Setter Property="BannerImage" Value="/Images/W_O-BG.png"/>

```

```

</Style>
</Window.Resources>
<Grid>
<syncfusion:WizardControl x:Name="wizardcontrol"
ItemContainerStyle="{StaticResource WizardPageStyle}" ItemsSource="{Binding
PageItems}">
<syncfusion:WizardControl.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Content}" TextWrapping="Wrap"/>
</DataTemplate>
</syncfusion:WizardControl.ItemTemplate>
</syncfusion:WizardControl>
</Grid>

```



Note: [ViewSample in GitHub.](#)

Selecting an Wizard Page

[SelectedWizardPage](#) property is used to select the wizard page from the WizardControl. Since the type of [SelectedWizardPage](#) is WizardPage, binding is needed to set selected wizard page through XAML. The following code explains how to select the wizard page in XAML and in C#,

XML

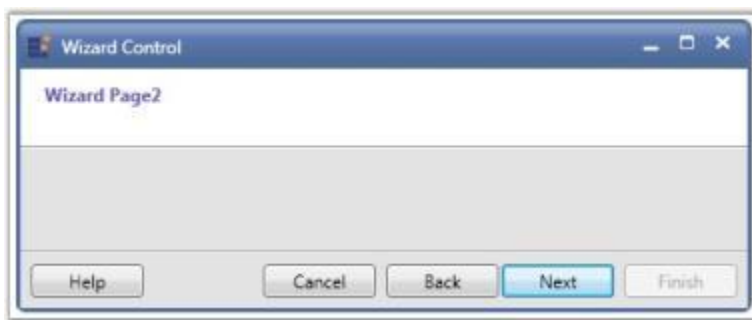
```

<syncfusion:WizardControl Name="WizardControl" SelectedWizardPage="{Binding
ElementName=wizard2}" Foreground="SlateBlue">
<syncfusion:WizardPage Title="Wizard Page1" x:Name="wizard1"/>
<syncfusion:WizardPage Title="Wizard Page2" x:Name="wizard2"/>
<syncfusion:WizardPage Title="Wizard Page3" x:Name="wizard3"/>
</syncfusion:WizardControl>

```

C#

```
//Initializing the Wizard control
WizardControl wizardControl = new WizardControl();
//Creating Wizard pages
WizardPage wizardPage1 = new WizardPage();
WizardPage wizardPage2 = new WizardPage();
WizardPage wizardPage3 = new WizardPage();
wizardControl.Foreground = Brushes.SlateBlue;
wizardPage1.Title = "Wizard Page1";
wizardPage2.Title = "Wizard Page2";
wizardPage3.Title = "Wizard Page3";
wizardControl.Items.Add(wizardPage1);
wizardControl.Items.Add(wizardPage2);
wizardControl.Items.Add(wizardPage3);
wizardControl.SelectedWizardPage = wizardPage2;
```



Title and Description

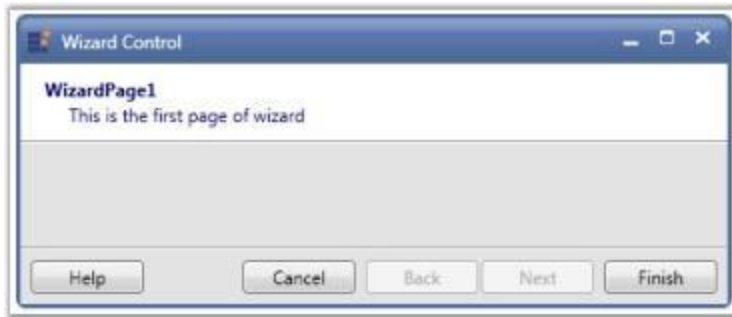
You can set the title and description for the [wizard page](#) by using the [Title](#) and [Description](#) properties respectively.

XML

```
<syncfusion:WizardControl Name="wizardControl">
  <syncfusion:WizardPage Name="wizardPage" Foreground="Navy"
    Title="WizardPage1" Description="This is the first page of wizard" />
</syncfusion:WizardControl>
```

C#

```
WizardControl wizardControl = new WizardControl();
WizardPage wizardPage = new WizardPage();
wizardControl.Items.Add(wizardPage);
wizardPage.Foreground = Brushes.Navy;
wizardPage.Title = "WizardPage1";
wizardPage.Description = "This is the first page of wizard";
```

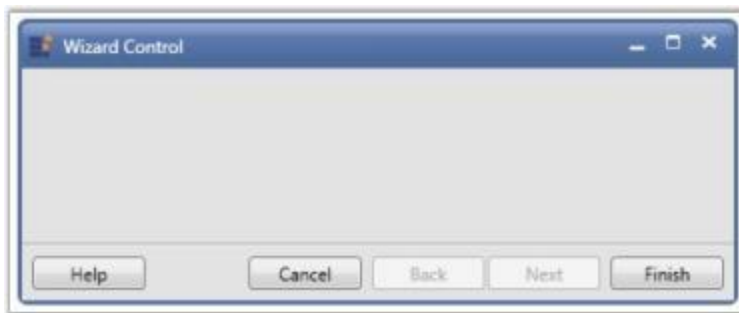


Wizard Page Type

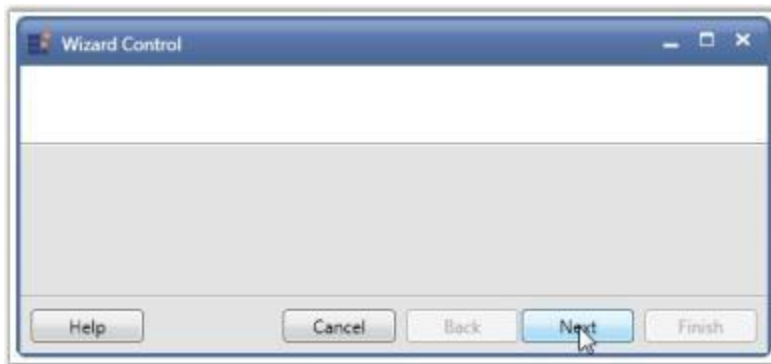
You can set the type of the [WizardPage](#) by using the [PageType](#) property. There are three types of wizard pages.

- Blank
- Interior
- Exterior

The [PageType](#) with Blank has no content or control in it.



The [PageType](#) with Interior value has the BannerImage occupies the top position of page.



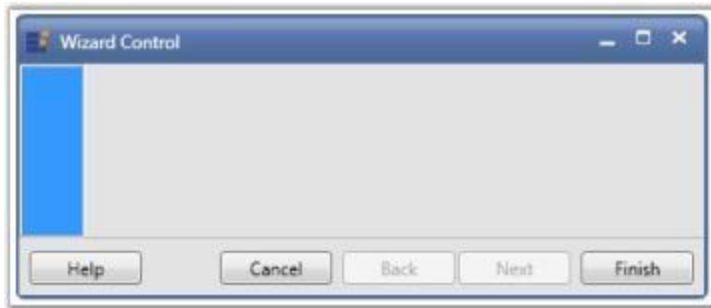
The [PageType](#) with Exterior value has the BannerImage occupies the left position of the page.

XML

```
<syncfusion:WizardControl Name="wizardControl">
  <syncfusion:WizardPage Name="wizardPage" PageType="Exterior"/>
</syncfusion:WizardControl>
```


C#

```
WizardControl wizardControl = new WizardControl();  
WizardPage wizardPage = new WizardPage();  
wizardControl.Items.Add(wizardPage);  
wizardPage.PageType = WizardPageType.Exterior;
```



Navigation Buttons of Wizard Page

This topic illustrates the following about Navigation buttons of [Wizard Page](#),

- Enabling or Disabling the Navigation Buttons
- Showing or Hiding the Navigation Buttons
- Text for the Navigation Buttons

Enabling or Disabling the Navigation Buttons

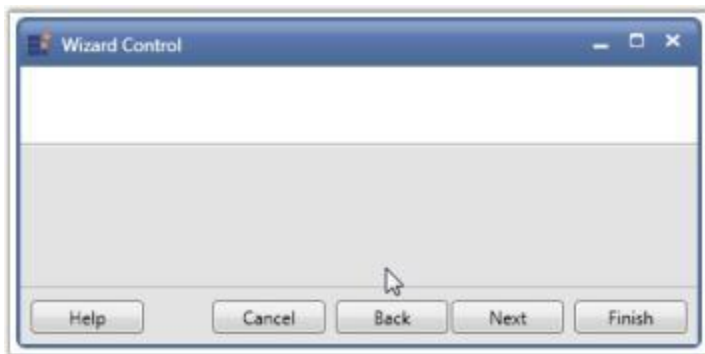
You can enable or disable the Back, Cancel, Next and Finish navigation buttons in the wizard control and wizard page. This is done by using the [BackEnabled](#), [CancelEnabled](#), [NextEnabled](#) and [FinishEnabled](#) properties.

XML

```
<syncfusion:WizardControl Name="wizardControl" BackEnabled="True"  
FinishEnabled="True"NextEnabled="True" CancelEnabled="True">  
<syncfusion:WizardPage Name="wizardPage"/>  
</syncfusion:WizardControl>
```

C#

```
WizardControl wizardControl = new WizardControl();  
WizardPage wizardPage = new WizardPage();  
wizardControl.Items.Add(wizardPage);  
wizardControl.NextEnabled = true;  
wizardControl.BackEnabled = true;  
wizardControl.FinishEnabled = true;  
wizardControl.CancelEnabled = true;
```



Showing or Hiding the Navigation Buttons

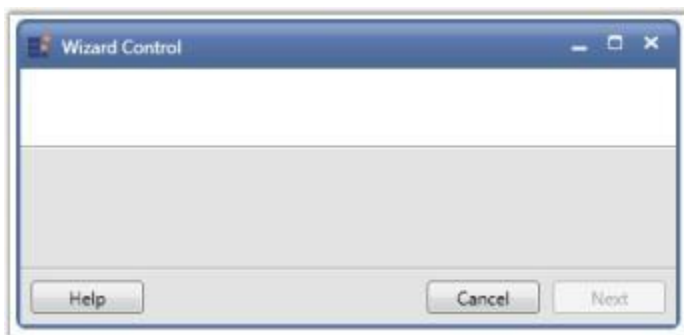
You can show or hide the Back, Cancel, Next, Help and Finish navigation buttons in the [Wizard Control](#) and wizard page by using the [BackVisible](#), [CancelVisible](#), [NextVisible](#), [HelpVisible](#) and [FinishVisible](#) properties.

XML

```
<syncfusion:WizardControl Name="wizardControl" BackVisible="False"
NextVisible="True" CancelVisible="True" HelpVisible="True"
FinishVisible="False">
<syncfusion:WizardPage Name="wizardPage"/>
</syncfusion:WizardControl>
```

C#

```
WizardControl wizardControl = new WizardControl();
WizardPage wizardPage = new WizardPage();
wizardControl.Items.Add(wizardPage);
wizardControl.CancelVisible = true;
wizardControl.BackVisible = false;
wizardControl.FinishVisible = true;
wizardControl.NextVisible = true;
wizardControl.HelpVisible = true;
```



Text for the Navigation Buttons

You can set custom text for the Back, Next, Finish, Help and Cancel navigation buttons in the [Wizard control](#) by using the [BackText](#), [NextText](#), [FinishText](#), [HelpText](#) and [CancelText](#) properties.

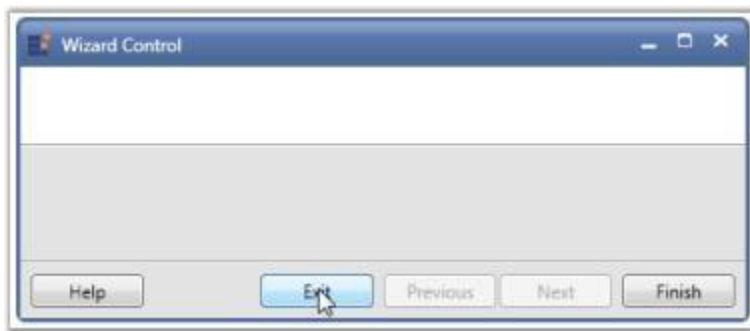
XML

```
<syncfusion:WizardControl Name="wizardControl" BackText="Previous"
HelpText="Help" CancelText="Exit" FinishText="Finish" NextText="Next">
```

```
<syncfusion:WizardPage Name="wizardPage"/>
</syncfusion:WizardControl>
```

C#

```
WizardControl wizardControl = new WizardControl();
WizardPage wizardPage = new WizardPage();
wizardControl.Items.Add(wizardPage);
wizardControl.NextText = "Next";
wizardControl.BackText = "Previous";
wizardControl.CancelText = "Exit";
wizardControl.HelpText = "Help";
wizardControl.FinishText = "Finish";
```



Note: You can set custom text for the navigation buttons in the wizard control but not for the navigation buttons in the wizard page.

Next Page and Previous Page Navigation

You can set the navigation to the Next and Previous pages of [WizardPage](#) by using the [NextPage](#) and [PreviousPage](#) properties.

XML

```
<syncfusion:WizardControl Name="wizardControl" >
<syncfusion:WizardPage Name="wizardPage1" Title="page1" NextPage="{Binding
ElementName=wizardPage4}"/>
<syncfusion:WizardPage Name="wizardPage2" Title="page2" PreviousPage="{Binding
ElementName=wizardPage3}"/>
<syncfusion:WizardPage Name="wizardPage3" Title="page3"/>
<syncfusion:WizardPage Name="wizardPage4" Title="page4"/>
</syncfusion:WizardControl>
```

C#

```
WizardControl wizardControl = new WizardControl();
WizardPage wizardPage1 = new WizardPage();
WizardPage wizardPage2 = new WizardPage();
```

```

WizardPage wizardPage3 = new WizardPage();
wizardControl.Foreground = Brushes.SlateBlue;
wizardPage1.Title = "Wizard Page1";
wizardPage2.Title = "Wizard Page2";
wizardPage3.Title = "Wizard Page3";
wizardControl.Items.Add(wizardPage1);
wizardControl.Items.Add(wizardPage2);
wizardControl.Items.Add(wizardPage3);
wizardPage1.NextPage = wizardPage3;
wizardPage3.PreviousPage = wizardPage1;

```

Closing the Wizard Window

You can close the [Wizard control](#) window by clicking the Finish or Cancel button by enabling the [FinishButtonClosesWindow](#) or [CancelButtonCancelsWindow](#) properties respectively.

XML

```

<syncfusion:WizardControl Name="wizardControl"
FinishButtonClosesWindow="True" CancelButtonCancelsWindow="True">
<syncfusion:WizardPage Name="wizardPage"/>
</syncfusion:WizardControl>

```

C#

```

WizardControl wizardControl = new WizardControl();
WizardPage wizardPage = new WizardPage();
wizardControl.Items.Add(wizardPage);
wizardControl.CancelButtonCancelsWindow = true;
wizardControl.FinishButtonClosesWindow = true;

```

Event for Next Button in Wizard Control

You can use the [Next](#) event in [Wizard Control](#) to perform any required operation when Next button is clicked.

XML

```

<syncfusion:WizardControl Name="wizardControl" CancelEnabled="False"
CancelVisible="False" Next="wizardControl_Next"
FinishButtonClosesWindow="True" CancelButtonCancelsWindow="True" >
<syncfusion:WizardPage Name="wizardPage1" Title="page1" />
<syncfusion:WizardPage Name="wizardPage2" Title="page2" />
<syncfusion:WizardPage x:Name="wizardPage3" Title="page3"/>
<syncfusion:WizardPage Name="wizardPage4" Title="page4"/>
</syncfusion:WizardControl>

```

C#

```

private void wizardControl_Next(object sender, RoutedEventArgs e)
{
    var obj = sender as WizardControl;
    if (obj.CancelVisible != true && obj.CancelEnabled != true)
    {
        obj.SelectedWizardPage = wizardPage2;
    }
}

```

```
}
```

Layout Related Features in WPF Wizard Control

This section illustrates the following Layout-related features of [Wizard Control](#).

Setting the Minimum Height for the Interior Wizard Page Header

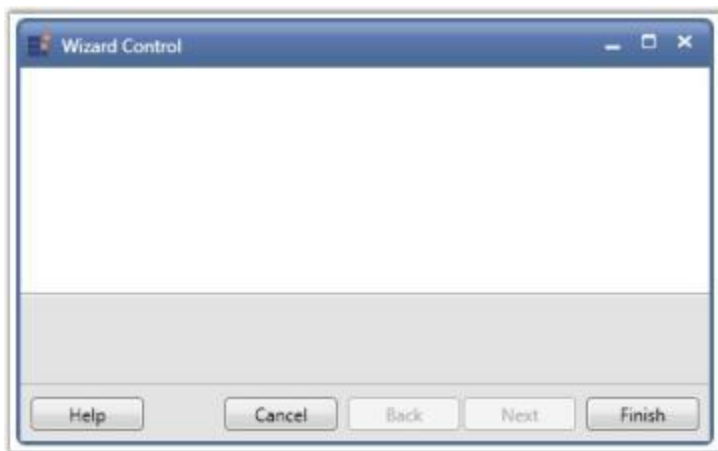
You can set the minimum height for the header of the Interior wizard page by using the [InteriorPageHeaderMinHeight](#) property in [Wizard Control](#).

XML

```
<syncfusion:WizardControl Name="wizardControl"
InteriorPageHeaderMinHeight="150">
<syncfusion:WizardPage Name="wizardPage" />
</syncfusion:WizardControl>
```

C#

```
WizardControl wizardControl = new WizardControl();
WizardPage wizardPage = new WizardPage();
wizardControl.Items.Add(wizardPage);
wizardControl.InteriorPageHeaderMinHeight = 150;
```



Setting the Banner Background Color

You can set the background color of the banner for the [Wizard Control](#) by using the [BannerBackground](#) property.

XML

```
<syncfusion:WizardControl Name="wizardControl">
<syncfusion:WizardPage Name="wizardPage" BannerBackground="Navy"/>
</syncfusion:WizardControl>
```

C#

```
WizardControl wizardControl = new WizardControl();
WizardPage wizardPage = new WizardPage();
wizardControl.Items.Add(wizardPage);
wizardPage.BannerBackground = Brushes.Navy;
```



Setting the Banner Image

You can set an image for the banner of the [Wizard Page](#) in [Wizard Control](#) using the [BannerImage](#) property.

Note: You can set the banner image either on the interior or exterior wizard page based on the wizard page type.

XML

```
<syncfusion:WizardControl Name="wizardControl">
  <syncfusion:WizardPage Name="wizardPage" BannerImage="/Image/sync.bmp"/>
</syncfusion:WizardControl>
```

C#

```
WizardControl wizardControl = new WizardControl();
WizardPage wizardPage = new WizardPage();
wizardControl.Items.Add(wizardPage);
wizardPage.BannerImage = new BitmapImage(new Uri("/Image/sync.bmp",
UriKind.RelativeOrAbsolute));
```



Setting Minimum Width for the Banner Image on the Exterior Wizard Page

You can set the minimum width of the banner image on the 'Exterior' wizard page by using the [ExteriorPageBannerImageMinWidth](#) property in [Wizard Control](#).

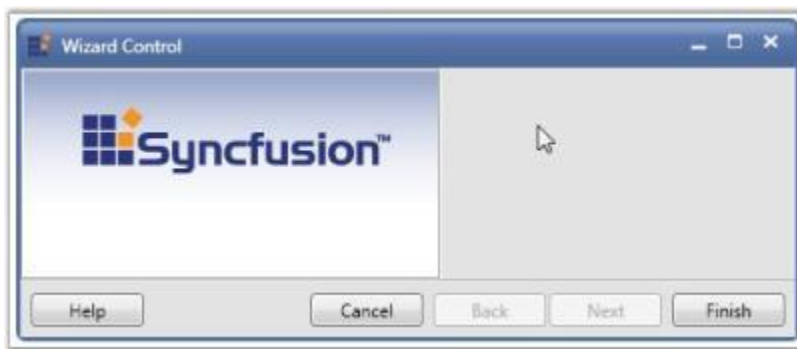
XML

```
<syncfusion:WizardControl Name="wizardControl"
  ExteriorPageBannerImageMinWidth="10">
```

```
<syncfusion:WizardPage Name="wizardPage" PageType="Exterior"
BannerImage="/Image/sync.bmp"/>
</syncfusion:WizardControl>
```

C#

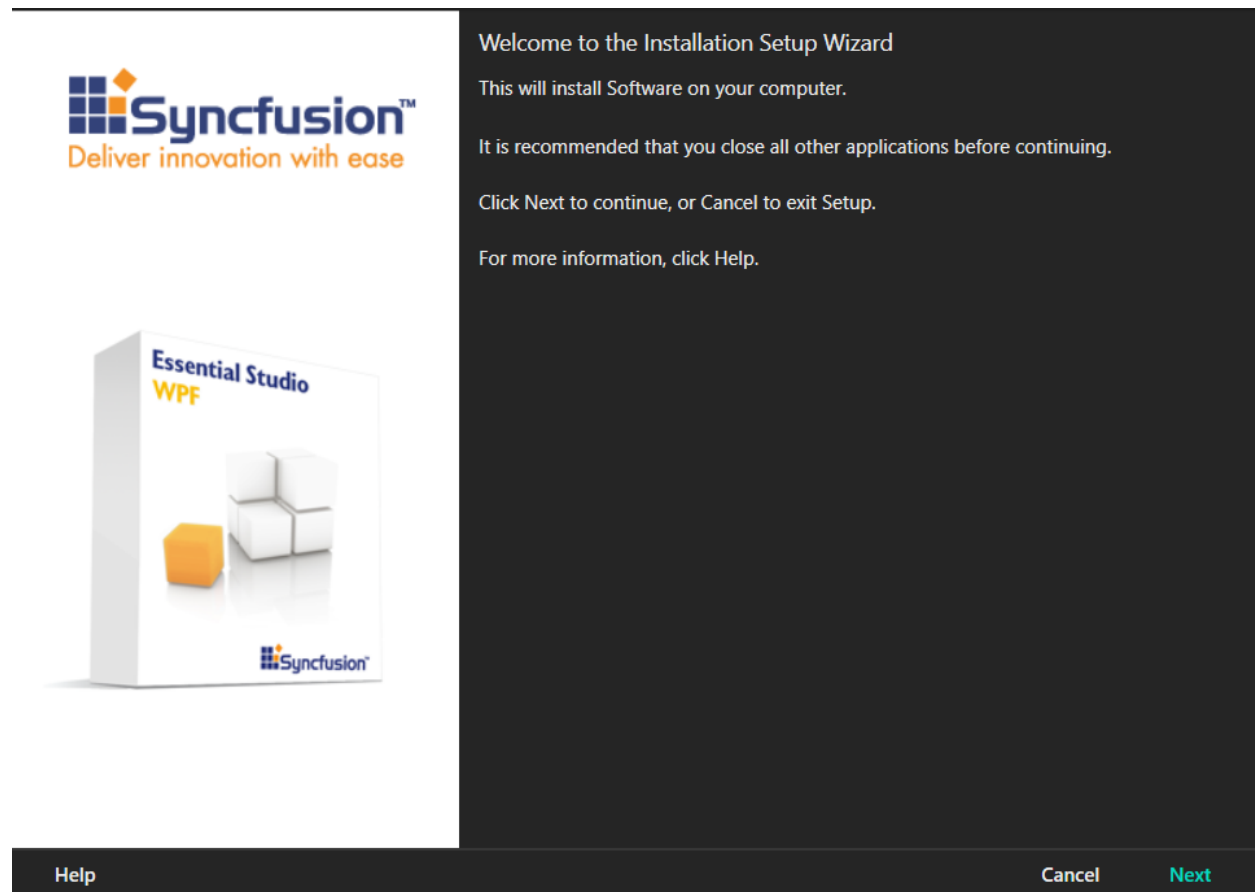
```
WizardControl wizardControl = new WizardControl();
grid.Children.Add(wizardControl);
WizardPage wizardPage = new WizardPage();
wizardControl.Items.Add(wizardPage);
wizardPage.BannerImage = new BitmapImage(new Uri("/Image/sync.bmp",
UriKind.RelativeOrAbsolute));
wizardPage.PageType = WizardPageType.Exterior;
wizardControl.ExteriorPageBannerImageMinWidth = 10;
```



Theme

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

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



AutoComplete (Classic)

[WPF AutoComplete \(Classic\) Overview](#)

Warning: This control has been marked as classic, please use [SfTextBoxExt \(AutoComplete\)](#) ("AutoComplete") instead.

The AutoComplete control provides you with the live drop-down hints as you enter the text in AutoComplete Textbox. It guides you by displaying the list of items from the Data Source. You can then select any item from the list instead of entering the whole text again.

[Feature summary](#)

Here are the core features of AutoComplete.

- FilePath, Registry & Custom Data Source support
- Data Binding support in case of Custom Data Source
- Selection Mode
- Auto Append
- Filter
- Custom Filtration
- History
- Popup Resizing
- Skins Support

Getting Started with WPF AutoComplete (Classic)

Structure of the AutoComplete control



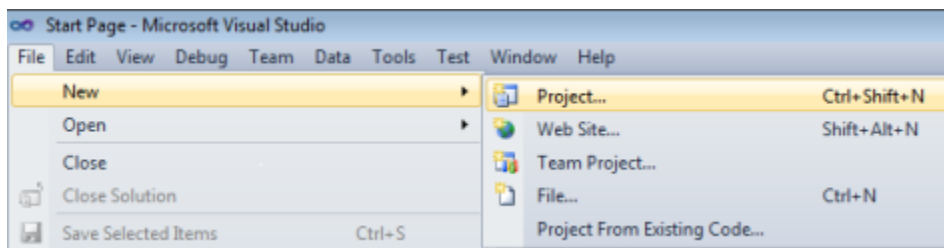
AutoComplete Control Structure

Add AutoComplete to an application

Following are the step-by-step instructions to add an AutoComplete control in a WPF application. The AutoComplete control can be created by using either C#, XAML code. It can also be created using Blend.

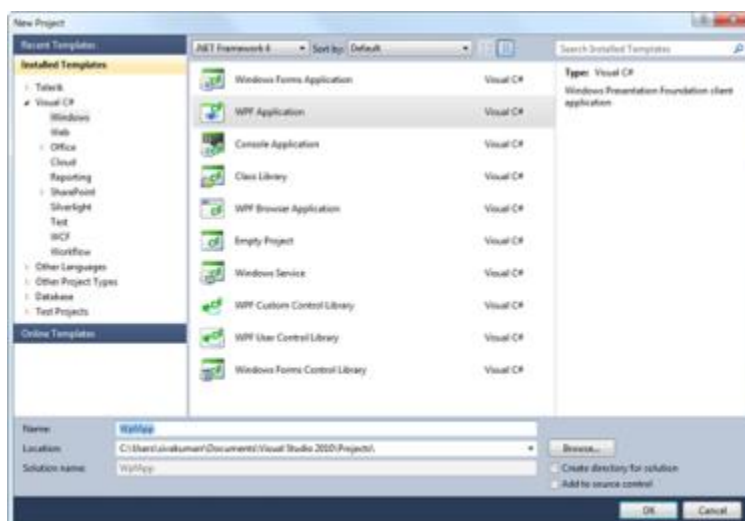
Creating AutoComplete using C#

1. Open Visual Studio, On the File menu click New -> Project. This opens the New Project Dialog box.



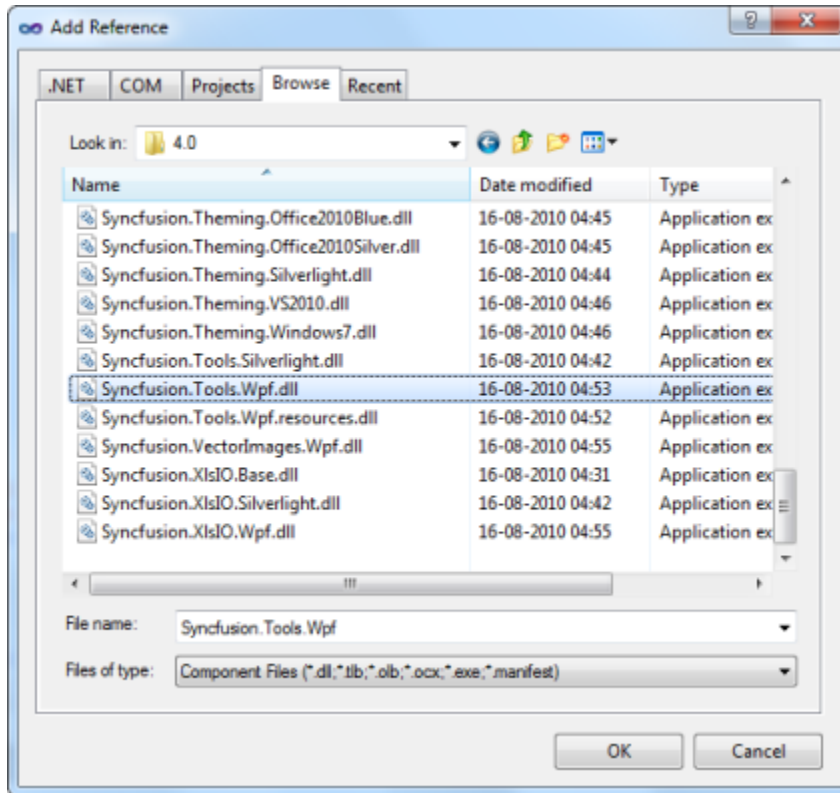
Creating New Project

2. In the Project Dialog window, select WPF application and, in the Name field type the name of the project. Click OK.



Creating New WPF Application

3. Go to Solution Explorer. Right-click References folder and click Add Reference. Add the Syncfusion.Tools.WPF.dll and Syncfusion.Shared.WPF assembly to the project References folder.



Adding Reference

XML

```
xmlns:syncfusion="clr-namespace:
Syncfusion.Windows.Tools.Controls;assembly=Syncfusion.Tools.Wpf"
```

4. Add Syncfusion.Tools.WPF reference in XAML and C# code as follows.

C#

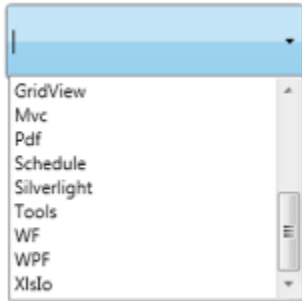
```
using Syncfusion.Windows.Tools.Controls;
```

5. Click and open the C# file. Add AutoComplete to the application.

C#

```
AutoComplete autoComplete1 = new AutoComplete();
List<String> productSource = new List<String>();
productSource.Add("WPF");
productSource.Add("Chart");
productSource.Add("GridView");
productSource.Add("WF");
productSource.Add("Xlsio");
```

```
productSource.Add("Business Intelligence");  
productSource.Add("Tools");  
productSource.Add("Silverlight");  
productSource.Add("Schedule");  
productSource.Add("Mvc");  
productSource.Add("Pdf");  
this.AutoComplete1.CustomSource = productSource;
```

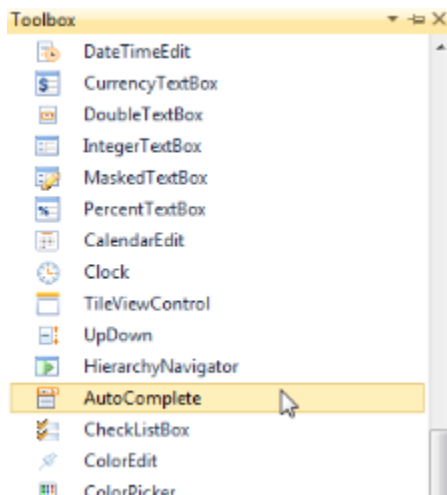


AutoComplete Created Using C#

[Create AutoComplete using XAML](#)

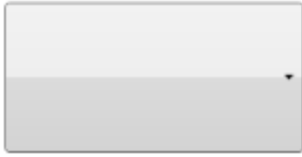
Following are the steps to create the AutoComplete by using VisualStudio in XAML as follows.

1. Create a new WPF application in Visual Studio. In Visual Studio Toolbox, click Syncfusion WPF Toolbox tab and select AutoComplete.



Select AutoComplete From ToolBox

2. Drag-and-drop the AutoComplete to Design View, to add AutoComplete to the application.

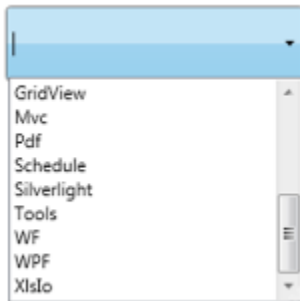


AutoComplete Drag and Drop from ToolBox

3. You can now customize the properties of AutoComplete in the Properties Window.

XML

```
<local:productSource x:Key="Src"/>
<syncfusion:AutoComplete x:Name="AutoComplete1" Source="Custom"
CustomSource="{StaticResource Src}"/>
```

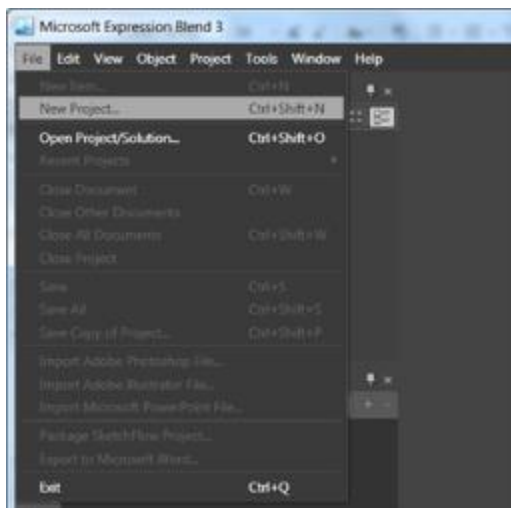


AutoComplete Created Using XAML

Create AutoComplete using expression blend

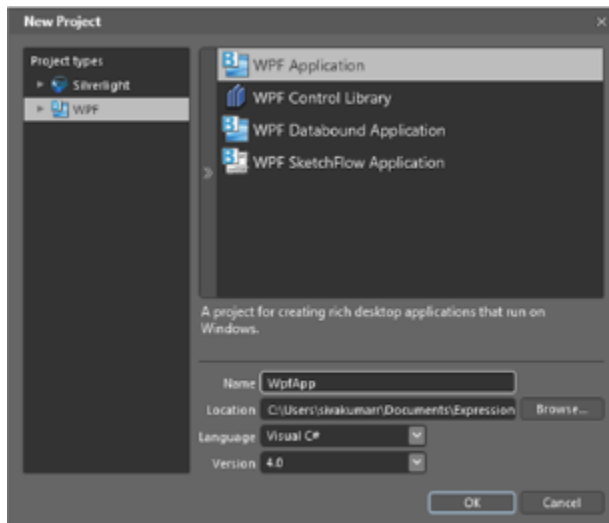
The AutoComplete control provides full Blend support. Here are the step-by-step instructions to create a WPF application in Blend.

1. Open Blend, On the File Menu click New Project. This opens the New Project dialog box.



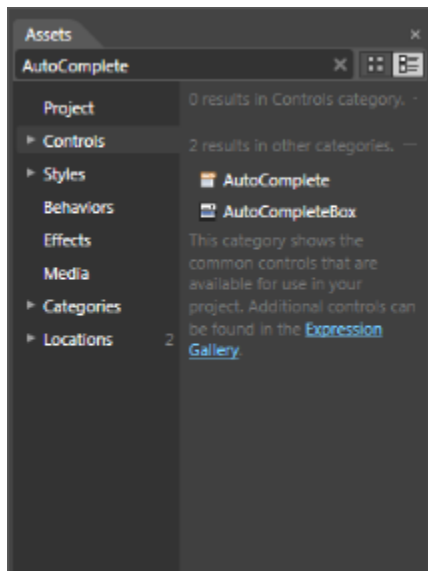
Create New project in Expression Blend

2. In the Project type's panel, select WPF application and then click OK.



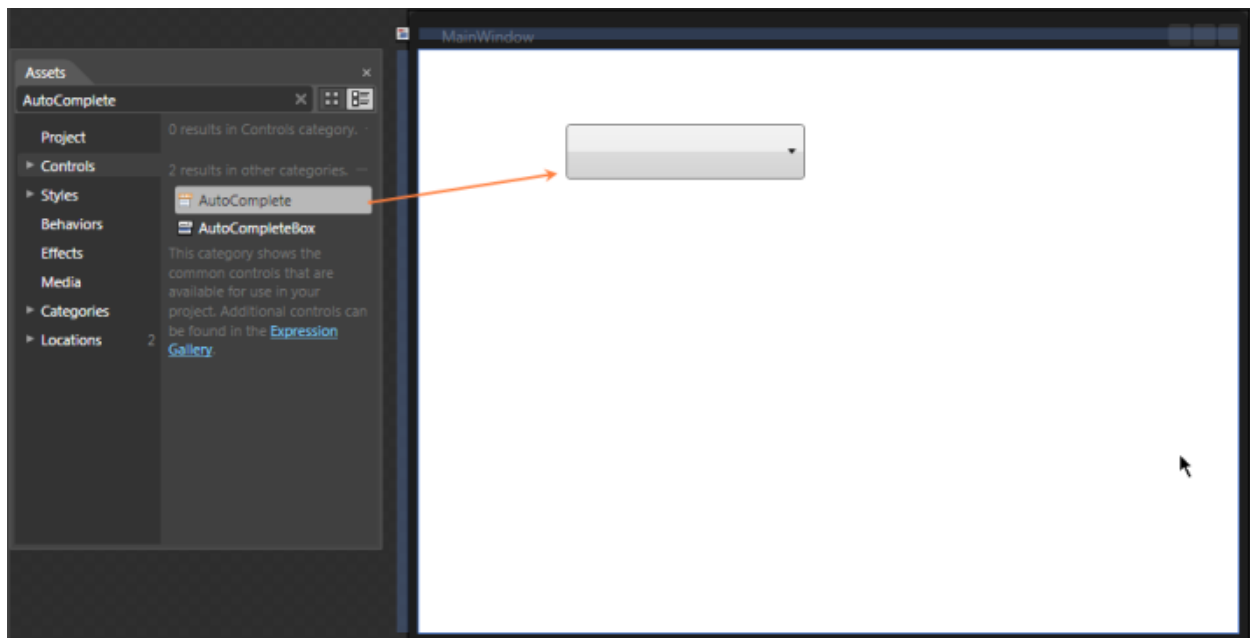
Create New WPF Application in Expression Blend

3. Add the following References with the sample project.
 - Syncfusion.Tools.WPF.dll
 - Syncfusion.Shared.WPF.dll
4. On the Window menu, select Assets. This opens the Assets Library dialog box. In the Search box, type AutoComplete. This displays the search results as shown below-.



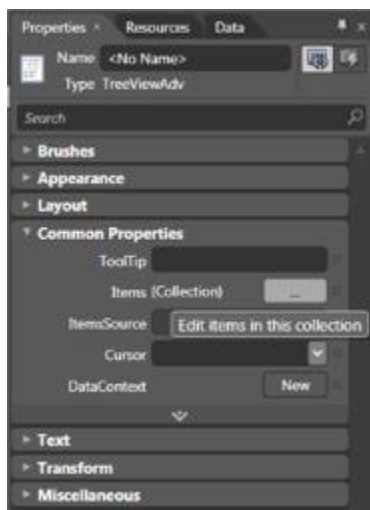
AutoComplete Displayed in Assets window

5. Drag the AutoComplete control to the Design View.



AutoComplete Drag & Drop from Asset window

6. You can now customize the properties of the AutoComplete in the Properties Window.



Properties Window

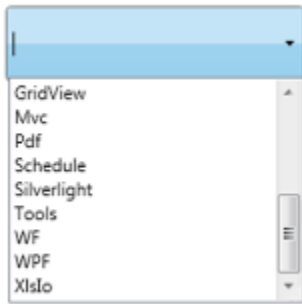
XML

```
<local:productSource x:Key="Src"/>
<syncfusion:AutoComplete x:Name="AutoComplete1" Source="Custom"
CustomSource="{StaticResource Src}"/>
```

C#

```
List<String> productSource = new List<String>();
productSource.Add("Diagram");
productSource.Add("Gauge");
```

```
productSource.Add("GridView");  
productSource.Add("Chart");  
productSource.Add("Business Intelligence");  
productSource.Add("Schedule");  
productSource.Add("Grid");  
productSource.Add("DocIo");  
productSource.Add("XlsIo");  
productSource.Add("Pdf");
```

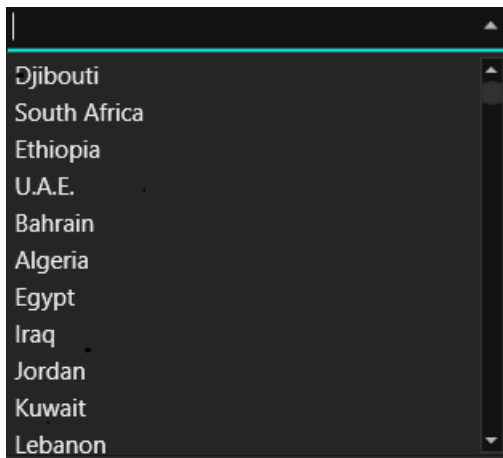


AutoComplete Created Using Blend

Theme

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

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



Basic Core Features in WPF AutoComplete (Classic)

AutoComplete supports basic core features which are listed as follows.

- [SelectedIndex](#)— Used to set and get the index of the selected item.
- [SelectedItem](#)—Used to get which item of the AutoComplete has been selected.
- [SelectedValue](#)—Used to get the value of the selected item, the value of the SelectedValue property will be set based on the value of the SelectedValuePath property.
- [SelectedValuePath](#)—Used to set the value of the SelectedValue property of the AutoComplete.

- [DisplayMemberPath](#)—Used to set the value for the items displayed in the drop-down list.
- [IsDropDownOpen](#)—Used to open or close the Drop-down list by setting its value as True or False.
- [SelectionChanged](#).
- [TextChanged](#).

Using basic core features in an application

In the [SelectionChanged](#) event the [SelectedIndex](#), [SelectedItem](#) & [SelectedValue](#) properties can be used in the application to get these property values. The properties and events listed can be used in the application as mentioned below.

C#

```
List<String> Products = new List<String>();
Products.Add("Diagram");
Products.Add("Gauge");
Products.Add("Chart");
Products.Add("Business Intelligence");
AutoComplete autoComplete1 = new AutoComplete();
autoComplete1.CustomSource = Products;
autoComplete1.SelectedIndex = 1;
autoComplete1.IsDropDownOpen = true;
autoComplete1.SelectionChanged +=
new SelectionChangedEventHandler(autoComplete1_SelectionChanged);
autoComplete1.TextChanged += new
PropertyChangedCallback(autoComplete1_TextChanged);
void autoComplete1_TextChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    this.textBlock.Text = this.autoComplete1.Text;
}
void autoComplete1_SelectionChanged(object sender, SelectionChangedEventArgs
e)
{
    MessageBox.Show("SelectedItem: " +
this.autoComplete1.SelectedItem.ToString()+ "\n" + "SelectedValue: "
+ this.autoComplete1.SelectedValue.ToString())
}
```

[Sample link](#)

WPF Sample Browser-> Tools -> Editors -> AutoComplete Demo

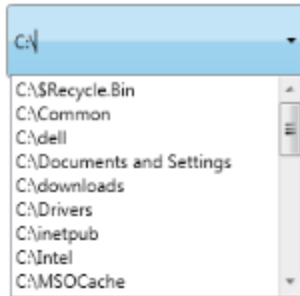
FilePath Registry Custom Data Source Support in WPF AutoComplete

AutoComplete can be used with different kinds of Data Source like FilePath, Registry & CustomSource. The Data

Source of the AutoComplete control can be set using the [Source](#) property.

When the value of the Source property is set as FilePath, the AutoComplete will displays the path in the local

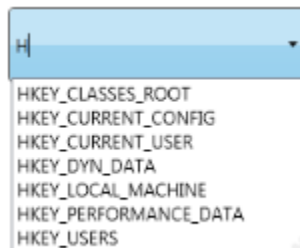
system as the source. This is illustrated in the following image.



Source—FilePath

When the value of the [Source](#) property is set as Registry, the AutoComplete loads the values from the Registry. It

is used when the Registry keys are required as input. This is illustrated in the image given below.

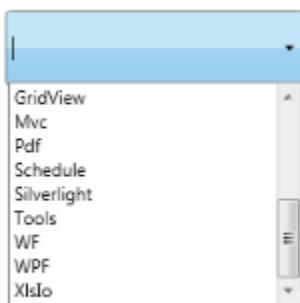


Source—Registry

When the value of the [Source](#) property is set as Custom, the AutoComplete loads the values from the Business objects

bounded to the AutoComplete control by using the [CustomSource](#) property. This is illustrated in the image given

below.



Source—Custom

[Adding data source support to an application](#)

AutoComplete can be used with different kinds of Data Sources using the [Source](#) property. This support can be added

to the application as mentioned in the following code example.

XML

```
<syncfusion:AutoComplete x:Name="AutoComplete1" Source="FilePath"/>
<syncfusion:AutoComplete x:Name="AutoComplete2" Source="Registry"/>
<syncfusion:AutoComplete x:Name="AutoComplete3" Source="Custom">
<syncfusion:AutoComplete.CustomSource>
<local:CustomerListCollection/>
</syncfusion:AutoComplete.CustomSource>
</syncfusion:AutoComplete>
```

C#

```
AutoComplete autoComplete1 = new AutoComplete();
this.autoComplete1.Source = SourceMode.FilePath;
AutoComplete autoComplete2 = new AutoComplete();
this.autoComplete2.SelectionMode = SourceMode.Registry;
AutoComplete autoComplete3 = new AutoComplete();
this.autoComplete3.SelectionMode = SourceMode.Custom;
List<String> products = new List<String>();
customSource.Add("Diagram");
customSource.Add("Gauge");
customSource.Add("Chart");
customSource.Add("Schedule");
customSource.Add("Grid");
customSource.Add("DocIo");
customSource.Add("XlsIo");
customSource.Add("Pdf");
customSource.Add("RichTextBox");
customSource.Add("ReportBuilder");
this.autoComplete3.CustomSource = products;
```

Tables for properties, methods, and events

Events

- [SourceChanged](#)

Sample link

WPF Sample Browser-> Tools -> Editors -> AutoComplete Demo

Data Binding in WPF AutoComplete (Classic)

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). You can bind the data to the AutoComplete through the [CustomSource](#) property. While binding the [CustomSource](#) to the AutoComplete, you must set the value of the DisplayMemberPath and the SelectedValuePath properties.

Adding data binding to an application

You can add the custom source for the AutoComplete by binding the source to the CustomSource property. You can use the DisplayMemberPath property to set the value for items that needs to be displayed in the drop-down list. Also you can use the [SelectedValuePath](#) property which can be used to set the value of the [SelectedValue](#) property.

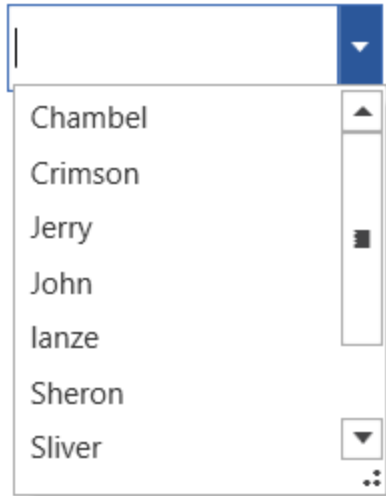
C#

```
//Model.cs
```

```
public class EmployeeList {
    public int EmployeeID { get; set; }
    public string Name { get; set; }
    public string Mailid { get; set; }
    public EmployeeList() { }
    public EmployeeList(string name, string mail, int id)
    {
        Name = name;
        Mailid = mail;
        EmployeeID = id;
    }
}
//ViewModel.cs
public class EmployeeListCollection : ObservableCollection<EmployeeList> {
    public EmployeeListCollection() {
        this.Add(new EmployeeList() { EmployeeID = 1001, Name = "John", Mailid =
"john@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1002, Name = "Jerry", Mailid =
"Jerry@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1004, Name = "lanze", Mailid =
"lanze@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1005, Name = "Chambel", Mailid =
"Chambel@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1006, Name = "Crimson", Mailid =
"Crimson@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1001, Name = "Smith", Mailid =
"john@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1002, Name = "Sheron", Mailid =
"Jerry@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1003, Name = "Sliver", Mailid =
"Brad@syncfusion.com" });
    }
}
```

XML

```
<syncfusion:AutoComplete x:Name="autoComplete"
Source="Custom"
DisplayMemberPath="Name"
SelectedValuePath="EmployeeID">
    <syncfusion:AutoComplete.CustomSource>
        <local:EmployeeListCollection/>
    </syncfusion:AutoComplete.CustomSource>
</syncfusion:AutoComplete>
```



AutoComplete Bound with Data Source

Note: View [Sample]() in GitHub

Tables for properties and events

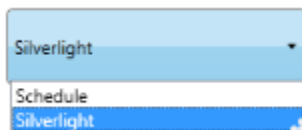
Events

- [SelectedValueChanged](#)
- [CustomSourceChanged](#)

Selection Mode Support in WPF AutoComplete (Classic)

AutoComplete supports two kinds of Selection Mode namely Single and Multiple. You can select the Mode using the [SelectionMode](#) property.

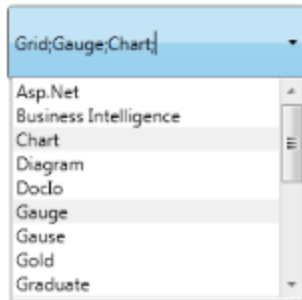
When the [SelectionMode](#) property is set as Single, only one item can be selected at a time. The following image illustrates the Single selection mode.



SelectionMode-Single

When the [SelectionMode](#) is set as Multiple, you can select multiple items by using the [SeparatorChar](#) property to separate the selected items. By default the SeparatorChar is “;”. This allows you to select multiple items by using the [SelectionMode](#) property. Once an item is selected the [SeparatorChar](#) is to be entered in the text box to select the next item.

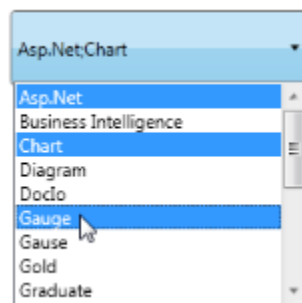
The following image illustrates the Multiple selection mode.



SelectionMode—Multiple

When the [SelectionMode](#) is set as Extended, you can select multiple items at a time by pressing the Ctrl key. While selecting the multiple items, the selected items will be separated by the [SeparatorChar](#) automatically.

The following image illustrates the Multiple selection mode.



SelectionMode—Extended

Adding Single, Multiple & Extended Selection Support to an Application

The [SelectionMode](#) property is used to attain these functionalities by setting its value as Single or Multiple or Extended. By default its value is Single. The following code snippet is used to set the [SelectionMode](#) property.

XML

```
<syncfusion:AutoComplete x:Name="AutoComplete2" SelectionMode="Multiple"/>
```

C#

```
AutoComplete autoComplete1 = new  
AutoComplete(); this.autoComplete2.SelectionMode = SelectionMode.Multiple;
```

Tables for properties and events

Events

- [SelectionModeChanged](#)

Sample link

WPF Sample Browser-> Tools -> Editors -> AutoComplete Demo

Auto Append Support in WPF AutoComplete (Classic)

Auto Append is used to guide the complete text by appending the entered text with suitable text from the data source, when a text is entered in the AutoComplete textbox. AutoComplete allows you to enable Auto Append using [IsAutoAppend](#) property.



Auto Append

Adding auto append support to an application

If the [IsAutoAppend](#) property is set as True, once you enter the text the AutoComplete guides you to complete text, by appending the entered text with suitable text from the data source. If this property is set as False the matched suitable text will not append with the entered text.

XML

```
<syncfusion:AutoComplete x:Name="AutoComplete1" IsAutoAppend="true"/>
```

C#

```
AutoComplete autoComplete1 = new  
AutoComplete(); this.autoComplete1.IsAutoAppend = true;
```

Tables for properties and events

Properties

- [IsAutoAppend](#)

Events

- [IsAutoAppendChanged](#)

Sample link

WPF Sample Browser-> Tools -> Editors -> AutoComplete Demo

Filter Support in WPF AutoComplete (Classic)

Filter support is used to filter the matched list of items from the linked source depending on the text entered in the AutoComplete textbox. AutoComplete allows the user to enable Filter the items using [IsFilter](#) property.



Filter Support

Adding filter support to an application

If the [IsFilter](#) property is set as True, once you enter text in the AutoComplete textbox, the matched list of items

will be displayed in the drop-down list. If this property is set as False the matched list of items will not be displayed in the drop-down list, instead all the items will be displayed.

XML

```
<syncfusion:AutoComplete x:Name="AutoComplete1" IsFilter="true"/></td></tr>
```

C#

```
AutoComplete autoComplete1 = new AutoComplete();this.autoComplete1.IsFilter = true;</td></tr>
```

Tables for properties, methods, and events

Events

- [IsFilterChanged](#)

Sample link

WPF Sample Browser-> Tools -> Editors -> AutoComplete Demo

Custom Filtration Support in WPF AutoComplete (Classic)

AutoComplete supports Custom Filtration of items, which allows you to specify three different search modes for displaying the drop-down list. The [StringMode](#) property is used to specify the search mode.

When the value of the [StringMode](#) property is set as StartChar, AutoComplete begins its search from starting index of the strings in the source list collection and the matching results will be displayed in the drop-down list. In the figure shown below, AutoComplete searches using the entered key “D” and displays the matched list.



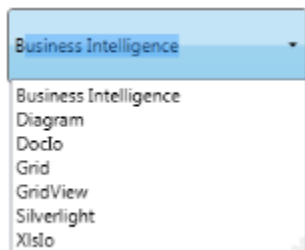
StringMode—StartChar

When the value of the StringMode property is set as IndexBased, starting index value can be set using the StringModelIndex property. In this mode AutoComplete begins its search from the user specified index of the strings in the source list collection and the matching results will be displayed in the drop-down list. In the figure shown below StringModelIndex value is set as “2” and the entered text is “i”. The AutoComplete displays the list of items which has “i” in the specified index.



StringMode—IndexBased

When the value of the StringMode property is set as AnyChar, the AutoComplete searches for the strings which has substrings entered in the AutoComplete control. In the figure shown below, based on the entered text "I", the AutoComplete displays the list of items which has as a substring in it.



StringMode—AnyChar

Using custom filtration support in an application

The [StringMode](#) property will be used to attain this functionality by setting its value as StartChar or IndexBased or AnyChar.

XML

```
<syncfusion:AutoComplete x:Name="AutoComplete1" StringMode="StartChar"/>
<syncfusion:AutoComplete x:Name="AutoComplete2" StringMode="IndexBased"/>
<syncfusion:AutoComplete x:Name="AutoComplete3" StringMode="AnyChar"/>
```

C#

```
AutoComplete autoComplete1 = new AutoComplete();
this.autoComplete1.StringMode = StringMode.StartChar;
AutoComplete autoComplete2 = new AutoComplete();
this.autoComplete2.StringMode = StringMode.IndexBased;
this.autoComplete2.StringModeIndex = 2;
AutoComplete autoComplete3 = new AutoComplete();
this.autoComplete3.StringMode = StringMode.AnyChar;
```

Sample Link

WPF Sample Browser-> Tools -> Editors -> AutoComplete Demo

History Support in WPF AutoComplete (Classic)

History support in AutoComplete means, reuse of the items which are already used in the AutoComplete textbox. AutoComplete allows you to enable this history support by setting the value of the [IsHistory](#) property to True. AutoComplete guides you to select an item from the list of items which are added to the history, by using the drop-down button to open the drop-down list.

Using history support in an application

Items can be added to the history using the [AddToHistory\(String string\)](#) and [AddToHistory\(Object obj\)](#) methods, only if that items are present in the data source used with the AutoComplete. Also it supports to save the history while closing the application and to load the history while opening the application using the [SaveHistory\(\)](#) and [LoadHistory\(\)](#) methods.

The following mentioned code example can be used to attain these functionalities.

C#

```
AutoComplete autoComplete1 = new AutoComplete();
autoComplete1.Loaded += new RoutedEventHandler
(autoComplete1_Loaded);
void autoComplete1_Loaded(object sender, RoutedEventArgs e)
{
    if (autoComplete1 != null)
        autoComplete1.LoadHistory();
}
autoComplete1.SelectionChanged += new
SelectionChangedEventHandler(autoComplete1_SelectionChanged);
private void autoComplete1_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    autoComplete1.AddHistory(autoComplete1.SelectedItem);
    autoComplete1.SaveHistory();
}
```

Tables for methods, and events

Methods

[AddHistory\(String\)](#)

[AddHistory\(Object\)](#)

[SaveHistory](#)

[LoadHistory](#)

[ClearAllHistory](#)

Sample link

WPF Sample Browser-> Tools -> Editors -> AutoComplete Demo

Popup Resize Support in WPF AutoComplete (Classic)

AutoComplete allows you to resize the drop-down list popup using the [CanResizePopup](#) property. If this property is set as True the thumb will be shown in the right bottom corner of the drop-down list which adjusts the height and the width of the popup at runtime. If this property is set as False, the visibility of the thumb will be collapsed and you will not be able to resize the popup at runtime.

Adding pop-up resizing support to an application

You can use [CanResizePopup](#) property to attain this functionality by setting the value as True.

XML

```
<syncfusion:AutoComplete x:Name="AutoComplete1" CanResizePopup="true" />
```

C#

```
AutoComplete autoComplete1 = new AutoComplete();
autoComplete1.CanResizePopup = true;
```

Sample link

WPF Sample Browser-> Tools -> Editors -> AutoComplete Demo

Appearance in WPF AutoComplete (Classic)

Theme

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

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



Show the shadow effect

You can show the shadow effect of AutoComplete by using [DropShadowEffect](#). You can also set the distance between the control and shadow by using [ShadowDepth](#) property.

Refer the below code for your reference.

XML

```
<syncfusion:AutoComplete Height="30" Width="150" Text="AutoComplete">
  <syncfusion:AutoComplete.Effect>
    <DropShadowEffect ShadowDepth="10" Color="LightGray" Opacity="2" />
  </syncfusion:AutoComplete.Effect>
</syncfusion:AutoComplete>
```

C#

```
AutoComplete autoComplete = new AutoComplete();
autoComplete.Height = 30;
autoComplete.Width = 150;
autoComplete.Text = "AutoComplete";
DropShadowEffect shadowEffect = new DropShadowEffect();
shadowEffect.ShadowDepth = 10;
shadowEffect.Color = Colors.LightGray;
shadowEffect.Opacity = 2;
autoComplete.Effect = shadowEffect;
grid.Children.Add(autoComplete);
```



How to

Bind the Business objects in WPF AutoComplete (Classic)

AutoComplete supports binding the Business objects as its Data Source.

The Business objects can be bound to the AutoComplete by using the CustomSource property and also by setting the value of the Source property as Custom. If the Source property value is set as FilePath or Registry, Binding of Business objects will not be supported.

Also when binding the Business objects, the DisplayMemberPath and the SelectedValuePath property values has to be given so that the DisplayMemberPath value displays in the drop-down list and the SelectedValuePath assigns values to the SelectedValue property.

You can bind the Business objects to the AutoComplete as mentioned below.

XML

```
<syncfusion:AutoComplete x:Name="AutoCompleteTextBox" Source="Custom"
DisplayMemberPath="Name" SelectedValuePath="EmployeeID">
<syncfusion:AutoComplete.CustomSource>
<local:EmployeeListCollection/>
</syncfusion:AutoComplete.CustomSource>
</syncfusion:AutoComplete></td></tr>
```

C#

```
public class EmployeeList
{
    public int EmployeeID { get; set; }
    public string Name { get; set; }
    public string Mailid { get; set; }
    public EmployeeList() { }
    public EmployeeList(string name, string mail, int id)
    {
        Name = name;
        Mailid = mail;
        EmployeeID = id;
    }
}

public class EmployeeListCollection : ObservableCollection<EmployeeList>
{
    public EmployeeListCollection()
    {
        this.Add(new EmployeeList() { EmployeeID = 1001, Name = "John", Mailid = "john@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1002, Name = "Jerry", Mailid = "Jerry@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1003, Name = "Brad", Mailid = "Brad@syncfusion.com" });
        this.Add(new EmployeeList() { EmployeeID = 1004, Name = "lanze", Mailid = "lanze@syncfusion.com" });
    }
}
```

```
this.Add(new EmployeeList() { EmployeeID = 1005, Name = "Chambel", Mailid =  
"Chambel@syncfusion.com" });  
this.Add(new EmployeeList() { EmployeeID = 1006, Name = "Crimson", Mailid =  
"Crimson@syncfusion.com" });  
}  
}
```

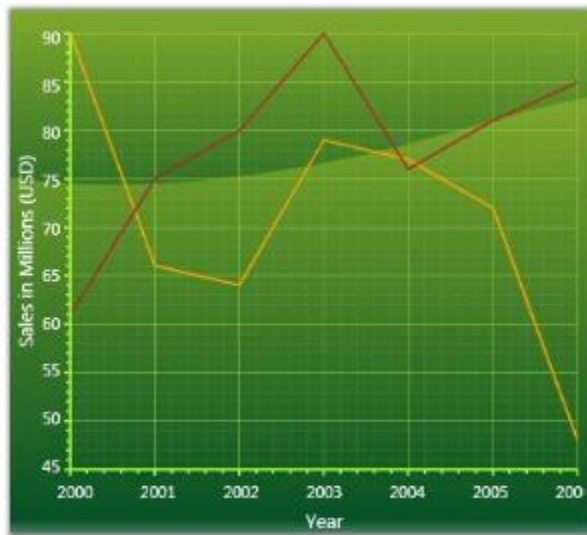
Chart (Classic)

WPF Chart (Classic) Overview

Essential Chart for WPF is a feature rich, highly customizable and presentable business Chart control. It can be used for easy understanding of large quantities of data and relationship between the data. Essential Chart enhances the readability and understandability of the raw data. With pre-built support for all kinds of list-based data sources and a very flexible template model, the control offers maximum flexibility with a very simple and straight-forward object model. The Chart control allows you to implement animations and user-interactions easily.

Essential Chart for WPF is intended for developers looking to add advanced, feature rich, visually appealing charts to the WPF applications.

The following image shows an example of a Chart control (line chart):



Real World Scenarios

Chart is used to show the graphical representation of two values. For example, a Line Chart can be used to depict the variation of sales in the recent years as shown in Figure 1.

Logarithmic charts can be used in Share Price charts where you plot between price and time. In logarithmic chart, you can identify the proportional change in price with respect to the change in time. Proportional change in price is used to observe the market sentiment.

Key Features

The following are the key features of Essential Chart for WPF:

- Chart customization-Essential Chart provides easy customization for each and every unit of chart such as Chart, Chart Area, Chart Series and Chart Legend

- Chart Data Binding-Essential Chart allows you to populate the chart with any kind of data source
- Chart Animation-Essential Chart supports various animation effects for good visualization
- Chart Templates-Chart control uses the benefits of WPF template concept, and enables to customize all parts of Chart with the templates customization option
- Chart Area-Multiple Chart Areas can be added to perform comparison of data at single view. Chart Area also comes with high layout customization.
- Chart Series-Highly customizable and interactive chart series can be added to Chart control
- Chart Types-Essential Chart supports more than 33 chart types
- 2-D and 3-D Appearance-Essential Chart supports 2-D and 3-D appearance of the Chart
- Chart Axis-Essential Chart allows to customizing the Chart Axis. Chart control also provides support of multiple axes
- Chart Legends-Essential Chart allows extensive customization of the legend. The position of the legend on the chart area as well as its representation aspects can be customized. Essential Chart also features modification of legend items using events. It also supports custom legend items that are not tied to any series of data.
- Chart Export, Import and Print-Essential Chart supports exporting, importing and printing functionalities

User Guide Organization

The product comes with numerous samples as well as an extensive documentation to guide you. This User Guide provides detailed information on the features and functionalities of the Essential Chart for WPF. It is organized into the following sections:

- Overview-This section gives a brief introduction to the product and its key features.
- Installation and Deployment-This section elaborates on the install location of the samples, license etc.
- What's New-This section lists the new features implemented for the latest release.
- Getting Started-This section guides you on getting started with WPF application and deploying Chart control etc.
- Concepts and Features-The features of the chart are illustrated with use case scenarios, code examples and screenshots under this section.

Document Conventions

The following conventions help you quickly identify the important sections of information, while using the content:

Convention	Description
Note	Represents important information.
Example	Represents an example.
Tip	Represents useful hints that helps you in using the controls and features.
Additional information	Represents additional information on the corresponding topic.

Getting Started with WPF Chart (Classic)

Feature Summary

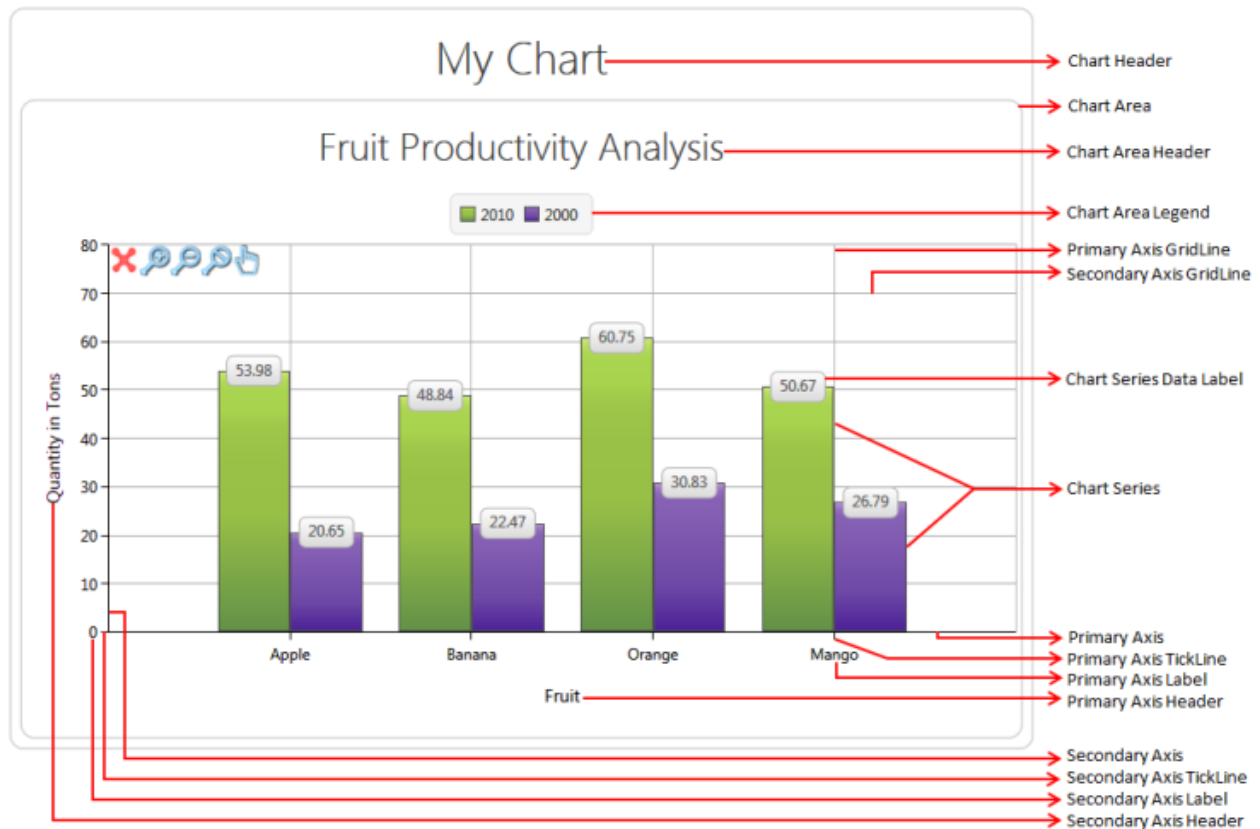
The complete summary of the features of the Chart control for WPF are listed as follows:

- Chart customization—Essential Chart provides easy customization for each and every unit of chart such as Chart, Chart Area, Chart Series and Chart Legend.
- Chart Data Binding—Essential Chart allows you to populate the chart with any kind of data source.
- Smart Tag Feature—Essential Chart provides sophisticated customization options.
- Chart Animation—Essential Chart supports various animation effects for good visualization.
- Chart Templates—Chart control uses the benefits of WPF template concept, and enables to customize all parts of Chart with the templates customization option.
- Chart Area—Multiple Chart Areas can be added to perform comparison of data at single view. Chart Area also comes with high layout customization.
- Chart Area Context Menu—WPF Chart has a built-in context menu which lets you change the chart type and color palette of a series and enable zooming.
- Chart Series—Highly customizable and interactive chart series can be added to Chart control.
- Chart Types—Essential Chart supports more than 33 chart types.
- 2-D and 3-D Appearance—Essential Chart supports 2-D and 3-D appearance of the Chart.
- Chart Axis—Essential Chart allows to customize the Chart Axis. Chart control also provides support of multiple axes.
- Chart Labels—Essential Chart allows you to customize the chart labels. The labels can be fetched from various data sources.
- Chart Skins—Essential Chart supports all basic OS themes and in addition supports 14 custom skins.
- Chart Legends—Essential Chart allows extensive customization of the legend. The position of the legend on the chart area as well as its representation aspects can be customized. Essential Chart also features modification of legend items using events. It also supports custom legend items that are not tied to any series of data.
- User Interaction—Essential Chart supports zooming and mouse events to enhance the user interaction.
- Chart Export, Import, and Print—Essential Chart supports exporting, importing and printing functionalities.
- Date Handling—Essential Chart provides built-in support for handling date. The data type of any series that is plotted on the chart can be set to Date Time. This also supports Date Time ranges and intervals.
- Multiple chart areas can share a single datetime x-axis for representing the data. This allows the user to use more than one area with a single axis.
- Enables synchronization of chart axis while using more than one area with a single axis.
- More than 34 chart types are available, which are highly and easily configurable with built-in support for creating stunning visual effects.
- High chart performance by implementing Fast Column, stacking column, Fast Scatter and HiLoOpenClose Chart types in WPF.
- Facility to select the zoomed area across both the Axis.
- Segmented horizontal and vertical striplines can be placed in the chart area.
- Complete customization support is available for the Chart Series.
- Animation feature is added to the chart with multiple animation options.

- Annotation with various Predefined shapes can be attached to the chart.
- Legend icons can be displayed according to the corresponding Chart Type.

Elaborate Structure of Chart Control

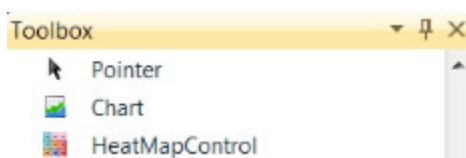
A chart is composed of various sub elements such as legends, areas, toolbars, headers, and footers. The following screenshot illustrates the elements of the Chart control.



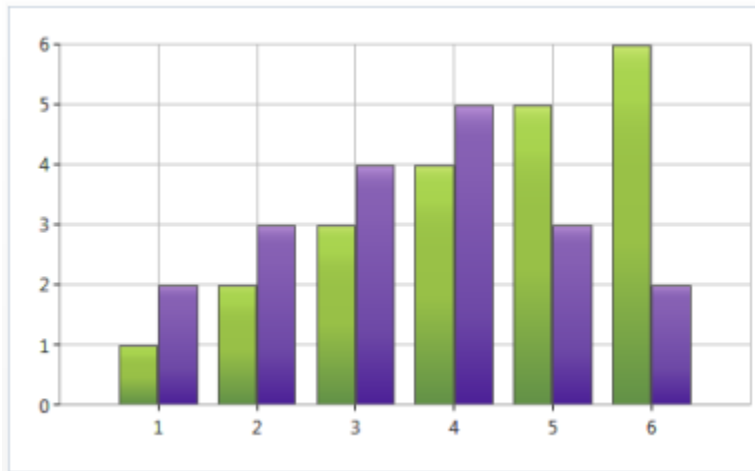
Creating a Simple Chart using Designer

Users can create a simple chart application with Syncfusion WPF Chart control without much of typing codes. It can be done with the help of Visual Studio designer and Syncfusion Smart Tag. The following are the steps to create a chart:

1. Create a new WPF application in VS2010. You can see the Chart control in the Toolbox of Visual Studio.



2. Drag the Chart control from the Toolbox and drop it to the designer. It generates a chart with two default series in the designer.

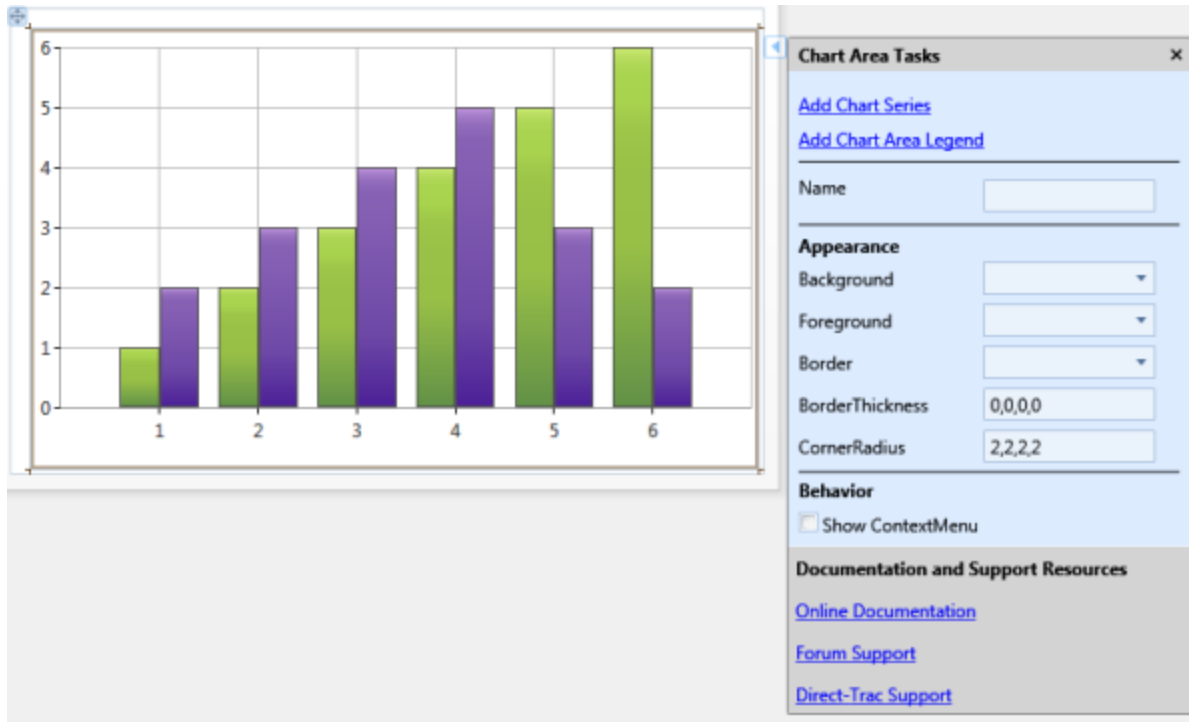


The following code is auto generated in the XAML window.

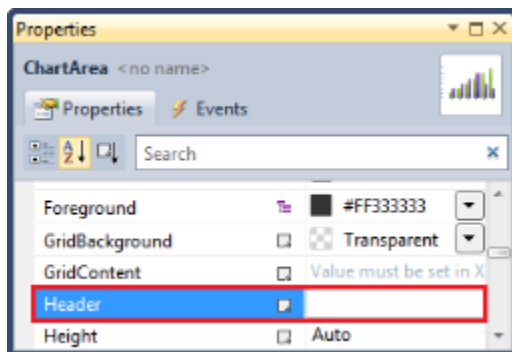
XML

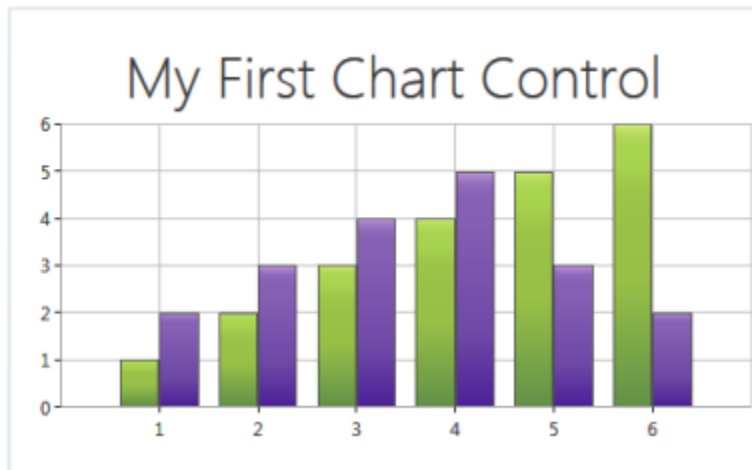
```
<syncfusion:Chart HorizontalAlignment="Left" Margin="10,10,0,0"
Name="chart1"
VerticalAlignment="Top">
<syncfusion:ChartArea ZoomInCoefficient="0.5" ZoomOutCoefficient="2">
<syncfusion:ChartSeries Data="1,1,2,2,3,3,4,4,5,5,6,6"
SegmentWidthMode="Fixed"/>
<syncfusion:ChartSeries Data="1,2,2,3,3,4,4,5,5,3,6,2"
SegmentWidthMode="Fixed"/>
</syncfusion:ChartArea>
</syncfusion:Chart>
```

3. Click the Expand button at the right corner of the chart area in the designer. It generates the Smart Tag for the chart area.

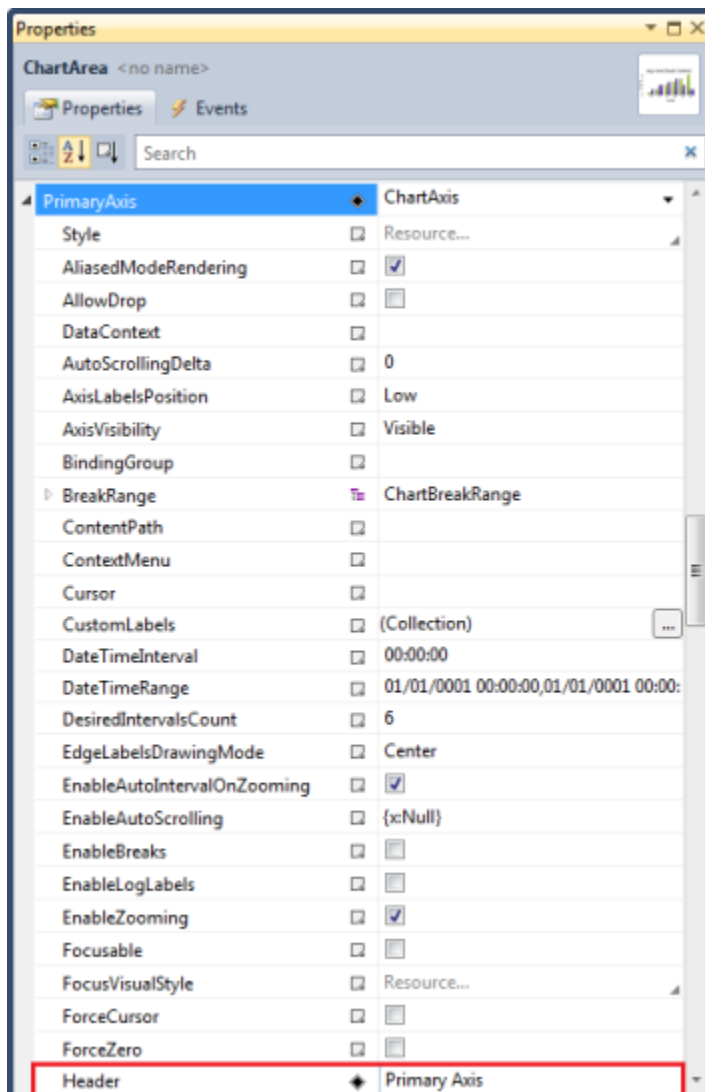


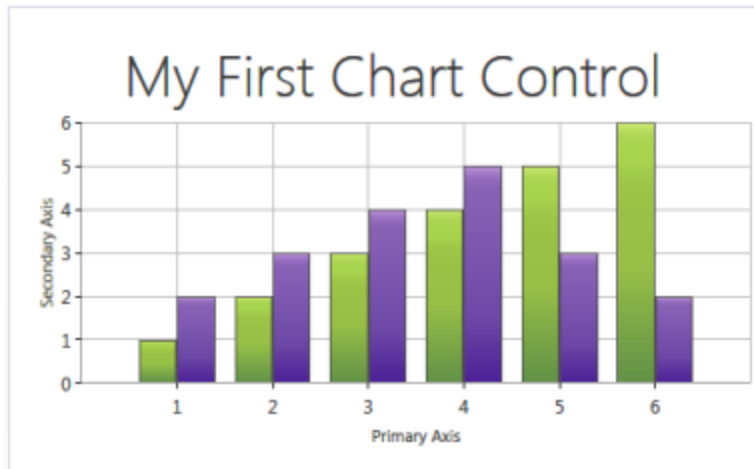
4. You can reuse this default series or else you can delete them and add a new series using the Smart Tag. Use DataSource property of ChartSeries to have the data collection bound to chart series, and to visualize the data in various forms such as bar chart, column chart, pie, pyramid, or other chart types. Refer to the section Chart Series to know more about data-binding to a chart series.
5. To add a title to the chart area, select the Properties tab in the chart area and enter a title for the chart area in the Header field.



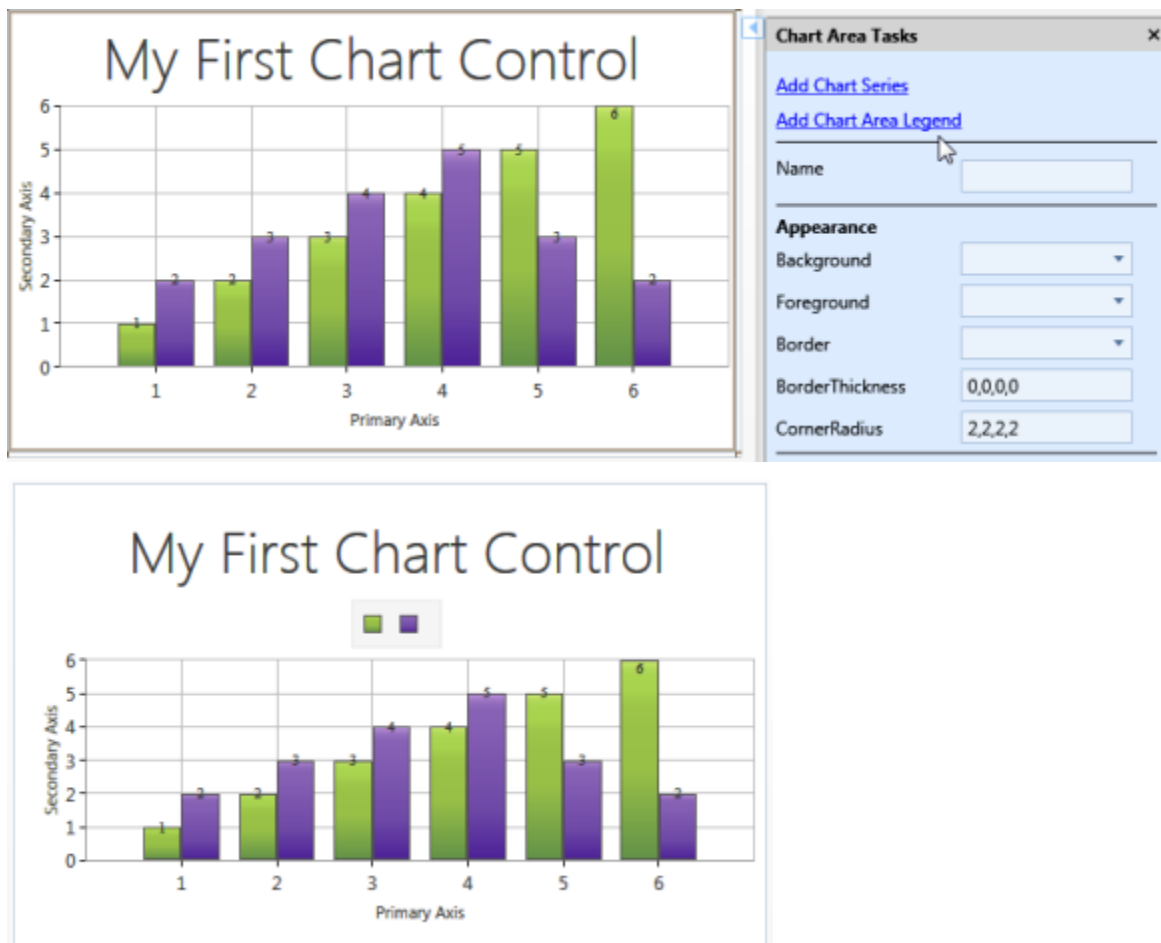


6. To set the axis header, expand the PrimaryAxis and enter the axis header in the Header field. Similarly, you can do this for the SecondaryAxis.

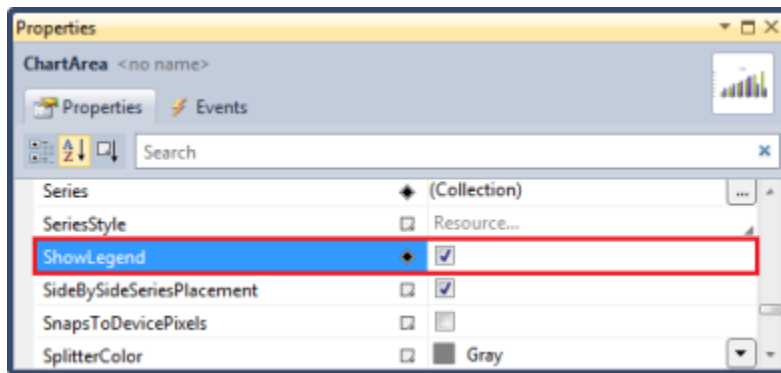




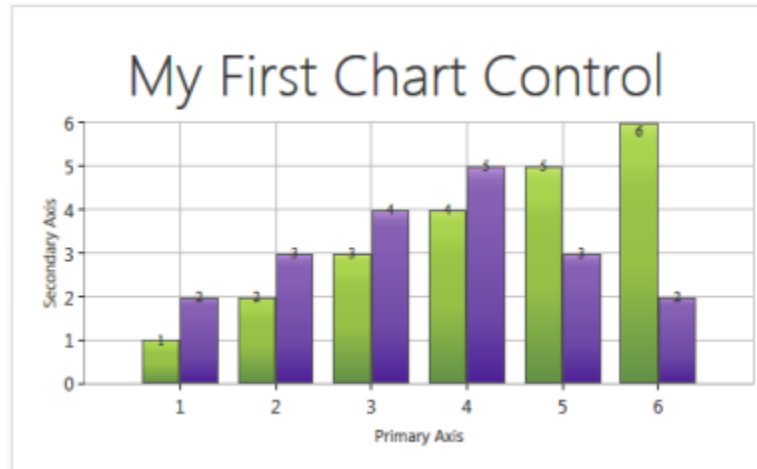
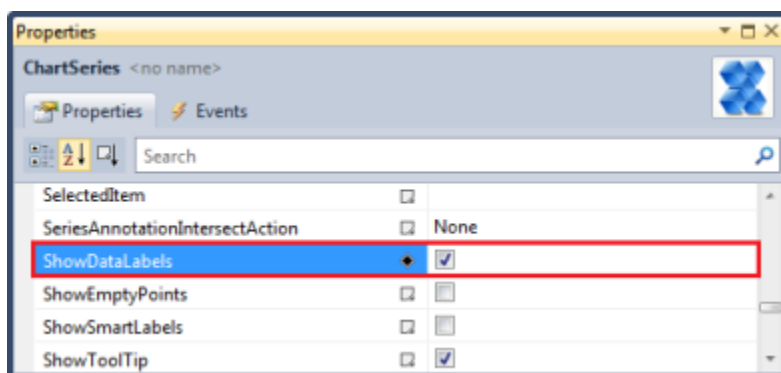
7. To add a legend to the chart area, click Add Chart Area Legend in the Smart Tag.



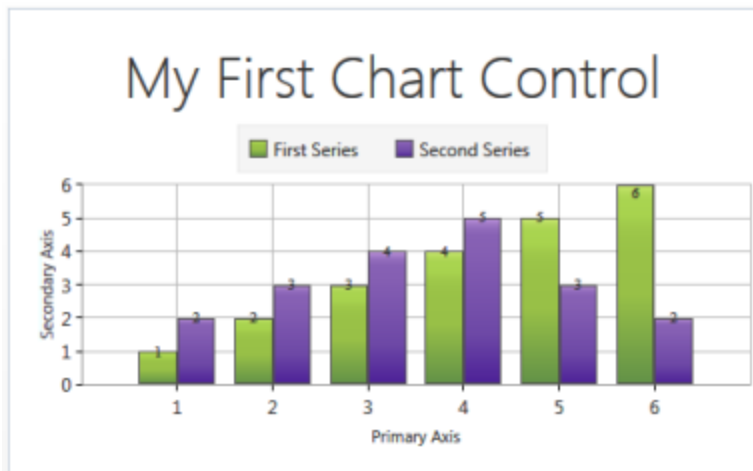
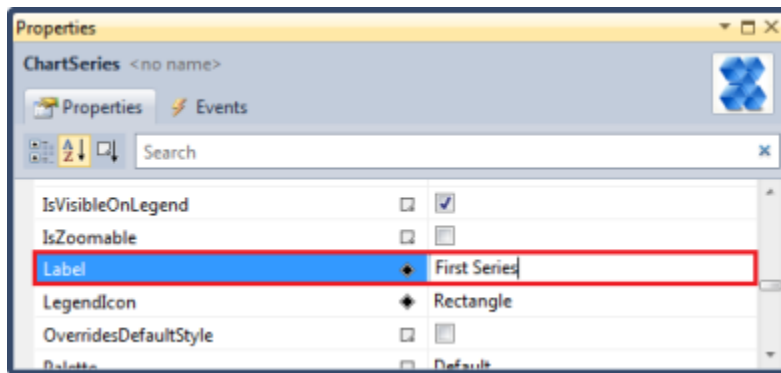
Note: The chart area legend can also be added in the chart area by setting the ShowLegend property of ChartArea as True.



8. To display the data labels, set the **ShowDataLabels** property of **ChartSeries** as **True**.



9. To set icon text of the chart legend, set the **Label** property of **ChartSeries**.

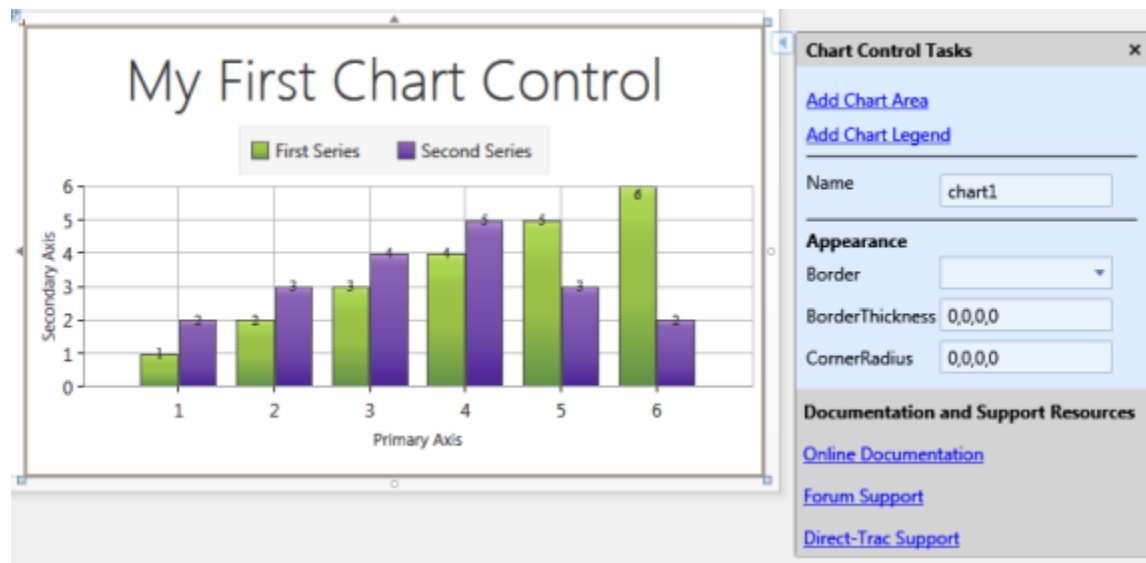
**XML**

```
<syncfusion:Chart HorizontalAlignment="Left" Margin="10,10,0,0"
Name="chart1"
VerticalAlignment="Top">
  <syncfusion:ChartArea ZoomInCoefficient="0.5" ZoomOutCoefficient="2"
Header="My First Chart Control">
    <syncfusion:ChartArea.Legend>
      <syncfusion:ChartLegend />
    </syncfusion:ChartArea.Legend>
    <syncfusion:ChartArea.SecondaryAxis>
      <syncfusion:ChartAxis Header="Secondary Axis" />
    </syncfusion:ChartArea.SecondaryAxis>
    <syncfusion:ChartArea.PrimaryAxis>
      <syncfusion:ChartAxis Header="Primary Axis" />
    </syncfusion:ChartArea.PrimaryAxis>
    <syncfusion:ChartSeries Label="FirstSeries" Data="1,1,2,2,3,3,4,4,5,5,6,6"
SegmentWidthMode="Fixed" ShowDataLabels="True"/>
    <syncfusion:ChartSeries Label="SecondSeries"
Data="1,2,2,3,3,4,4,5,5,3,6,2"
SegmentWidthMode="Fixed" ShowDataLabels="True"/>
  </syncfusion:ChartArea>
</syncfusion:Chart>
```

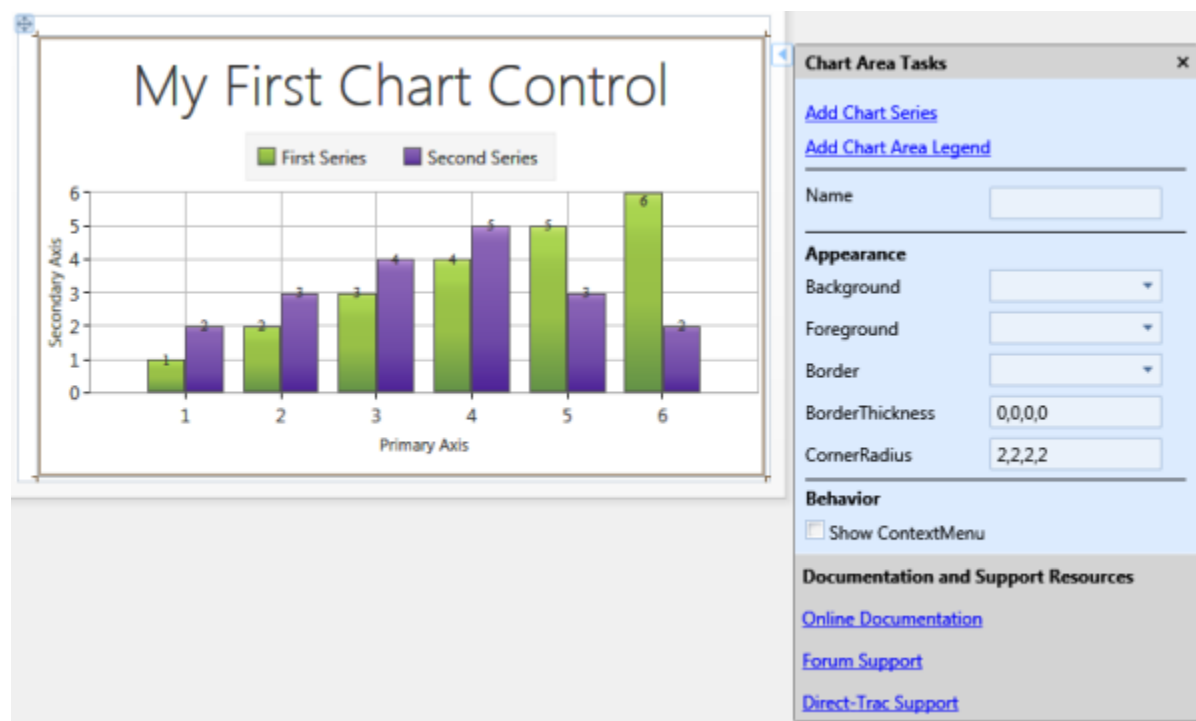
Chart Control Smart Tag Support

Smart Tag appears for the chart, chart area and chart legend. By using the Smart Tag, you can add chart areas, legends, and series to the chart. In addition, you can customize the chart appearance using the Smart Tag in the designer.

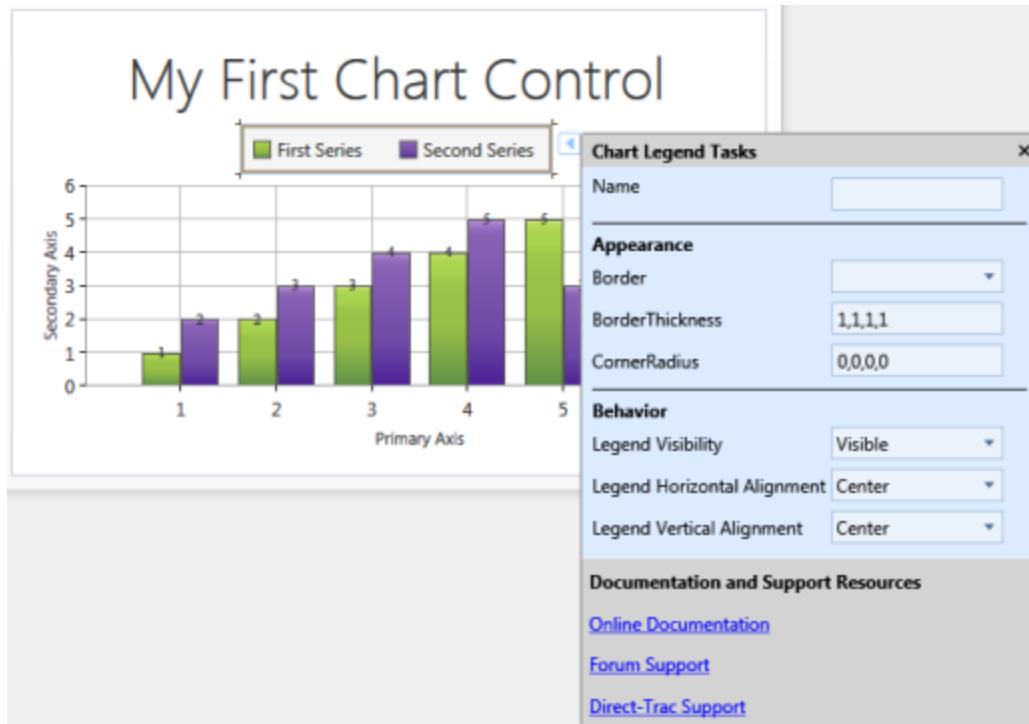
The following screenshot illustrates the Smart Tag for chart.



The following screenshot illustrates the Smart Tag for chart area.



The following screenshot illustrates the Smart Tag for chart legend.



Creating a Data-Binding Application for Chart Control

To create a data-binding application for Chart control:

1. Add the following Syncfusion assembly references to your project.

I. Syncfusion.Chart.WPF

II. Syncfusion.Shared.WPF

III. Syncfusion.Core

2. Create a namespace reference to Syncfusion.Chart.WPF using the Syncfusion global namespace reference schemas.syncfusion.com. The Chart control is available in the Syncfusion.Chart.WPF namespace of Syncfusion.Chart.WPF dll.

XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
```

C#

```
using Syncfusion.Windows.Chart;
```

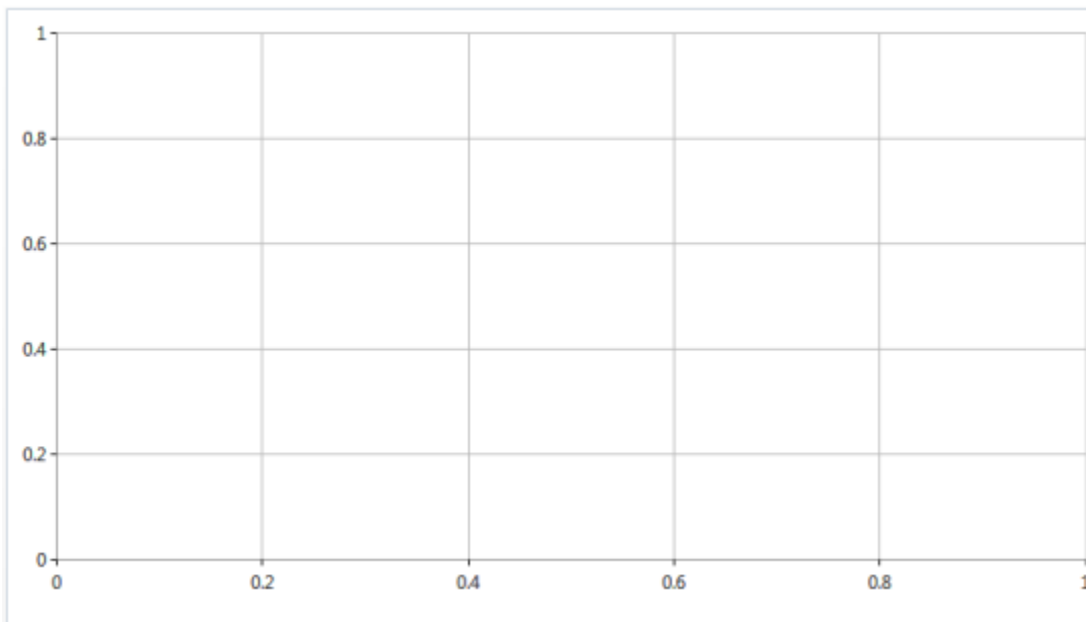
3. To create a simple series, initialize the Chart, ChartArea, and ChartSeries as in the following code example.

XML

```
<syncfusion:Chart>  
  <syncfusion:ChartArea>  
    <syncfusion:ChartSeries />  
  </syncfusion:ChartArea>  
</syncfusion:Chart>
```

CSHARP

```
Chart chart = new Chart();  
//Adding a new area.  
chart.Areas.Add(new ChartArea());  
//Adding a new series.  
chart.Areas[0].Series.Add(new ChartSeries());  
//Assigning window's content property.  
this.Content = chart;
```



4. To set a header to the chart area, use the Header property of ChartArea.

XML

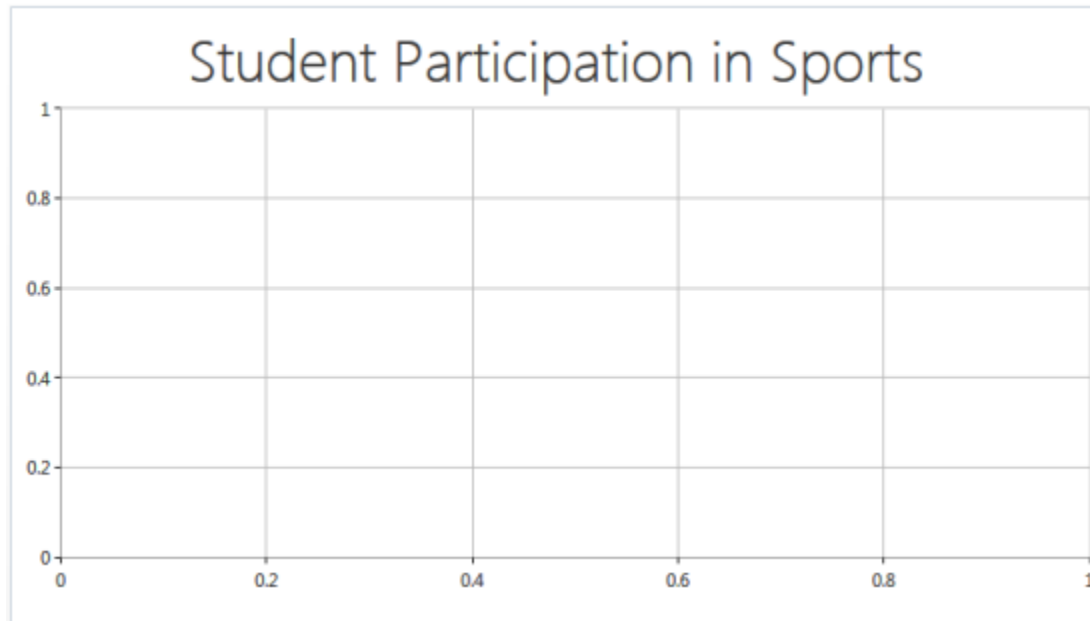
```
<syncfusion:Chart>  
  <syncfusion:ChartArea Header="Student Participation in Sports">  
    <syncfusion:ChartSeries/>  
  </syncfusion:ChartArea>  
</syncfusion:Chart>
```

CSHARP

```
Chart chart = new Chart();  
//Adding a new area.  
chart.Areas.Add(new ChartArea());  
//Adding a new series.  
chart.Areas[0].Series.Add(new ChartSeries());
```



```
//Assigning Header to chart area.
chart.Areas[0].Header = "Student Participation in Sports";
//Assigning window's content property.
this.Content = chart;
```



5. To set headers to the primary and secondary axes, use the Header property of ChartAxis.

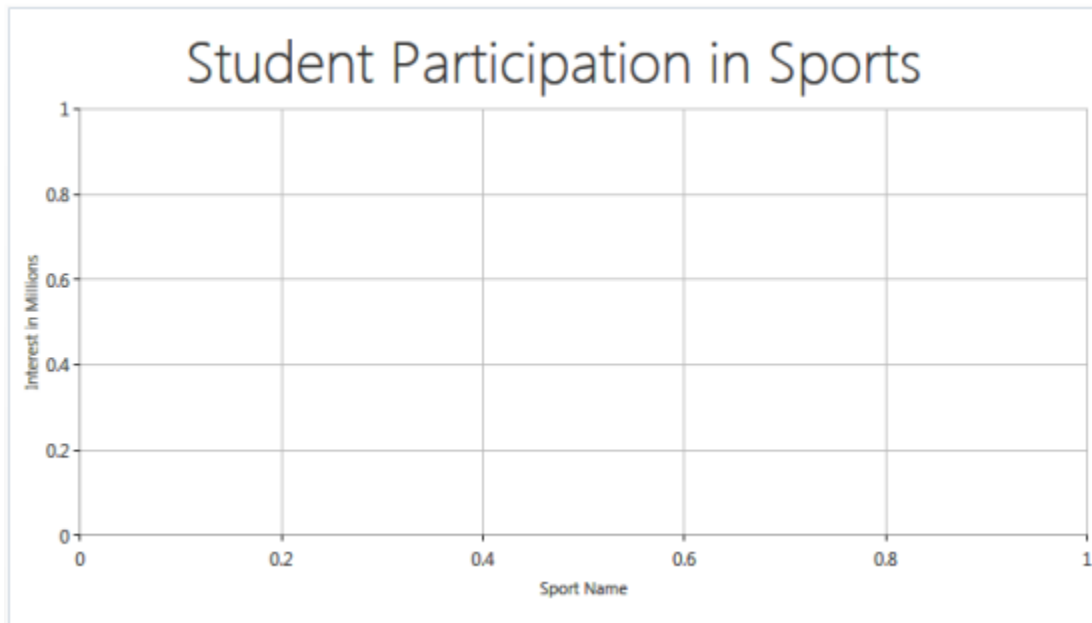
XML

```
<syncfusion:Chart>
  <syncfusion:ChartArea Header="Student Participation in Sports">
    <syncfusion:ChartArea.PrimaryAxis>
      <syncfusion:ChartAxis Header="Sport Name"/>
    </syncfusion:ChartArea.PrimaryAxis>
    <syncfusion:ChartArea.SecondaryAxis>
      <syncfusion:ChartAxis Header="Interest in Millions"/>
    </syncfusion:ChartArea.SecondaryAxis>
  </syncfusion:ChartArea>
</syncfusion:Chart>
```

CSHARP

```
Chart chart = new Chart();
//Adding a new area.
chart.Areas.Add(new ChartArea());
//Adding a new series.
chart.Areas[0].Series.Add(new ChartSeries());
//Assigning Header to chart area.
chart.Areas[0].Header = "Student Participation in Sports";
//Adding header to primary axis
chart.Areas[0].PrimaryAxis.Header = "Sport Name";
//Adding header to secondary axis
```

```
chart.Areas[0].SecondaryAxis.Header = "Interest in Millions";
//Assigning window's content property.
this.Content = chart;
```



6. Use the DataSource property of ChartSeries to bind collection objects with the chart series and to visualize data from collection.
7. Create a class to represent a chart point. Each instance of this class is visualized as a chart point in the chart series. Make sure that the class has two properties, one for positioning a point along x-axis and another one for positioning a point along y-axis.

Note: Y mapping property should be a double compatible type, and x mapping property can be double string, DateTime, TimeSpan, and etc.

C#

```
public class Sport
{
    private int sportid;
    private string sportname;
    private double interest;
    public Sport()
    {
    }
    public Sport(int sportid, string sportname, double interests)
    {
        this.sportid = sportid;
        this.sportname = sportname;
        this.interest = interests;
    }
    public int SportID
    {
        get { return sportid; }
        set
        {

```

```

sportid = value;
}
}
public string SportName
{
    get { return sportname; }
    set
    {
        sportname = value;
    }
}
public double Interest
{
    get { return interest; }
    set
    {
        interest = value;
    }
}
}

```

8. Create a Model class with a collection property which is a collection of the previously defined class. For more details about the types of collection that chart supports refer to the section 4.2.2. Populate the collection property. In the following code example, the collection is populated using simple code simulation.

CSHARP

```

// Namespace to be included for ObservableCollection.
using System.Collections.ObjectModel;
public class Model
{
    public Model()
    {
        SportDetails = new ObservableCollection<Sport>();
        SportDetails.Add(new Sport(101, "Golf", 9));
        SportDetails.Add(new Sport(102, "Soccer", 37));
        SportDetails.Add(new Sport(103, "Hockey", 10));
        SportDetails.Add(new Sport(104, "Rugby", 17));
        SportDetails.Add(new Sport(105, "Shuttle", 30));
        SportDetails.Add(new Sport(106, "Cricket", 15));
        SportDetails.Add(new Sport(107, "Baseball", 6));
        SportDetails.Add(new Sport(108, "Tennis", 10));
    }
    public ObservableCollection<Sport> SportDetails
    {
        get;
        set;
    }
}

```

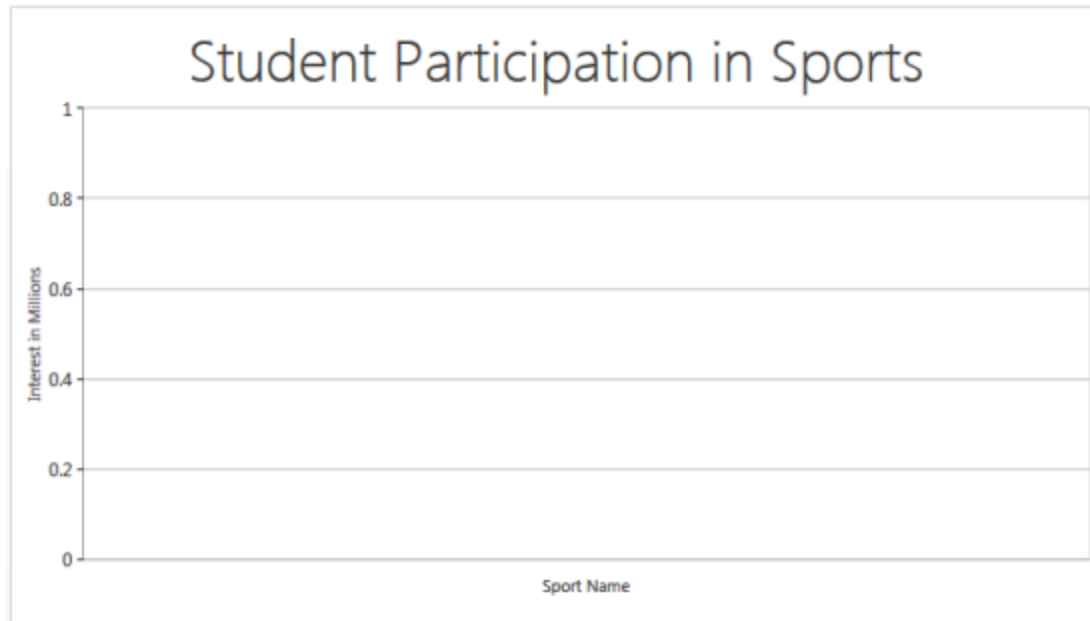
9. Bind the collection property to the DataSource property of ChartSeries.

XML

```
//Reference to Application
xmlns:local="clr-namespace:GettingStarted"
.....
<Grid.DataContext>
<local:Model/>
</Grid.DataContext>
<syncfusion:Chart>
<syncfusion:ChartArea Header="Student Participation in Sports">
<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis Header="Sport Name"/>
</syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartAxis Header="Interest in Millions"/>
</syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartSeries DataSource="{Binding SportDetails}"/>
</syncfusion:ChartArea>
</syncfusion:Chart>
```

CSHARP

```
Chart chart = new Chart();
//Adding a new area.
chart.Areas.Add(new ChartArea());
//Adding a new series.
chart.Areas[0].Series.Add(new ChartSeries());
//Assigning Header to chart area.
chart.Areas[0].Header = "Student Participation in Sports";
//Adding header to primary axis
chart.Areas[0].PrimaryAxis.Header = "Sport Name";
//Adding header to secondary axis
chart.Areas[0].SecondaryAxis.Header = "Interest in Millions";
//Assigning DataContext with the model
this.DataContext = new Model();
//Creating a new binding for DataSource
Binding dataSrcBinding = new Binding("SportDetails")
{ Source = this.DataContext };
// Binding sport details to Datasource
BindingOperations.SetBinding(chart.Areas[0].Series[0],
ChartSeries.DataSourceProperty,
dataSrcBinding);
//Assigning window's content property.
this.Content = chart;
```



10. If you do not have any model in your application, you can simply assign the DataSource with a collection object (which is a collection of objects of the class stated in point 7).

CSHARP

```
// Namespace to be included for ObservableCollection.
using System.Collections.ObjectModel;
public class SportCollection : ObservableCollection<Sport>
{
    public SportCollection()
    {
        this.Add(new Sport() { SportID = 101, SportName = "Golf", Interest = 9 });
        this.Add(new Sport() { SportID = 102, SportName = "Soccer", Interest = 37 });
        this.Add(new Sport() { SportID = 103, SportName = "Hockey", Interest = 10 });
        this.Add(new Sport() { SportID = 104, SportName = "Rugby", Interest = 17 });
        this.Add(new Sport() { SportID = 105, SportName = "Shuttle", Interest = 30 });
        this.Add(new Sport() { SportID = 106, SportName = "Cricket", Interest = 15 });
        this.Add(new Sport() { SportID = 107, SportName = "Baseball", Interest = 6 });
        this.Add(new Sport() { SportID = 108, SportName = "Tennis", Interest = 10 });
    }
}
```

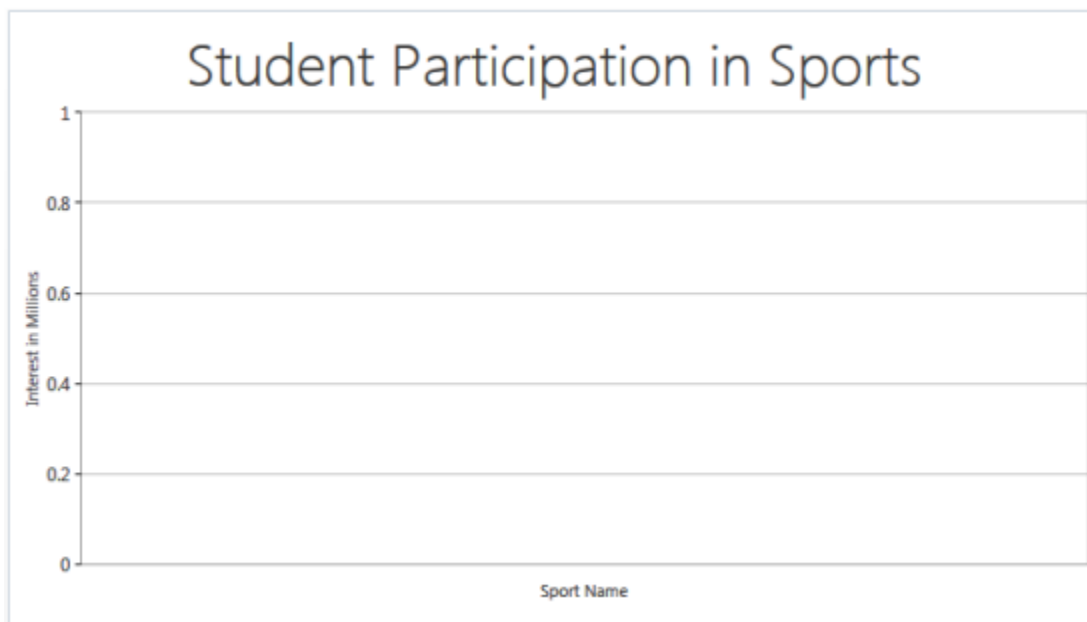
XML

```
//Reference to application
xmlns:local="clr-namespace:GettingStarted"
.....
<Window.Resources>
```

```
<local:SportCollection x:Key="sportCollection"/>
</Window.Resources>
<syncfusion:Chart>
<syncfusion:ChartArea Header="Student Participation in Sports">
<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis Header="Sport Name"/>
</syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartAxis Header="Interest in Millions"/>
</syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartSeries DataSource="{StaticResource sportCollection}"/>
</syncfusion:ChartArea>
</syncfusion:Chart>
```

CSHARP

```
Chart chart = new Chart();
//Adding a new area.
chart.Areas.Add(new ChartArea());
//Adding a new series.
chart.Areas[0].Series.Add(new ChartSeries());
//Assigning a header to chart area.
chart.Areas[0].Header = "Student Participation in Sports";
//Adding header to primary axis
chart.Areas[0].PrimaryAxis.Header = "Sport Name";
//Adding header to secondary axis
chart.Areas[0].SecondaryAxis.Header = "Interest in Millions";
// Assigning sport collection to Datasource
chart.Areas[0].Series[0].DataSource = new SportCollection();
//Assigning Window's content property.
this.Content = chart;
```



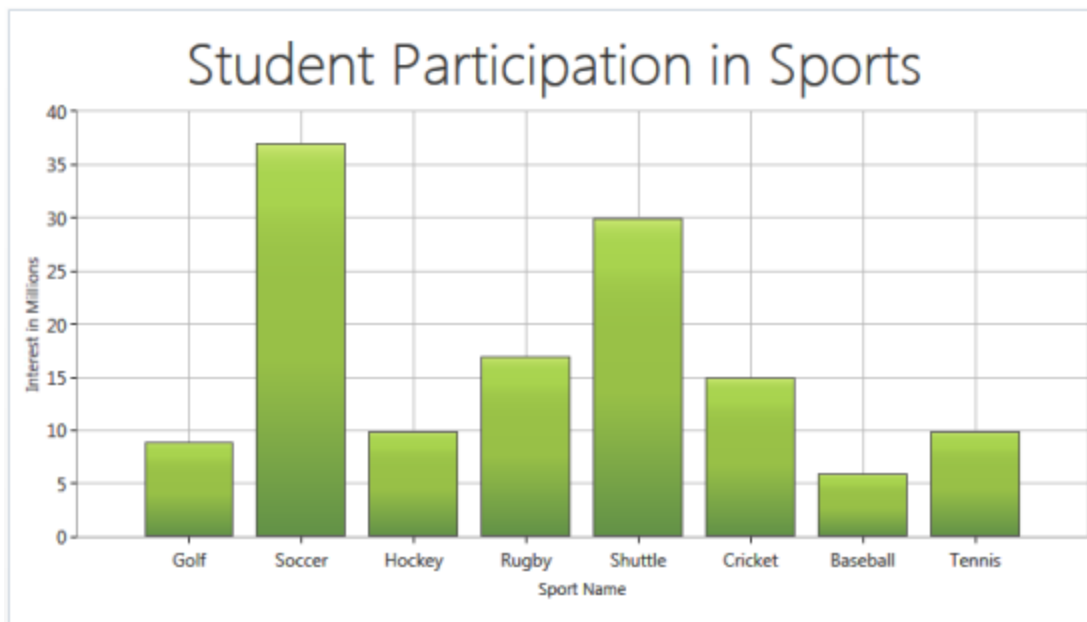
11. Specify the `BindingPathX` and `BindingPathsY` with mapping names. The mapping names are simply the names of the properties whose value has to be taken for x-axis value and y-axis values, respectively. Running your project can have a chart with a column type loaded with the data points specified by you.

XML

```
<syncfusion:Chart>
<syncfusion:ChartArea Header="Student Participation in Sports">
<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis Header="Sport Name"/>
</syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartAxis Header="Interest in Millions"/>
</syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartSeries DataSource="{StaticResource sportCollection}"
BindingPathX="SportName"
BindingPathsY="Interest"/>
</syncfusion:ChartArea>
</syncfusion:Chart>
```

C#

```
Chart chart = new Chart();
//Adding a new area.
chart.Areas.Add(new ChartArea());
//Adding a new series.
chart.Areas[0].Series.Add(new ChartSeries());
//Assigning header to chart area.
chart.Areas[0].Header = "Student Participation in Sports";
//Adding header to primary axis
chart.Areas[0].PrimaryAxis.Header = "Sport Name";
//Adding header to secondary axis
chart.Areas[0].SecondaryAxis.Header = "Interest in Millions";
// Assigning sport collection to Datasource
chart.Areas[0].Series[0].DataSource = new SportCollection();
//Setting BindingPathX for series
chart.Areas[0].Series[0].BindingPathX = "SportName";
//Setting BindingPathsY for series
chart.Areas[0].Series[0].BindingPathsY = new string[] { "Interest" };
//Assigning Window's content property.
this.Content = chart;
```



12. To add data labels to chart points, set `ShowDataLabels` property of `ChartSeries` as true. You can modify the default values and appearance of the labels using the `AdornmentsInfo` property of `ChartSeries` (instance of the `ChartAdornmentsInfo`). The `ChartAdornmentsInfo` class is used to configure the settings of data label positions, label contents, symbols, and etc.

XML

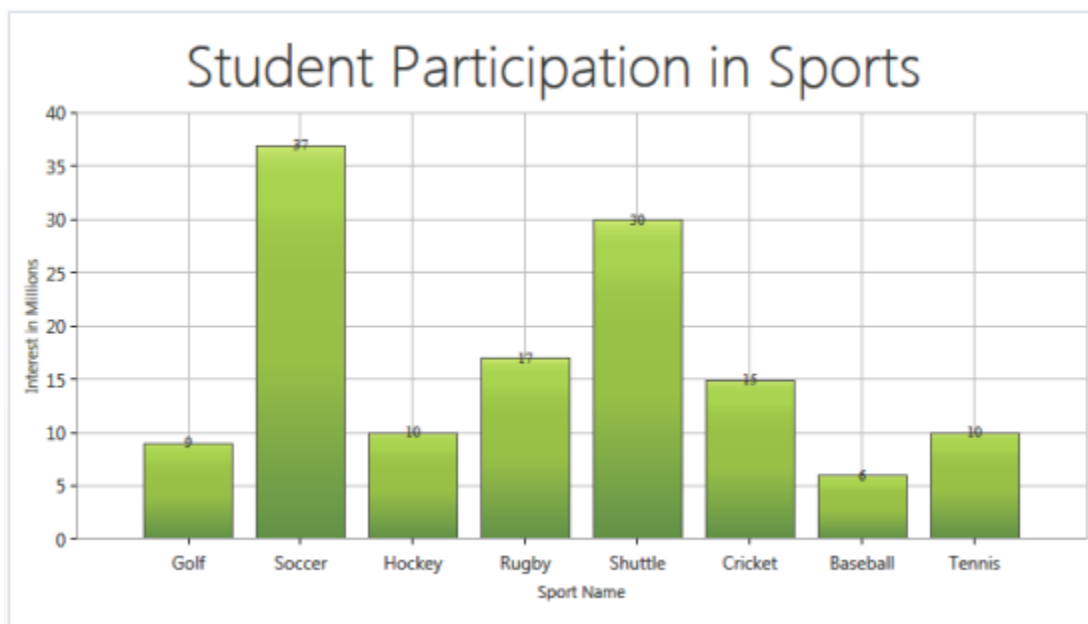
```
<syncfusion:Chart>
  <syncfusion:ChartArea Header="Student Participation in Sports">
    <syncfusion:ChartArea.PrimaryAxis>
      <syncfusion:ChartAxis Header="Sport Name"/>
    </syncfusion:ChartArea.PrimaryAxis>
    <syncfusion:ChartArea.SecondaryAxis>
      <syncfusion:ChartAxis Header="Interest in Millions"/>
    </syncfusion:ChartArea.SecondaryAxis>
    <syncfusion:ChartSeries ShowDataLabels="True"
      DataSource="{StaticResource sportCollection}"
      BindingPathX="SportName"
      BindingPathsY="Interest"/>
  </syncfusion:ChartArea>
</syncfusion:Chart>
```

C#

```
Chart chart = new Chart();
//Adding a new area.
chart.Areas.Add(new ChartArea());
//Adding a new series.
chart.Areas[0].Series.Add(new ChartSeries());
//Assigning header to chart area.
chart.Areas[0].Header = "Student Participation in Sports";
//Adding header to primary axis
chart.Areas[0].PrimaryAxis.Header = "Sport Name";
```



```
//Adding header to secondary axis
chart.Areas[0].SecondaryAxis.Header = "Interest in Millions";
// Assigning sport collection to Datasource
chart.Areas[0].Series[0].DataSource = new SportCollection();
//Setting BindingPathX for series
chart.Areas[0].Series[0].BindingPathX = "SportName";
//Setting BindingPathsY for series
chart.Areas[0].Series[0].BindingPathsY = new string[] { "Interest" };
//Setting DataLabels to chart series.
chart.Areas[0].Series[0].ShowDataLabels = true;
//Assigning window's content property.
this.Content = chart;
```



13. To add legends to the chart area, set ShowLegend property of ChartArea as true.

XML

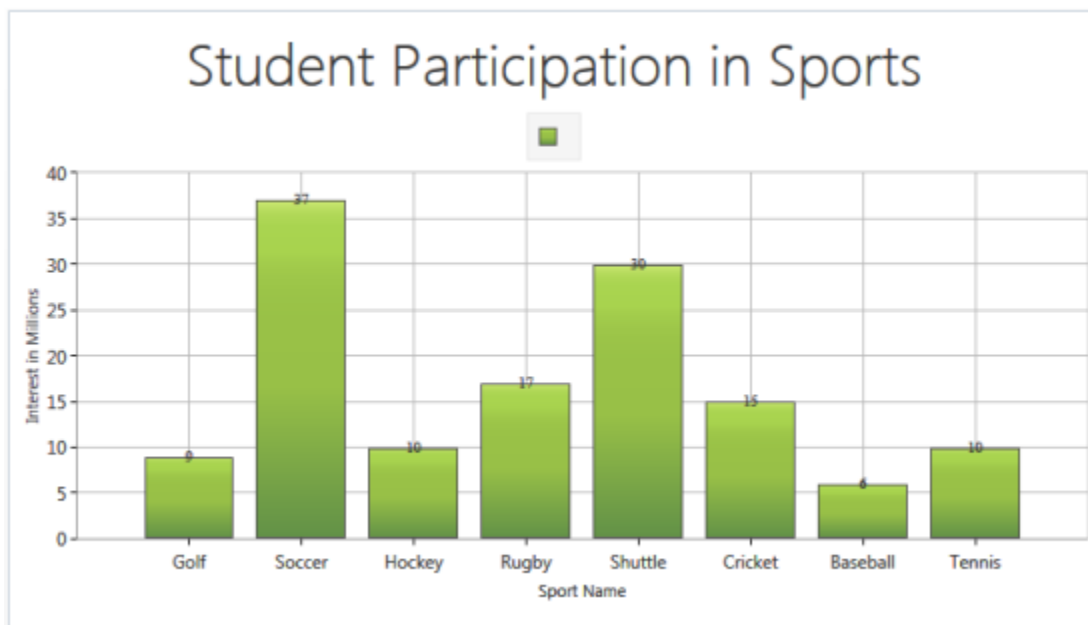
```
<syncfusion:Chart>
  <syncfusion:ChartArea ShowLegend="True"
    Header="Student Participation in Sports">
    <syncfusion:ChartArea.PrimaryAxis>
      <syncfusion:ChartAxis Header="Sport Name"/>
    </syncfusion:ChartArea.PrimaryAxis>
    <syncfusion:ChartArea.SecondaryAxis>
      <syncfusion:ChartAxis Header="Interest in Millions"/>
    </syncfusion:ChartArea.SecondaryAxis>
    <syncfusion:ChartSeries ShowDataLabels="True"
      DataSource="{StaticResource sportCollection}"
      BindingPathX="SportName"
      BindingPathsY="Interest"/>
    </syncfusion:ChartArea>
  </syncfusion:Chart>
```

CSHARP

```

Chart chart = new Chart();
//Adding a new area.
chart.Areas.Add(new ChartArea());
//Setting Legend to chart area.
chart.Areas[0].ShowLegend = true;
//Adding a new series.
chart.Areas[0].Series.Add(new ChartSeries());
//Assigning Header to chart area.
chart.Areas[0].Header = "Student Participation in Sports";
//Adding header to primary axis
chart.Areas[0].PrimaryAxis.Header = "Sport Name";
//Adding header to secondary axis
chart.Areas[0].SecondaryAxis.Header = "Interest in Millions";
// Assigning sport collection to Datasource
chart.Areas[0].Series[0].DataSource = new SportCollection();
//Setting BindingPathX for series
chart.Areas[0].Series[0].BindingPathX = "SportName";
//Setting BindingPathsY for series
chart.Areas[0].Series[0].BindingPathsY = new string[] { "Interest" };
//Setting DataLabels to chart series.
chart.Areas[0].Series[0].ShowDataLabels = true;
//Assigning window's content property.
this.Content = chart;

```



14. To have an icon label in the legend, set the Label property of ChartSeries with a required string. To modify the default icon of the legend, use the LegendIcon property of ChartSeries.

XML

```

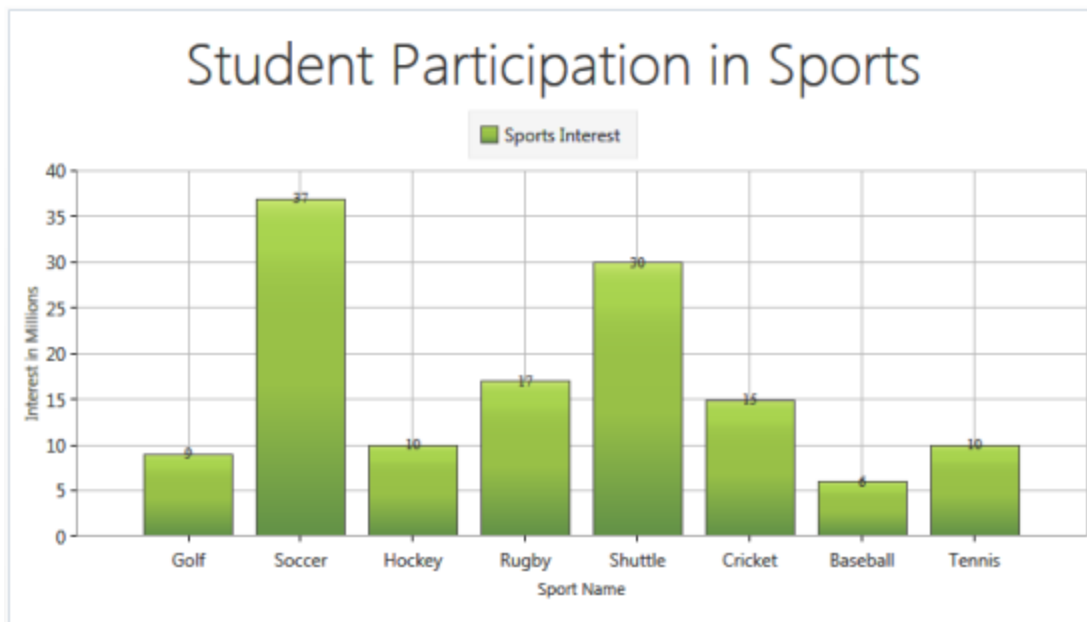
<syncfusion:Chart>
  <syncfusion:ChartArea ShowLegend="True"
    Header="Student Participation in Sports">

```

```
<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis Header="Sport Name"/>
</syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartAxis Header="Interest in Millions"/>
</syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartSeries Label="Sports Interest"
ShowDataLabels="True"
DataSource="{StaticResource sportCollection}"
BindingPathX="SportName"
BindingPathsY="Interest"/>
</syncfusion:ChartArea>
</syncfusion:Chart>
```

CSHARP

```
Chart chart = new Chart();
//Adding a new area.
chart.Areas.Add(new ChartArea());
//Setting legend to chart area.
chart.Areas[0].ShowLegend = true;
//Adding a new series.
chart.Areas[0].Series.Add(new ChartSeries());
//Assigning header to chart area.
chart.Areas[0].Header = "Student Participation in Sports";
//Adding header to primary axis
chart.Areas[0].PrimaryAxis.Header = "Sport Name";
//Adding header to secondary axis
chart.Areas[0].SecondaryAxis.Header = "Interest in Millions";
// Assigning sport collection to Datasource
chart.Areas[0].Series[0].DataSource = new SportCollection();
//Setting BindingPathX for series
chart.Areas[0].Series[0].BindingPathX = "SportName";
//Setting BindingPathsY for series
chart.Areas[0].Series[0].BindingPathsY = new string[] { "Interest" };
//Setting data labels to chart series.
chart.Areas[0].Series[0].ShowDataLabels = true;
//Setting legend icon text by assigning label of chart series.
chart.Areas[0].Series[0].Label = "First Series";
//Assigning Window's content property.
this.Content = chart;
```



15. To change the default legend icon, set the `LegendIcon` property of `ChartSeries` with an icon type.

XML

```
<syncfusion:Chart>
  <syncfusion:ChartArea ShowLegend="True"
    Header="Student Participation in Sports">
    <syncfusion:ChartArea.PrimaryAxis>
      <syncfusion:ChartAxis Header="Sport Name"/>
    </syncfusion:ChartArea.PrimaryAxis>
    <syncfusion:ChartArea.SecondaryAxis>
      <syncfusion:ChartAxis Header="Interest in Millions"/>
    </syncfusion:ChartArea.SecondaryAxis>
    <syncfusion:ChartSeries LegendIcon="Pentagon"
      Label="Sports Interest"
      ShowDataLabels="True"
      DataSource="{StaticResource sportCollection}"
      BindingPathX="SportName"
      BindingPathsY="Interest"/>
    </syncfusion:ChartArea>
  </syncfusion:Chart>
```

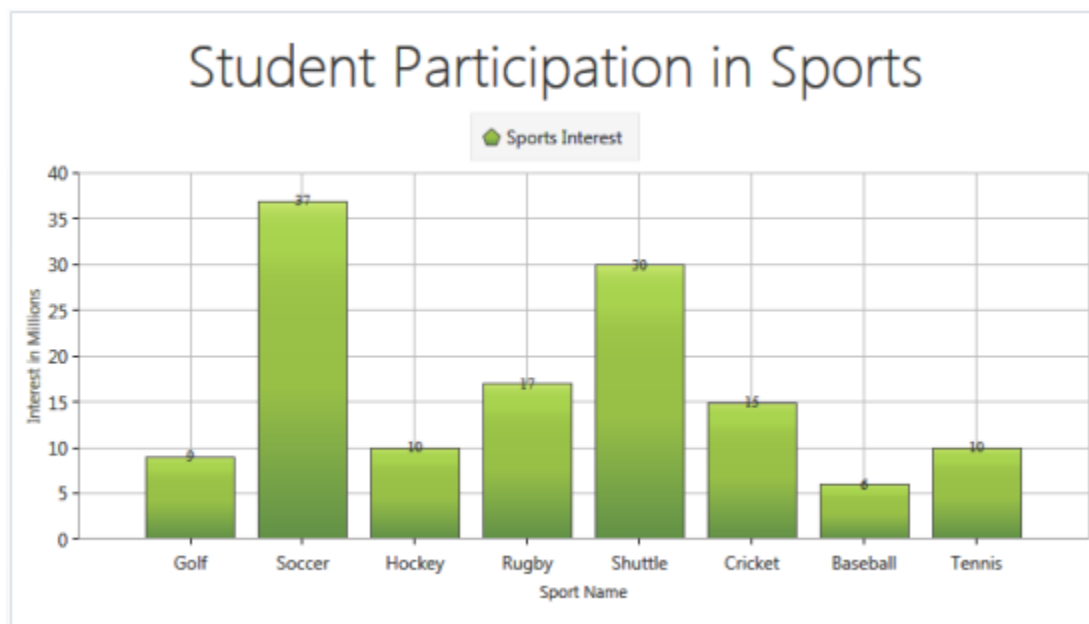
CSHARP

```
Chart chart = new Chart();
//Adding a new area.
chart.Areas.Add(new ChartArea());
//Setting legend to chart area.
chart.Areas[0].ShowLegend = true;
//Adding a new series.
chart.Areas[0].Series.Add(new ChartSeries());
//Assigning header to chart area.
chart.Areas[0].Header = "Student Participation in Sports";
//Adding header to primary axis
```

```

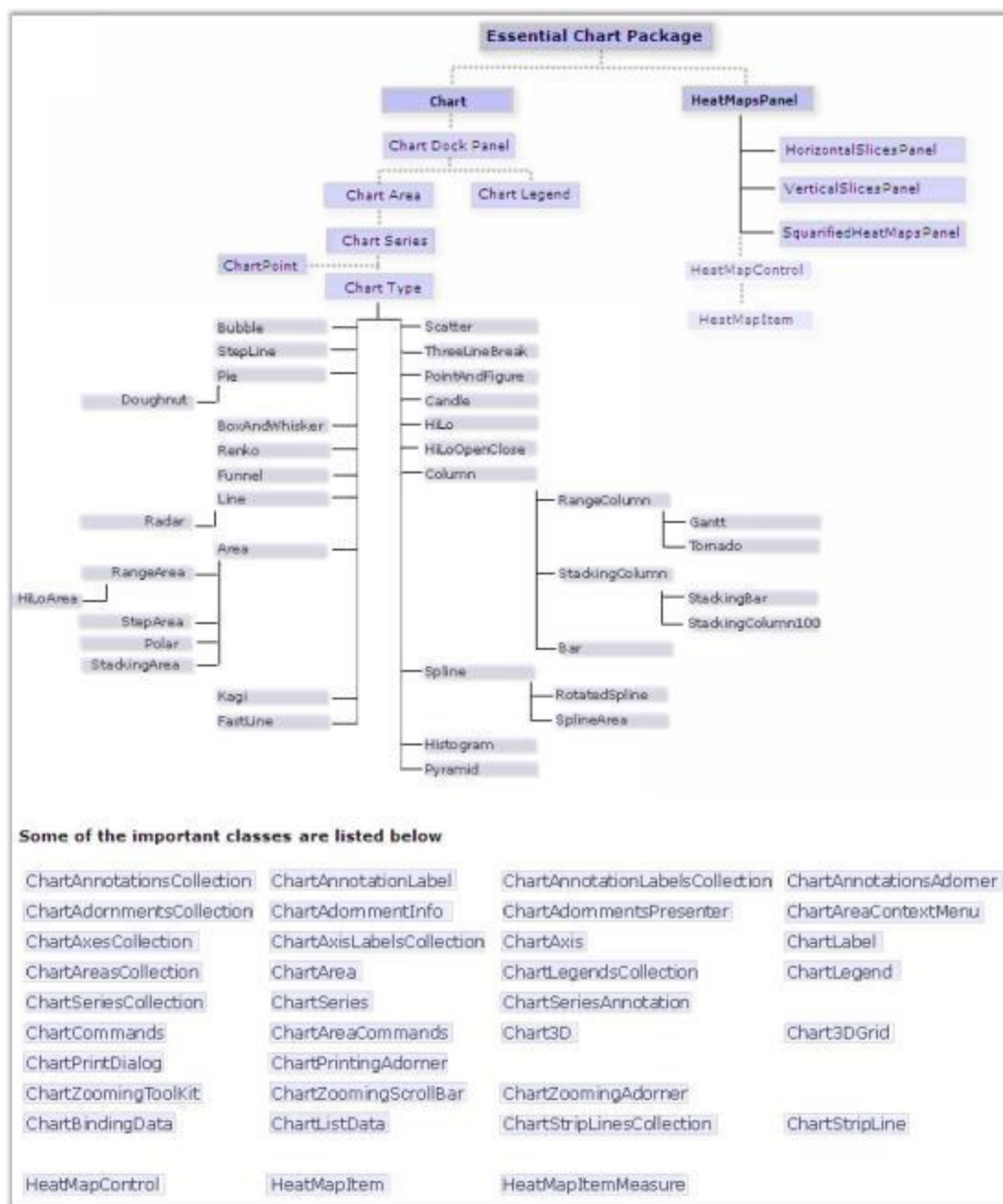
chart.Areas[0].PrimaryAxis.Header = "Sport Name";
//Adding header to secondary axis
chart.Areas[0].SecondaryAxis.Header = "Interest in Millions";
// Assigning sport collection to Datasource
chart.Areas[0].Series[0].DataSource = new SportCollection();
//Setting BindingPathX for series
chart.Areas[0].Series[0].BindingPathX = "SportName";
//Setting BindingPathsY for series
chart.Areas[0].Series[0].BindingPathsY = new string[] { "Interest" };
//Setting DataLabels to chart series.
chart.Areas[0].Series[0].ShowDataLabels = true;
//Setting legend icon text by assigning label of chart series.
chart.Areas[0].Series[0].Label = "First Series";
//Setting legend icon text by assigning Label of chart series.
chart.Areas[0].Series[0].Label = "First Series";
//Assigning Window's content property.
this.Content = chart;

```



Class Diagram

The class diagram for Essential Chart for WPF is as follows.



Chart

A chart is a type of information graphic or graphic organizer that represents tabular numeric data and / or functions. Chart is often used to make it easier to understand large quantities of data and the relationship between different parts of the data. Charts can be read more quickly than the raw data using which they are created.

Property	Description
AnnotationLabels	gets or sets the Labels
AnnotationLabelTemplate	gets or sets the LabelTemplate

Area	gets the collection of ChartArea
AreasPanel	gets or sets the areas panel template
CornerRadius	gets or sets the corner radius
Legends	gets the collection of Chart Legends

Method	Description
CopyToClipboard	copies the chart to clipboard
OnApplyTemplate	invoked whenever application code or internal processes call ApplyTemplate()
Print	prints the chart by the specified print Area
Save	saves the chart to the file with specified file name using encoder
SaveToXps	saves to xps format
SwitchPrintingMode	switches to the printing mode

Data Binding in WPF Chart (Classic)

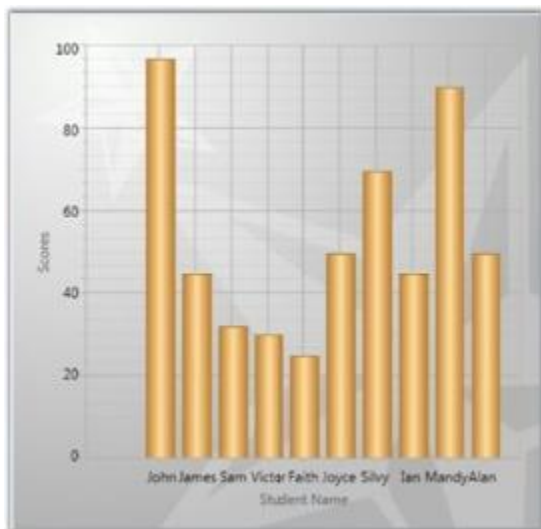
IList Data Source

Simple IList-based instances can be easily bound to the Chart. The following code example illustrates how to bind IList-based instances as the data source to Chart.

C#

```
public IList marks()
{
    List<mark> markList = new List<mark>();
    markList.Add(new mark() { ID = 0, Name = "John", Mark1 = 97, Mark2 = 99,
    Mark3 = 85, Mark4 = 92 });
    markList.Add(new mark() { ID = 1, Name = "James", Mark1 = 45, Mark2 = 35,
    Mark3 = 48, Mark4 = 42 });
    markList.Add(new mark() { ID = 2, Name = "Sam", Mark1 = 32, Mark2 = 65, Mark3
    = 67, Mark4 = 78 });
    markList.Add(new mark() { ID = 3, Name = "Victor", Mark1 = 30, Mark2 =
    39, Mark3 = 38, Mark4 = 56 });
    markList.Add(new mark() { ID = 4, Name = "Faith", Mark1 = 25, Mark2 = 45, Mark3
    = 77, Mark4 = 19 });
    markList.Add(new mark() { ID = 5, Name = "Joyce", Mark1 = 50, Mark2 = 35, Mark3
    = 54, Mark4 = 55 });
    markList.Add(new mark() { ID = 6, Name = "Silvy", Mark1 = 70, Mark2 = 28,
    Mark3 = 35, Mark4 = 45 });
    markList.Add(new mark() { ID = 7, Name = "Ian", Mark1 = 45, Mark2 = 85, Mark3 =
    77, Mark4 = 19 });
    markList.Add(new mark() { ID = 8, Name = "Mandy", Mark1 = 90, Mark2 = 45,
    Mark3 = 54, Mark4 = 55 });
    markList.Add(new mark() { ID = 9, Name = "Alan", Mark1 = 50, Mark2 = 28,
    Mark3 = 25, Mark4 = 45 });
    return markList;
}
```

The following screenshot illustrates how a Chart Series is associated to the Chart by using IList-based instances.



Data-Binding in WPF Chart (Classic) for Child Level Properties

XML Data Source

Chart provides built-in support to bind XML data created through XmlDataProvider instances. The following code example illustrates this.

XML

```
<XmlDataProvider x:Key="myXmlData">
  <x:XData>
    <Travel xmlns="">
      <TravelDetails DayCount="1" Kms="30" Day="Mon"/>
      <TravelDetails DayCount="2" Kms="45" Day="Tue"/>
      <TravelDetails DayCount="3" Kms="50" Day="Wed"/>
      <TravelDetails DayCount="4" Kms="60" Day="Thu"/>
      <TravelDetails DayCount="5" Kms="55" Day="Fri"/>
      <TravelDetails DayCount="6" Kms="75" Day="Sat"/>
      <TravelDetails DayCount="7" Kms="55" Day="Sun"/>
      <TravelDetails DayCount="8" Kms="40" Day="Mon"/>
      <TravelDetails DayCount="9" Kms="30" Day="Tue"/>
    </Travel>
  </x:XData>
</XmlDataProvider>

<syncfusion:ChartSeries DataSource="{Binding Source={StaticResource
myXmlData}, XPath=Travel/TravelDetails}" BindingPathX="DayCount"
BindingPathsY="Kms" IsIndexed="False" Name="series1" Label="Series1"
Type="FastStackingColumn" Interior="{StaticResource SeriesAInterior}" />
```

C#

```
chart1.Areas[0].Series[0].DataSource = xmlDataProvider1;
chart1.Areas[0].Series[0].BindingPathX = "Month";
chart1.Areas[0].Series[0].BindingPathsY = new string[] { "Sales" };
```




ObservableCollection Data Source

Essential Chart provides support to bind data to an ObservableCollection or INotifyCollectionChanged collection. Also, the chart automatically gets updated when any changes are made to the data source.

The following code illustrates how to bind an ObservableCollection as data source to Chart.

XML

```
<Window.Resources>
<local:Sports x:Key="sportinterest"/>
</Window.Resources>
<sfchart:ChartSeries DataSource="{StaticResource sportinterest}"
Type="Column" BindingPathX="SportName" BindingPathsY="Interest"/>
```

C#

```
// Namespace to be included for INotifyPropertyChanged interface.
using System.ComponentModel;
// Namespace to be included for ObservableCollection.
using System.Collections.ObjectModel;
public class Sport : INotifyPropertyChanged
{
    private int sportid;
    private string sportname;
    private double interest;
    public event PropertyChangedEventHandler PropertyChanged;
    public Sport()
    {
    }
    public Sport(int sportid, string sportname, double interests)
    {
        this.sportid = sportid;
        this.sportname = sportname;
        this.interest = interests;
    }
    public override string ToString()
    {
        return sportname.ToString();
    }
    public int SportID
```

```

{
    get { return sportid; }
    set
    {
        sportid = value;
        OnPropertyChanged("SportID");
    }
}

public string SportName
{
    get { return sportname; }
    set
    {
        sportname = value;
        OnPropertyChanged("SportName");
    }
}

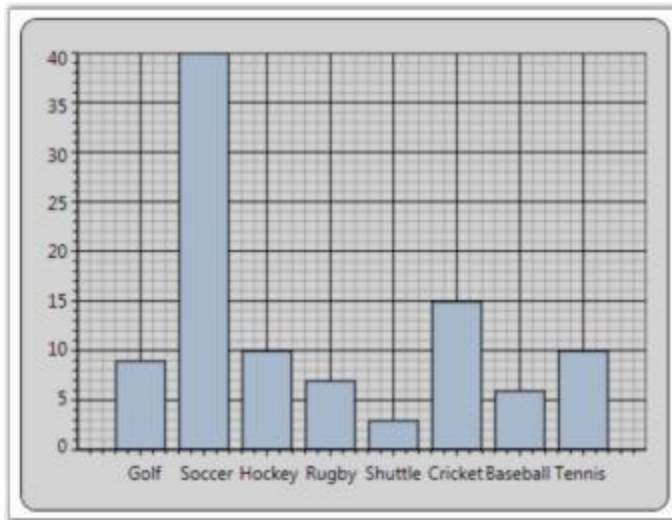
public double Interest
{
    get { return interest; }
    set
    {
        interest = value;
        OnPropertyChanged("Interest");
    }
}

protected void OnPropertyChanged(string info)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (handler != null)
    {
        handler(this, new PropertyChangedEventArgs(info));
    }
}

public class Sports : ObservableCollection<Sport>
{
    public Sports() : base()
    {
        Add(new Sport(101, "Golf", 9));
        Add(new Sport(102, "Soccer", 40));
        Add(new Sport(103, "Hockey", 10));
        Add(new Sport(104, "Rugby", 7));
        Add(new Sport(105, "Shuttle", 3));
        Add(new Sport(106, "Cricket", 15));
        Add(new Sport(107, "Baseball", 6));
        Add(new Sport(108, "Tennis", 10));
    }
}

```

The following screenshot illustrates how a Chart Series is associated to the Chart by using ObservableCollection data source.



CollectionViewSource Data Source

You can bind `CollectionViewSource` as a Chart Series Data Source to Chart. Chart listens to the changes in the source and gets updated automatically. The following code illustrates how to bind `CollectionViewSource` as data source to Chart.

XML

```
<Window.Resources>
<local:SalesinLocation x:Key="saleslocation"/>
<CollectionViewSource Source="{StaticResource saleslocation}" x:Key="cvs" >
<CollectionViewSource.SortDescriptions>
<scm:SortDescription PropertyName="LocationID" />
</CollectionViewSource.SortDescriptions>
</CollectionViewSource>
</Window.Resources>
<sfchart:Chart>
<sfchart:ChartArea>
<sfchart:ChartSeries Name="series1" Label="Sales" Type="Column"
DataSource="{Binding Source={StaticResource cvs}}"
BindingPathX="LocationName" BindingPathsY="Sales"/>
</sfchart:ChartArea>
</sfchart:Chart>
```

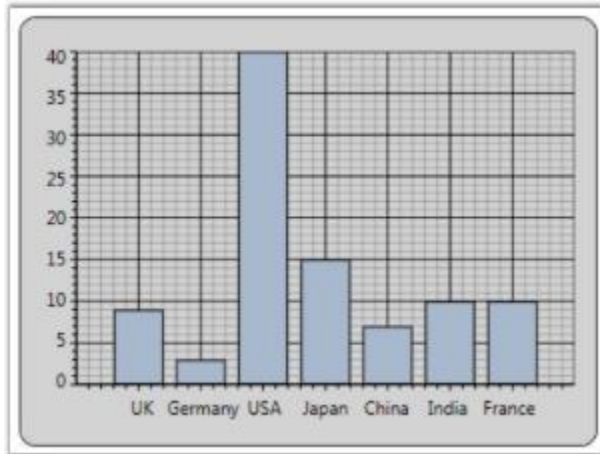
C#

```
// Namespace to be included for INotifyPropertyChanged interface.
using System.ComponentModel;
// Namespace to be included for ObservableCollection.
using System.Collections.ObjectModel;
public class Production : INotifyPropertyChanged
{
    private double prodid;
    private string locationname;
    private double sales;
    public event PropertyChangedEventHandler PropertyChanged;
    public Production()
    {
    }
}
```

```
public Production(double prodid, string locationname, double sales)
{
    this.prodid = prodid;
    this.locationname = locationname;
    this.sales = sales;
}
public override string ToString()
{
    return locationname.ToString();
}
public double LocationID
{
    get { return prodid; }
    set
    {
        prodid = value;
        OnPropertyChanged("ProdId");
    }
}
public string LocationName
{
    get { return locationname; }
    set
    {
        locationname = value;
        OnPropertyChanged("LocationName");
    }
}
public double Sales
{
    get { return sales; }
    set
    {
        sales = value;
        OnPropertyChanged("Sales");
    }
}
protected void OnPropertyChanged(string info)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (handler != null)
    {
        handler(this, new PropertyChangedEventArgs(info));
    }
}
class SalesinLocation : ObservableCollection<Production>
{
    public SalesinLocation() : base()
    {
        Random rand = new Random(DateTime.Now.Millisecond);
        Add(new Production(101, "UK", 9));
        Add(new Production(102, "Germany", 3));
        Add(new Production(103, "USA", 40));
        Add(new Production(104, "Japan", 15));
        Add(new Production(105, "China", 7));
        Add(new Production(106, "India", 10));
    }
}
```

```
Add(new Production(107, "France", 10));
}
}
```

The following screenshot illustrates how a Chart Series is associated to the Chart by using CollectionViewSource data source.



LINQ Data Source

Chart lets you to directly bind LINQ results as the Data Source for a Chart Series. The following code illustrates how to bind LINQ results as the data source for the Chart Series.

C#

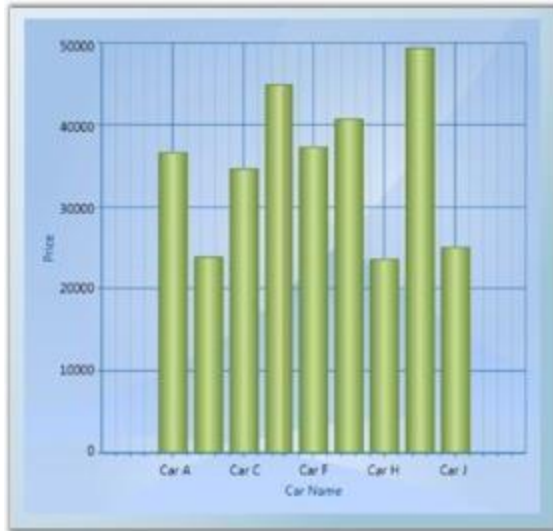
```
// Namespace to be included for XDocument.
using System.Xml.Linq;
public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();
        // Create Data to associate with the series.
        CreateDataToSeries();
        // Add Data Source with the series. Queries Price value more than 30,000.
        this.SetDataSource(carlist.Where(s => s.Price > 30000).ToList());
    }
    // Add Data Source with the series.
    private void SetDataSource(IList source)
    {
        Chart1.Areas[0].Series.Clear();
        ChartSeries series = new ChartSeries();
        // Data Source for series 1.
        series.DataSource = source;
        // Binding X with Name.
        series.BindingPathX = "Name";
        // Binding Y with Price.
        series.BindingPathsY = new string[] { "Price" };
        // Add series to Chart.
        Chart1.Areas[0].Series.Add(series);
    }
    // Create Data to associate with the series.
}
```

```

IList<Car> carlist;
private IList<Car> CreateDataToSeries()
{
    carlist = new List<Car>();
    carlist.Add(new Car("Car A", 36700, 200, 28));
    carlist.Add(new Car("Car B", 23970, 170, 23));
    carlist.Add(new Car("Car C", 34675, 160, 22));
    carlist.Add(new Car("Car D", 44950, 180, 36));
    carlist.Add(new Car("Car E", 74950, 150, 18));
    carlist.Add(new Car("Car F", 37300, 190, 25));
    carlist.Add(new Car("Car G", 40765, 200, 26));
    carlist.Add(new Car("Car H", 23799, 150, 22));
    carlist.Add(new Car("Car I", 49400, 160, 29));
    carlist.Add(new Car("Car J", 25149, 200, 22));
    return carlist;
}
switch (queryby)
{
    case "Price":
        this.SetDataSource(carlist.Where(s => s.Price <
double.Parse(value)).ToList());
        break;
    case "MaximumSpeed":
        this.SetDataSource(carlist.Where(s => s.MaximumSpeed <
double.Parse(value)).ToList());
        break;
    case "Mileage":
        this.SetDataSource(carlist.Where(s => s.Mileage <
double.Parse(value)).ToList());
        break;
}
}
// Add class Car.
class Car
{
    public string Name { get; set; }
    public double Price { get; set; }
    public double MaximumSpeed { get; set; }
    public double Mileage { get; set; }
    public Car(string name, double price, double maxspeed, double mileage)
    {
        this.Name = name;
        this.Price = price;
        this.MaximumSpeed = maxspeed;
        this.Mileage = mileage;
    }
}

```

The following screenshot illustrates how a Chart Series is associated to the Chart by using LINQ results.



Data-Binding in WPF Chart (Classic) for Child Level Properties

The child level properties can be bound to the chart series using `BindingPathX` and `BindingPathsY` values. `BindingPathX` and `BindingPathsY` are the properties that belong to chart series which holds the x-axis and y-axis binding path values.

The following code example elaborates on the data binding for the child level properties-End and Start, for the Product class.

C#

```
public class Product
{
    public int ID { get; set; }
    public ProductInfo Info { get; set; }
    public TestClass Start
    {
        get
        {
            return Info.Start;
        }
    }
    public int End
    {
        get
        {
            return Info.End;
        }
    }
}

public class ProductInfo
{
    public int Start { get; set; }
    public int End { get; set; }
}

//series1 is the chart series
this.series1.BindingPathX = "ID";
this.series1.BindingPathsY = new string[] { "Info.Start", "Info.End" };
```

XML

```
<syncfusion:ChartSeries Name="series1" Type="Gantt"
Interior="{StaticResource SeriesAInterior}" Label="Allotted Days"
StrokeThickness="0.4" IsIndexed="False" BindingPathX="ID"
BindingPathY="Info.Start,Info.End" />
```

Data Binding Support

The Data Binding feature has generated common DataModel based on the user bind underlying objects. You can use this DataModel to perform sorting and grouping operations. This can be done by using the DataModel.View property of ChartSeries control. You can also bind ObservableCollection, List, IList, CollectionViewSource, DataTable, LINQresults, ITypedList, IBindingList, BindingList and XML data into the Chart.

The View is internally generated automatically based on the object collection bind on the DataSource property of ChartSeries. You can also bind the external collection view objects into the Chart. The external collection view, SortDescription and GroupDescription are internally listened by Chart DataModel and the internal View.

This Data Binding engine improves the performance of chart with effectively handling the chart data. The following are some of the useful tips to improve the chart performance.

- Instead of using built-in IsSortData and IsIndexed feature, you can sort the underlying business object in application level and then initialize it to DataSource property of ChartSeries control.
- Use BeginInit and EndInit method when you add, delete and replace multiple objects at a same time in ChartSeries control. It helps to improve the performance.
- Disable Auto Range and Interval feature. You can manually initialize the Range and Interval to ChartAxis to get better performance.

Use Case Scenarios

In Stock market, data gets updated in a timely manner. Initial data can be bind to chart and later you can update the underlying object. Based on that new stock information added in your underlying object, the DataModel and the Chart Visual can refresh automatically.

Adding Data Binding to an Application

You can bind chart data into the DataSource property of ChartSeries control. The View is generated internally for the user bound data and the chart can render in visual. The BindingPathX and BindingPathsY properties are used to initialize the property name of the binded data. This property values is used as coordinate values for X and Y direction in the ChartAxis control.

XML

```
<syncfusion:ChartSeries DataSource="{StaticResource data}"
BindingPathX="ProductID" BindingPathsY="Price" />
```

C#

```
ProductDetails data = this.Resources["data"] as ProductDetails;
//Initialize the business object into the ChartSeries.
series.DataSource = data;
series.BindingPathX = "ProductID";
series.BindingPathsY = new string[] { "Price" };
```



```
//To add the SortDescription in DataModel View.
SortDescription desc = new SortDescription("Price",
ListSortDirection.Descending);
chart.Areas[0].Series[0].DataModel.View.SortDescriptions.Add(desc);
```

Property	Description	Type	Data Type	Reference links
DataModel	This DataModel has internally initialized based on the user bind data on DataSource property of ChartSeries control. It generates View for user bind DataSource.	CLR.	ChartDataModel	NA

ChartDataModel Table

Property	Description	Type	Data Type	Reference links
View	This property helps to add SortDescription and GroupDescription features. This property initializes the common view for all Collection objects initially.	CLR.	ICollectionViewAdv	NA

ICollectionDataPoint interface

ICollectionDataPoint interface is a point that contains the information about the data of the chart series, X and Y coordinates of the point, and the segment. The ICollectionDataPoint interface objects can be initialized internally based on the ChartPoint and the object input by the user.

Properties

The following table lists the properties of the ICollectionDataPoint interface.

Property name	Description	Data Type
X	Represents the X value of the DataPoint.	Double
Y	Represents the Y value of the DataPoint.	Double
ParentSegment	Represents the parent segment of the DataPoint.	ChartSegment
Visible	Sets the visibility of the DataPoint.	Bool
Item	Includes X,Y and custom values to represent the point.	Object
Label	Value represents the ContentPath.	String
IsEmpty	Indicates whether the point is empty.	Bool
Values	Value array should be used to represent the range of Y values that correspond to one X value.	Double
EmptyPoint	Value indicates the Empty Point.	Bool

Following are the code that describes the IChartDataPoint interfaces.

C#

```
class CustomPoint : IChartDataPoint
{
    public CustomPoint(double X, double Y)
    {
        this.X = X;
        this.Y = Y;
        this.Values = new double[] { Y };
    }
    public double X { get; set; }
    public double Y { get; set; }
    public double[] Values { get; set; }
    public bool IsEmpty { get; set; }
    public bool EmptyPoint { get; set; }
    public string Label { get; set; }
    public bool Visible { get; set; }
    public object StringItem { get; set; }
    public ChartSegment ParentSegment { get; set; }
    public object Item { get; set; }
    public object Clone()
    {
        CustomPoint customPoint = new CustomPoint(this.X, this.Y);
        //...
        // Filling proper fields.
        //...
        return customPoint;
    }
    public void Dispose()
    {
    }
}
// Custom collection strongly typed as CustomPoint that implements
// IChartData.
class CustomChartPointsCollection : ObservableCollection<CustomPoint>,
IChartData
{
    public new IChartDataPoint this[int index]
    {
        get { return base[index]; }
    }
    public ChartValueType XValueType { get; set; }
    #region IDisposable Members
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    public void Dispose()
    {
    }
    #endregion
}
//Using classes
//...
//Creating a new chart1 with area and series.
//...
```

```
CustomChartPointsCollection customCollection = new  
CustomChartPointsCollection();  
customCollection.Add(new CustomPoint(1, 3));  
customCollection.Add(new CustomPoint(2, 5));  
customCollection.Add(new CustomPoint(3, 2));  
customCollection.Add(new CustomPoint(4, 8));  
chart1.Areas[0].Series[0].Data = customCollection;
```

Chart-Area in WPF Chart (Classic)

Adding Chart Area

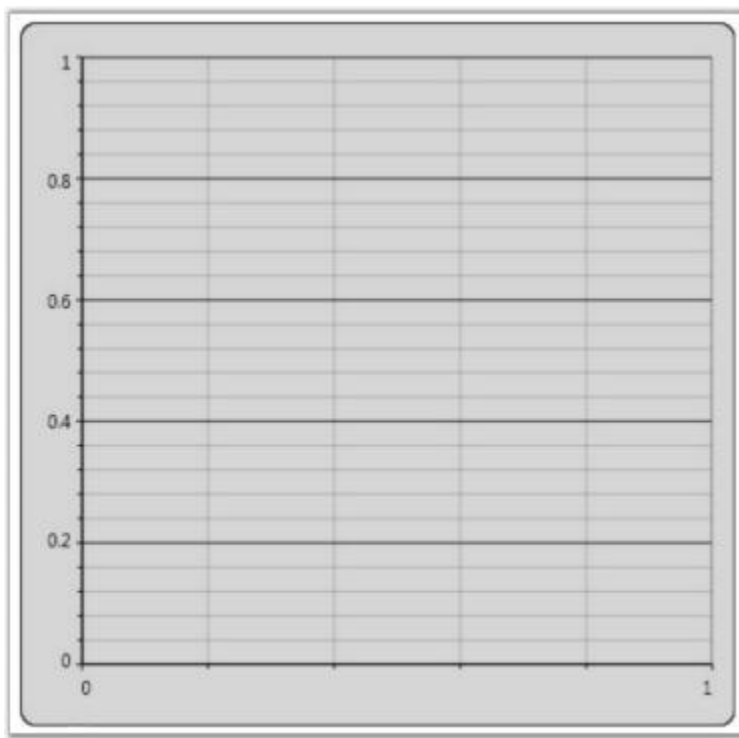
Once you add the Chart control, the first thing to do is to add a Chart Area and and Chart Series. The following code example illustrates how to do this.

XML

```
<sfchart:Chart>  
<sfchart:ChartArea>  
<sfchart:ChartSeries/>  
</sfchart:ChartArea>  
</sfchart:Chart>
```

C#

```
ChartArea area = new ChartArea();  
Chart1.Areas.Add(area);  
ChartSeries series = new ChartSeries();  
area.Series.Add(series);
```



Multiple Areas

Essential Chart provides support to add multiple Chart Areas to a Chart to visualize related data side by side. The following code example illustrates this.

XML

```
<sfchart:Chart>
  <sfchart:ChartArea Background="LightGray" GridBackground="White">
    <sfchart:ChartSeries/>
  </sfchart:ChartArea>
  <sfchart:ChartArea Background="WhiteSmoke" GridBackground="White" >
    <sfchart:ChartSeries/>
  </sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
ChartArea area = new ChartArea();
area.Background = Brushes.LightGray;
area.GridBackground = Brushes.White;
Chart1.Areas.Add(area);
ChartArea area1 = new ChartArea();
area1.Background = Brushes.WhiteSmoke;
area1.GridBackground = Brushes.White;
Chart1.Areas.Add(area1);
```

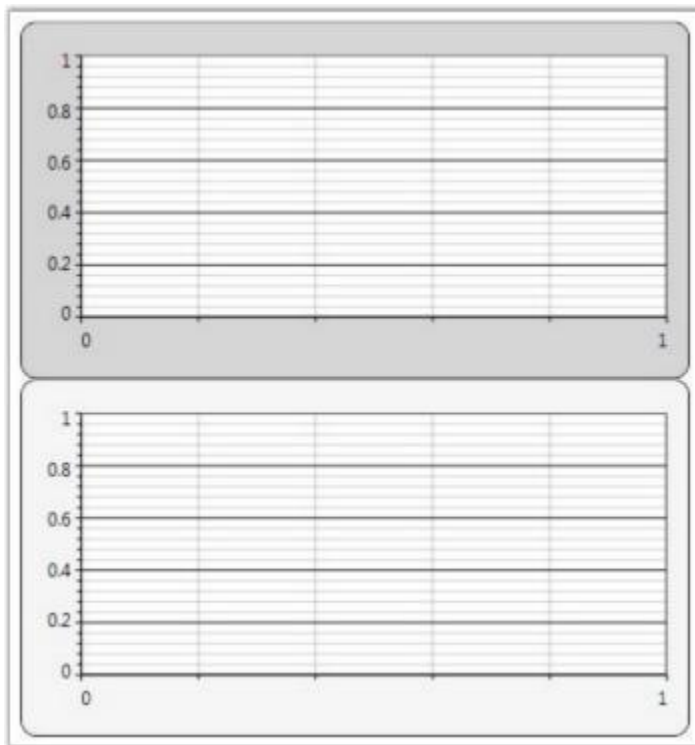


Chart-Area in WPF Chart (Classic) Header

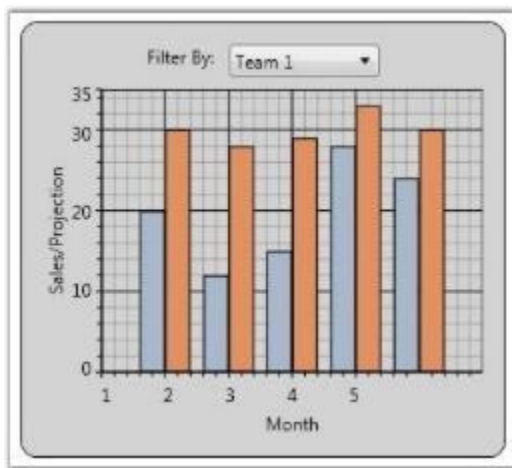
Chart enables you to add headers to the Chart Area object. Any element can be added as a Chart Area header by using the Header property of the ChartArea class.

XML

```

<sfchart:Chart>
<sfchart:ChartArea Background="LightGray" GridBackground="White">
<sfchart:ChartArea.Header>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Filter By:"></TextBlock>
<ComboBox Width="100" Margin="10, 0, 0, 0">
<ComboBoxItem>Team 1</ComboBoxItem>
<ComboBoxItem>Team 2</ComboBoxItem>
<ComboBoxItem>Team 3</ComboBoxItem>
</ComboBox>
</StackPanel>
</sfchart:ChartArea.Header>
<!--Chart Series initialization code is hidden for brevity.-->
<sfchart:ChartSeries/>
</sfchart:ChartArea>
</sfchart:Chart>

```

**Chart Header**

Essential Chart for WPF enables users to set the title for a chart.

Name of Property	Description	Type of Property	Value It Accepts	Property Syntax	Sub Properties
Header	Sets the title of the chart.	Dependency Property	Object/ 'Chart Header'	<syncfusion:Chart Name="chart1" Header="Chart Header">	Sub Property Name : HeaderAlignment Type: HorizontalAlignment / HorizontalAlignment.Left

Setting the Title for a Chart

Set the title for a chart by using the following code.

XML

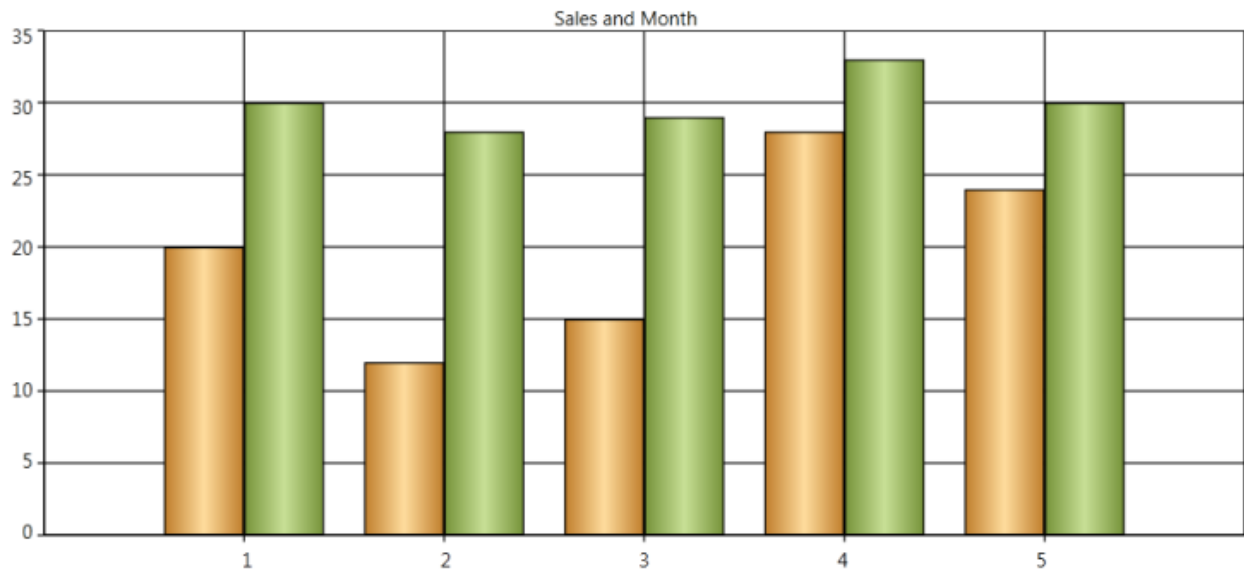
```

<sfchart:Chart Name="Chart1" Header="Sales and Month">
</sfchart:Chart>

```

C#

```
chart1.Header = " Sales and Month "
```

*Customizing Chart Title*

Users can customize the chart header using a text block, text box, rectangle, or border control.

Customize the chart header by using the following code.

XML

```
<sfchart:Chart Name="Chart1" >
  <sfchart:Chart.Header>
    <TextBlock Text="Sales and Month" FontSize="16" Foreground="Blue"
      FontStyle="Italic" FontWeight="Bold" Margin="-5" />
  </sfchart:Chart.Header>
</sfchart:Chart>
```



Chart-Area in WPF Chart (Classic) Context Menu

WPF Chart has a built-in context menu which can be enabled by setting the `ChartArea.IsContextMenuEnabled` property to `true`. This context menu lets you change the Chart Type of a series and Color Palettes, and enable Zooming.

Default Context Menu

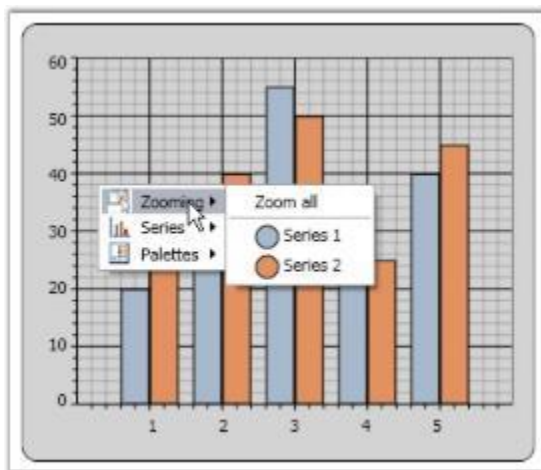
The following code example illustrates how to display the built-in context menu of Chart Area.

XML

```
<syncfusion:Chart >  
<syncfusion:ChartArea IsContextMenuEnabled="True" />  
</syncfusion:Chart>
```

C#

```
area.IsContextMenuEnabled = true;
```

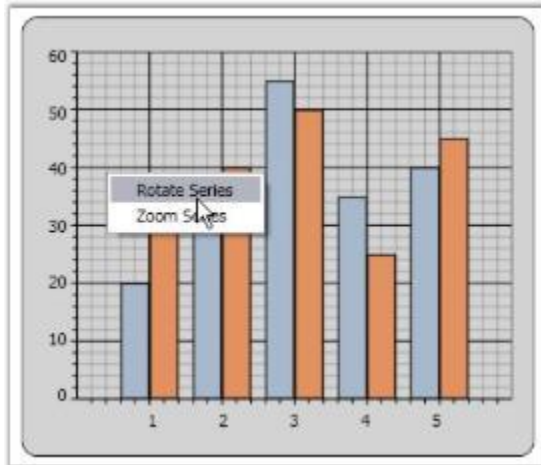


Custom Context Menu

You can also customize the context menu items to display any desired text. The following code example illustrates this.

C#

```
ContextMenu contextMenu = new ContextMenu();  
contextMenu.Items.Add("Rotate Series");  
contextMenu.Items.Add("Zoom Series");  
Chart1.Areas[0].ContextMenu = contextMenu;
```



Background

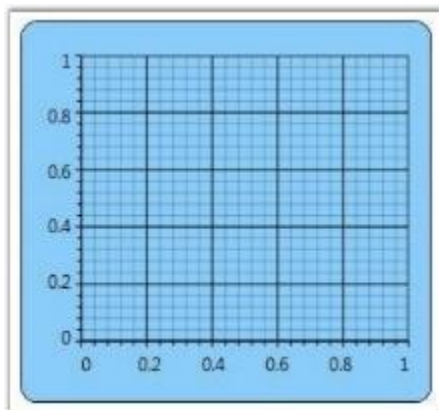
You can customize the background of the Chart Area to suit the application by specifying custom brushes for the Background property. The following code example illustrates this.

XML

```
<sfchart:Chart>
  <sfchart:ChartArea Background="LightSkyBlue">
    <sfchart:ChartSeries/>
  </sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
ChartArea area = new ChartArea();
area.Background = Brushes.LightSkyBlue;
Chart1.Areas.Add(area);
```



Grid Background

The GridBackground property of the Chart Area is used to change the color of the inner area of the Chart Area object. The following code example illustrates how to set this property.

XML

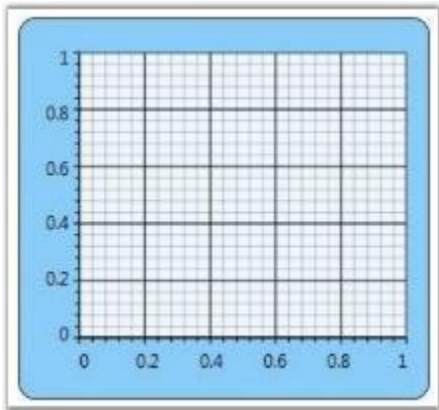
```
<sfchart:Chart>
```



```
<sfchart:ChartArea Background="LightSkyBlue" GridBackground="AliceBlue">
  <sfchart:ChartSeries/>
</sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
ChartArea area = new ChartArea();
area.Background = Brushes.LightSkyBlue;
area.GridBackground = Brushes.AliceBlue;
Chart1.Areas.Add(area);
```



Essential Chart alternatively provides options to apply two backgrounds to a single Chart Area. The following properties of the ChartArea class are used for this purpose.

Property	Description
AlternatingBackground	specifies the alternating background
AlternatingFillMode	enum property that specifies whether Even or Odd interval should be filled with the alternating background color
AlternatingFillDirection	orientation property that specifies whether the alternating background should be applied horizontally or vertically

The following code example illustrates how to set the preceding properties.

XML

```
<sfchart:Chart>
  <sfchart:ChartArea Background="LightSkyBlue" GridBackground="AliceBlue"
    AlternatingGridBackground="LightPink" AlternatingFillMode="Even"
    AlternatingFillDirection="Horizontal">
    <sfchart:ChartSeries/>
  </sfchart:ChartArea>
</sfchart:Chart>
```

C#

```

ChartArea area = new ChartArea();
area.Background = Brushes.LightSkyBlue;
area.GridBackground = Brushes.AliceBlue;
area.AlternatingGridBackground = Brushes.LightPink;
area.AlternatingFillMode = AlternatingFillMode.Even;
area.AlternatingFillDirection = Orientation.Horizontal;
Chart1.Areas.Add(area);

```

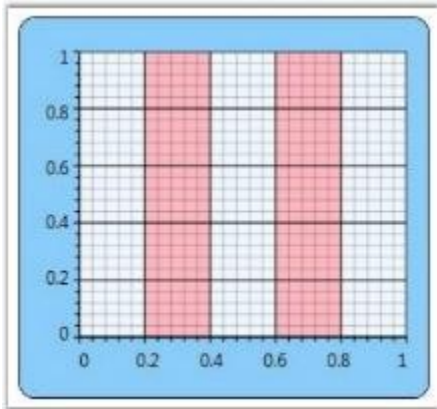


Chart Watermark Support

Essential Chart supports watermarks which can be used to show text or images behind the chart area.

- Watermarks can be rotated by specifying the rotation angle for the text or image watermark.
- The opacity of a watermark can be set as a percentage.
- Watermarks can be aligned horizontally or vertically with respect to the chart area.

Use Case Scenarios

- Watermarking is used in copyright protection systems. They are intended to prevent or detect illegal copying of charts by unauthorized users.
- Also, chart watermarks provide indisputable proof of the image origin or its author.
- Owner identification like a username can be added to the chart so it is easy to find the owner.
- A unique and secure digital signature can be added to each chart. This can be used to verify the origin and generation details of a chart.

Properties

Property	Description	Type	Data Type
Watermark	Sets the text or image watermark behind chart area.	Dependency	Brush

Sample Link

To access the chart watermark demo:

1. Open the Syncfusion Dashboard.
2. Select User Interface.
3. Click the WPF drop-down list and select Explore Samples.

4. Browse to the path Chart.WPF\Samples\3.5\WindowsSamples\Chart Area\Chart Watermark Demo

Adding Watermark to an Application

Image Watermark

XML

```
<syncfusion:ChartArea.Watermark>                <VisualBrush Stretch="None"
Opacity="0.5" AlignmentX="Center"
AlignmentY="Top">                                <VisualBrush.Visual>                <Image
Name="img" Source="/WatermarkDemo;component/SyncLogo1.png">
<Image.LayoutTransform>                            <RotateTransform Angle="-
45"/>                                </Image.LayoutTransform>                </Image>
</VisualBrush.Visual>                            </VisualBrush>
</syncfusion:ChartArea.Watermark>
```

C#

```
Image image = new Image() { Source = new
BitmapImage(new Uri(@"D:\WatermarkDemo\SyncLogo1.png")),
LayoutTransform = new RotateTransform() { Angle = -45 } };
this.chartArea.Watermark = new VisualBrush() {
Visual = image, Stretch = Stretch.None,
AlignmentX = AlignmentX.Center, AlignmentY = AlignmentY.Top,
Opacity = 0.5 };;
```



Text Watermark

XML

```
<syncfusion:ChartArea.Watermark>                                <VisualBrush Stretch="None"
Opacity="0.8" AlignmentX="Right"
AlignmentY="Bottom">                                <VisualBrush.Visual>
<TextBlock Name="txt" Text="Syncfusion" FontSize="64" Foreground="Red"
FontFamily="Microsoft Sans Serif">
<TextBlock.LayoutTransform>                                <RotateTransform
Angle="325"/>                                </TextBlock.LayoutTransform>
</TextBlock>                                </VisualBrush.Visual>                                </VisualBrush>
</syncfusion:ChartArea.Watermark>
```

C#

```
TextBlock text = new TextBlock() { Text =
"Syncfusion", FontSize = 64, Foreground =
Brushes.Red, FontFamily = new FontFamily("Microsoft Sans
Serif"), LayoutTransform = new RotateTransform() { Angle =
325 } }; this.chartArea.Watermark = new VisualBrush()
{ Visual = text, Stretch = Stretch.None,
AlignmentX = AlignmentX.Right, AlignmentY =
AlignmentY.Bottom, Opacity = 0.5 };
```

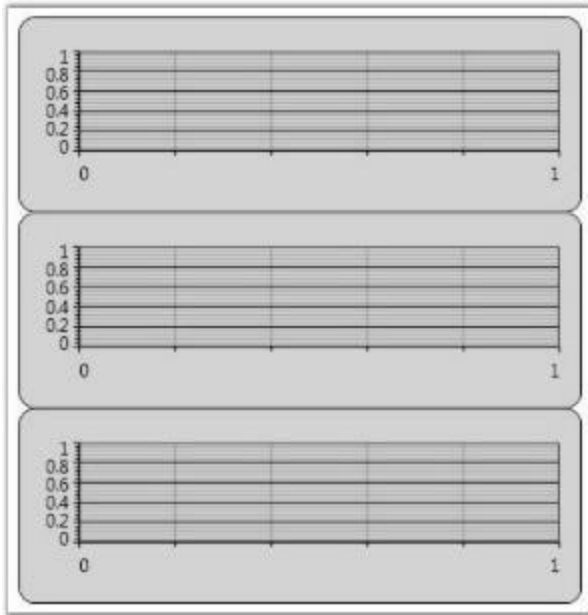


Chart-Area in WPF Chart (Classic) Layout Customization

Upon adding multiple Chart Areas to Chart, you may want to customize the layout in which these multiple Chart Areas are rendered. You can do so by specifying a custom container for these Chart Areas through the `AreasPanel` property of the Chart control. Any container such as Grid, Stack Panel, Dock Panel, Canvas or Wrap Panel can be used.

Built-in Chart Grid

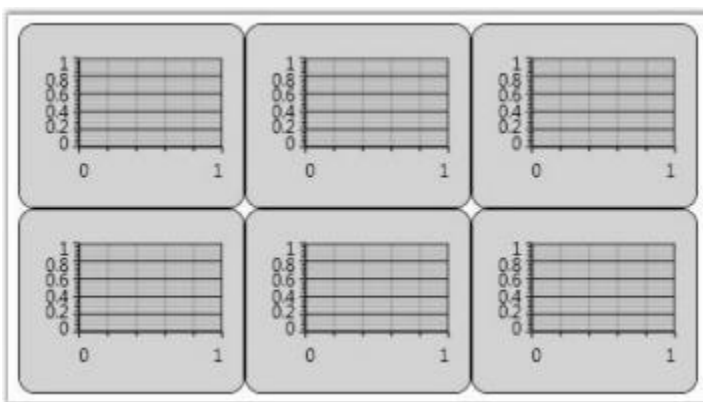
ChartGrid is the container that is used by Chart, by default, to host the Chart Areas. The following screenshot illustrates the default settings of ChartGrid in which all the Chart Areas are arranged one after the other.



However, the default settings of the `ChartGrid` can be customized to display the Chart Areas side by side. The following code example illustrates how this can be done by using the `Orientation` and `AutoRowCount` properties.

XML

```
<sfchart:Chart>
  <sfchart:Chart.AreasPanel>
    <ItemsPanelTemplate>
      <sfchart:ChartGrid Orientation="Horizontal"
        AutoRowCount="2"></sfchart:ChartGrid>
    </ItemsPanelTemplate>
  </sfchart:Chart.AreasPanel>
</sfchart:Chart>
```



The `ChartGrid` also provides options to control the aspect ratio (number of rows / number of columns) of the layout grid. Take a look at the `ChartGrid` class reference for more information.

Other Panels

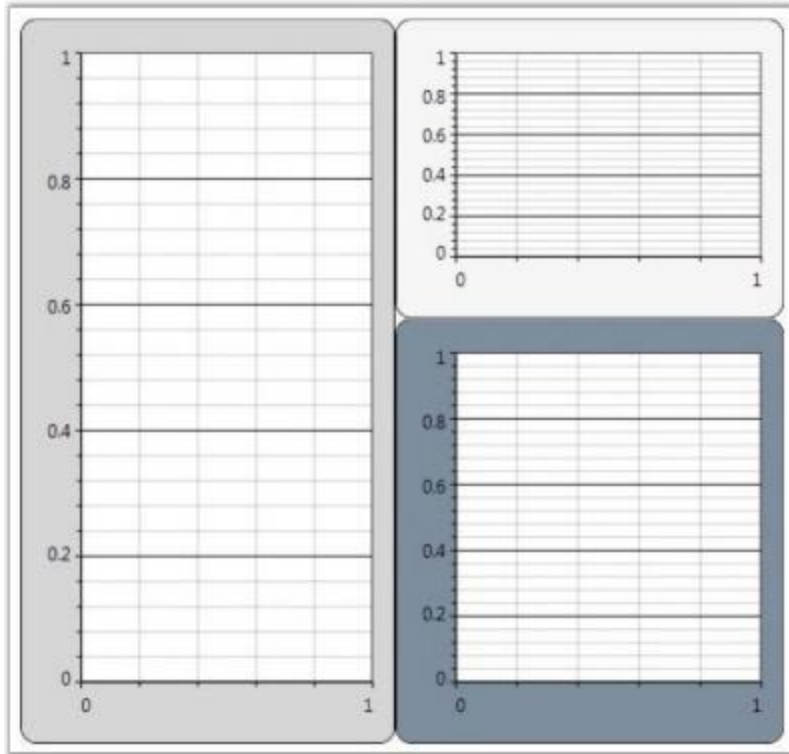
Also you can plug-in any kind of `Panel` as the container for the Chart Area. The following code example illustrates how to use a `DockPanel` as the `Areas Panel` of the Chart.

XML

```
<sfchart:Chart>
<sfchart:Chart.AreasPanel>
<ItemsPanelTemplate>
<DockPanel/>
</ItemsPanelTemplate>
</sfchart:Chart.AreasPanel>
<sfchart:ChartArea Background="LightGray" GridBackground="White"
DockPanel.Dock="Top" Height="200">
<sfchart:ChartSeries/>
</sfchart:ChartArea>
<sfchart:ChartArea Background="WhiteSmoke" GridBackground="White"
DockPanel.Dock="Top" Height="200">
<sfchart:ChartSeries/>
</sfchart:ChartArea>
<sfchart:ChartArea Background="LightSlateGray" GridBackground="White" >
<sfchart:ChartSeries/>
</sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
ChartArea area = new ChartArea();
area.Background = Brushes.LightGray;
area.GridBackground = Brushes.White;
Chart1.Areas.Add(area);
ChartArea area1 = new ChartArea();
area1.Background = Brushes.WhiteSmoke;
area1.GridBackground = Brushes.White;
Chart1.Areas.Add(area1);
ChartArea area2 = new ChartArea();
area2.Background = Brushes.LightSlateGray;
area2.GridBackground = Brushes.White;
Chart1.Areas.Add(area2);
```



A sample which demonstrates how to use different containers as the Areas Panel is available in the following sample installation path.

..My Documents\Syncfusion\EssentialStudio\Version
Number\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Area\Custom Panel Demo

Synchronization of Chart Axis

Essential Chart WPF lets you align multiple chart areas by using a single primary axis. You can synchronize the chart axis by using the following code example.

XML

```
<sfChart:SyncChartAreas>
  <sfChart:SyncChartAreas.PrimaryAxis>
    <sfChart:ChartAxis Name="year" LabelFontSize="8" ValueType="Double"
      RangePadding="None"
      RangeCalculationMode="ConsistentAcrossChartTypes"
      sfChart:ChartArea.ShowGridLines="True" Header="Year">
    </sfChart:ChartAxis>
  </sfChart:SyncChartAreas.PrimaryAxis>
  <sfChart:SyncChartAreas.Areas>
    <sfChart:ChartArea Background="Transparent">
      <sfChart:ChartArea.SecondaryAxis>
        <sfChart:ChartAxis Name="Literacy" IntersectAction="Hide" LabelFontSize="8"
          ValueType="Double" Header="Literacy Growth"
          RangePadding="None" sfChart:ChartArea.ShowGridLines="False" />
      </sfChart:ChartArea.SecondaryAxis>
      <sfChart:ChartSeries Name="series22" Label="Increase In Population"
        BindingPathX="Year" BindingPathsY="IncreaseInPopulation"
        Type="Area" StrokeThickness="2" Interior="Green" >
    </sfChart:ChartSeries.YAxis>
  </sfChart:SyncChartAreas.Areas>
</sfChart:SyncChartAreas>
```



```

<sfChart:ChartAxis sfChart:ChartArea.ShowGridLines="False" Header="Increase
In Population" OpposedPosition="True"
Orientation="Vertical" LabelFontSize="8" ValueType="Double"
RangePadding="None" />
</sfChart:ChartSeries.YAxis>
</sfChart:ChartSeries>
<sfChart:ChartSeries Name="series21" Label="Literacy Growth"
BindingPathX="Year" BindingPathsY="LiteracyGrowth" Type="Spline"
StrokeThickness="2" Interior="DarkBlue"/>
</sfChart:ChartArea>
</sfChart:SyncChartAreas.Areas>
</sfChart:SyncChartAreas>

```

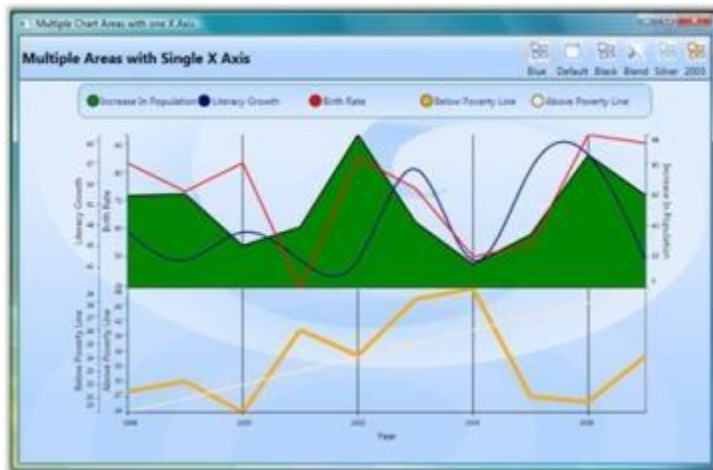
C#

```

SyncChartAreas syncAreas = new SyncChartAreas();
ChartAxis primaryAxis = new ChartAxis();
syncAreas.PrimaryAxis = primaryAxis;
ChartArea area1 = new ChartArea();
syncAreas.Areas.Add(area1);
ChartArea area2 = new ChartArea();
syncAreas.Areas.Add(area2);

```

Run the code. The following output is displayed.

*Additional Zooming Functionality for SyncChartAreas*

Essential chart now supports the concepts of Zoomin, Zoomout and panning in the SyncChartArea.

Adding Additional Zooming Functionality

Add Additional Zooming Functionality, by using the following code.

XML

```

<sfChart:SyncChartAreas IsContextMenuEnabled="True">

```

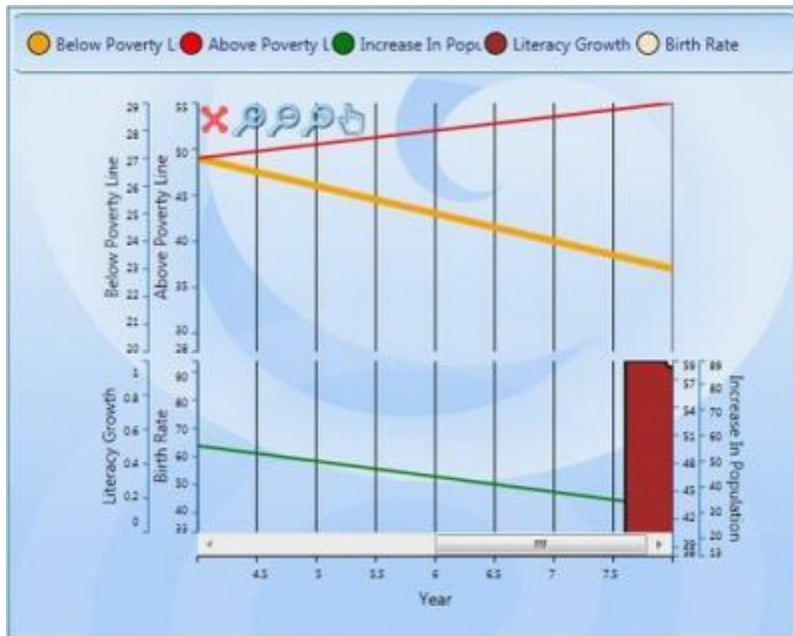
C#

```

syncArea.IsContextMenuEnabled = true;

```

When the code runs, the following output displays.



IDictionary

IDictionary supports for binding the Dictionary list to the chart series BindingPathX and BindingPathsY Values.

[Sample Data Source](#)

IDictionary is useful to bind this kind of data source.

Note: In this sample data, 0, 1, 2, 3 ...6 are the Keys and the CompanyExpenses are the Values.

C#

```
SortedList Expenditure = new SortedList();
expenditure.Add(0, new CompanyExpense() { x = "Production", y = 20d });
expenditure.Add(1, new CompanyExpense() { x = "Facilities", y = 23d });
expenditure.Add(2, new CompanyExpense() { x = "Insurance", y = 12d });
expenditure.Add(3, new CompanyExpense() { x = "Licenses", y = 3d });
expenditure.Add(4, new CompanyExpense() { x = "Labor", y = 28d });
expenditure.Add(5, new CompanyExpense() { x = "Legal", y = 2d });
expenditure.Add(6, new CompanyExpense() { x = "Taxes", y = 10d });
```

[Binding Data Source](#)

Bind the data source, by using the following code.

Note: Set BindingPath to Key to bind the key of the dictionary to BindingPathX or BindingPathsY.

XML

```
<syncfusion:ChartSeries BindingPathX="Key" BindingPathsY="y"
DataSource={StaticResource expenditure}>
```

C#

```
Series.BindingPathX="Key";
```

```
Series.BindingPathsY = "Y";
```

Lazy Loading Support for Chart WPF

Lazy Loading Support reduces the loading time of chart area. This belongs to Chart area class.

Adding Lazy Loading Support

Set EnableLazyLoading property as true to add this feature.

Add Lazy Loading Support, by using the following code.

XML

```
<syncfusion:ChartArea EnableLazyLoading="True" IsContextMenuEnabled="True">
```

C#

```
Area1.EnableLazyLoading = true;
```

Revamping 3D charts in Chart WPF Feature

Charts are implemented in 3D with customization to improve the look and feel of the chart.

Display 3D Chart, by using the following code:

XML

```
//To display the chart type in 3d mode
<syncfusion:ChartArea.Chart3DSettings>
<syncfusion:Chart3D BackWallThickness="0"
ShowBackWall="False" LeftWallBackground="Transparent" LeftWallThickness="0"
BottomWallBackground="Transparent" BottomWallThickness="0"/>
</syncfusion:ChartArea.Chart3DSettings>
```

C#

```
//To set the background
Chart1.Areas[0].Chart3DSettings.BackWallBackground = Brushes.AliceBlue;
// To set the back wall thickness
Chart1.Areas[0].Chart3DSettings.BackWallThickness = 0.5;
//To set the back ground for the Bottom wall
Chart1.Areas[0].Chart3DSettings.BottomWallBackground = Brushes.Black;
// To set the thickness for bottom wall
Chart1.Areas[0].Chart3DSettings.BottomWallThickness = 0.5;
// To set the Camera projection
Chart1.Areas[0].Chart3DSettings.CameraProjection =
CameraProjection.Orthographic;
Light light=new AmbientLight(Color.FromArgb(200, 200, 200, 200));
//To set the chart light
Chart1.Areas[0].Chart3DSettings.ChartLight = light;
// To set the back ground for left wall
Chart1.Areas[0].Chart3DSettings.LeftWallBackground = Brushes.Yellow;
// To set the Thickness for Left wall
Chart1.Areas[0].Chart3DSettings.LeftWallThickness = 0.5;
// TO set the background for right wall
Chart1.Areas[0].Chart3DSettings.RightWallBackground = Brushes.Green;
// To set the thickness for Rightwall
```

```

Chart1.Areas[0].Chart3DSettings.RightWallThickness = 0.5;
Chart1.Areas[0].Chart3DSettings.RotateOnMouseDown = true;
Chart1.Areas[0].Chart3DSettings.ShowBackWall = true;
Chart1.Areas[0].Chart3DSettings.ShowBottomWall = true;
Chart1.Areas[0].Chart3DSettings.ShowPrimaryAxis = true;
Chart1.Areas[0].Chart3DSettings.ShowRightWall = true;
Chart1.Areas[0].Chart3DSettings.ShowSecondaryAxis = true;
Chart1.Areas[0].Chart3DSettings.ShowTopWall = true;
Chart1.Areas[0].Chart3DSettings.TopWallBackground = Brushes.IndianRed;
Chart1.Areas[0].Chart3DSettings.TopWallThickness = 0.5;
Chart1.Areas[0].Chart3DSettings.ViewDefaultRotate = 1.5;
Chart1.Areas[0].Chart3DSettings.ViewDefaultTilt = 2.5;
Chart1.Areas[0].Chart3DSettings.ViewDefaultTurn = 10; Screen

```

When the code runs, the following output displays.



Interactive Cursors

The Interactive cursor hints the X value and Y value of a specific data point, as indicated by horizontal and vertical intersecting lines. These lines can be dragged to specific data.

Property Details

The following table contains the property details of the table.

Name of Property	Description	Type of Property	Value It Accepts
IsInversedLabel	To inverse the label	Dependency Property	Bool
EnableVerticalMove	To set the vertical movement	Dependency Property	Bool
EnableHorizontalMove	To set the horizontal movement	Dependency Property	Bool
OffsetX	To set the Offset X	Dependency Property	Double
OffsetY	To set the Offset Y	Dependency Property	Double

HorizontalLabelTemplate	To set the Template for Horizontal label	Dependency Property	DataTemplate
VerticalLabelTemplate	To set the Template for vertical label	Dependency Property	DataTemplate
IsBindWithSegment	To set or unset the IsBindWithSegment	Dependency Property	Bool
CursorVisibility	To set the visibility for the cursor	Dependency Property	Bool
CursorStrokeThickness	To set the stroke thickness for Cursor	Dependency Property	Double
HorizontalCursorStroke	To set the stroke for the horizontal cursor	Dependency Property	Brushes
VerticalCursorStroke	To set the stroke for the vertical cursor	Dependency Property	Brushes
BindWithMoseMoveOnSegment	To set or unset the bindwith mousemove on segments	Dependency Property	Bool

Adding Interactive Cursors

Add Interactive Cursors, by using the following code.

XML

```
//sets the interactive cursor
<Chart:InteractiveCursor
IsInversedLabel="True" OffsetX="100" OffsetY="100"
CursorStrokeThickness="3" VerticalLabelVisibility="Visible"
HorizontalLabelVisibility="Visible" Cursor="Hand"
IsBindWithSegment="true" HorizontalCursorStroke="Magenta"
VerticalCursorStroke="Magenta">
//sets the vertical template for the Vertical Label
<Chart:InteractiveCursor.VerticalLabelTemplate>
<DataTemplate>
<Grid>
<Label Foreground="GreenYellow" Content="{Binding}"/>
</Grid>
</DataTemplate>
</Chart:InteractiveCursor.VerticalLabelTemplate>
//sets the Horizontal template for the horizontal Label
<Chart:InteractiveCursor.HorizontalLabelTemplate>
<DataTemplate>
<Grid>
<Label Foreground="GreenYellow" Content="{Binding}"/>
</Grid>
</DataTemplate>
</Chart:InteractiveCursor.HorizontalLabelTemplate>
</Chart:InteractiveCursor>
</Chart:SyncChartAreas.InteractiveCursors>
```

C#

```

// sets the cursor visibility
this.chart1.Areas[0].InteractiveCursors[0].CursorVisibility = Visibility.
Visible;
// sets the label visibility
this.chart1.Areas[0].InteractiveCursors[0].LabelVisibility = Visibility.
Visible;
// sets the cursor
this.chart1.Areas[0].InteractiveCursors[0].Cursor = Cursors. Hand;
//Sets the stroke for the Horizontal cursor
this.chart1.Areas[0].InteractiveCursors[0].HorizontalCursorStroke = Brushes.
Magenta;
//Sets the stroke for vertical cursor
this.chart1.Areas[0].InteractiveCursors[0].VerticalCursorStroke = Brushes.
Magenta;
// sets the is bind with segment
this.chart1.Areas[0].InteractiveCursors[0].IsBindWithSegment = true;
// Sets the BindWithMouseMoveSegment
this.chart1.Areas[0].InteractiveCursors[0].BindWithMoseMoveOnSegment = true;
// sets the label as inversed
this.chart1.Areas[0].InteractiveCursors[0].IsInversedLabel = true;
// To set the OffsetX
this.chart1.Areas[0].InteractiveCursors[0].OffsetX = 100;
//To set the OffsetY
this.chart1.Areas[0].InteractiveCursors[0].OffsetY = 100;
//To set the stroke thickness for the cursor
this.chart1.Areas[0].InteractiveCursors[0].CursorStrokethickness = 3;

```

[Splitter for SyncChartArea should be implemented](#)

Essential Chart WPF is now enhanced with Splitter for SyncChartArea. This is useful to differentiate the implementation of more than one Chart Area.

[Property Details](#)

The following table contains the property details.

Name of Property	Description	Type of Property	Value It Accepts
SplitterVisibility	Sets the visibility for the splitter.	Dependency Property	Enum of the type SplitterVisibility
SplitterWidth	Sets the width for the Splitter.	Dependency Property	Double
SplitterStroke	Sets the stroke for the splitter.	Dependency Property	Brush
SplitterColor	Sets the color for the splitter.	Dependency Property	Brush

[Adding Splitter for SyncChartArea](#)

Add Splitter for SyncChartArea, by using the following code.

XML

```
<sfChart:SyncChartAreas Name="syncChart" SplitterVisibility="ShowAlways"
SplitterStroke="Red" SplitterColor="Blue" SplitterWidth="2" />
```

C#

```
this.SyncChart.SplitterVisibility = SplitterVisibility.ShowAlways;
this.SyncChart.SplitterStroke = Brushes.Red;
this.SyncChart.SplitterColor = Brushes.Blue;
this.SyncChart.SplitterWidth = 2;
```

Bind Array kind of Objects in Chart WPF

Essential chart now supports binding of array kind of values in BindingPathX and BindingPathsY.

This feature is useful when we use an array or list as data source.

C#

```
<syncfusion:ChartSeries Name="series1" BindingPathsY="Y" BindingPathX="X[0]"
Type="FastStackingColumn" Stroke="Black" DataSource="{Binding}" />
```

We can bind the data values with the index "X[0]" .

*Sample Data Source***C#**

```
public class TestClass
{
    public double Y { get; set; }
    public double[] X { get; set; }
}
public class TestClassCollection : ObservableCollection<TestClass>
{
    public TestClassCollection()
    {
        this.Add(new TestClass { Y = 1, X = new double[] { 12, 3.9 } });
        this.Add(new TestClass { Y = 2, X = new double[] { 3, 3.9 } });
    }
}
```

In the above code:

TestClassCollection is collection of TestClass which has an array object and a double object.

When we bind this kind of data source to the chart series, we can specify the BindingpathX and BindingPathsY values using this feature.

C#

```
item.DataContext = new TestClassCollection1
item.BindingPathX = "X[0]";
item.BindingPathsY = new string[] { "Y" };
```

SmallChange and LargeChange Properties for Chart Area Scrolling Bar

Formerly, the chart area's scrolling bar value change could not be customized by users. This feature enables the user to specify the large and small change values similar to the Slider control.

Properties

Property	Description	Type	Data Type
VerticalBarSmallChange	A double property used to specify the small value change of the vertical scrollbar.	Attached	double
VerticalBarLargeChange	A double property used to specify the large value change of the vertical scrollbar.	Attached	double
HorizontalBarSmallChange	A double property used to specify the small value change of the horizontal scrollbar.	Attached	double
HorizontalBarLargeChange	A double property used to specify the large value change of the horizontal scrollbar.	Attached	double

Adding SmallChange and LargeChange Properties for Chart Area Scrolling Bar to an Application

XML

```
<syncfusion:ChartArea Name="area"
syncfusion:ChartZoomingScrollBar.VerticalBarSmallChange="{Binding
ElementName=ver_sma,Path=Value,Mode=TwoWay}"
syncfusion:ChartZoomingScrollBar.VerticalBarLargeChange="{Binding
ElementName=ver_lar,Path=Value,Mode=TwoWay}"
syncfusion:ChartZoomingScrollBar.HorizontalBarLargeChange="{Binding
ElementName=hor_lar,Path=Value,Mode=TwoWay}"
syncfusion:ChartZoomingScrollBar.HorizontalBarSmallChange="{Binding
ElementName=hor_sma,Path=Value,Mode=TwoWay}" >
<syncfusion:ChartSeries Type="Area" StrokeThickness="1.0"
InactiveSeriesOpacityOnZoom="0.25" x:Name="ser1" DataSource="{Binding
ZoomingModel}" BindingPathX="Id" BindingPathsY="YValue" IsIndexed="False"
Label="Anomaly" />
</syncfusion:ChartArea>
```

C#

```
ChartZoomingScrollBar.SetHorizontalBarLargeChange(area, 0.5);
ChartZoomingScrollBar.SetHorizontalBarSmallChange(area, 0.1);
ChartZoomingScrollBar.SetVerticalBarLargeChange(area, 0.4);
ChartZoomingScrollBar.SetVerticalBarSmallChange(area, 0.1);
```

Additional Zooming Functionality for SyncChartAreas

Essential Chart now supports the concepts of Zoomin, Zoomout, and panning in the SyncChartArea

Adding Additional Zooming Functionality

Add additional zooming functionality by using the following code.

XML

```
<sfChart:SyncChartAreas IsContextMenuEnabled="True">
```

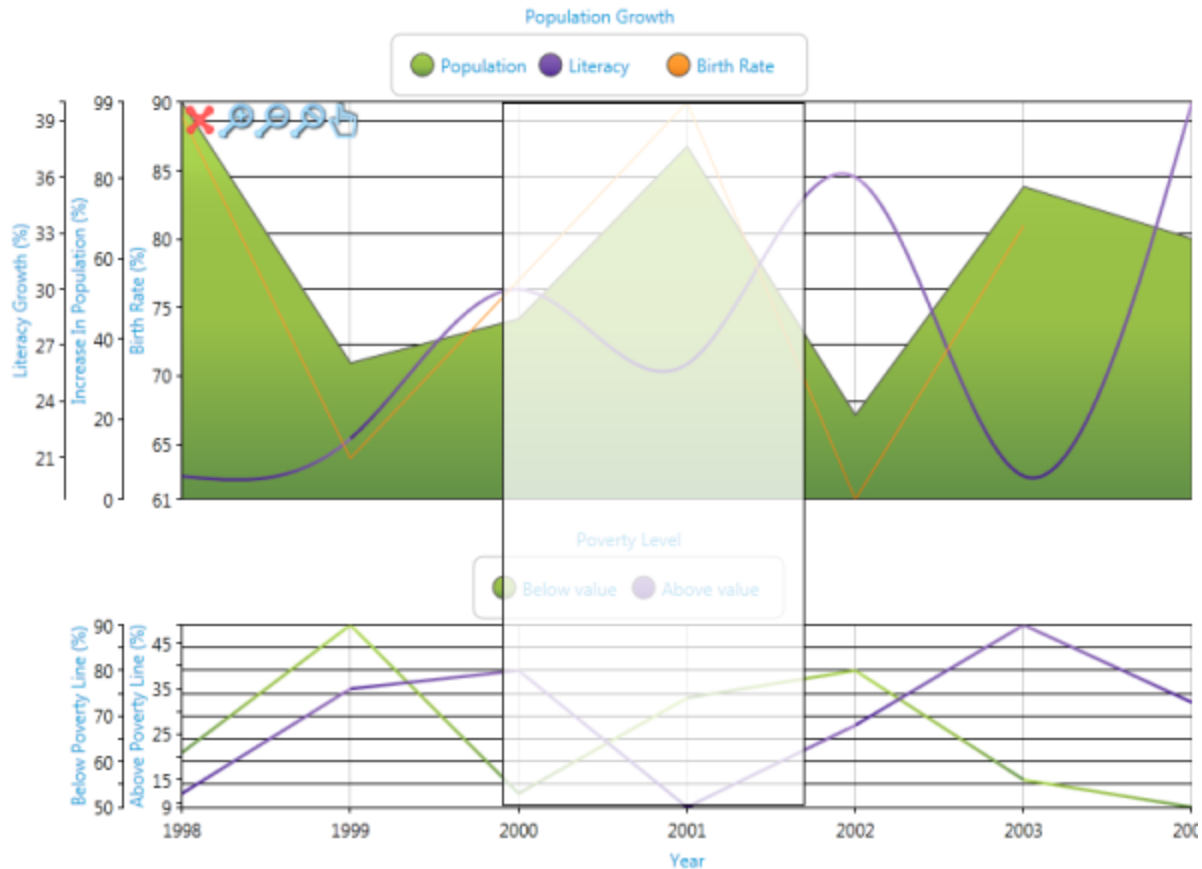

C#

```
syncArea.IsContextMenuEnabled = true;
```

*Rectangular Selection Zooming in SyncChartAreas*

SyncChartArea also has support for sector zooming as in ordinary chart areas.

- SyncChart can be zoomed in using sector zooming instead of the Zoom In icon from the Zooming toolkit.



[Sample Link](#)

Essential Chart WPF > User Interaction > Zooming and Scrolling Demo

XML

```
<sfChart:SyncChartAreas Name="syncChart" EnableMouseDragZooming="True">
</sfChart:SyncChartAreas>
```

C#

```
syncChart.EnableMouseDragZooming = true;
```

Chart-Series in WPF Chart (Classic)

Populating Chart Series

The following are the three steps that should be followed:

- Adding Chart Series
- Data Binding
- Sorting the Series Data

Adding Chart Series

You can add one or more Chart Series to a Chart Area to plot points in the Chart. Note that you can specify one of the several built-in Chart types for rendering the series points.

XML

```
<sfchart:Chart>
<sfchart:ChartArea>
<sfchart:ChartSeries/>
</sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
ChartSeries series = new ChartSeries();
Chart1.Areas[0].Series.Add(series);
```

You can then add points to the series using one of the following Data Binding techniques.

Data Binding

The most common and convenient approach to populate a Chart Series is by simply binding the Chart Series to a business object list. The following properties are used for this purpose.

Property Table

Property	Description
DataSource	takes any IEnumerable instance as the Data Source
BindingPathX	specifies the member / field in the specified data source that contains the x values for the series
BindingPathsY	specifies the members / fields in the specified data source that contain the y values for the seriesNote that some Chart Types require more than one y value and hence this property is of type String Array.

All the common data sources are supported by the Chart control. The following are some of the data sources supported.

- IList instances
- ObservableCollection
- XmlDataProvider
- CollectionViewSource
- LINQ results
- Other compatible data sources

Note that the chart plot will automatically update when the bound data sends a change notification.

The following code example illustrates how to bind the Chart control to an XmlDataProvider.

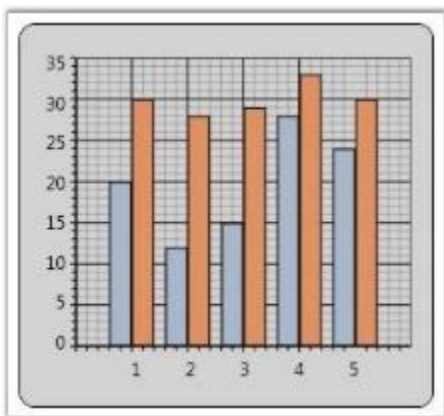
XML

```
<Window.Resources>
<XmlDataProvider x:Key="myXmlData">
<x:XData>
<Products xmlns="">
<Product Sales="20" Projected="30" Month="1"/>
<Product Sales="12" Projected="28" Month="2"/>
```

```

<Product Sales="15" Projected="29" Month="3"/>
<Product Sales="28" Projected="33" Month="4"/>
<Product Sales="24" Projected="30" Month="5"/>
</Products>
</x:XData>
</XmlDataProvider>
</Window.Resources>
<Grid>
<sfchart:Chart Name="Chart2">
<sfchart:ChartArea >
<sfchart:ChartSeries DataSource="{Binding Source={StaticResource myXmlData},
XPath=Products/Product}" BindingPathX="Month"
BindingPathsY="Sales" Type="Column" Label="Actual Sales">
</sfchart:ChartSeries>
<sfchart:ChartSeries DataSource="{Binding Source={StaticResource myXmlData},
XPath=Products/Product}" BindingPathX="Month"
BindingPathsY="Projected" Type="Column" Label="Projected Sales">
</sfchart:ChartSeries>
</sfchart:ChartArea>
</sfchart:Chart>
</Grid>

```



Sorting the Series Data

The data can either be sorted or unsorted. If you are sure that the data passed to the collection is sorted, then you can turn off the sorting feature by using the following code.

C#

```
Series.IsSortData = false;
```

Series Customization

Chart Series can be customized with various properties. This section discusses the following topics.

Interior

The interior of the Chart Series can be set by using the Interior property.

XML

```

<Window.Resources>
<!--To be added in window resources.-->

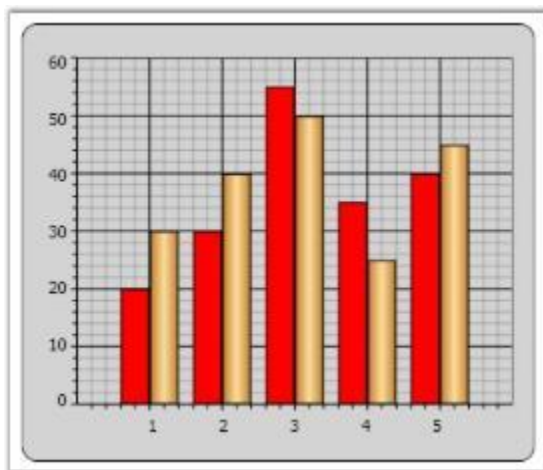
```

```

<LinearGradientBrush x:Key="SeriesAInterior" EndPoint="1,0.5"
StartPoint="0,0.5">
<LinearGradientBrush.GradientStops>
<GradientStop Color="#FFC07E2C" Offset="0"/>
<GradientStop Color="#FFFDD9E" Offset="0.5"/>
<GradientStop Color="#FFC07E2C" Offset="1"/>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
</Window.Resources>
<!--To be added in window resources.-->
<syncfusion:ChartSeries Interior="Red" Data="1 20 2 30 3 55 4 35 5 40" />
<syncfusion:ChartSeries Interior="{StaticResource SeriesAInterior}"
Label="Series 2" Data="1 30 2 40 3 50 4 25 5 45" />

```

The following screenshot illustrates Chart Series Interior settings.



IsVisible

Essential Chart for WPF enables you to show / hide the Chart Series by using the IsVisible boolean property provided by the ChartSeries class.

XML

```

<syncfusion:Chart >
<syncfusion:ChartArea IsContextMenuEnabled="True" >
<syncfusion:ChartSeries Label="Series 1" IsVisible="False" Data=" 1 35 2 45
3 30 4 25 5 40" />
<syncfusion:ChartSeries Label="Series 2" Data=" 1 30 2 40 3 50 4 20 5 45" />
</syncfusion:ChartArea>
</syncfusion:Chart>

```

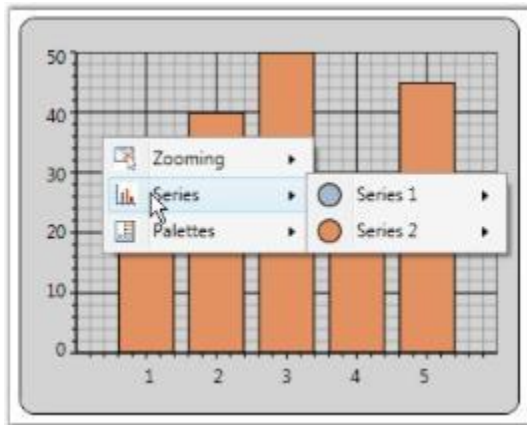
C#

```

ChartSeries series = new ChartSeries();
series.IsVisible = false;

```

The following screenshot illustrates Chart with Series 1 invisible.



IsRotated

Chart Series can be rotated by using the `ChartSeries.IsRotated` property.

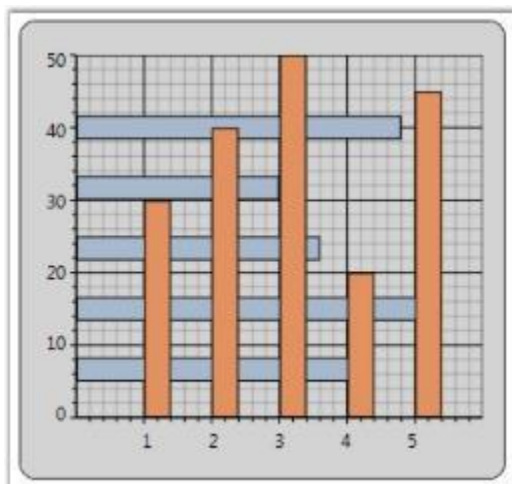
XML

```
<syncfusion:Chart >
<syncfusion:ChartArea >
<syncfusion:ChartSeries Label="Series 1" IsRotated="True" Data=" 1 35 2 45 3
30 4 25 5 40" />
<syncfusion:ChartSeries Label="Series 2" Data=" 1 30 2 40 3 50 4 20 5 45" />
</syncfusion:ChartArea>
</syncfusion:Chart>
```

C#

```
area.Series[0].IsRotated = true;
```

The following screenshot illustrates Chart with Series 1 rotated.



Label

The text displayed in the Chart Legends and the Chart Area context menu can be customized by using the `ChartSeries.Label` property.

XML

```
<syncfusion:ChartArea>
<syncfusion:ChartSeries Type="Column" Label="Series 1"/>
<syncfusion:ChartSeries Type="Column" Label="Series 2"/>
</syncfusion:ChartArea>
```

C#

```
area.Series[0].Label = "Series 1";
area.Series[1].Label = "Series 2";
```

The following screenshot illustrates Chart with customized Series Labels.

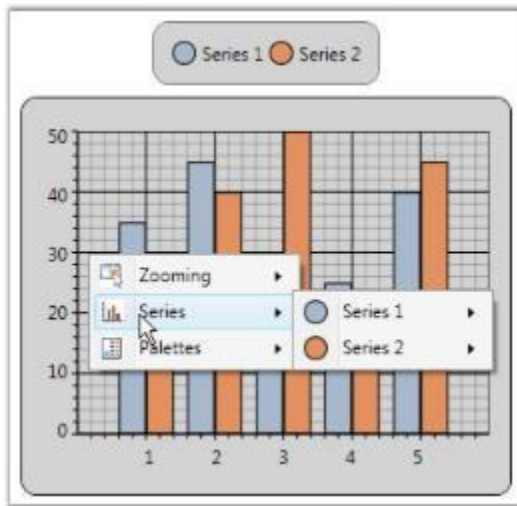
**Chart-Series in WPF Chart (Classic) Types**

Chart control supports 34 types of charts, among which 12 types are supported in 3D mode as well. The Type property is used to specify the Chart Type.

XML

```
<Window.Resources>
<local:ProductSalesCollection x:Key="SeriesData1"/>
</Window.Resources>
<sfchart:Chart>
<sfchart:ChartArea>
<sfchart:ChartSeries Type="Area" DataSource="{StaticResource SeriesData1}"
BindingPathX="Year" BindingPathsY="Sales"/>
</sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
ChartSeries series = new ChartSeries();
series.DataSource = new ProductSalesCollection();
series.BindingPathX = "Year";
series.BindingPathsY = new string[] { "Sales" };
Chart1.Areas[0].Series.Add(series);
```

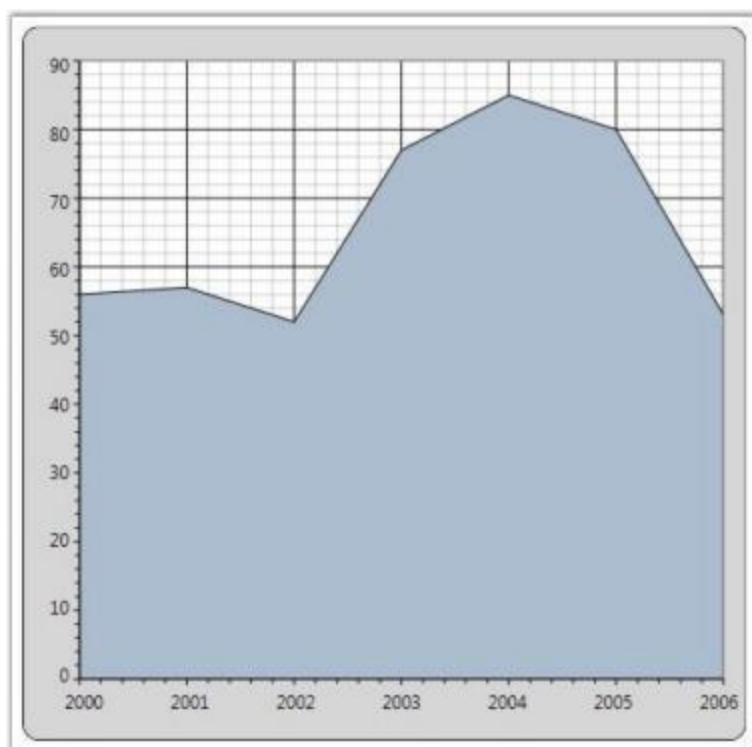


Chart-Series in WPF Chart (Classic) Look and Feel

Chart for WPF provides various options to customize the look and feel of the Chart Series. The following are some of the properties that are used for this purpose.

Property

Property	Description
Interior	specifies the fill color of chart series
Stroke	specifies the border color of the chart series segment
StrokeThickness	specifies the thickness of the chart series segment border

The following code example illustrates how to set the preceding properties.

XML

```
<Window.Resources>
<local:ProductSalesCollection x:Key="SeriesData1"/>
</Window.Resources>
<sfchart:Chart>
<sfchart:ChartArea>
<sfchart:ChartSeries Type="Area" DataSource="{StaticResource SeriesData1}"
BindingPathX="Year" BindingPathsY="Sales"
Interior="LightCoral" Stroke="Black" StrokeThickness="1.5"/>
</sfchart:ChartArea>
</sfchart:Chart>
```

C#


```

ChartSeries series = new ChartSeries();
series.DataSource = new ProductSalesCollection();
series.BindingPathX = "Year";
series.BindingPathsY = new string[] { "Sales" };
series.Interior = Brushes.LightCoral;
series.Stroke = Brushes.Black;
series.StrokeThickness = 1.5;
Chart1.Areas[0].Series.Add(series);

```

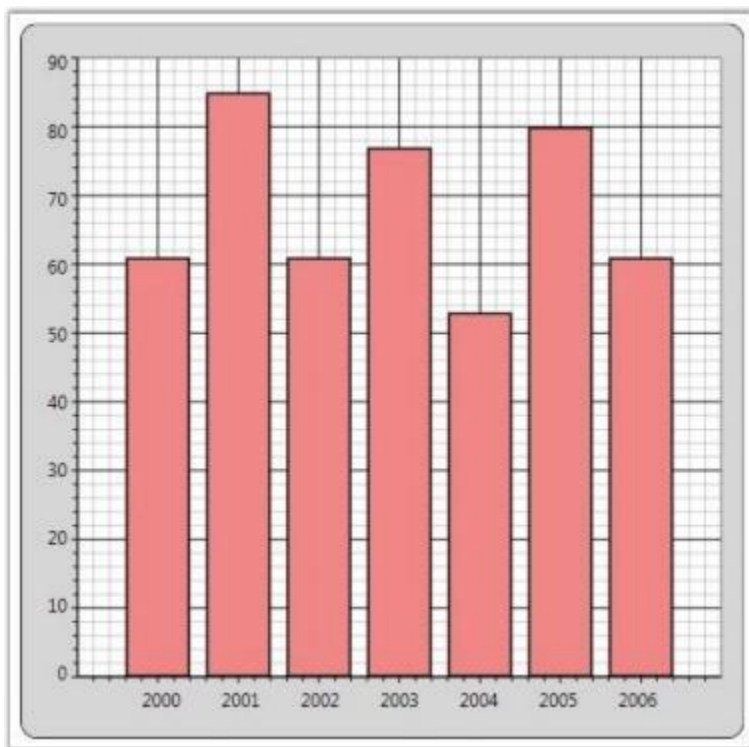


Chart-Series in WPF Chart (Classic) Template

Chart for WPF enables you to apply custom Data Templates to the Chart Series. By applying the custom data templates, the Chart Series Segments can be altered. The following code example illustrates how to create a sample data template to draw a Line Chart.

Note: Data templates cannot be customized for a single segment in a series. As each series is drawn as a single segment, we cannot customize the individual segments.

XML

```

<Window.Resources>
  <!--Chart Series Data-->
  <XmlDataProvider x:Key="myXmlData">
    <x:XData>
      <Products xmlns="">
        <Product Sales="20" Projected="30" Month="1"/>
        <Product Sales="12" Projected="28" Month="2"/>
        <Product Sales="15" Projected="29" Month="3"/>
        <Product Sales="28" Projected="33" Month="4"/>
        <Product Sales="24" Projected="30" Month="5"/>
      </Products>
    </x:XData>
  </XmlDataProvider>

```

```

</x:XData>
</XmlDataProvider>
<!--Line Chart Template-->
<DataTemplate x:Key="Templatel">
<Line X1="{Binding X1}" X2="{Binding X2}" Y1="{Binding Y1}" Y2="{Binding
Y2}" StrokeThickness="2" Stroke="{Binding Interior}"
StrokeDashArray="3,2" />
</DataTemplate>
</Window.Resources>
<sfchart:Chart>
<sfchart:ChartArea Background="LightGray" GridBackground="White">
<sfchart:ChartSeries Template="{StaticResource Templatel}"
DataSource="{Binding Source={StaticResource myXmlData},
XPath=Products/Product}" BindingPathX="Month" BindingPathsY="Sales"
Interior="Red" Type="Line"/>
</sfchart:ChartArea>
</sfchart:Chart>

```

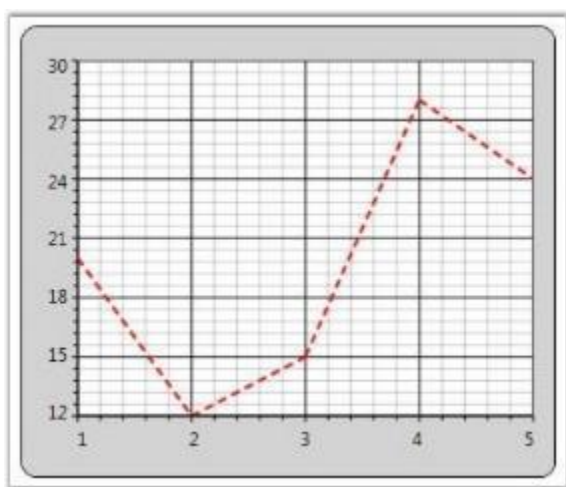


Chart-Series in WPF Chart (Classic) Adornments

Chart Series Adornments are used to display values in a Chart Segment related to it. Values from data points (x value, y value) or other properties from a data source can be displayed. ChartAdornmentsInfo class is used to display Chart Series Adornments. ChartAdornmentsInfo class provides the following properties to customize the chart series adornments.

Property Table

Property	Description
LabelContentPath	specifies the value to display in the AdornmentsFor example, to display the x value, LabelContentPath will be DataPoint.X.
Symbol	allows selecting one symbol among the eleven predefined symbols or custom symbol options
SymbolInterior	specifies the interior of the predefined symbols added to the adornments
SymbolHeight	specifies the height of the predefined symbols added to the adornments
SymbolWidth	specifies the width of the predefined symbols added to the adornments

SymbolTemplate	specifies the symbol to be displayed in the adornments when the Symbol property is set to <i>Custom</i>
VerticalAlignment	specifies the Vertical alignment of the Adornment text
HorizontalAlignment	specifies the Horizontal alignment of the adornments
LabelTemplate	customizes the look and feel of the adornments being displayed

The following code example illustrates how to display adornments in a Chart Series.

XML

```
<Window.Resources>
  <!--Chart Series Data-->
  <XmlDataProvider x:Key="myXmlData">
    <x:XData>
      <Products xmlns="">
        <Product Sales="20" Projected="30" Month="1"/>
        <Product Sales="12" Projected="28" Month="2"/>
        <Product Sales="15" Projected="29" Month="3"/>
        <Product Sales="28" Projected="33" Month="4"/>
        <Product Sales="24" Projected="30" Month="5"/>
      </Products>
    </x:XData>
  </XmlDataProvider>
  <!--ChartAdornmentInfo LabelTemplate-->
  <DataTemplate x:Key="Lb1txt1">
    <TextBlock Name="TB1" Text="{Binding}" FontSize="13" Foreground="Red"
      TextAlignment="Justify" VerticalAlignment="Center" FontWeight="Bold" />
  </DataTemplate>
</Window.Resources>
<!--Chart with Adornments-->
<sfchart:Chart>
  <sfchart:ChartArea Background="LightGray" GridBackground="White">
    <sfchart:ChartArea.PrimaryAxis>
      <sfchart:ChartAxis sfchart:ChartArea.ShowGridLines="False" />
    </sfchart:ChartArea.PrimaryAxis>
    <sfchart:ChartSeries Type="Column" DataSource="{Binding
      Source={StaticResource myXmlData}, XPath=Products/Product}"
      BindingPathX="Month" BindingPathsY="Sales" Interior="LightSkyBlue"
      Stroke="Black" StrokeThickness="1.5">
      <sfchart:ChartSeries.AdornmentsInfo>
        <sfchart:ChartAdornmentInfo LabelTemplate="{StaticResource Lb1txt1}"
          LabelContentPath="DataPoint.X" Visible="True"
          VerticalAlignment="Top" />
      </sfchart:ChartSeries.AdornmentsInfo>
    </sfchart:ChartSeries>
  </sfchart:ChartArea>
</sfchart:Chart>
```

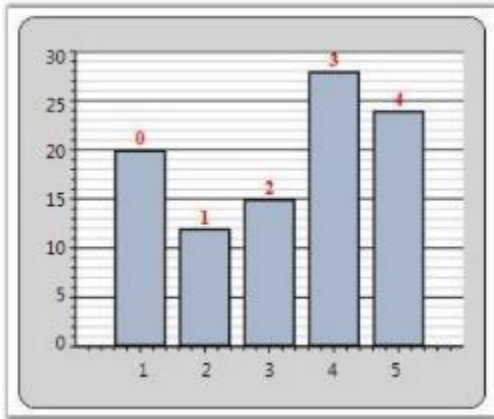
C#

```
ChartSeries series = Chart1.Areas[0].Series[0];
ChartAdornmentInfo adornments = series.AdornmentsInfo;
```

```

adornments.LabelContentPath = "DataPoint.X";
adornments.LabelTemplate = this.Resources["Lb1txt1"] as DataTemplate;
adornments.Visible = true;
adornments.VerticalAlignment = VerticalAlignment.Top;

```



Symbol Template for Chart Adornment

The following Symbol Templates can be set for ChartAdornmentInfo.

- Predefined Symbol Template (set from one among the twelve predefined symbols)
- Custom Template (custom Data Template set by users)

Predefined Symbol Template

You can set any one of the following predefined symbols for ChartAdornmentInfo.

- Cross
- Diamond
- Ellipse
- Hexagon
- HorizontalLine
- InvertedTriangle
- Pentagon
- Plus Square
- Triangle
- VerticalLine

The following code example illustrates how to apply predefined symbol templates to chart adornments.

XML

```

<Syncfusion:ChartSeries Type="Column" DataSource="{StaticResource
collection1}" BindingPathX="X" BindingPathsY="Y" IsIndexed="True"
Stroke="Black" StrokeThickness="1.5">
<Syncfusion:ChartSeries.AdornmentsInfo>
<Syncfusion:ChartAdornmentInfo Visible="True" Symbol="Pentagon"
SymbolInterior="Green" SymbolHeight="25" SymbolWidth="25" />
</Syncfusion:ChartSeries.AdornmentsInfo>
</Syncfusion:ChartSeries>

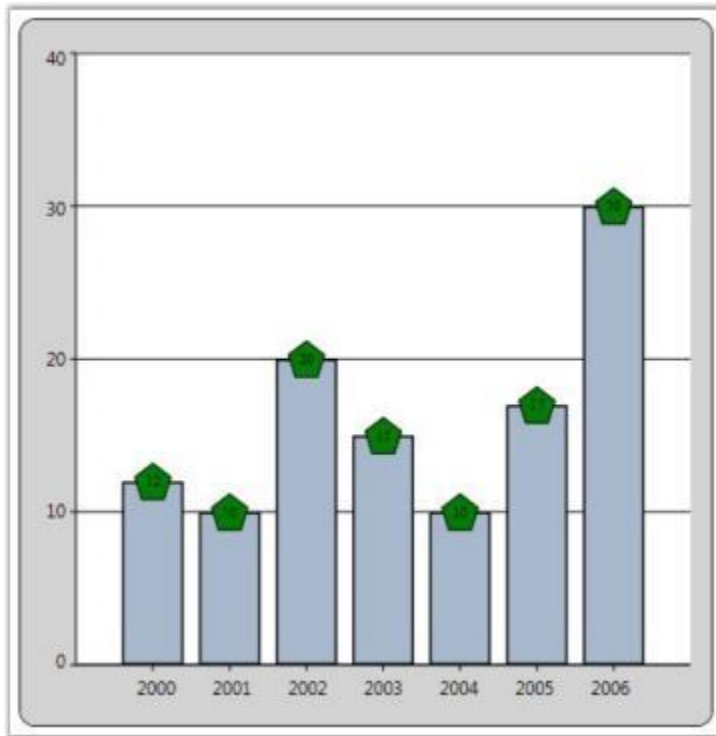
```

C#

```

series1.AdornmentsInfo.Visible = true;
series1.AdornmentsInfo.Symbol = Symbol.Pentagon;
series1.AdornmentsInfo.SymbolInterior = Brushes.Green;
series1.AdornmentsInfo.SymbolHeight = 25;
series1.AdornmentsInfo.SymbolWidth = 25;

```

*Custom Symbol Template*

The following code example illustrates how to apply custom symbol templates to chart adornments.

XML

```

<Window.Resources>
<local:ProductSalesCollection x:Key="SeriesData1"/>
<DataTemplate x:Key="Lb1txt1">
<TextBlock Name="TB1" Text="{Binding}" FontSize="11" Foreground="Black"
TextAlignment="Justify" VerticalAlignment="Center"
FontWeight="Bold">
</TextBlock>
</DataTemplate>
<DataTemplate x:Key="SymbolTemplate">
<Rectangle Stroke="Black" Fill="Red" Width="10" Height="10"/>
</DataTemplate>
</Window.Resources>
<sfchart:Chart>
<sfchart:ChartArea>
<sfchart:ChartSeries Type="Area" DataSource="{StaticResource SeriesData1}"
BindingPathX="Year" BindingPathsY="Sales"
Interior="LightCoral" Stroke="Black" StrokeThickness="1.5">
<sfchart:ChartSeries.AdornmentsInfo>

```

```

<sfchart:ChartAdornmentInfo SymbolTemplate="{StaticResource SymbolTemplate}"
LabelTemplate="{StaticResource Lbltxt1}"
LabelContentPath="DataPoint.X" Visible="True" VerticalAlignment="Top"/>
</sfchart:ChartSeries.AdornmentsInfo>
</sfchart:ChartSeries>
</sfchart:ChartArea>
</sfchart:Chart>

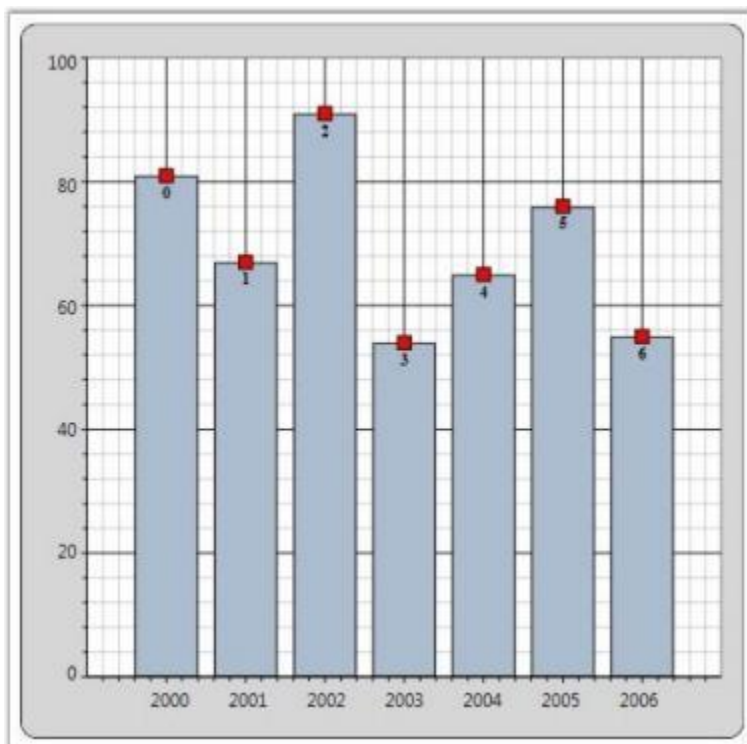
```

C#

```

ChartSeries series = new ChartSeries();
series.DataSource = new ProductSalesCollection();
series.BindingPathX = "Year";
series.BindingPathsY = new string[] { "Sales" };
series.Stroke = Brushes.Black;
series.StrokeThickness = 1d;
Chart1.Areas[0].Series.Add(series);
ChartAdornmentInfo adornments = series.AdornmentsInfo;
adornments.LabelContentPath = "DataPoint.X";
adornments.LabelTemplate = this.Resources["Lbltxt1"] as DataTemplate;
adornments.Visible = true;
adornments.VerticalAlignment = VerticalAlignment.Bottom;
adornments.SymbolTemplate = this.Resources["SymbolTemplate"] as
DataTemplate;

```

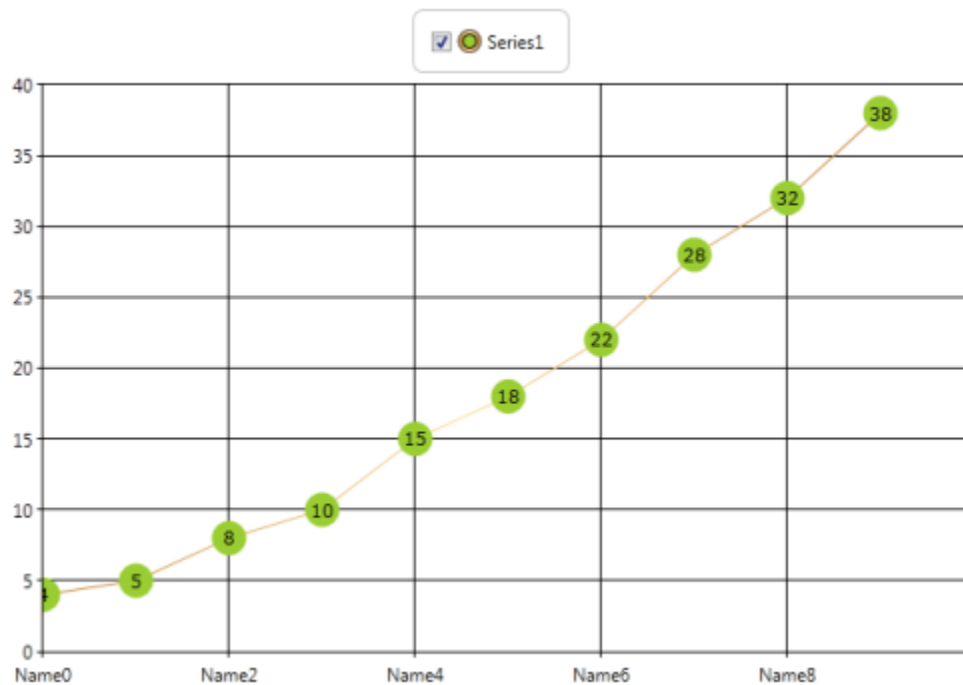
*Adornment Support for Fast Chart Types*

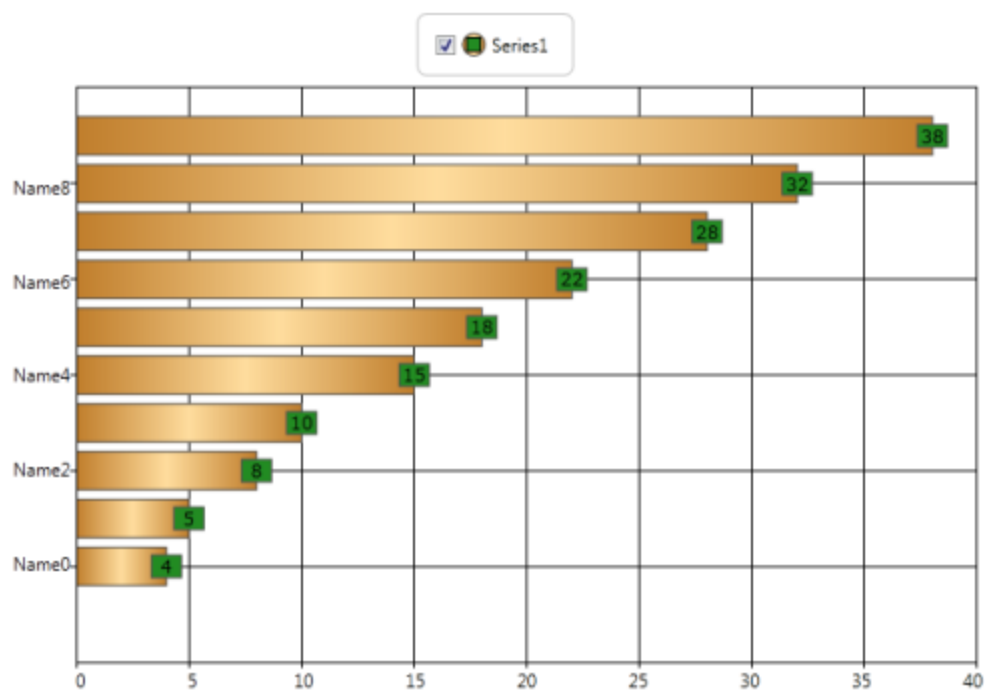
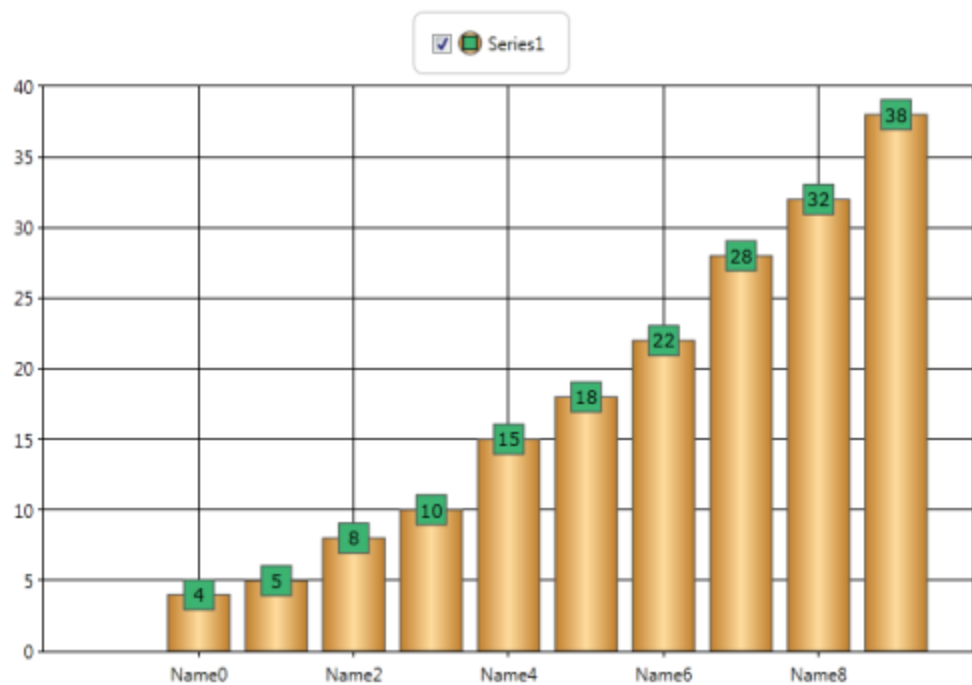
This feature helps the user use adornments in fastchart types too. Considering the performance of Fastchart types, only limited adornment support for fastchart types have been provided.

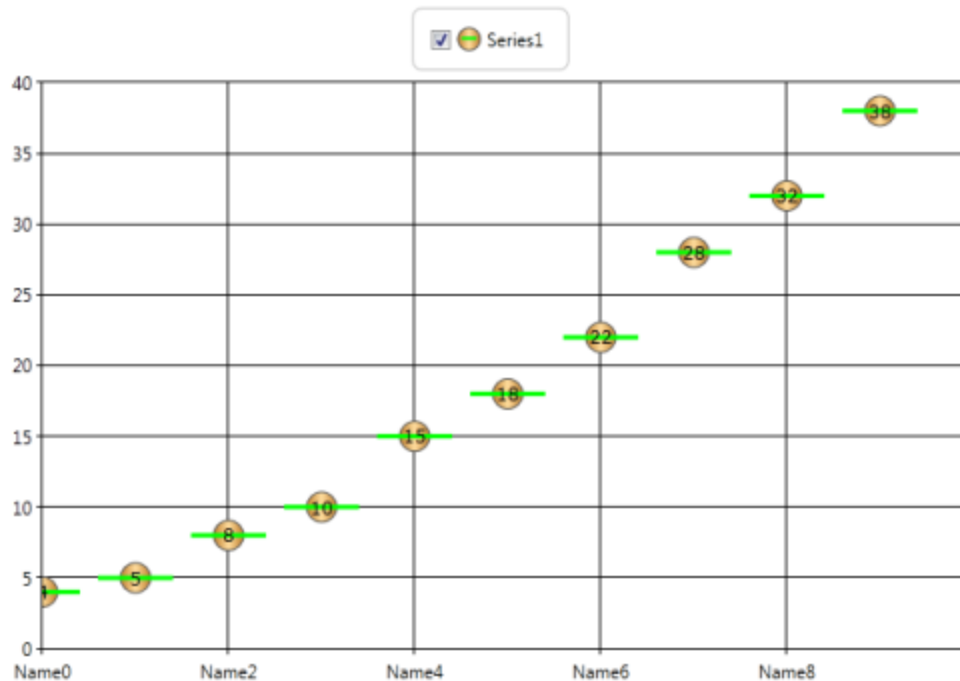
- Limited adornment symbols are supported for Fastchart types. They are
- Ellipse
- Square
- Horizontal line
- Vertical line
- Symbol interior can be changed.
- Symbol height and width can be changed.

Use Case Scenarios

Most fastchart types are used in stock market charts. Users can display the data of the point in the adornment.







[Sample Link](#)

Essential Chart WPF > Chart Series > Adornments Configuration Demo

Adding Fastchart Types with Adornments to an Application

XML

```
<Syncfusion:ChartSeries.AdornmentsInfo>
<Syncfusion:ChartAdornmentInfo Visible="True" Symbol="Ellipse"
SymbolInterior="Red" SymbolHeight="20" SymbolWidth="20" />
</Syncfusion:ChartSeries.AdornmentsInfo>
</Syncfusion:ChartSeries>
```

C#

```
series1.AdornmentsInfo.Visible = true;
series1.AdornmentsInfo.Symbol = Symbol.Ellipse;
series1.AdornmentsInfo.SymbolInterior = Brushes.Red;
series1.AdornmentsInfo.SymbolHeight = 20;
series1.AdornmentsInfo.SymbolWidth = 20;
```

Chart Segment Labels

Labels can be displayed in the chart types such as Pie, Doughnut, Pyramid and Funnel. Chart Segment Labels can be used to display information like x value, y value, percentage, y value of total and datetime. This feature has been implemented based on the ChartAdornmentsInfo class. The following code example illustrates how to add Chart Segment Labels.

XML

```
<sfchart:ChartArea>
<sfchart:ChartSeries Type="Pie">
```

```

<sfchart:ChartSeries DataSource="{Binding Source={StaticResource myXmlData},
XPath=Products/Product}" BindingPathX="Month"
BindingPathsY="Sales" >
<sfchart:ChartSeries.AdornmentsInfo>
<sfchart:ChartAdornmentInfo LabelContentPath="SegmentLabel" Visible="True"
SegmentShowLine="False"
SegmentLabelContent="Percentage" SegmentLabelFontSize="12"/>
</sfchart:ChartSeries.AdornmentsInfo>
</sfchart:ChartSeries>
</sfchart:ChartArea>

```

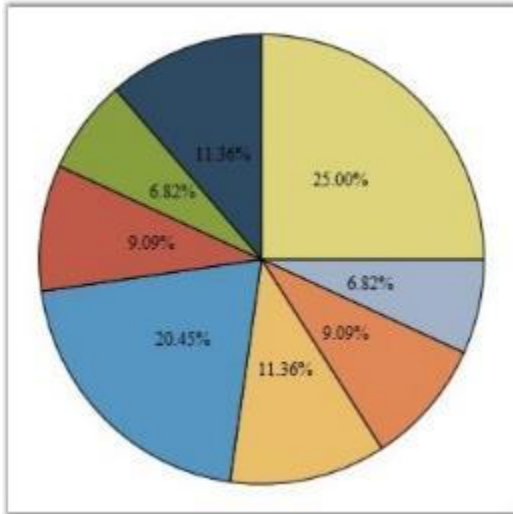


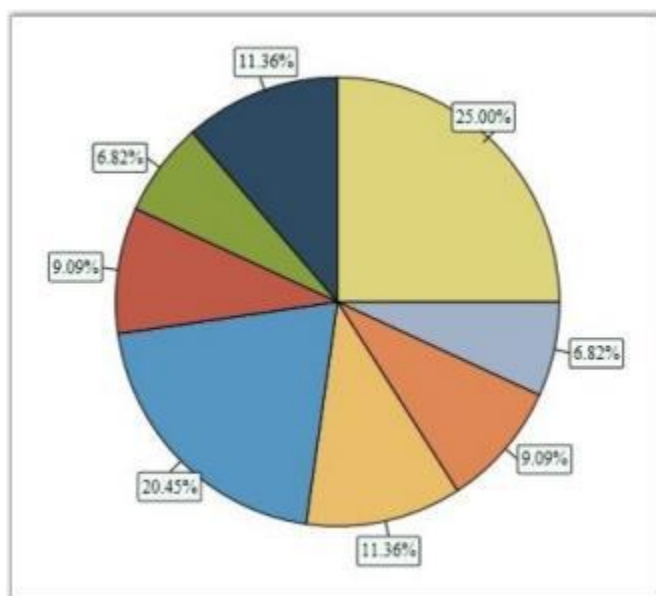
Chart supports applying custom templates to the chart segment labels. The following code example illustrates this.

XML

```

<Window.Resources>
<DataTemplate x:Key="labelsTemplate">
<Border CornerRadius="1" Margin="0" Padding="2" BorderBrush="Black"
Background="MintCream" BorderThickness="1">
<ContentPresenter Content="{Binding}" />
</Border>
</DataTemplate>
<DataTemplate x:Key="connectorTemplate">
<Line X1="0" X2="10" Y1="0" Y2="0" Stroke="Black"/>
</DataTemplate>
</Window.Resources>
<sfchart:ChartSeries DataSource="{Binding Source={StaticResource myXmlData},
XPath=Products/Product}" BindingPathX="Month"
BindingPathsY="Sales" Type="Pie">
<sfchart:ChartSeries.AdornmentsInfo>
<sfchart:ChartAdornmentInfo LabelContentPath="SegmentLabel"
SegmentIsOut="False" SegmentLabelContent="Percentage" Visible="True"
SegmentShowLine="True" SegmentLabelFontSize="12"
LabelTemplate="{StaticResource labelsTemplate}"
ConnectorTemplate="{StaticResource connectorTemplate}" />
</sfchart:ChartSeries.AdornmentsInfo>
</sfchart:ChartSeries>

```



The following properties are used to customize the look and feel of the Chart Segment Labels.

Property

Name	Type	Description	Default Value
SegmentLabelContent	Enum	specifies value of the labelThe options included are as follows.DateTimePercentageXValueYofTotalYValue	YValue
SegmentIsOut	Bool	indicates whether label is inside or outside the segment	False
SegmentShowLine	Bool	indicates whether line is shown from the segment to label	False
HorizontalAlignment	HorizontalAlignment	specifies horizontal label alignment with respect to segment	Center
VerticalAlignment	VerticalAlignment	specifies vertical label alignment with respect to segment	Center
SegmentLabelDateTimeFormat	String	specifies segment label data time format	"dd/MM/yyyy"
SegmentLabelFontFamily	FontFamily	specifies font of the label	Times New Roman
SegmentLabelFontSize	Int32	specifies font size of the label	10
SegmentLabelFontWeight	FontWeight	specifies font weight of the label	Normal
SegmentLabelFormat	String	specifies label format	"0.00"

SegmentLabelRotation	double	specifies rotation angle for the label	0d
ConnectorTemplate	DataTemplate	label's connector template	Null
LabelContentPath	String	label's content path	"SegmentLabel"
LabelTemplate	DataTemplate	label's template	Null
SymbolTemplate	DataTemplate	symbol's template	Null

Note: To show the segment lines, the SegmentShowLine property must be set to true and a ConnectorTemplate must be associated.

Chart-Series in WPF Chart (Classic) Empty Points

Essential Chart provides support for Empty Points. The data collection that is passed to the chart can have NaN or infinite values that will be considered as Empty Points. You can also hide the empty points by setting the ShowEmptyPoints property to *false*.

XML

```
<syncfusion:ChartSeries Type="Column" Name="series1"
EmptyPointInterior="Red" EmptyPointStyle="SymbolAndInterior"
Interior="Green" IsIndexed="False" ShowEmptyPoints="True" Stroke="Black"
StrokeThickness="1"/>
```

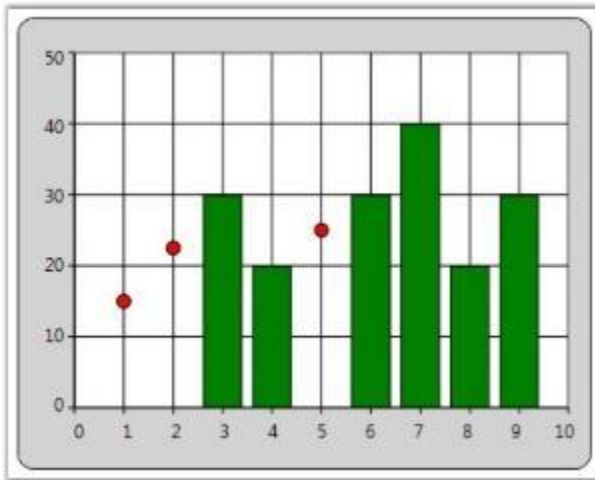
C#

```
// Display Empty Points.
series1.ShowEmptyPoints = true;
// Set Empty Point style.
series1.EmptyPointStyle = EmptyPointStyle.SymbolAndInterior;
// Set Empty Point symbol color.
series1.EmptyPointInterior = Brushes.Red;
// Collection with Nan values.
public IList products()
{
    Random rand = new Random(DateTime.Now.Millisecond);
    List<product> productList = new List<product>();
    productList.Add(new product() { ProdId = 1, Prodname = "Rice", Price =
double.NaN, Stock = 3.5 });
    productList.Add(new product() { ProdId = 2, Prodname = "Wheat", Price =
double.NaN, Stock = 5.8 });
    productList.Add(new product() { ProdId = 3, Prodname = "Oil", Price = 30,
Stock = 2.1 });
    productList.Add(new product() { ProdId = 4, Prodname = "Corn", Price = 20,
Stock = 5.1 });
    productList.Add(new product() { ProdId = 5, Prodname = "Gram", Price =
double.NaN, Stock = 2.0 });
    productList.Add(new product() { ProdId = 6, Prodname = "Milk", Price = 30,
Stock = 1.5 });
    productList.Add(new product() { ProdId = 7, Prodname = "Oil", Price = 40,
Stock = 2.0 });
}
```

```

productList.Add(new product() { ProdId = 8, Prodname = "Corn", Price = 20,
Stock = 2.5 });
productList.Add(new product() { ProdId = 9, Prodname = "Butter", Price = 30,
Stock = 1.5 });
return productList;
}

```



Empty Point Symbol customization

This feature enables you to customize the marker for the empty point. You can differentiate the points using symbol or interior color. This support is available for all chart types except Fast chart type.

Property

Property	Description	Type	Data Type	Reference links
EmptyPointSymbolTemplate	Used to apply user defined template for the empty point symbol.	Dependency Property	DataTemplate	NA

Customizing Chart Empty Point Symbol

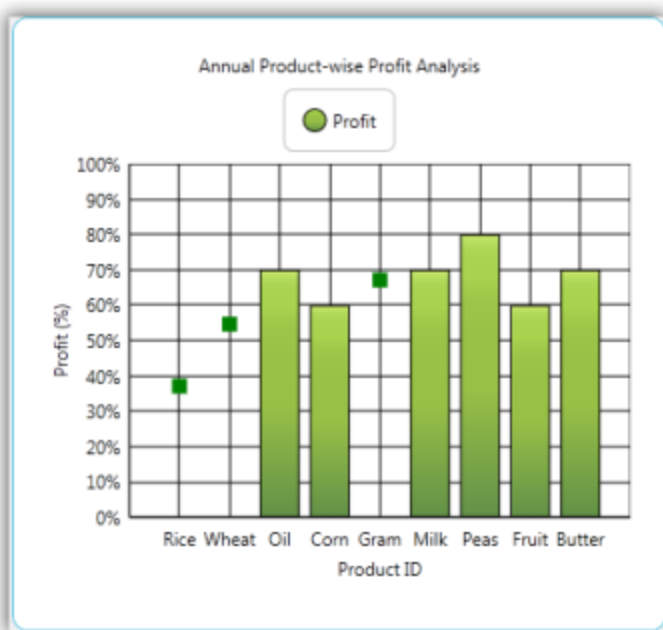
You can customize the empty point symbol using the *EmptyPointSymbolTemplate* property. The following code illustrates this:

XML

```

<!--Data Template for Empty point symbol-->
<DataTemplate x:Key="EmptyTemp">
    <Grid>
        <Rectangle Fill="Green" Margin="0,0,0,10" Width="10" Height="10" />
    </Grid>
</DataTemplate>
<!-- Adding Data Template for Chart series-->
<syncfusion:ChartSeries Name="series1" Label="Profit"
EmptyPointSymbolTemplate="{StaticResource EmptyTemp}">

```



[Sample Link](#)

To view a sample:

1. Open the Syncfusion Dashboard.
2. Select User Interface.
3. Click the WPF drop-down list and select Explore Samples.
4. Navigate to *Chart.WPF\Samples\3.5\WindowsSamples\Chart Customization*

[Empty Point Default Value](#)

This feature enables you to specify the default value of the empty point. You can either set this as zero or the average of nearest value on the adjacent side.

[Properties](#)

Property	Description	Type	Data Type	Reference links
EmptyPointValue	Specifies whether empty point has to show zero or average value.	Dependency Property	EmptyPointValue	NA

Customizing the Default Value of the Empty Point

You can customize the default value of the empty point using the *EmptyPointValue* property.

Set the EmptyPointValue property to *Zero*, the default empty point value will be zero.

The following code illustrates this:

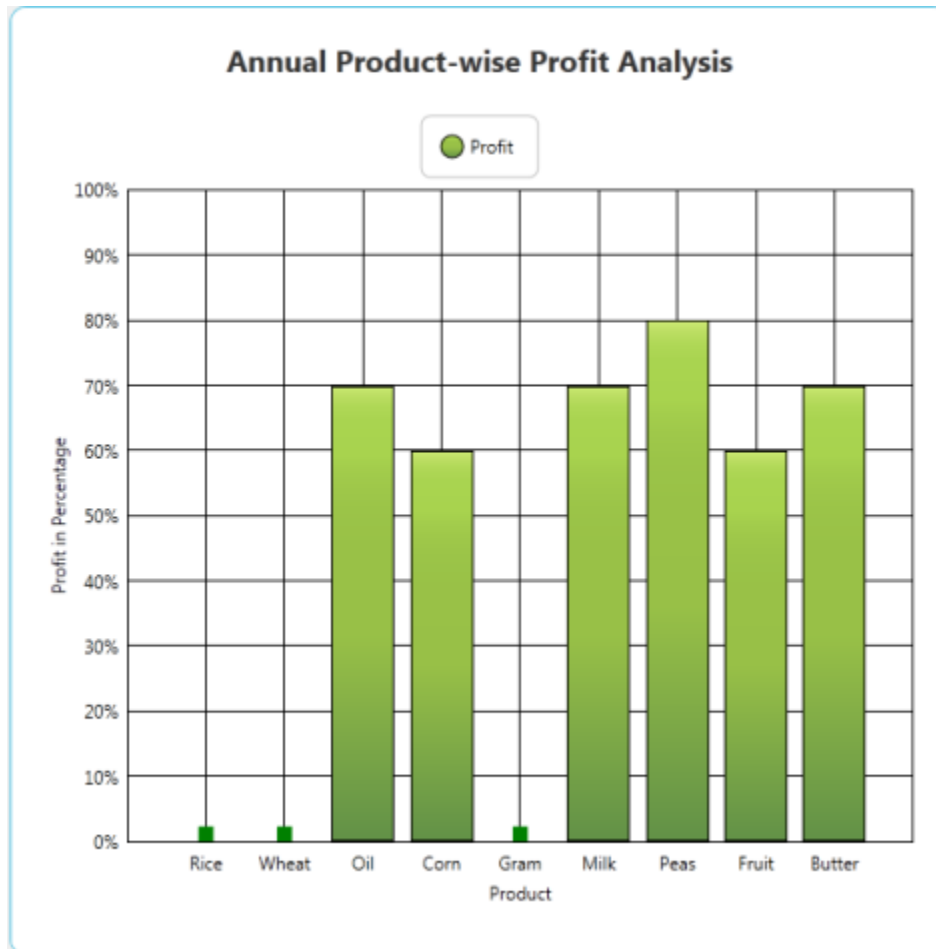
XML

```
<syncfusion:ChartSeries Name="series1" EmptyPointValue="Zero"
EmptyPointStyle="Symbol" ShowEmptyPoints="True"
EmptyPointSymbolTemplate="{StaticResource EmptyTemp}"
```

```
Stroke="Black" StrokeThickness="1"/>
```

C#

```
series1.EmptyPointValue = EmptyPointValue.Zero;
```



Set the EmptyPointValue property to Average, the default empty point value will be the average of nearest value on the adjacent side.

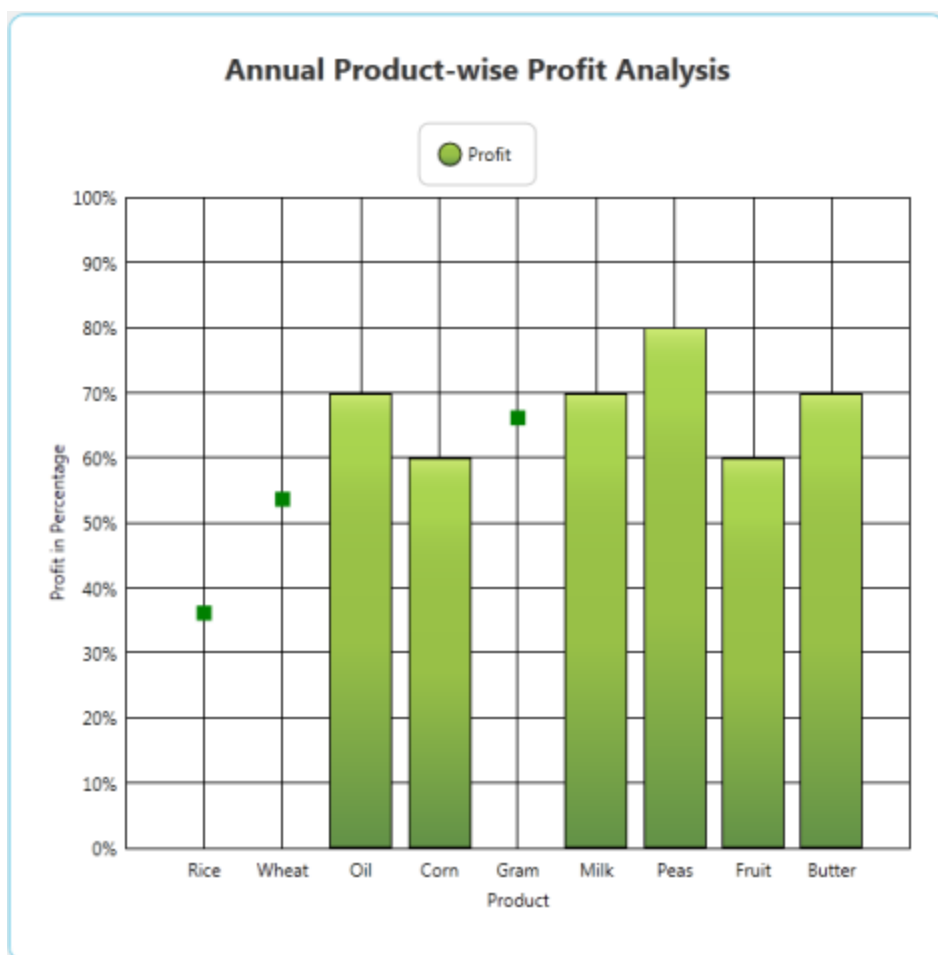
The following code illustrates this:

XML

```
<syncfusion:ChartSeries Name="series1" EmptyPointValue="Average"
EmptyPointStyle="Symbol" ShowEmptyPoints="True"
EmptyPointSymbolTemplate="{StaticResource EmptyTemp}"
Stroke="Black" StrokeThickness="1"/>
```

C#

```
series1.EmptyPointValue = EmptyPointValue.Average;
```



Applying Different Colors to Chart Series Segments

Essential Chart allows setting different colors to each data point in a chart series by using built-in color palettes and custom palettes.

- Using the ColorEach property, the segments of a chart series can be applied with various colors.
- Chart series segments are allowed to be set with unique colors without specifying an interior color for individual segments of chart series.

Use Case Scenarios

With the help of this feature, users can differentiate huge numbers of data points with different colors through which chart segments can be easily identified.

Tables for Properties, Methods, and Events

Properties

Property	Description	Type	Data Type
ColorEach	Specifies whether each data point of a series is shown in a different color.	Dependency	Bool
Palette	To set the color palette for chart series segments.	Dependency	ChartColorPalette

CustomPalette	To set the custom color palette for chart series segments.	Dependency	Brush[]
---------------	--	------------	---------

[Sample Link](#)

To access the chart series multi-color segments demo:

1. Open the Syncfusion Dashboard.
2. Select User Interface.
3. Click the WPF drop-down list and select Explore Samples.
4. Browse to the path Chart.WPF\Samples\3.5\WindowsSamples\Chart Series\Series Multi-Color Segments Demo.

Adding Colorful Chart Series Segments to an Application

[Built-in Palette](#)

XML

```
<syncfusion:ChartSeries DataSource="{Binding ProductModel}"
ColorEach="True" Palette="Gradient"
BindingPathX="Months" BindingPathsY="Sales">
</syncfusion:ChartSeries>
```

C#

```
this.Series1.ColorEach = true;      this.Series1.Palette =
ChartColorPalette.Gradient;
```



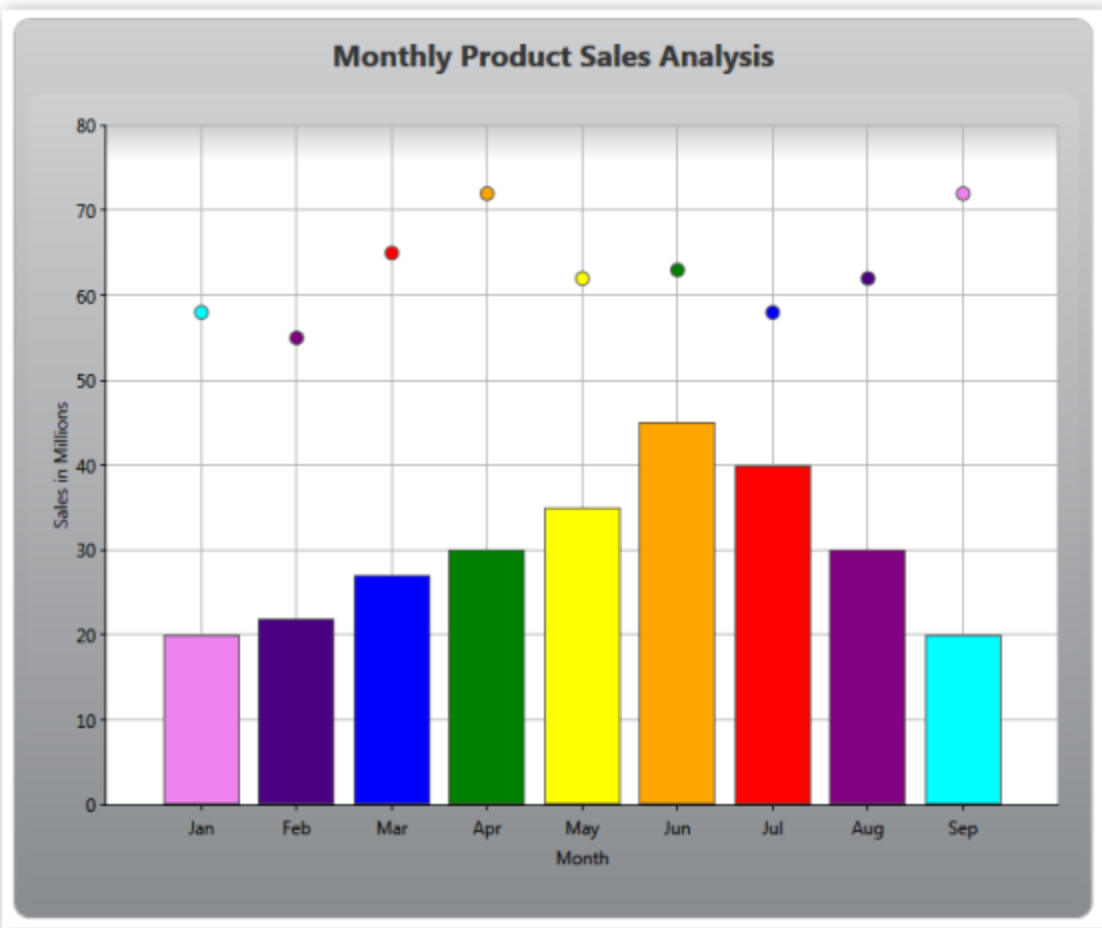
Custom Palette

XML

```
<syncfusion:ChartSeries DataSource="{Binding ProductModel}" ColorEach="True"
  Palette="Custom" BindingPathX="Months" BindingPathsY="Sales">
  <syncfusion:ChartStyleModel.CustomPalette>
    <x:Array Type='Brush'>
      <SolidColorBrush Color='Violet'></SolidColorBrush><SolidColorBrush
        Color='Indigo'></SolidColorBrush><SolidColorBrush
        Color='Blue'></SolidColorBrush><SolidColorBrush
        Color='Green'></SolidColorBrush><SolidColorBrush
        Color='Yellow'></SolidColorBrush><SolidColorBrush
        Color='Orange'></SolidColorBrush><SolidColorBrush
        Color='Red'></SolidColorBrush>
    </x:Array>
  </syncfusion:ChartStyleModel.CustomPalette></syncfusion:ChartSeries>
```

C#

```
this.Series1.ColorEach = true; this.Series1.Palette =
  ChartColorPalette.Custom; this.Series1.CustomPalette = new
  Brush[] { Brushes.Violet, Brushes.Indigo, Brushes.Blue, Brushes.Green, Brushes.Yel
    low, Brushes.Orange, Brushes.Red };
```



Highlighting Series

Chart for WPF lets you to "highlight" all the data points in a series when you move the mouse over any one of the data points in the series, or over the legend items corresponding to a Chart Series. This is achieved by binding the `ChartSeries.Interior` to the `ChartSeries.Highlighted` dependency property, such that all the data point segments in the series will get "highlighted" when this property changes.

The following code example illustrates this.

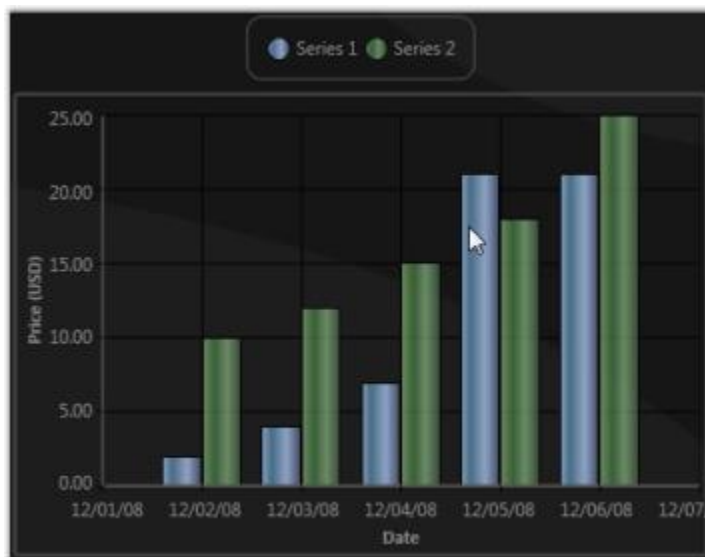
XML

```
<!--Include this in the Window's Resources section.-->
<local:HighlightedToOpacityConverter x:Key="myOpcConverter"/>
<sfchart:ChartSeries Name="series1" Label="Series 1" Type="Column"
DataSource="{StaticResource SeriesData1}" BindingPathX="Date"
BindingPathsY="Y1">
<sfchart:ChartSeries.Interior>
<!--Increasing the Opacity of the Interior when this series is highlighted.-->
<LinearGradientBrush EndPoint="0,0.5" StartPoint="1,0.5" Opacity="{Binding
ElementName=series1, Path=Highlighted,
Converter={StaticResource myOpcConverter}}">
<GradientStop Color="#FF434865" Offset="0"/>
<GradientStop Color="#FF7598BF" Offset="0.947"/>
<GradientStop Color="#FF496B82" Offset="0.75"/>
```

```
<GradientStop Color="#FF8DA4C7" Offset="0.365"/>
</LinearGradientBrush>
</sfchart:ChartSeries.Interior>
</sfchart:ChartSeries>
```

C#

```
public class HighlightedToOpacityConverter : IValueConverter
{
    #region IValueConverter Members
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        bool highlighted = (bool)value;
        if (highlighted)
            return 1;
        else
            return 0.65;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
    #endregion
}
```



A sample which demonstrates Series highlighting feature is available in the following sample installation path.

..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Series\Series Highlight Demo

Highlighting Data Points

Chart for WPF lets you highlight a single data point segment when you move the mouse pointer over the data point segment. The ChartSegment.Highlighted dependency property is used for this purpose. This

property is set to *true* when you move the mouse pointer over a data point segment. Also, you can create custom templates that utilize this dependency property to change the interior of the data point segment when the mouse pointer is moved it.

The following code example illustrates this.

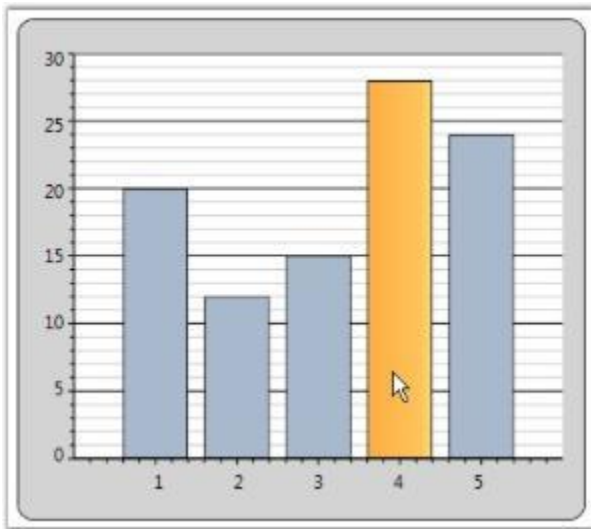
XML

```
<!--Include this in the Window's Resources section.-->
<!--Define the interior that will be used as the "highlight" color.-->
<LinearGradientBrush x:Key="MouseHoverInterior" StartPoint="0,0.5"
EndPoint="1,0.5">
  <GradientStop Color="#FFFDAE41" Offset="0"/>
  <GradientStop Color="#FFFD55C" Offset="0.860442"/>
  <GradientStop Color="#FFFDDDD77" Offset="0.989014"/>
  <GradientStop Color="#FFFDDDD77" Offset="1"/>
</LinearGradientBrush>
<!--Chart Series Data-->
<XmlDataProvider x:Key="myXmlData">
  <x:XData>
    <Products xmlns="">
      <Product Sales="20" Projected="30" Month="1"/>
      <Product Sales="12" Projected="28" Month="2"/>
      <Product Sales="15" Projected="29" Month="3"/>
      <Product Sales="28" Projected="33" Month="4"/>
      <Product Sales="24" Projected="30" Month="5"/>
    </Products>
  </x:XData>
</XmlDataProvider>
<!--Custom column template where the interior color of the data point
segment is changed when the mouse pointer is over it.
(ChartSegment.Highlighted property change)-->
<DataTemplate x:Key="ColumnTemplate">
  <Canvas>
    <Grid Canvas.Left="{Binding X}" Canvas.Top="{Binding Y}" Width="{Binding
Width}" Height="{Binding Height}">
      <Border Name="ColumnRect" VerticalAlignment="Bottom" Width="{Binding Width}"
Height="{Binding Height}"
      >
        <Border.Style>
          <Style TargetType="{x:Type Border}">
            <Setter Property="BorderThickness" Value="1"/>
            <Setter Property="BorderBrush" Value="Black"/>
            <Setter Property="Background" Value="{Binding Interior}"/>
          <Style.Triggers>
            <DataTrigger Value="True">
              <DataTrigger.Binding>
                <Binding Path="Highlighted"/>
              </DataTrigger.Binding>
              <Setter Property="Background" Value="{StaticResource MouseHoverInterior}"/>
            </DataTrigger>
          </Style.Triggers>
        </Style>
      </Border.Style>
    </Border>
  </Grid>
</Canvas>
```

```

</DataTemplate>
<!--Refer to the preceding template while defining a Chart Series.-->
<sfchart:ChartSeries Name="series1" Template="{StaticResource
ColumnTemplate}" Type="Column"
DataSource="{Binding Source={StaticResource myXmlData},
XPath=Products/Product}" BindingPathX="Month" BindingPathsY="Sales"
Stroke="Black" StrokeThickness="1.5" />

```



A sample which demonstrates Data Point highlighting feature is available in the following sample installation path.

..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Series\Data Point Highlight Demo

Selecting Points

Chart for WPF lets you implement list box-like selection of data point segments in your chart. The following steps illustrate this.

1. Bind the Chart to a CollectionViewSource

Wrap your data in a CollectionViewSource and bind this to a Chart Series.

XML

```

<!--Create a CollectionViewSource.-->
<local:MyDataCollection x:Key="SeriesData1"/>
<CollectionViewSource x:Key="cvs" Source="{StaticResource SeriesData1}" />
<!--Bind this to a Chart Series.-->
<sfchart:ChartSeries DataSource="{Binding Source={StaticResource cvs}}"
Template="{StaticResource SeriesTemplate}" Type="Column" BindingPathX="Date"
BindingPathsY="Y2" />

```

The CollectionViewSource has a CurrentItem property which tracks the "selected item". The Chart control listens to this property change and updates the corresponding data point's ChartSegment.IsSelected property appropriately.

2. Create a Custom Template

Create a custom template that renders a data-point segment with a "selected" look and feel, when the `ChartSegment.IsSelected` property changes to `true`.

XML

```
<!-- This template helps in 2 ways. 1) It enables to bind to IsSelected
property to change the selected segment color. 2) It enables to listen to
Canvas.MouseDown event to change selection.-->
<DataTemplate x:Key="SeriesTemplate">
  <!--Change the CollectionView.CurrentItem in the handler.-->
  <Canvas MouseDown="Canvas_MouseDown">
    <Grid Canvas.Left="{Binding X}" Canvas.Top="{Binding Y}" Width="{Binding
Width}" Height="{Binding Height}">
      <Border Name="ColumnRect" VerticalAlignment="Bottom" Width="{Binding Width}"
Height="{Binding Height}">
        <Border.Style>
          <Style TargetType="{x:Type Border}">
            <Setter Property="BorderThickness" Value="1"/>
            <Setter Property="BorderBrush" Value="Black"/>
            <Setter Property="Background" Value="{Binding Interior}"/>
          </Style>
          <Style.Triggers>
            <DataTrigger Value="True">
              <DataTrigger.Binding>
                <!-- By binding to IsSelected you can change the background of
"CollectionView.CurrentItem".-->
                <Binding Path="IsSelected"/>
              </DataTrigger.Binding>
              <Setter Property="Background" Value="{StaticResource MouseHoverInterior}"/>
            </DataTrigger>
          </Style.Triggers>
        </Border.Style>
      </Border>
    </Grid>
  </Canvas>
</DataTemplate>
<!--Defines the interior that will be used as the "highlight" color.-->
<LinearGradientBrush x:Key="MouseHoverInterior" StartPoint="0,0.5"
EndPoint="1,0.5">
  <GradientStop Color="#FFFDAE41" Offset="0"/>
  <GradientStop Color="#FFFDC55C" Offset="0.860442"/>
  <GradientStop Color="#FFFDDDD77" Offset="0.989014"/>
  <GradientStop Color="#FFFDDDD77" Offset="1"/>
</LinearGradientBrush>
```

This will cause the `CollectionView.CurrentItem` to be rendered distinctly. Note that the `CurrentItem` can be changed by a different Control bound to the same `CollectionView`, and this change will be automatically reflected in the Chart.

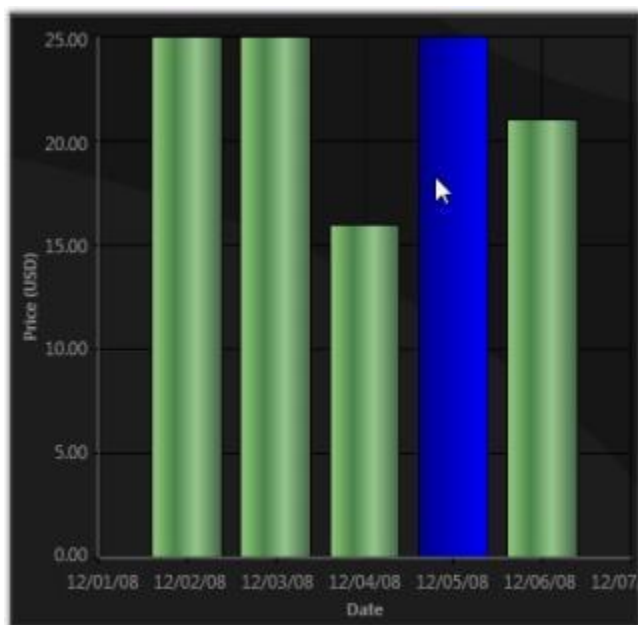
3. Change `CollectionView.CurrentItem`

Change it when any Mouse Button is pressed while the Mouse Pointer is moved over a Data Point Segment.

To change the CurrentItem when any mouse button is pressed while the mouse pointer is moved over a data point segment, listen to the MouseDown event of the top-level Canvas in the template, as illustrated in the following code.

CSHARP

```
private void Canvas_MouseDown(object sender, MouseButtonEventArgs e)
{
    // Get the corresponding Chart Segment.
    ChartSegment seg =
        ((ChartSeriesPresenter.ChartSegmentPresenter) ((Canvas) sender).TemplatedParent).Segment;
    // Get the corresponding bound CollectionView.
    CollectionView cv = seg.Series.DataSource as CollectionView;
    // Set the CurrentItem in the CollectionView.
    cv.MoveCurrentToPosition(seg.CorrespondingPoints[0].Index);
}
```



A sample which demonstrates Data Point highlighting feature is available in the following sample installation path.

..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Series\Selectable Data Points Demo

Side-By-Side Series

Series can be placed side by side or overlapped by using the ChartArea.SideBySideSeriesPlacement property. This is especially used when multiple HiLo type series are used in the Chart. HiLo type series that get stacked and plotted can be separated and placed side by side by using this property.

XML

```
<syncfusion:ChartArea SideBySideSeriesPlacement="True">
```

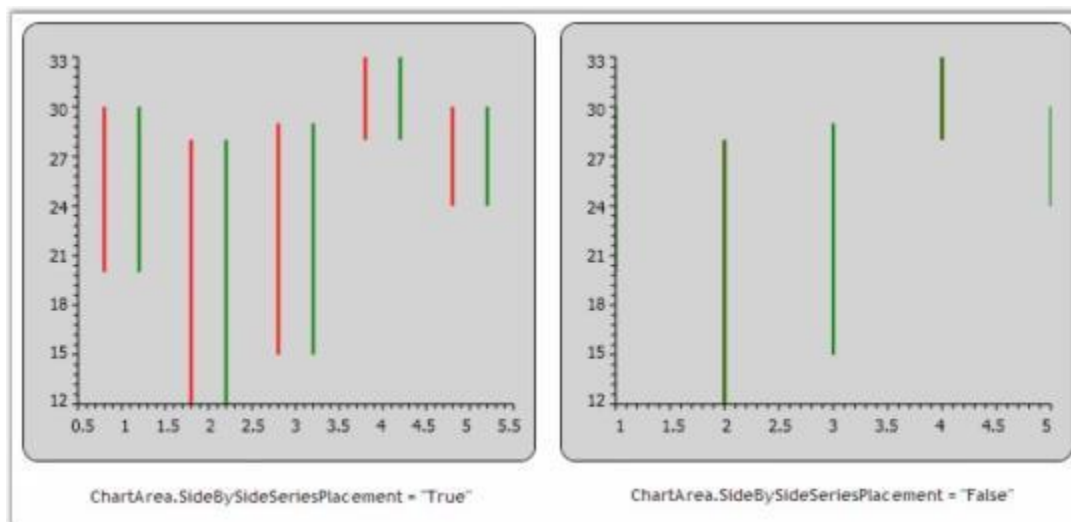


```
<syncfusion:ChartSeries Type="HiLo" />
</syncfusion:ChartArea>
```

C#

```
area.SideBySideSeriesPlacement = true;
```

The following screenshot illustrates Chart Series placed side by side.

**AutoDiscard Property**

AutoDiscard can be used to set range for primary axis in a chart. This property belongs to ChartSeries class. It can be set to three enumerations namely:

- None-The user can provide custom range values for primary axis, when the AutoDiscard property is set to *None*.
- ExtendRange-The range values will be automatically extended based on the difference between the start and end range values, when the AutoDiscard property is set to *ExtendRange*.
- ResetRange-The range values will be reset based on the interval, when set to *ResetRange*,

Note: The AutosetRange property of the primary axis needs to be set to False, when the AutoDiscard property is set to ExtendRange or ResetRange.

The following code example illustrates the usage of the AutoDiscard property, when set to various enumerations.

XML

```
<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis Header="Time" LabelForeground="LightGray"
Value Type="Double" IsAutoSetRange="False"/>
</syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartSeries Label="Measurement 1" Type="FastLine"
Interior="Orange" AutoDiscard="None" Range="0,10"/>
or
<syncfusion:ChartSeries Label="Measurement 1" Type="FastLine"
Interior="Orange" AutoDiscard="ExtendRange" />
```

```
or
<syncfusion:ChartSeries Label="Measurement 1" Type="FastLine"
Interior="Orange" AutoDiscard="ResetRange" />
```

Controlling the Visibility of Chart Legend Items

Essential Chart WPF now provides support to toggle the visibility of the Chart Legend Items. This is achieved by using the `VisibilityOnLegend` property.

Property	Description
VisibilityOnLegend	Sets the visibility of Legend Items. It includes the following options. Visible-Items in the Legend will be Visible. Hidden-Items in the Legend will be Hidden. Collapsed-Items in the Legend will be Collapsed.

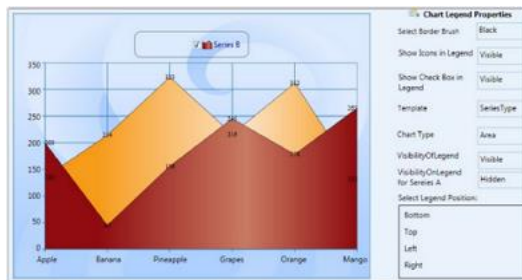
The following code example illustrates how to set this property.

XML

```
<syncfusion:ChartSeries Name="SeriesB" Type="Bar"
VisibilityOnLegend="Hidden" BindingPathX="FruitName"
BindingPathsY="FruitID,NumberOfFruits,Price,Year" Label="Series B"
Stroke="#FF000000" StrokeThickness="0.5" >
</syncfusion:ChartSeries>
```

C#

```
Chart1.Areas[0].Series[0].VisibilityOnLegend = Visibility.Hidden;
```



Methods

The `LegendItemSource` method associated with this feature can also be used to control the visibility of the Legend Items. The following code example illustrates how to use this method.

C#

```
public void LegendItemSource (ChartSeries chartSeries)
{
    ChartSeriesCollection collection = new ChartSeriesCollection();
    foreach (ChartSeries item in chartSeries.Area.Series)
    {
        if (item.IsVisibleOnLegend == true && item.VisibilityOnLegend !=
            Visibility.Collapsed)
        {
            collection.Add(item);
        }
    }
}
```

```

}
}
Legend.ItemsSource = collection;
}

```

Events

The OnVisibilityOnLegend event is triggered when the value of the VisibilityOnLegend property is changed. The following code example illustrates how to handle this event.

C#

```

public static readonly DependencyProperty VisibilityOnLegendProperty =
DependencyProperty.Register("VisibilityOnLegend", typeof(Visibility),
typeof(ChartSeries), new PropertyMetadata(Visibility.Visible, new
PropertyChangedCallback(OnVisibilityOnLegend)));
public Visibility VisibilityOnLegend
{
    get { return (Visibility)GetValue(VisibilityOnLegendProperty); }
    set { SetValue(VisibilityOnLegendProperty, value); }
}
private static void OnVisibilityOnLegend(DependencyObject d,
DependencyPropertyChangedEventArgs args)
{
    ChartSeries type = (ChartSeries)d;
    type.Area.LegendItemSource(type);
}

```

Creating Predefined Shapes for Annotation Objects

Predefined shapes for annotation objects are used to point at specific information about a point in the chart series. For example: Circle, Down arrow, and so on. The following table describes more about the annotation shapes:

Property

Name of the Property	Description	Values it accepts
AnnotationShape	Dependency Property	Enum of type AnnotationShapes
Fill	Dependency Property	Colors from Brushes

Note: The AnnotationShape property helps create the required shape for the annotation and the Fill property helps fill the shape selected with required color.

The following code example illustrates creation of a circle with orange fill at points 5 on X series and point 45 on Y series in a Chart.

XML

```

<syncfusion:ChartSeries DataSource="{Binding Source={StaticResource
myXmlData}, XPath=Products/Product}" BindingPathX="Month"
BindingPathsY="Sales" IsIndexed="False" Name="series1" Label="Series1"
Type="Area" Interior="{StaticResource SeriesAInterior}">
<syncfusion:ChartSeries.Annotations>
<syncfusion:AnnotationsCollection >

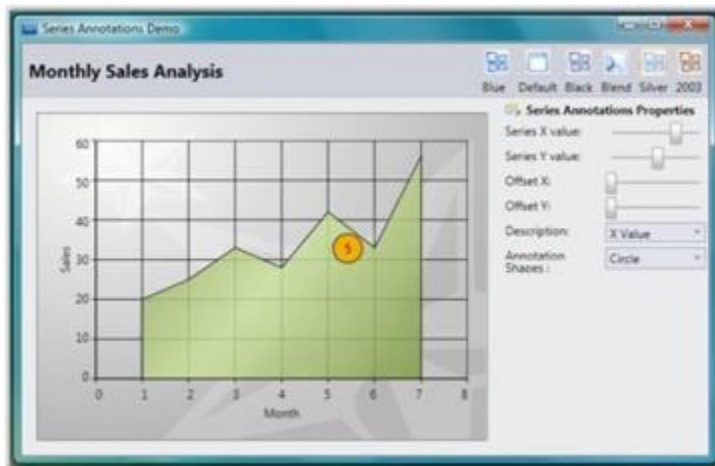
```

```
<syncfusion:ChartSeriesAnnotation x:Name="serAnnot" X="5" Y="45" OffsetX="0"
OffsetY="0" AnnotationShape="Circle"
Fill="Orange" Stroke="Black" />
</syncfusion:AnnotationsCollection>
</syncfusion:ChartSeries.Annotations>
</syncfusion:ChartSeries>
```

C#

```
Chart1.Areas[0].Series[0].Annotations.Items[0].AnnotationShape =
AnnotationShapes.Circle;
Chart1.Areas[0].Series[0].Annotations.Items[0].X = 5;
Chart1.Areas[0].Series[0].Annotations.Items[0].Y = 45;
Chart1.Areas[0].Series[0].Annotations.Items[0].OffsetX = 0;
Chart1.Areas[0].Series[0].Annotations.Items[0].OffsetY = 0;
Chart1.Areas[0].Series[0].Annotations.Items[0].Fill = Brushes.Orange;
Chart1.Areas[0].Series[0].Annotations.Items[0].Stroke = Brushes.Black;
```

Run the sample. The following output is provided.

**Empty point support for FastLine Chart type**

Essential chart WPF is now supports Empty point for Fast Line Chart type.

The data collection that is passed to the chart may have NaN values, this is an empty points.

If data points bounded with chart does not give any value then chart renders empty points in chart series.

This feature is useful when you are not able to get exact value for a particular data.

e.g. In population analysis if you do not get the result for previous years then we can use Empty data value.

Adding Empty Point

Add Empty Point to the Chart, by using the following code.

Set ShowEmptyPoints to True to enable Empty Point.

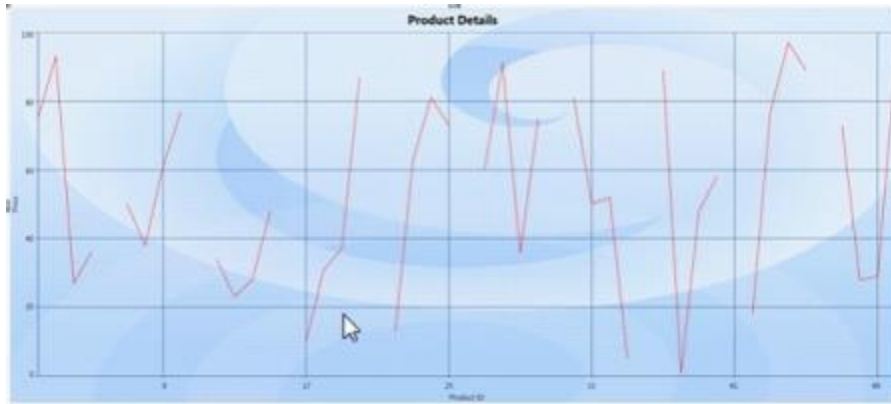
XML

```
<syncfusion:ChartSeries Name="series1" ShowEmptyPoints="True"
```

```
Type="FastLine" Interior="Red" Stroke="Black" DataSource="{Binding}"/>
```

C#

```
Series1.ShowEmptyPoints = true;
```



Customization support for FastChart types

This enables the users to customize the Fast chart types like FastScatter, FastColumn, FastStackingColumn, and FastHiLoOpenClose. Using this feature, users can customize the Stroke, Stroke thickness, and interior of each chart segment of the series.

Adding Customization Support

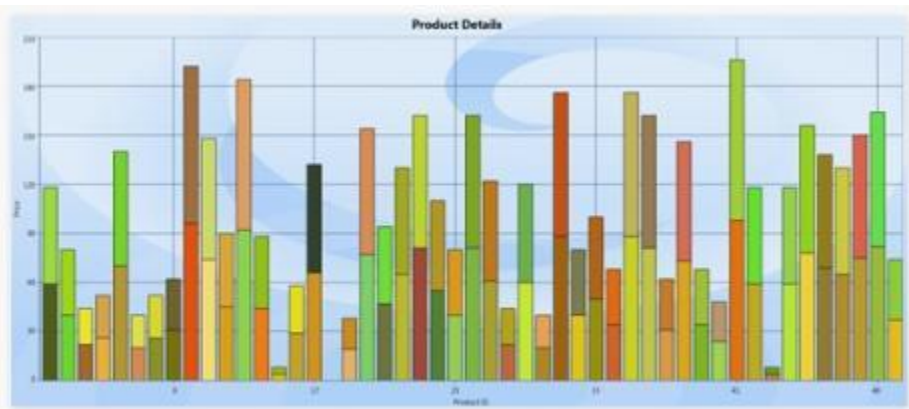
Add customization support for FastChart types, by using the following code.

XML

```
<syncfusion:ChartSeries Name="series1" Type="FastStackingColumn"
FastSegmentProperties="{Binding Converter={StaticResource interiorConverter}}
"
Stroke="Black" DataSource="{Binding}"/>
```

C#

```
FastSegmentPropertiesCollection list = new
FastSegmentPropertiesCollection();
FastSegmentProperties segmentProperty = new FastSegmentProperties {
Stroke=Brushes.Black,
StrokeThickness=1, Interior = brush };
list.Add(segmentProperty);
series1.FastSegmentProperties= list;
```



Smart Labels Support

Essential Chart ships with the enhancement of smart labels support to avoid the overlapping of adornment labels using the enum property `AdornmentIntersectAction`. The following actions can be taken when labels overlap:

- Hide
- AdjustAroundPoints
- None

Advantage of Using Smart Labels

1. Avoids the overlap of segment labels.
2. To view the label clearly and place the labels around the data points.
3. Additional connector lines are shown between a label and its corresponding chart point.

Properties

Property	Description	Type	Data Type
ShowSmartLabels	Sets the smart labels for the series.	Dependency Property	Boolean
AdornmentIntersectAction	Sets the intersect action for the adornments.	Dependency Property	Enum

Sample Link

1. Open the WPF sample browser.
2. Select the Chart control from the sample browser.
3. Chart > Smart Labels > Smart Label Demo.

Adding Smart Labels Support to an Application

The following code examples are used to add smart labels to the chart series.

XML

```
<sync:ChartSeries x:Name="series2"
AdornmentIntersectAction="AdjustAcrossPoints">
```

```

ShowSmartLabels="True"
<sync:ChartSeries.AdornmentsInfo>
<sync:ChartAdornmentInfo x:Name="adorn" Visible="True" Symbol="Square"
SymbolHeight="20" SymbolWidth="20"
SymbolInterior="LightBlue" />
</sync:ChartSeries.AdornmentsInfo>
</sync:ChartSeries>

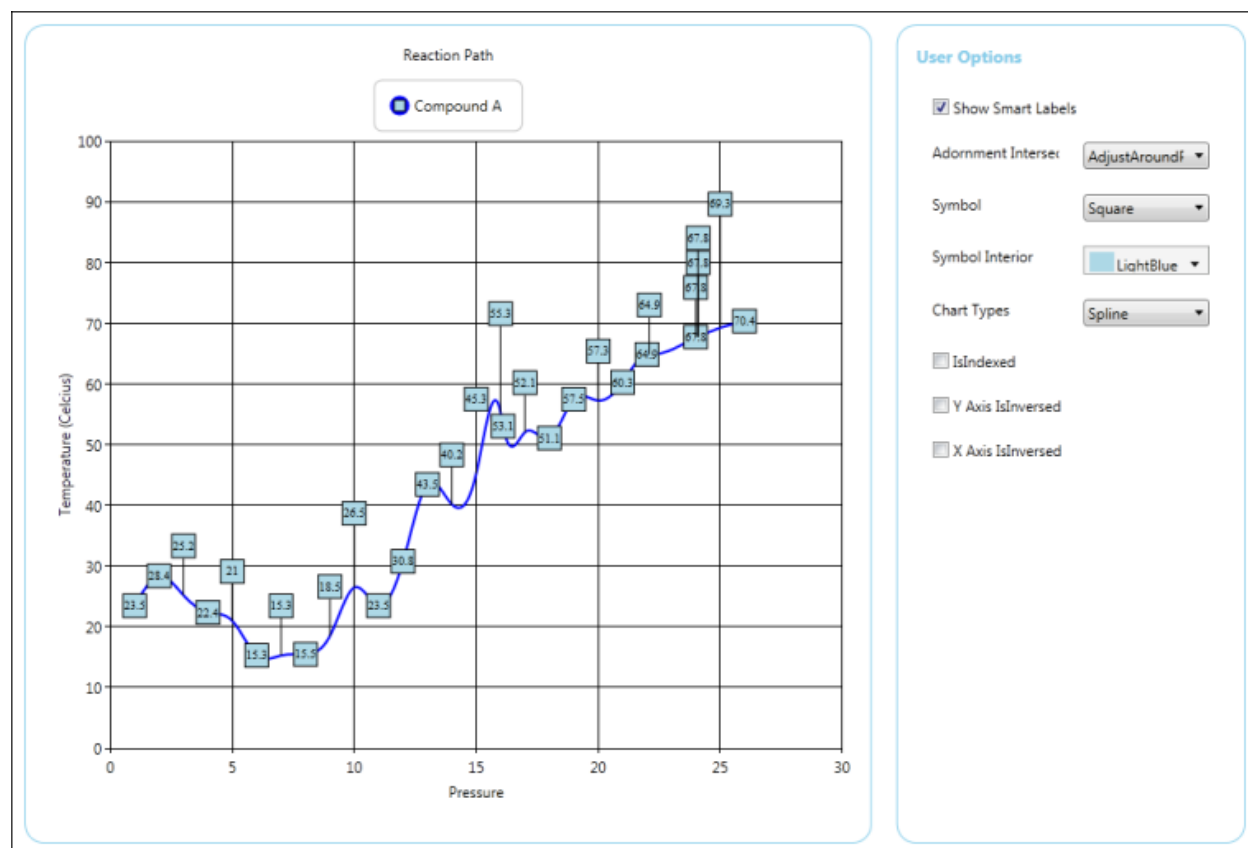
```

C#

```

series.ShowSmartLabels = true;
series.AdornmentIntersectAction =
AdornemntIntersectActions.AdjustAcrossPoints;

```



The following code examples are used to add smart labels to the chart series with intersect action set to Hide.

XML

```

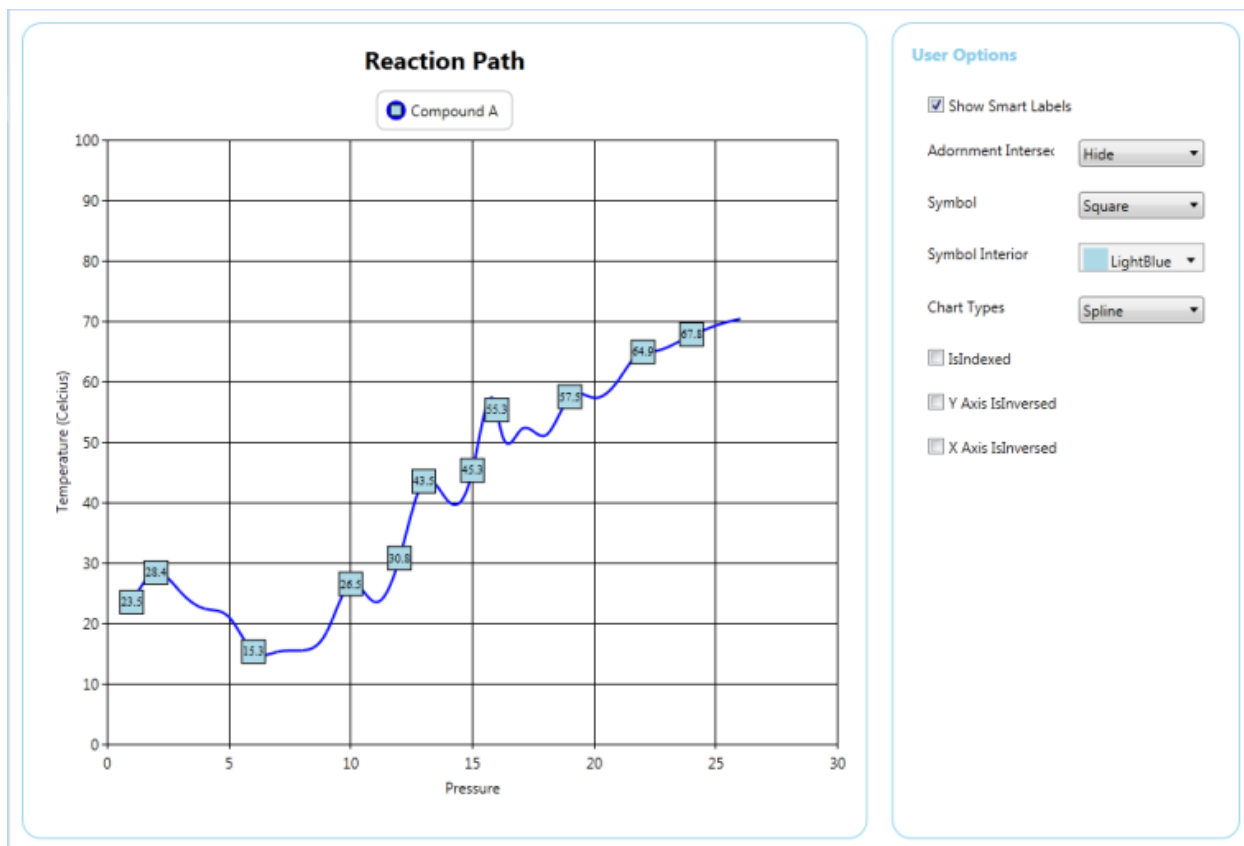
<sync:ChartSeries x:Name="series2"
AdornmentIntersectAction="Hide"
ShowSmartLabels="True"
<sync:ChartSeries.AdornmentsInfo>
<sync:ChartAdornmentInfo x:Name="adorn" Visible="True" Symbol="Square"
SymbolHeight="20" SymbolWidth="20"
SymbolInterior="LightBlue" />
</sync:ChartSeries.AdornmentsInfo>

```

```
</sync:ChartSeries>
```

C#

```
series.ShowSmartLabels = true;
series.AdornmentIntersectAction =
AdornemntIntersectActions.AdjustAcrossPoints;
```



The following code examples are used to add smart labels to the chart series with intersect action set to Hide.

XML

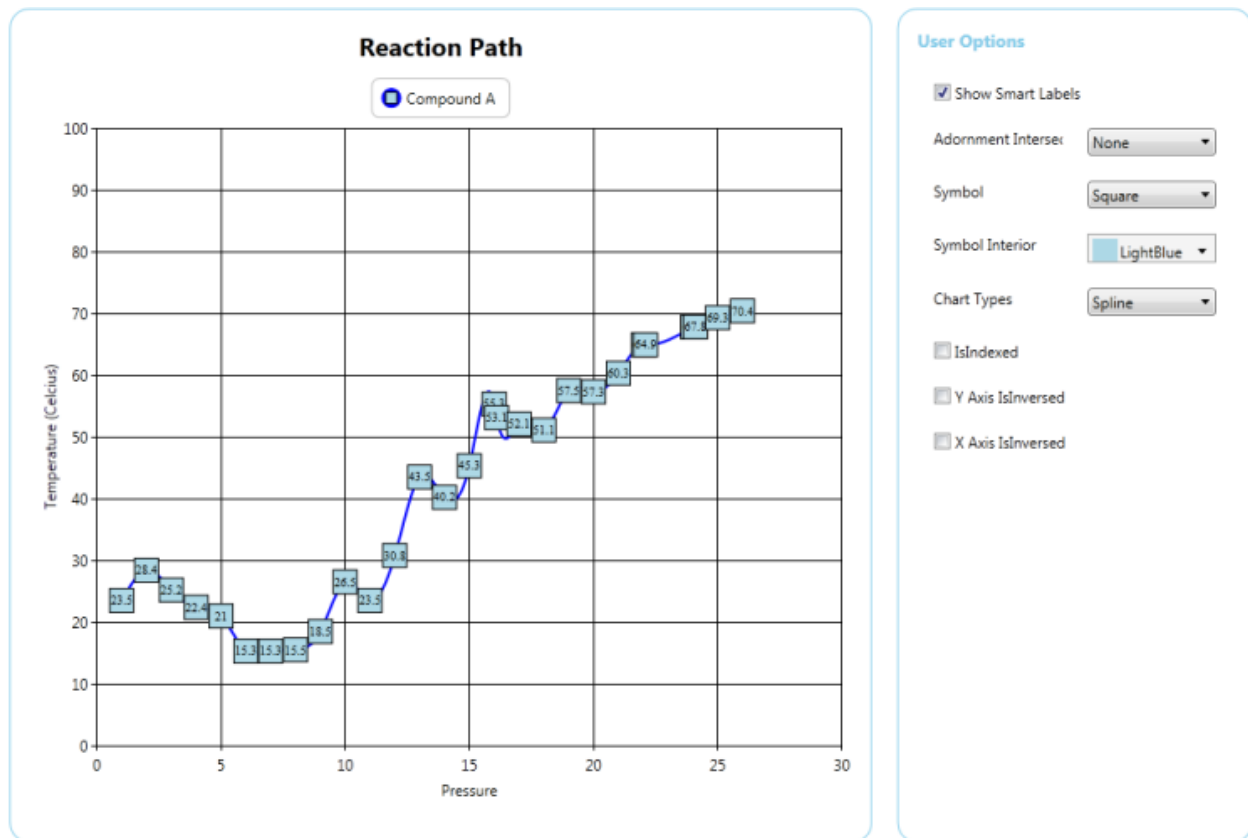
```
<sync:ChartSeries x:Name="series2"
AdornmentIntersectAction="None"
ShowSmartLabels="True"
>
<sync:ChartSeries.AdornmentsInfo>
<sync:ChartAdornmentInfo x:Name="adorn" Visible="True" Symbol="Square"
SymbolHeight="20" SymbolWidth="20"
SymbolInterior="LightBlue" />
</sync:ChartSeries.AdornmentsInfo>
</sync:ChartSeries>
```

C#

```
series.ShowSmartLabels = true;
series.AdornmentIntersectAction =
```



```
AdornemntIntersectActions.None;
```



Technical Indicators

Overview

Technical Indicators are used to predict future price levels by analyzing a set of recorded data. They are used to improve the data analysis capabilities of a chart with regard to information on pricing, volume, and other metrics used in business calculations.

Essential Chart supports the following Technical Indicators:

- Accumulation Distribution
- Average True Range
- Bollinger Band
- Exponential Moving Average
- Simple Average
- Triangular Average
- MACD
- Momentum
- Stochastic
- RSI

A technical indicator is a graphic representation of price action. Some technical indicators overlay on the trading chart, and some reside on the bottom of the chart.

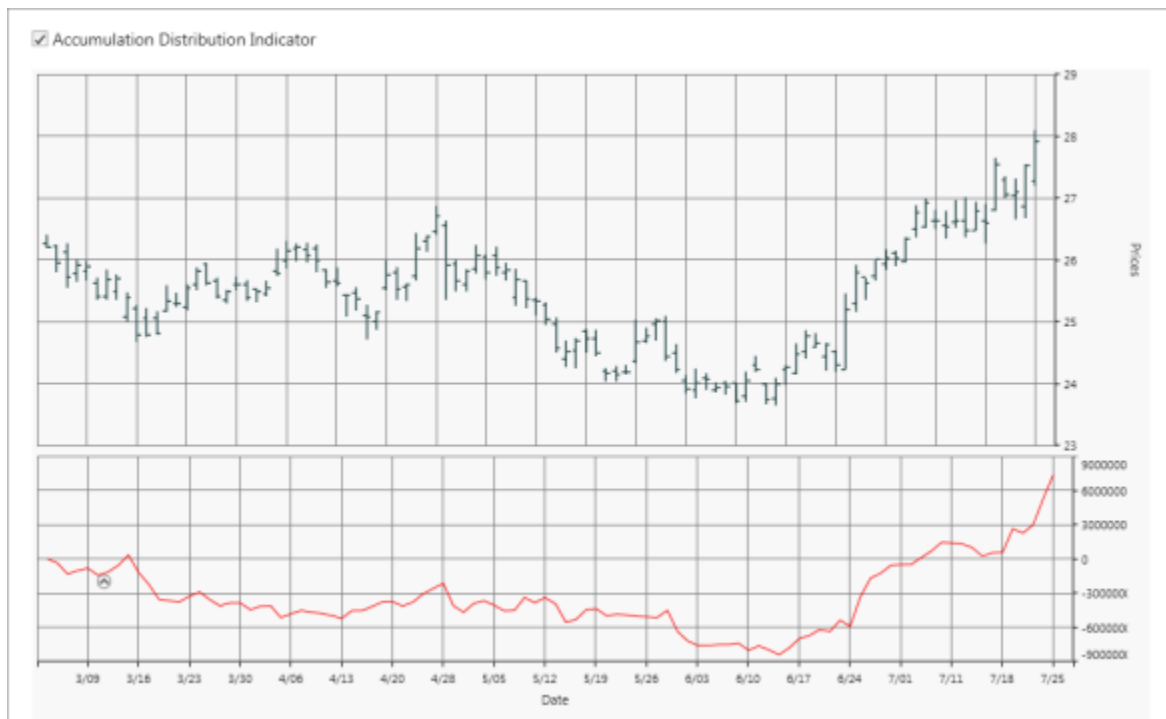
Accumulation Distribution

Accumulation Distribution Indicator is a volume indicator which was essentially designed to measure underlying supply and demand. It accomplishes this by trying to determine whether traders are actually accumulating (buying) or distributing (selling).

Property	Description
SignalLineColor	Specifies the color for the signal line of the Accumulation Distribution technical indicator.

XML

```
<sync:ChartSeries.Indicators>
<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="AccumulationDistribution"
sync:ChartAccumulationDistribution.SignalLineColor="Red">
</sync:ChartTechnicalIndicator>
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>
```



Average True Range

Average True Range (ATR) Indicator is a technical analysis volatility indicator. The indicator does not provide an indication of price trend, simply the degree of price volatility. The average true range is an N-day exponential moving average of the true range values.

XML

```
<sync:ChartSeries.Indicators>
```

```

<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="AverageTrueRange" >
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>

```

Bollinger Band

Bollinger Bands consist of a band of three lines which are plotted in relation to security prices. The line in the middle is usually a Simple Moving Average (SMA) set to a period of 20 days (The type of trend line and period can be changed by the trader; however a 20 day moving average is by far the most popular). The SMA then serves as a base for the Upper and Lower Bands. The Upper and Lower Bands are used as a way to measure volatility by observing the relationship between the Bands and price. Typically the Upper and Lower Bands are set to two standard deviations away from the SMA (The Middle Line); however the number of standard deviations can also be adjusted by the trader.

Bollinger Band Properties

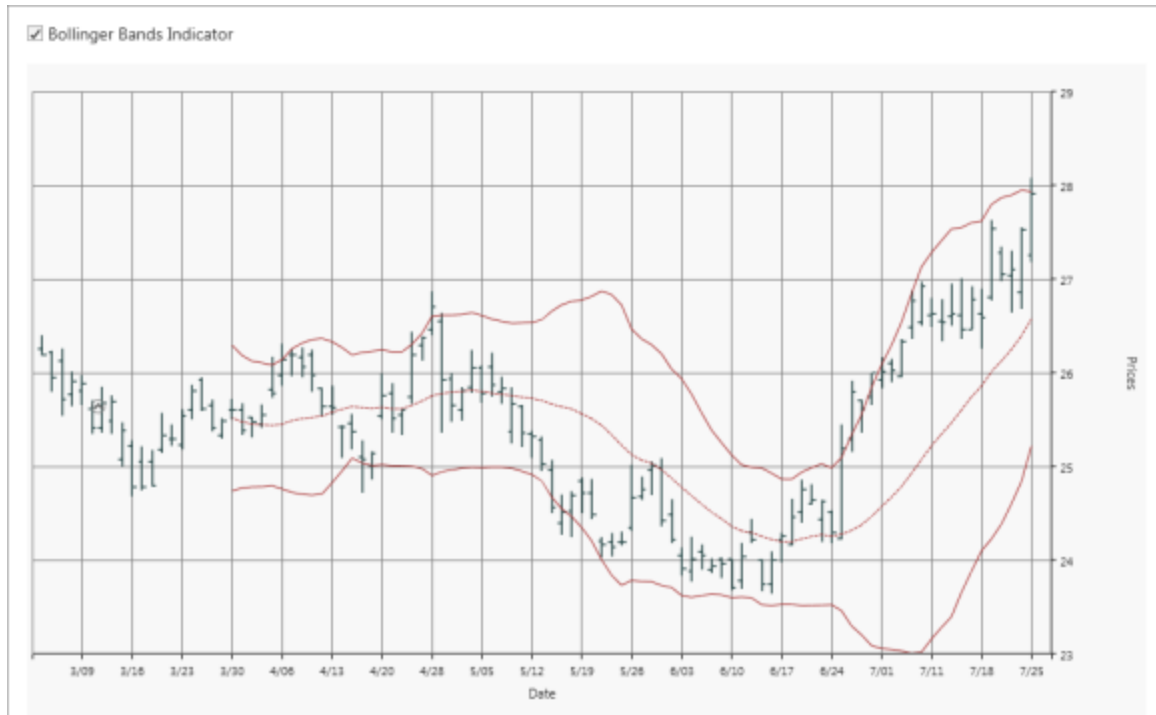
Property	Description
SignalLineColor	Specifies the color for the signal line of the Bollinger Band technical indicator.
UpperLineColor	Specifies the color for the upper line of the Bollinger Band technical indicator.
LowerLineColor	Specifies the color for the lower line of the Bollinger Band technical indicator.
BollingerMovingAverage	Indicates the identification of a trend.

XML

```

<sync:ChartSeries.Indicators>
<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="BollingerBands"
sync:ChartBollingerBand.LowerLineColor ="Blue"
sync:ChartBollingerBand.UpperLineColor="Red"
sync:ChartBollingerBand.SignalLineColor ="Green"
sync:ChartBollingerBand.BollingerMovingAverage="50" >
</sync:ChartTechnicalIndicator>
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>

```



Exponential Moving Average

The exponential moving average (EMA) is a weighted average of a price data which put a higher weight on recent data point.

Property	Description
SignalLineInterior	Specifies the color for the signal line of the Exponential Moving Average technical indicator.Â
ExponentialMovingAverage	Indicates the identification of a trend.

XML

```

<sync:ChartSeries.Indicators>
<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="ExponentialAverage"
sync:ChartExponentialAverage.ExponentialAverage="50"
sync:ChartExponentialAverage.SignalLineInterior="Red" >
</sync:ChartTechnicalIndicator>
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>

```

Simple Average

The simple moving average is an arithmetic mean of price data. It is calculated by summing up each interval's price and dividing the sum by the number of intervals covered by the moving average. For instance, adding the closing prices of an instrument for the most recent 25 days, and then dividing it by 25 will get you the 25 day moving average.

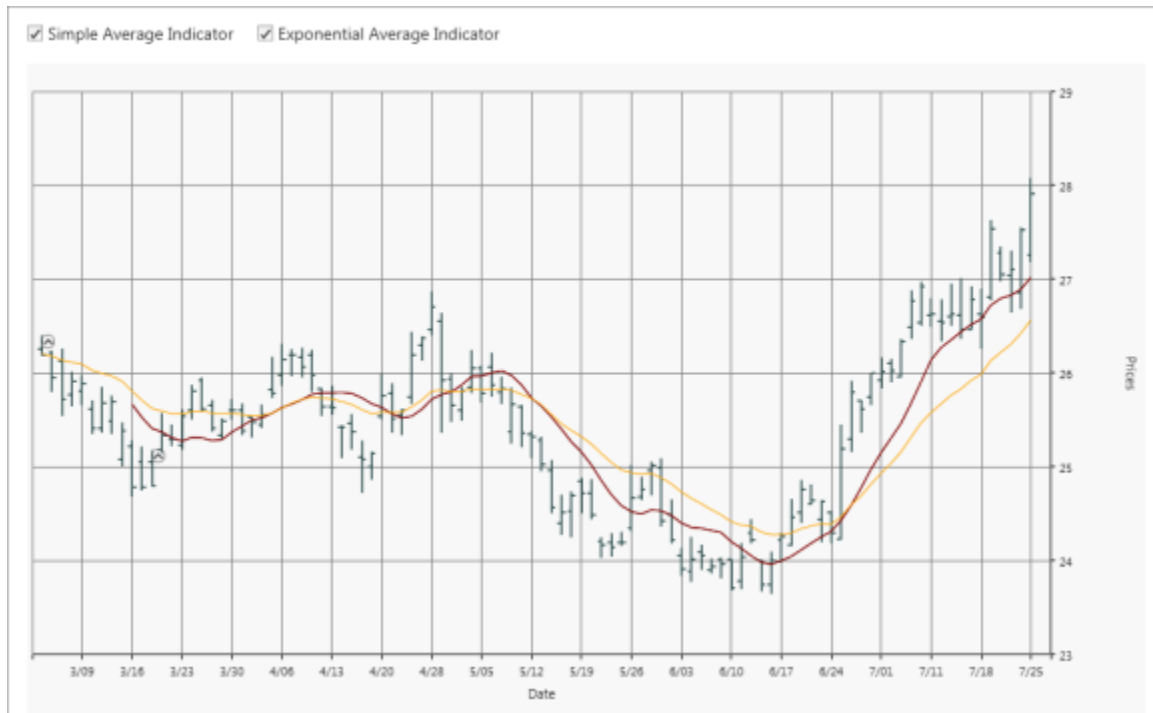
Property	Description
SignalLineInterior	Specifies the color for the signal line of the Simple Average technical indicator.
MovingAverage	Indicates the identification of a trend.

XML

```

<sync:ChartSeries.Indicators>
<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="TriangularAverage"
sync:ChartSimpleAverage.MovingAverage="20"
sync:ChartSimpleAverage.SignalLineInterior="Red" />
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>

```

**Triangular Average**

The Moving Average Triangular indicator measures a simple arithmetic average of prices, specified by the input Price and creates a simple arithmetic average of this average. The length of every of these averages is one more than half the value specified in the input Length, rounded to a whole number. This uses all the price data from the latest number of bars specified by the input Length, but with the smoothing effect of so-called averaging the average.

Property	Description
SignalLineInterior	Specifies the color for the signal line of the Triangular Average technical indicator.

TriangularMovingAverage	Indicates the identification of a trend.
-------------------------	--

XML

```

<sync:ChartSeries.Indicators>
<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="TriangularAverage"
sync:ChartTriangularAverage.TriangularAverage="20"
sync:ChartTriangularAverage.SignalLineInterior="Red" />
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>

```

MACD

MACD is an extremely popular indicator used in technical analysis. MACD can be used to identify aspects of a security's overall trend. Most notably these aspects are momentum, as well as trend direction and duration.

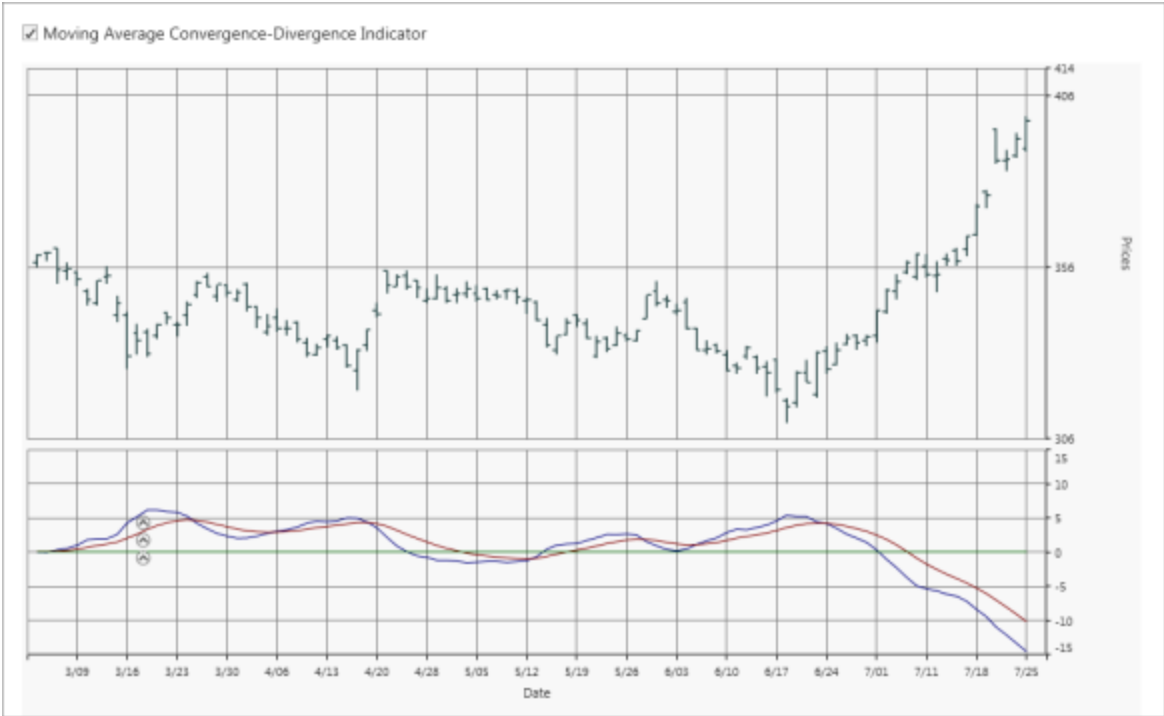
Property	Description
SignalLineInterior	Specifies the color for the signal line of the MACD technical indicator.
ConvergenceLineColor	Specifies the color for the convergence line of the MACD technical indicator.
DivergenceLineColor	Specifies the color for the divergence line of the MACD technical indicator.

XML

```

<sync:ChartSeries.Indicators>
<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="MACD"
sync:ChartMACD.SignalLineInterior="Red"
sync:ChartMACD.ConvergenceLineColor="Yellow"
sync:ChartMACD.DivergenceLineColor="Blue" />
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>

```



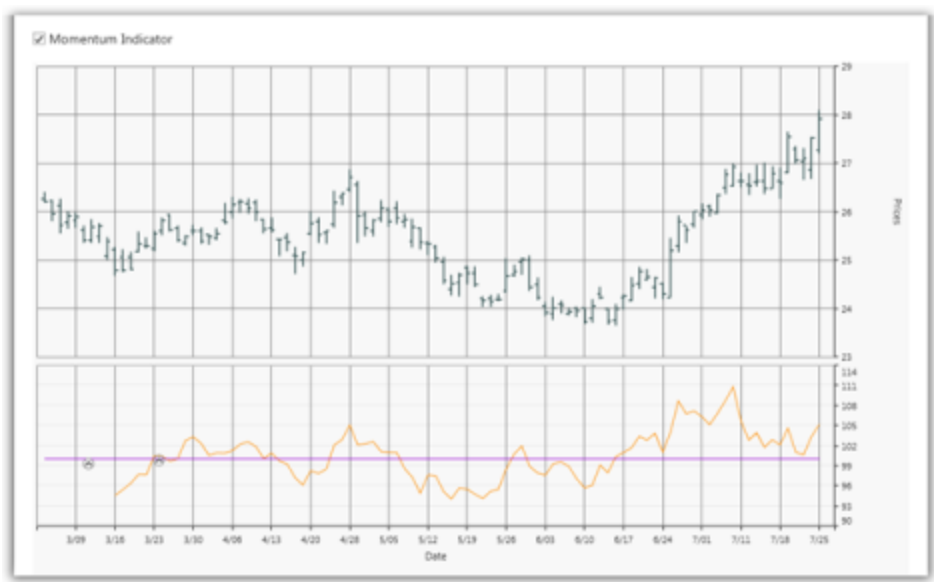
Momentum

Momentum and rate of change (ROC) are simple technical analysis indicators showing the difference between today's closing price and the close N days ago.

Property	Description
MomentumTimeSpan	Measures the amount that a security's price has changed over a given time span.

XML

```
<sync:ChartSeries.Indicators>
<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="Momentum"
sync:ChartMomentum.MomentumTimeSpan="10" />
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>
```



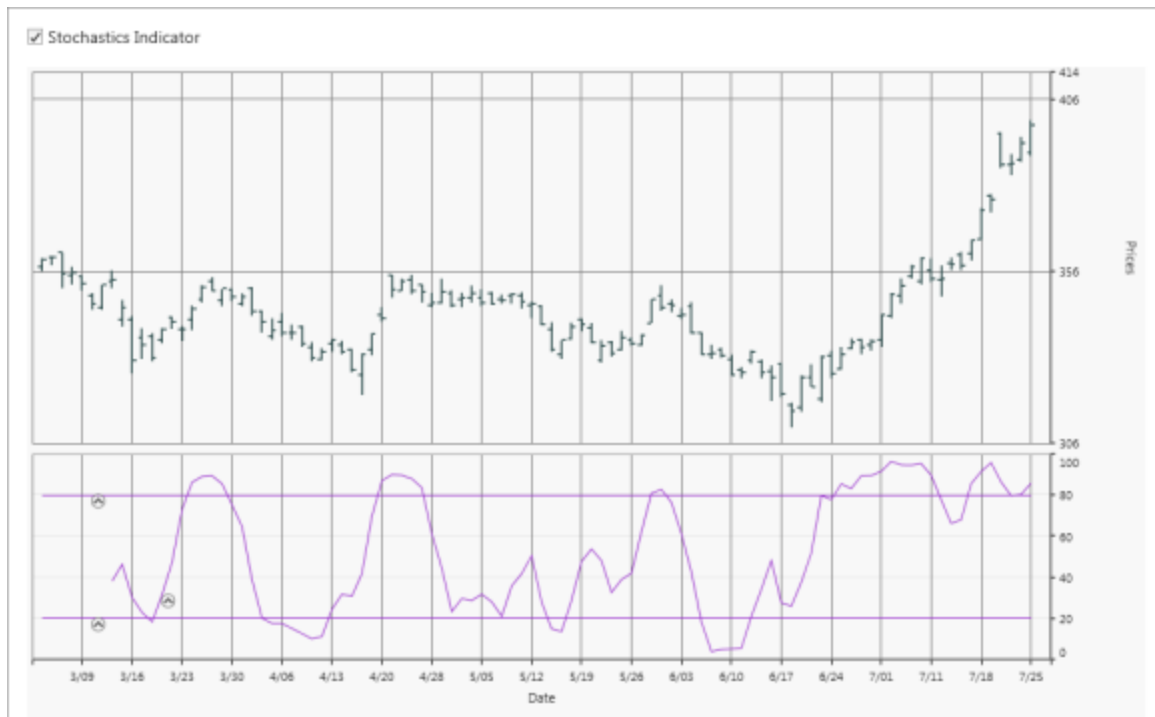
Stochastic

The Stochastic Oscillator is a range bound momentum oscillator. The Stochastic indicator is designed to display the location of the close compared to the high or low range over a user-defined number of periods. Typically, the Stochastic Oscillator is used for three things: Identifying overbought and oversold levels, spotting divergences, and also identifying bull and bear set ups or signals.

Property	Description
SignalLineColor	Specifies the color for the signal line of the Stochastic technical indicator.
UpperLineColor	Specifies the color for the upper line of the Stochastic technical indicator.
LowerLineColor	Specifies the color for the lower line of the Stochastic technical indicator.

XML

```
<sync:ChartSeries.Indicators>
<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="Stochastics"
sync:ChartStochastics.LowerLineColor="Red"
sync:ChartStochastics.SignalLineColor="Green"
sync:ChartStochastics.UpperLineColor="Blue" />
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>
```

RSI

The Relative Strength Index (RSI) is a well-versed momentum-based oscillator which is used to measure the speed (velocity) as well as the change (magnitude) of directional price movements. Essentially RSI, when graphed, provides a visual mean to monitor both the current, as well as historical, strength and weakness of a particular market.

XML

```
<sync:ChartSeries.Indicators>
<sync:IndicatorCollection>
<sync:IndicatorCollection.Items>
<sync:ChartTechnicalIndicator IndicatorType="RelativeStrengthIndex" />
</sync:IndicatorCollection.Items>
</sync:IndicatorCollection>
</sync:ChartSeries.Indicators>
```

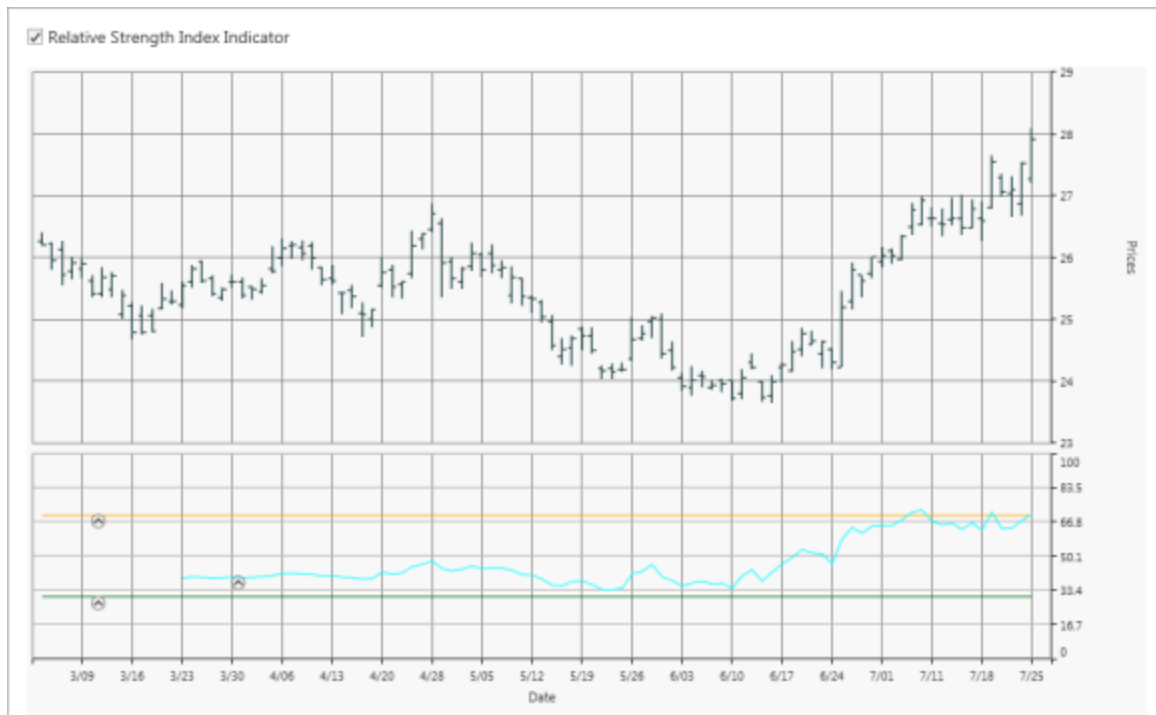


Chart-Types in WPF Chart (Classic)

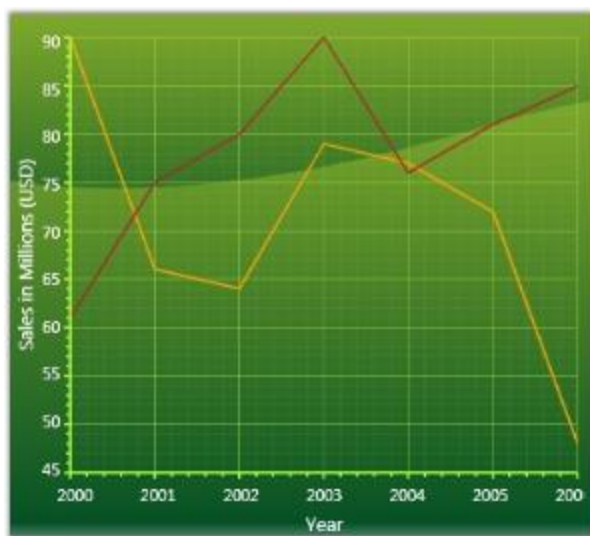
Line Charts

Line Chart

Line Charts join points on a plot using straight lines showing trends in data at equal intervals. Line charts treats the input as non-numeric, categorical information, equally spaced along the x-axis. This is appropriate for categorical data, such as text labels, but can produce unexpected results when the x values consist of numbers.

When rendered in 3D the plot looks like a ribbon and hence such types are also referred to as Ribbon or Strip Charts.

The appearance of the lines and the points can be configured with options such as the colors used, thickness of the lines and the symbols displayed.



Data Requirements

The following are the details about Line Chart:

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Template

While setting template the following parameters can be used.

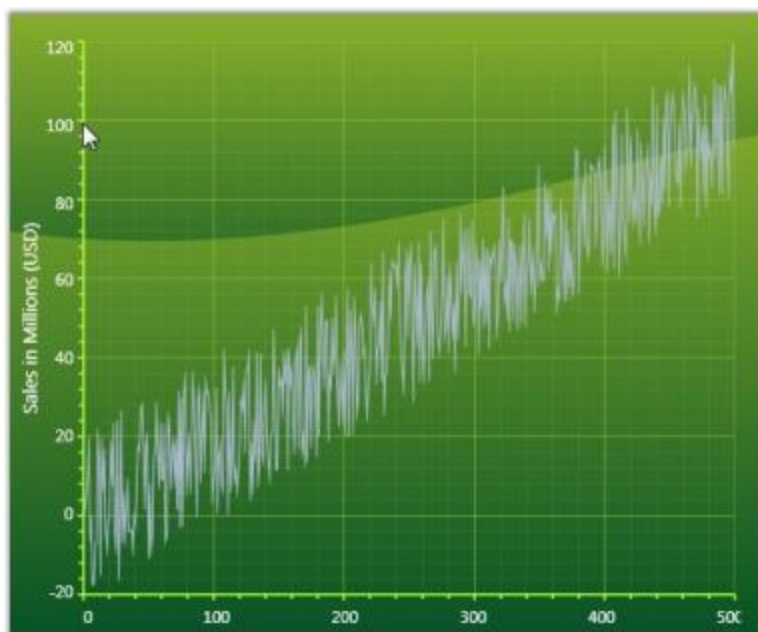
Name	Type	Description
X1	double	x-coordinate of first point
Y1	double	y-coordinate of first point
X2	double	x-coordinate of second point
Y2	double	y-coordinate of second point
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Line Chart Types is available in the following sample installation path.

..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Line Chart Demo

Fast Line Chart

Use a Fast Line chart instead of a Line chart when displaying a large number of data points in the chart. This chart type improves performance by foregoing some features in the Line chart.



Data Requirements

The following are the details about Fast Line Chart:

Details	
Number of y values per point	One
Number of points	one or more
Number of series	one or more

Template

While setting template the following parameters can be used:

Name	Type	Description
X1	Double	x-coordinate of first point
Y1	Double	y-coordinate of first point
X2	Double	x-coordinate of second point
Y2	Double	y-coordinate of second point
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Line Chart Types is available in the following sample installation path.

..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Line Chart Demo

Customization of FastLine

Essential Chart enables you to customize the look and feel of the Fastline chart.

Following types of lines are supported for FastLine chart:

- Dash line
- Dot line
- Dash-dot line
- Dash-dot-dot line

Property

Property	Description	Type	Data Type	Reference links
Pen	Gets and sets various types of pen for drawing fastchat type.	Attached property for ChartFastSeriesPresenter	System.Drawing.Pen	Pen Class

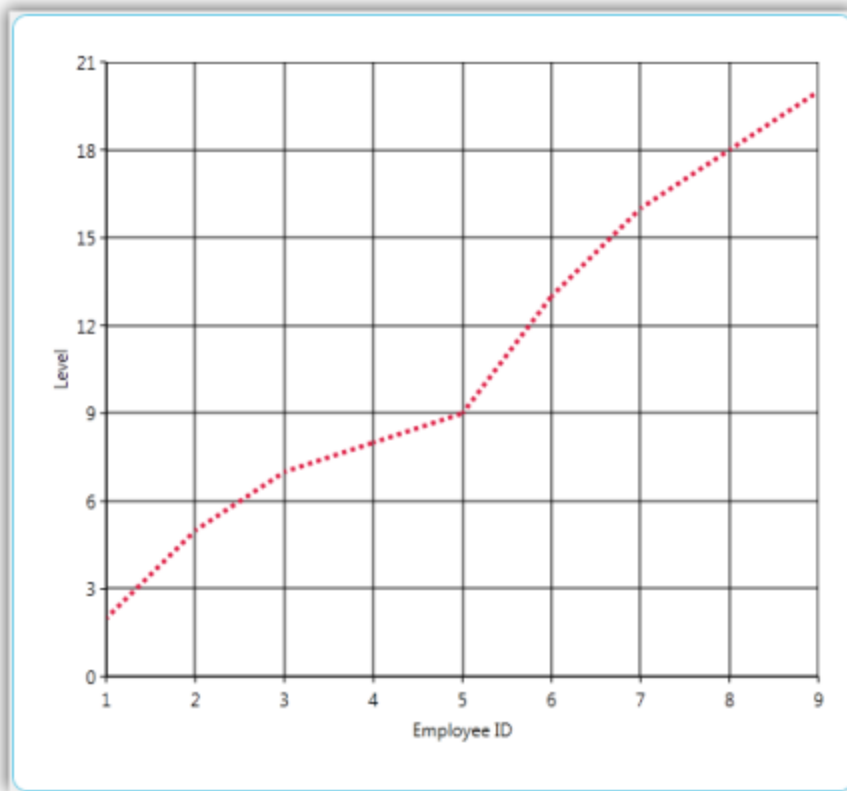
Customizing Fastline

To customize the FastLine chart, use the Pen property of ChartFastSeriesPresenter.

The following code illustrates this:

XML

```
<syncfusion:ChartSeries Name="series" Type="FastLine" IsIndexed="True"
Interior="LightBlue" >
<syncfusion:ChartFastSeriesPresenter.Pen>
<Pen Brush="Black" DashCap="Round" DashStyle="DashDot" StartLineCap="Round"
EndLineCap="Round"/>
</syncfusion:ChartFastSeriesPresenter.Pen>
</syncfusion:ChartSeries>
```



[Sample Link](#)

To view a sample

1. Open the Syncfusion Dashboard.
2. Select User Interface.
3. Click the WPF drop-down list and select Explore Samples.

4. Navigate to

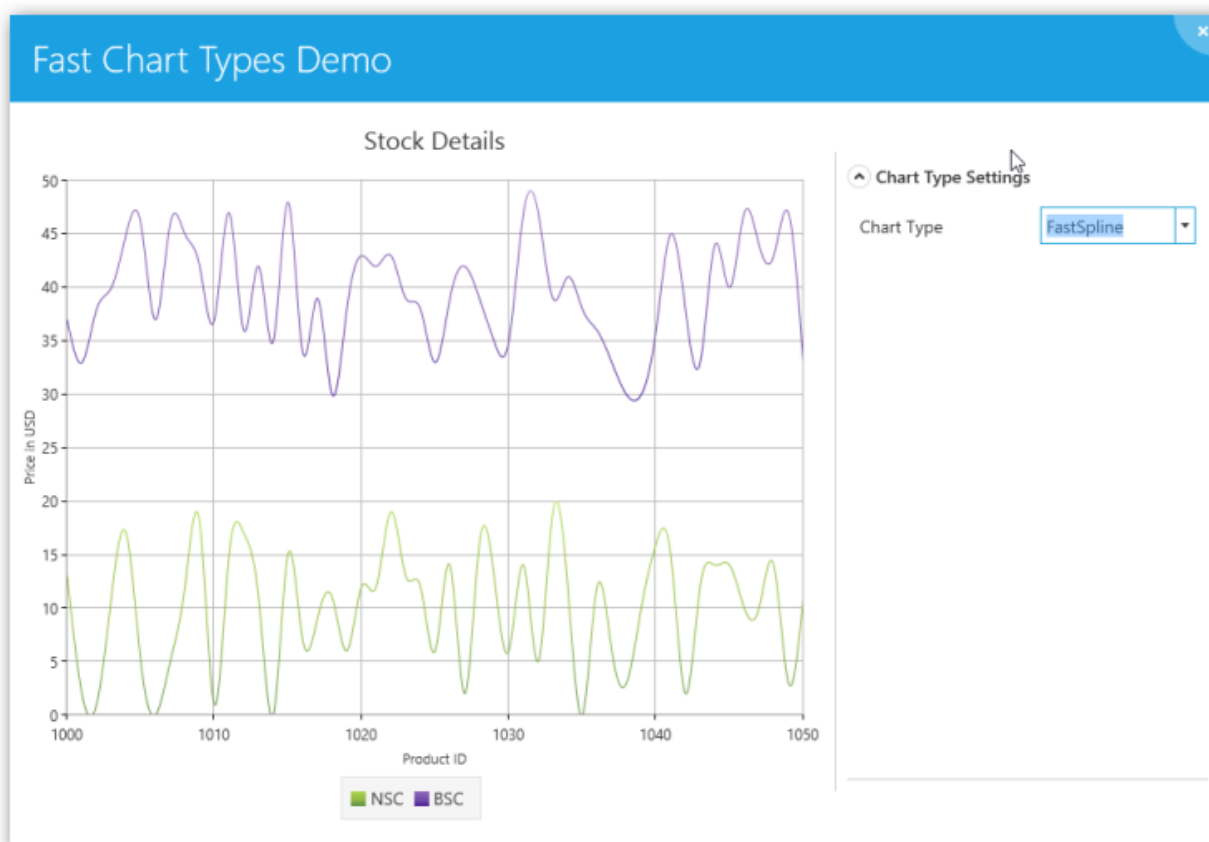


Chart.WPF\Samples\3.5\WindowsSamples\Chart Customization\ FastLine Customization Demo

FastSpline Chart Series

FastSpline is used to render a large number of data points as a smooth line in a fast manner and helps in improving the performance of the chart. Its view is similar to the Line chart, but plots a fitted curve through each data point in series. FastSpline draws a smooth flowing line through all data points in the data set.

The appearance of the FastSpline can be configured with options such as interior, thickness of the Lines

Chart Type Characteristic

The following table lists the main characteristic of the FastSpline chart.

Details	
Number of Y values per point	One.
Number of Series	One or more.
Cannot be Combined with	Pie, Bar, Stacked Bar, Polar, Radar.

The following screenshot depicts a FastSpline chart:

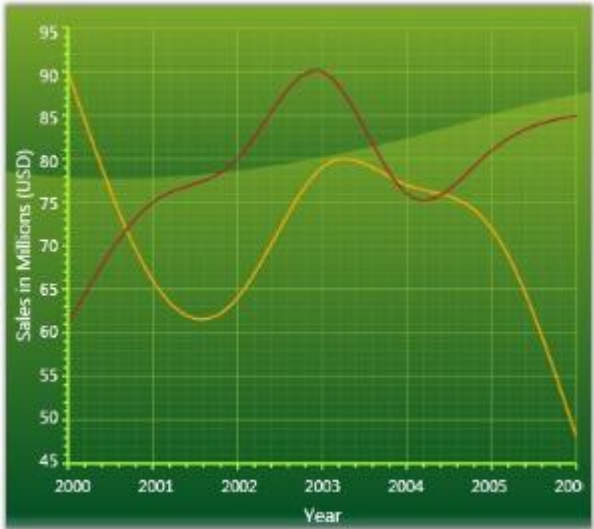
![[Chart-Controlsimg70]](Chart-Controlsimages/Chart-Controls_img70.png)

Use Case Scenarios

FastSpline chart is useful in the following fields where high volume data is used.

- Marketing
- Scientific Research
- Economics

Properties

Property	Description
Geometry	 <p>Draws the spline when set to paths <i>Data</i> property.</p>

Template for FastSpline

The following code example illustrates the template for the FastSpline chart:

XML

```
<DataTemplate x:key="{x:Type local:ChartFastSplineSegment}">
  <Grid>
    <Path Stroke="{Binding Interior}" StrokeThickness="{Binding
      StrokeThickness}" Data="{Binding Geometry}" ToolTip="{Binding ToolTip}"/>
    <local:FastSplinePresenter Points="{Binding Points}" Series="{Binding
      Series}" ToolTip="{Binding ToolTip}" />
  </Grid>
</DataTemplate>
```

Adding FastSpline Chart to an Application

To add FastSpline chart to an application do the following:

1. Create a new WPF application in VS2012.
2. Create a chart sample with ChartArea and ChartAxis.
3. To get the FastSpline chart, specify the Type property in ChartSeries as "FastSpline".

Code Example

The following code example illustrates the usage of FastSpline charts.

XML

```

<sfchart:Chart>
  <sfchart:ChartArea Background="LightGray" GridBackground="White">
    <sfchart:ChartSeries Template="{StaticResource Template1}"
      DataSource="{Binding Source={StaticResource
myXmlData},XPath=Products/Product}" BindingPathX="Month"
      BindingPathsY="Sales" Interior="Red" Type="FastSpline"/>
  </sfchart:ChartArea>
</sfchart:Chart>

```

Spline Chart

Spline Chart is similar to a Line Chart except that it connects the different data points using splines instead of straight lines.

When rendered in 3D the plot looks like a ribbon and hence such types are also referred to as Ribbon or Strip Charts.

The appearance of the lines and the points can be configured with options such as the colors used, thickness of the lines and the symbols displayed.

![[Chart-Controlsimg71](Chart-Controlsimages/Chart-Controls_img71.jpeg)]

Data Requirements

The following are the details about Spline Chart:

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Spline Settings

Name	Type	Container	Description
ChartSplineType.SplineCoefficient	double	ChartSeries	attached property which lets you control the spline curvature

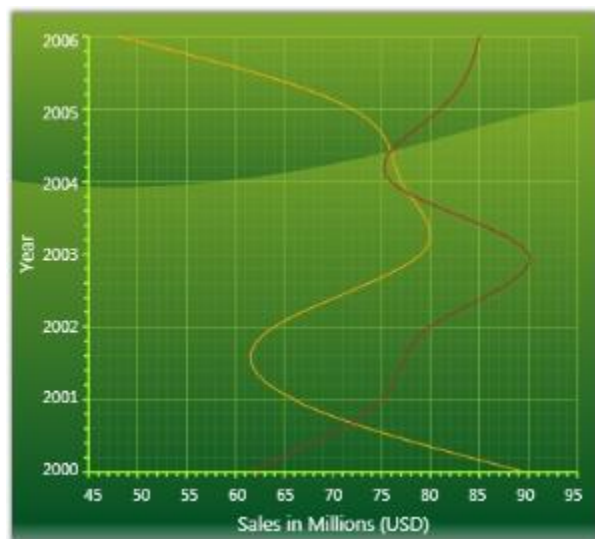
Template

While setting template the following parameters can be used.

Name	Type	Description
X1	double	x-coordinate of first point
Y1	double	y-coordinate of first point
X2	double	x-coordinate of second point

Y2	double	y-coordinate of second point
Geometry	Geometry	segment geometry
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Line Chart Types is available in the following sample installation path.



..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Line Chart Demo

Rotated Spline Chart

A Rotated Spline Chart is similar to a Spline Chart. The only difference is that it would be rotated. It plots one or several series of data and joins each series by smooth, rotated spline curves instead of straight lines.

The following image shows a sample Rotated Spline Chart.

![[Chart-Controlsimg72](Chart-Controlsimages/Chart-Controls_img72.jpeg)]

Data Requirements

The following are the details about Rotated Spline Chart:

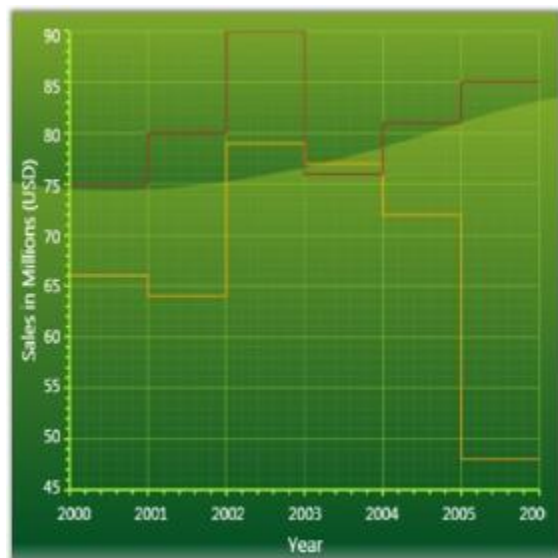
Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Rotated Spline Settings

Name	Type	Container	Description
------	------	-----------	-------------

ChartSplineType.SplineCoefficient	double	ChartSeries	attached property which lets you control the spline curvature
-----------------------------------	--------	-------------	---

A sample which demonstrates Line Chart Types is available in the following sample installation path.



..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Line Chart Demo

Step Line Chart

Step Line Charts use horizontal and vertical lines to connect data points resulting in a step like progression.

![[Chart-Controlsimg73]](Chart-Controlsimages/Chart-Controls_img73.jpeg)

Data Requirements

The following are the details about Step Line Chart:

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

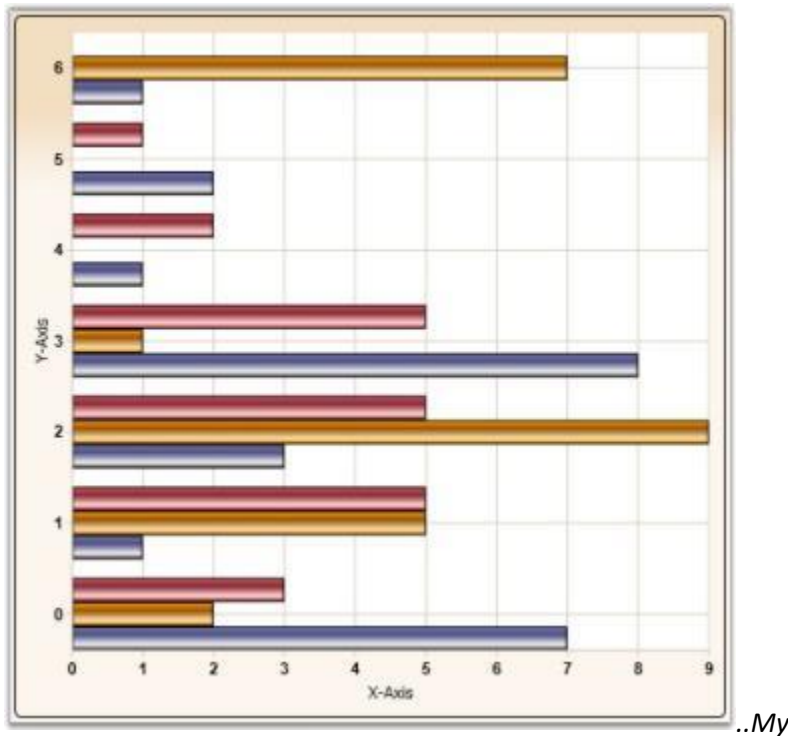
Template

While setting template the following parameters can be used:

Name	Type	Description
X1	double	x-coordinate of first point
Y1	double	y-coordinate of first point
X2	double	x-coordinate of second point

Y2	double	y-coordinate of second point
StepX	double	x-coordinate of transient point
StepY	double	y-coordinate of transient point
Points	PointCollection	collection of segment points
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Line Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Line Chart Demo

Bar Charts

Bar Chart

Bar Chart is the simplest and most versatile of statistical diagrams. It displays horizontal bars for each point in the series and points from adjacent series are drawn as bars next to each other. It is also available with a 3-D visual effect.

Bar Charts can be used to compare values across categories, for showing the variations in the value of an item over time or for showing the values of several items at a single point in time.

Another good reason to use bar charts is when you realize that the number of a data series fits better in a horizontal format. If you have long gaps between different values and you also have many items to compare, the bar chart type is the best one to use.

The following image shows a multi series Bar Chart.

![[Chart-Controlsimg74]](Chart-Controlsimages/Chart-Controls_img74.jpeg)

Data Requirements

The following are the details about Bar Chart:

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Bar Chart Settings

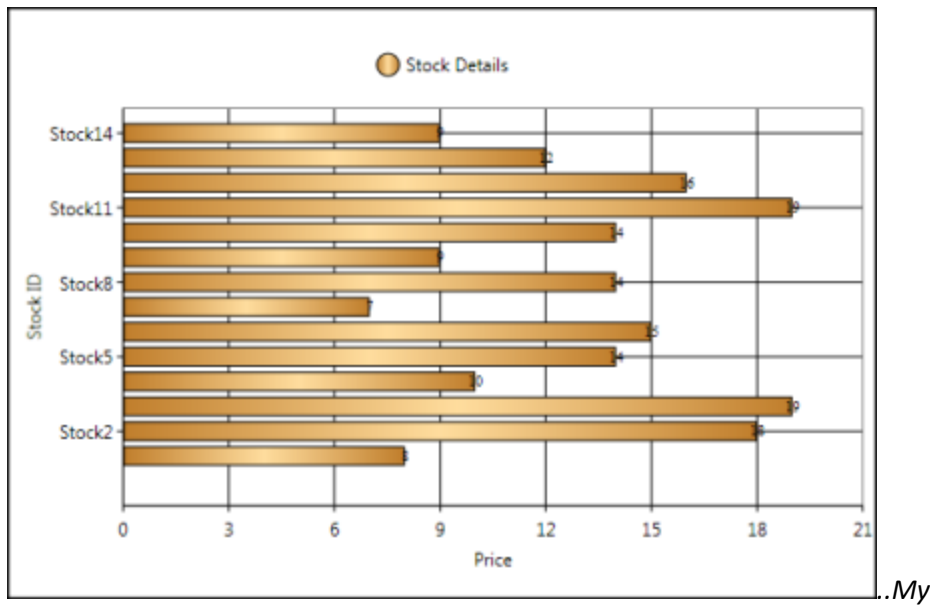
Name	Type	Container	Description
ChartType.Spacing	double	ChartArea	Attached property that specifies the space between columnsPossible values lie between 0 and 1. Default value is 0.2.

Template

While setting template the following parameters can be used.

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Bar Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Bar Chart Demo

FastBar ChartType

FastBar Chart is similar to Bar chart as it uses horizontal bars to display different values of one or more items. It is inherited from FastColumn ChartType with rotated Series, to achieve better performance over Bar ChartType.

It is used for comparing frequencies, counts, total and average of data in different categories. It is ideal for showing the variations in the value of an item over time.

The following points give the advantages of FastBar over Bar charts:

- They load faster than the Bar charts.
- They ensure high performance for displaying data.
- They can be used as real time charts to render a huge number of data points.

Use Case Scenarios

FastBar ChartType can be used in Stock Market where large amounts of data need to be rendered. It allows users to add large numbers of DataPoints thereby delivering a better performance.

Adding FastBar ChartType to an Application

To add FastBar ChartType to the application:

1. Select FastBar from the enum of type ChartTypes.
2. Assign it to the Type property in the ChartSeries. This can be added using xaml and also using C# code as given in the following code examples.

XML

```
<syncfusion:ChartSeries Type="FastBar"/>
Areal.Series[0].Type = ChartTypes.FastBar;
```

![[Chart-Controlsimages75]](Chart-Controlsimages/Chart-Controls_img75.png)

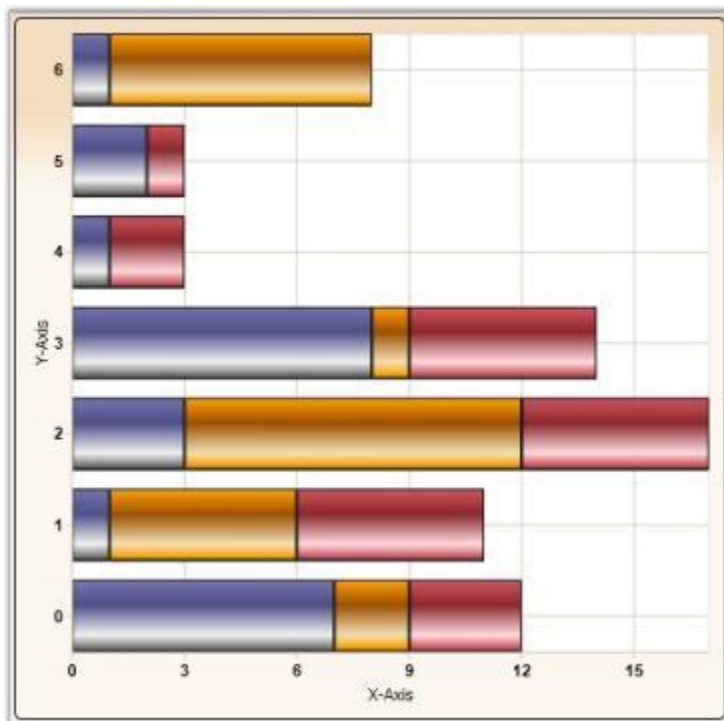
[Sample Link](#)

To run the UI WPF sample

1. Open Essential Studio Dashboard by selecting Start -> Program -> Syncfusion-> Essential Studio <-> Dashboard.
2. Select Run Locally Installed Samples, from the WPF drop-down list on the User Interface pane.
3. Select Chart in the sample browser.
4. Select ChartPerformance->FastChartTypesDemo on the Essential Chart pane and click the Run Sample button.

To open the sample project

5. Go to the following sample location in your system:



“<sample installation

location>\Syncfusion\EssentialStudio\Version Number
\WPF\Chart.WPF\Samples\3.5\WindowsSamples\ChartPerformance\FastChartTypesDemo”

6. This location contains two sub folders CS and VB. You can open the sample projects from the respective folders based on your application development language.

Stacking Bar Chart

Stacking Bar Charts are similar to regular bar charts except that the Y values stack on top of each other in the specified series order. This helps visualize the relationship of parts to the whole.

The following image shows a sample Stacking Bar Chart.

![[Chart-Controlsimages76]](Chart-Controlsimages/Chart-Controls_img76.jpeg)

Data Requirements

The following are the details about Stacking Bar Chart:

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Stacking Bar Chart Properties

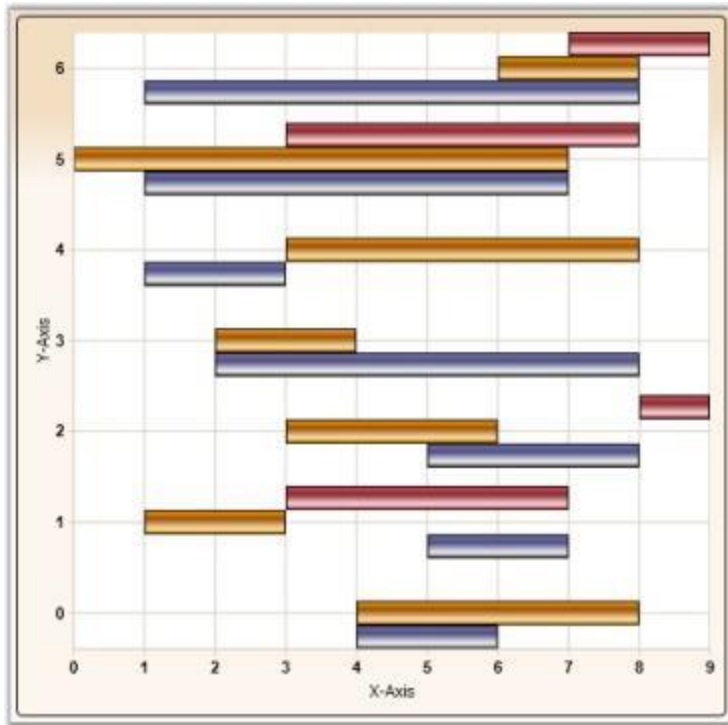
Name	Type	Container	Description
ChartType.Spacing	double	ChartArea	Attached property that specifies the space between columnsPossible values lie between 0 and 1. Default value is 0.2.

Template

While setting template the following parameters can be used:

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
IsUpper	bool	true “ if this is upper column
IsLower	bool	true “ if this is lower column
Series	ChartSeries	reference to series-owner

A sample which demonstrates Bar Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version
Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Bar Chart Demo

Gantt Chart

Frequently used in project management, a Gantt chart provides a graphical illustration of a schedule to help plan, coordinate, and track specific tasks in a project.

![[Chart-Controlsimg77]](Chart-Controlsimages/Chart-Controls_img77.jpeg)

Data Requirements

The following are the details about Gantt Chart:

Details	
Number of y values per point	two
Number of points	one or more
Number of series	one or more

Gantt Chart Settings

Gantt Chart Setting

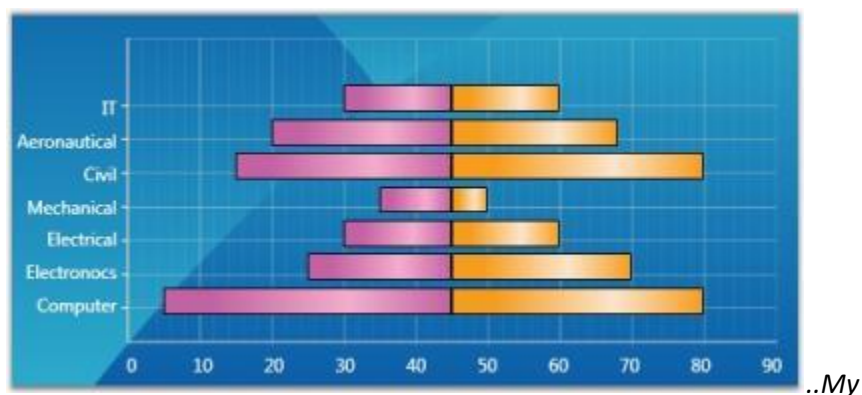
Name	Type	Container	Description
ChartType.Spacing	double	ChartArea	attached property that specifies the space between columnsPossible values lie between 0 and 1. Default value is 0.2.

Template

While setting template the following parameters can be used.

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Gantt Chart Type is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Gantt Chart Demo

Tornado Chart

The Tornado chart is a bar chart which shows the variability of an output to several different inputs. Variability is displayed using relative lengths of bars across a range. It is mainly used in sensitivity analysis. It shows how different random factors can influence the prognostic outcome.

![[Chart-Controlsimg78](Chart-Controlsimages/Chart-Controls_img78.jpeg)]

Data Requirements

The following are the details about Tornado Chart:

Details	
Number of y values per point	two
Number of points	one or more
Number of series	one or more

Tornado Settings

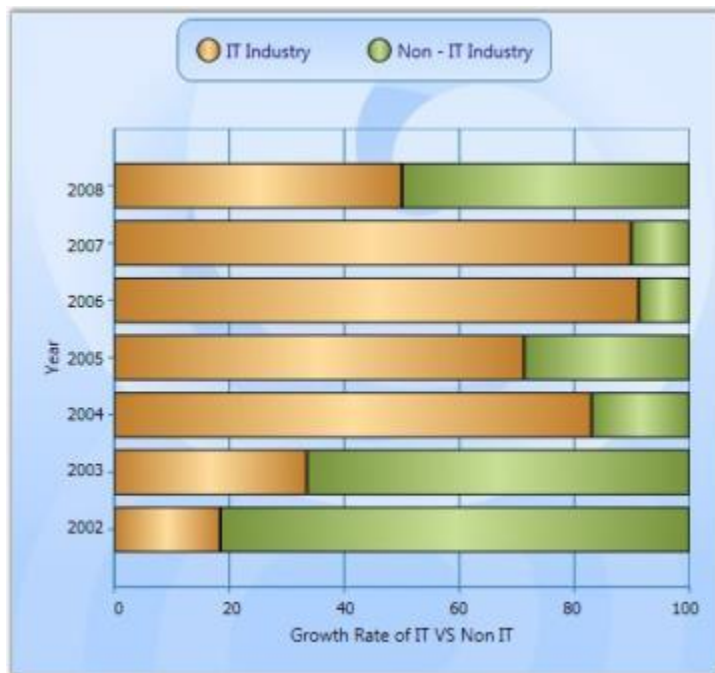
Name	Type	Container	Description
ChartType.Spacing	double	ChartArea	attached property that specifies the space between columnsPossible values lie between 0 and 1. Default value is 0.2.

Template

While setting template the following parameters can be used:

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Tornado Chart Type is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Tornado Chart Demo

Stacked Bar 100 Chart

In the 100 percent Stacked Bar Chart, the cumulative proportion of each stacked element always totals 100 percent. This type of chart is great to visualize the relative contribution of each series values to the whole.

The following image shows a sample Stacking Bar 100 Chart.

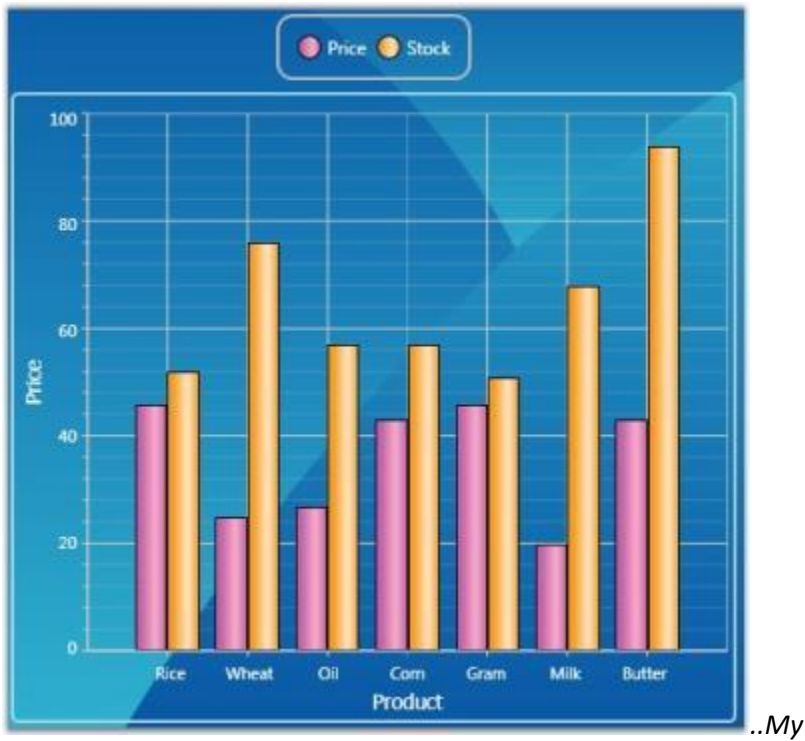
![Chart-Controlsimg79](Chart-Controlsimages/Chart-Controls_img79.jpeg)

Data Requirements

The following are the details about Stacking Bar 100 Chart:

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

A sample which demonstrates Bar Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Bar Chart Demo

Column Charts

Column Chart

Column Charts are among the most common chart types that are being used. It uses vertical bars (called columns) to display different values of one or more items. It is similar to a bar chart except that here the bars are vertical and not horizontal. Points from adjacent series are drawn as bars next to each other.

It is used for comparing the frequency, count, total or average of data in different categories. It is ideal for showing the variations in the value of an item over time.

The following image shows a multi series Column Chart.

![[Chart-Controlsimg80]](Chart-Controlsimages/Chart-Controls_img80.jpeg)

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Custom Properties

Custom Property

Name	Type	Container	Description
ChartType.Spacing	double	ChartArea	sets the interval between columnsPossible values lie between 0 and 1.

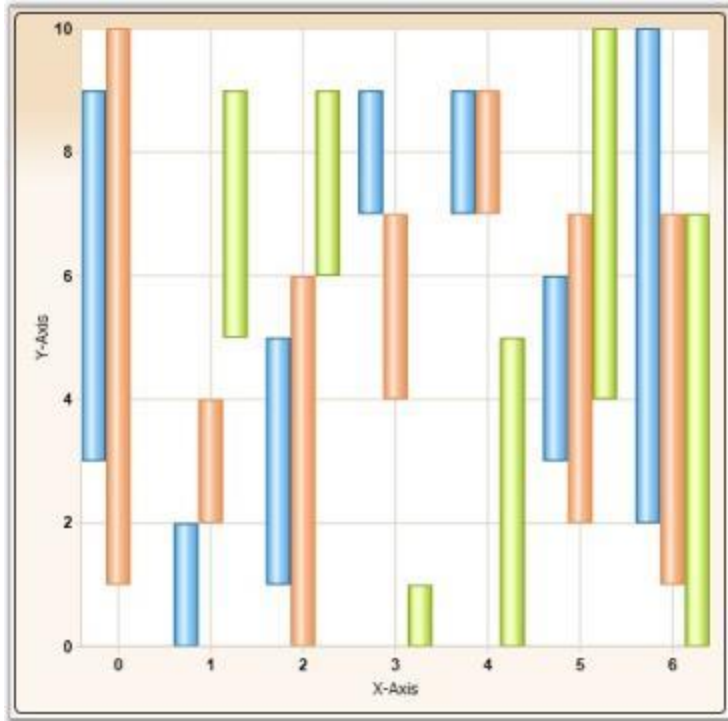
Template

While setting template the following parameters can be used:

Template Parameter

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Column Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Column Chart Demo

See Also

Chart Types

[Column Range Chart](#)

Column Range Chart is similar to the Column Chart except that each column is rendered over a range. Therefore the user must specify the y-axis Starting and Ending values for each point.

The following figure shows a Column Range Chart.

![Chart-Controlsimg81](Chart-Controlsimages/Chart-Controls_img81.jpeg)

Data Requirements

Data Requirement

Details	
Number of y values per point	two
Number of points	one or more
Number of series	one or more

[Custom Properties](#)

Custom Property

Name	Type	Container	Description
------	------	-----------	-------------

ChartType.Spacing	double	ChartArea	sets the interval between columnsPossible values lie between 0 and 1.
-------------------	--------	-----------	---

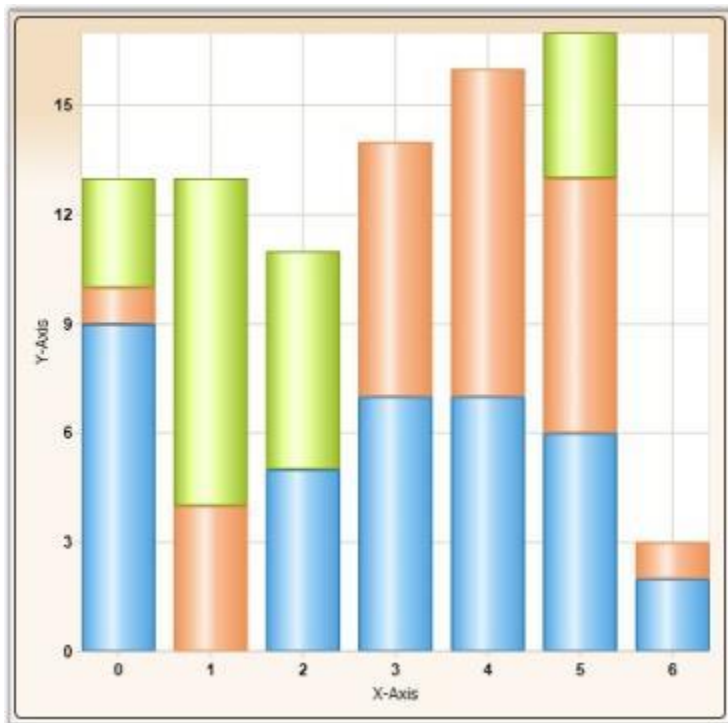
Template

While setting template the following parameters can be used:

Template Parameter

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Column Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Column Chart Demo

Stacking Column Chart

Stacking Column Charts are similar to regular column charts except that the Y values stack on top of each other in the specified series order. This helps visualize the relationship of parts to the whole.

The following image shows a sample Stacking Column Chart.

![[Chart-Controlsimg82]](Chart-Controlsimages/Chart-Controls_img82.jpeg)

Data Requirements

Data Requirement

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Custom StackingColumn100 Properties

Custom Stacking Column 100 Property

Name	Type	Container	Description
ChartStackingColumn100Type.ShowValueAsProbability	bool	ChartArea	y-axis range is set between 0 and 100. If true, the y-axis range is set between 0 and 1. Default value is <i>false</i> .

Template

While setting template, the following parameters can be used.

Template Parameter

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
IsUpper	bool	true “ if this is upper column
IsLower	bool	true “ if this is lower column
Series	ChartSeries	reference to series-owner

Stacking Negative Series

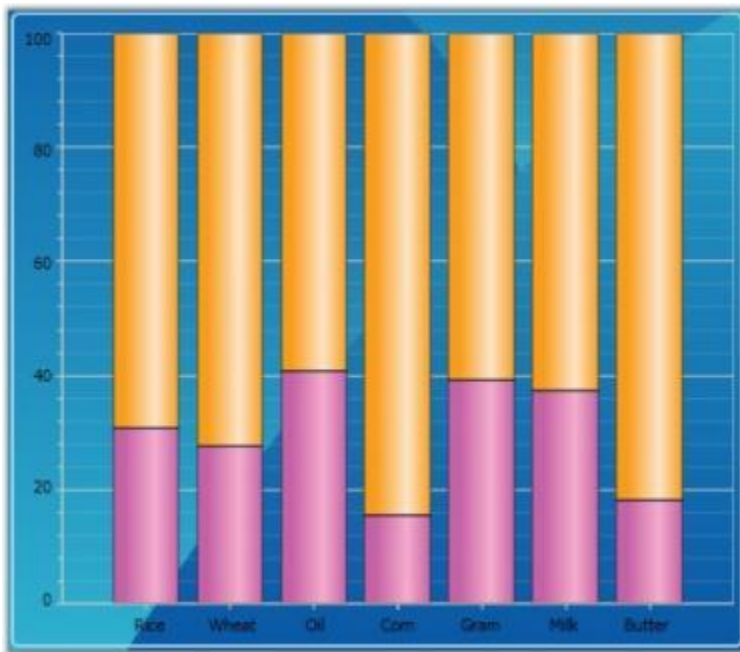
When negative values are added, Stacking Column chart can be made to be stacked separately in the chart area, above and below the y-axis 0.

Following code is used to do this.

C#

```
ChartStackingColumnType.SetRequiresNegativeSeriesStack(this.chartArea2,  
true);
```

A sample which demonstrates Column Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Column Chart Demo

Stacking Column 100 Chart

In the 100 % Stacked Column Chart, the cumulative proportion of each stacked element always totals 100%. This type of chart is great to visualize the relative contribution of each series values to the whole.

The following image shows a sample Stacking Column 100 Chart.

![[Chart-Controlsimg83](Chart-Controlsimages/Chart-Controls_img83.jpeg)]

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Spline Settings

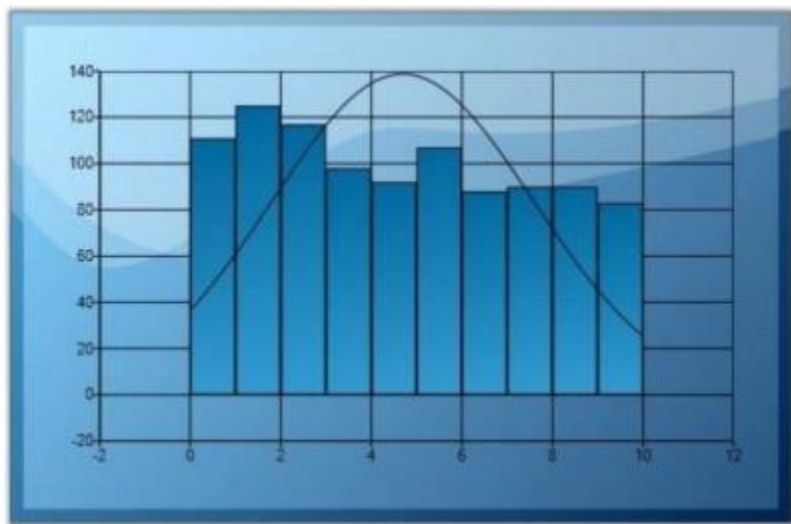
Name	Type	Container	Description
ChartStackingColumn100.ShowValueAsProbability	bool	ChartArea	y-axis range is set from 0 - 100If true, y-axis range is set from 0 - 1. Default value is <i>false</i> .

Template

While setting template, the following parameters can be used.

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
IsUpper	bool	<i>true</i> “ if this is upper column
IsLower	bool	<i>true</i> “ if this is lower column
Percentage	double	indicates the percentage this point takes up
Series	ChartSeries	reference to series-owner

A sample which demonstrates Column Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Column Chart Demo

See Also

Chart Types


Histogram Chart

Histogram is a bar (column) chart of a frequency distribution in which the widths of the bars are proportional to the classes into which the variable has been divided and the heights of the bars are proportional to the class frequencies. The categories are usually specified as non overlapping intervals of some variable. The categories (bars) must be adjacent. In addition, the chart has the capability to draw a normal distribution curve.

Histograms are useful data summaries that convey the following information:

- The general shape of the frequency distribution. (normal, exponential, etc.)
- Symmetry of the distribution and whether it is skewed.
- Modality - unimodal, bimodal or multimodal.

The shape of the distribution conveys important information such as the probability distribution of the data.

![[Chart-Controls

Data Requirements

Data Requirement

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

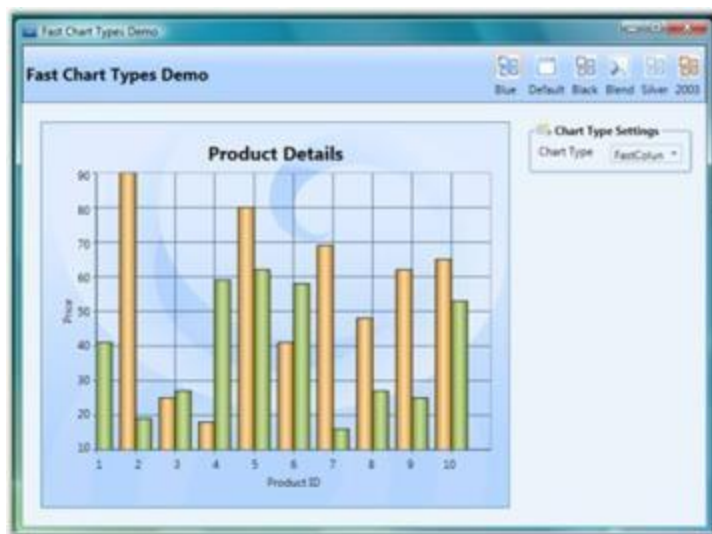
Histogram Settings

Histogram Setting

Name	Type	Container	Description
ChartHistogramType.IntervalOfHistogram	double	ChartArea	attached property that specifies the Interval which leads for one column
ChartHistogramType.DrawNormalDistribution	bool	ChartSeries	specifies whether to draw Normal Distribution Line

Template

While setting template the following parameters can be used:

*Template Parameter*

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

Fast Column Chart

Fast Column Chart is similar to Column chart as it uses vertical bars (called columns) to display different values of one or more items. Points from adjacent series are drawn as bars next to each other.

It is used for comparing the frequency, count, total or average of data in different categories. It is ideal for showing the variations in the value of an item over time.

The following points mark the advantages of Fast Column over Column charts:

- The Fast Column charts are rendered using drawing visuals.
- They load faster than the Column charts.
- They ensure high performance for displaying data.
- They can be used as real time charts to render huge number of data points.

The Chart type Fast Column is added in the Enum of type ChartTypes.

![[Chart-Controlsimg85]](Chart-Controlsimages/Chart-Controls_img85.jpeg)

Data Requirements

Details

Number of y values per point	one
Number of points	one or more
Number of series	one or more

Custom Properties

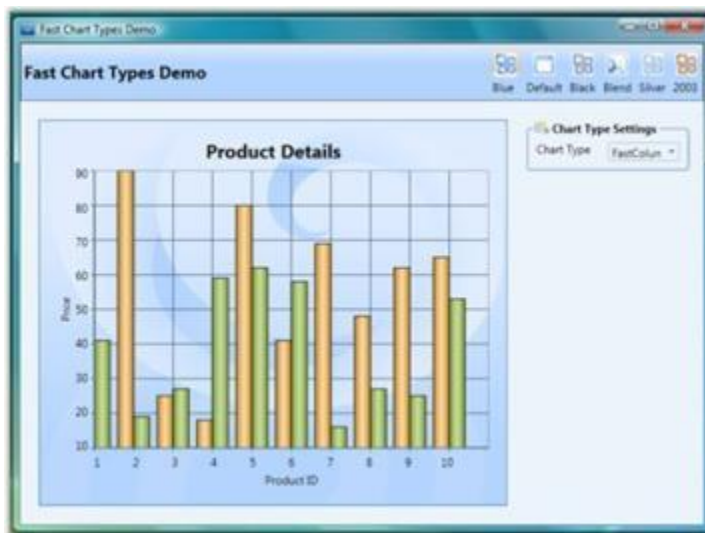
Name	Type	Container	Description
ChartType.Spacing	double	ChartArea	sets the interval between columnsPossible values lie between 0 and 1.

Template

While setting template the following parameters can be used:

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Fast Column Chart Type is available in the following sample installation path.



..My Documents\Syncfusion\Essential Studio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Performance-> Fast chart types

The following code example illustrate the usage of Fast Column charts.

XML

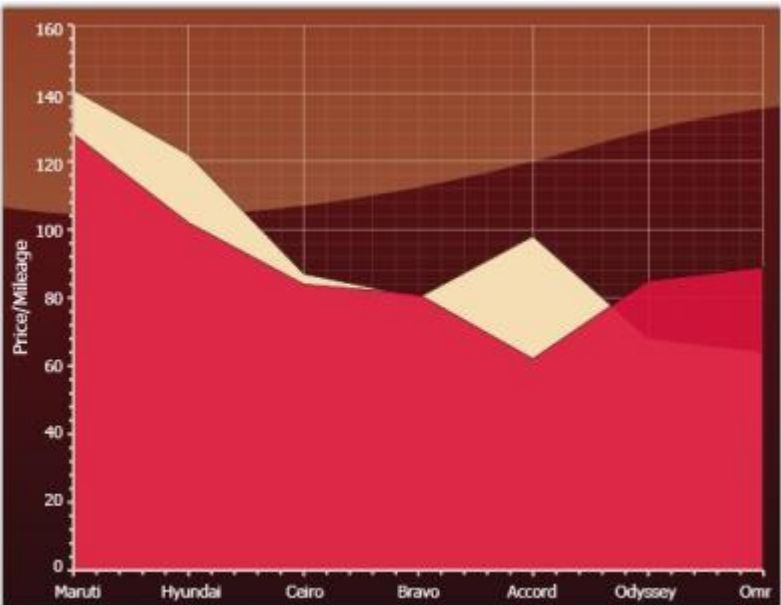
```
<syncfusion:ChartSeries Type="FastColumn" Name="series1" Stroke="Black"
DataSource="{Binding}"/>
```

C#

```
ChartSeries series = new ChartSeries();
series.Type = ChartTypes.FastColumn;
```

Run the sample.

A Fast Column chart is displayed pertaining to the data source it is bound to.



![[Chart-Controls
Controlsimages/Chart-Controls_img86.jpeg]](Chart-Controlsimages/Chart-Controls_img86.jpeg)

Area Charts

Area Chart

The Area Chart connects the Y-points using straight lines and forms an area covered by the above lines and X-axis. This area is then shaded with a specified color or gradient.

Multiple series can be plotted on the same chart and alpha-blended interior color can be used on the exterior chart to make the interior chart show through.

The following image shows a multi series Area Chart.

![[Chart-Controlsimg87]](Chart-Controlsimages/Chart-Controls_img87.jpeg)

Data Requirements

Details	
Number of y values per point	one

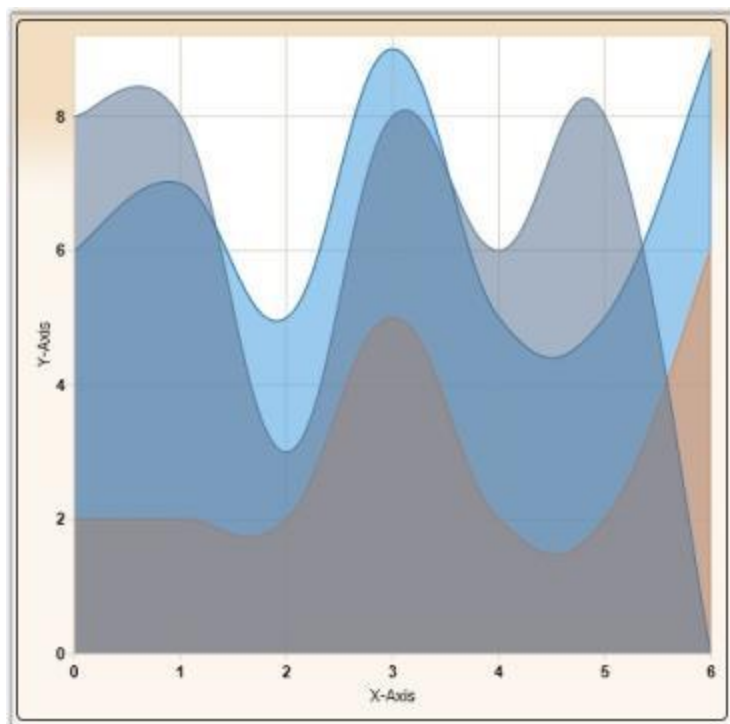
Number of points	two or more
Number of series	one or more

Template

The following parameters can be used while specifying template.

Name	Type	Description
X1	double	x-coordinate of first point
Y1	double	y-coordinate of first point
X2	double	x-coordinate of second point
Y2	double	y-coordinate of second point
IsStartSegment	bool	<i>true</i> - if this is start segment
IsEndSegment	bool	<i>true</i> - if this is end segment
Geometry	Geometry	segment geometry
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Area Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Area Chart Demo

[Spline Area Chart](#)

Spline Area Chart is similar to an Area Chart with the only difference being the way in which the points of a series are connected. It connects each series of points by a smooth spline curve.

Multiple series can be plotted on the same chart and alpha-blended interior color can be used on the exterior chart to make the interior chart(s) show through.

The following image shows a multi series Spline Area Chart.

![[Chart-Controlsimg88]](Chart-Controlsimages/Chart-Controls_img88.jpeg)

[Data Requirements](#)

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

[SplineArea Properties](#)

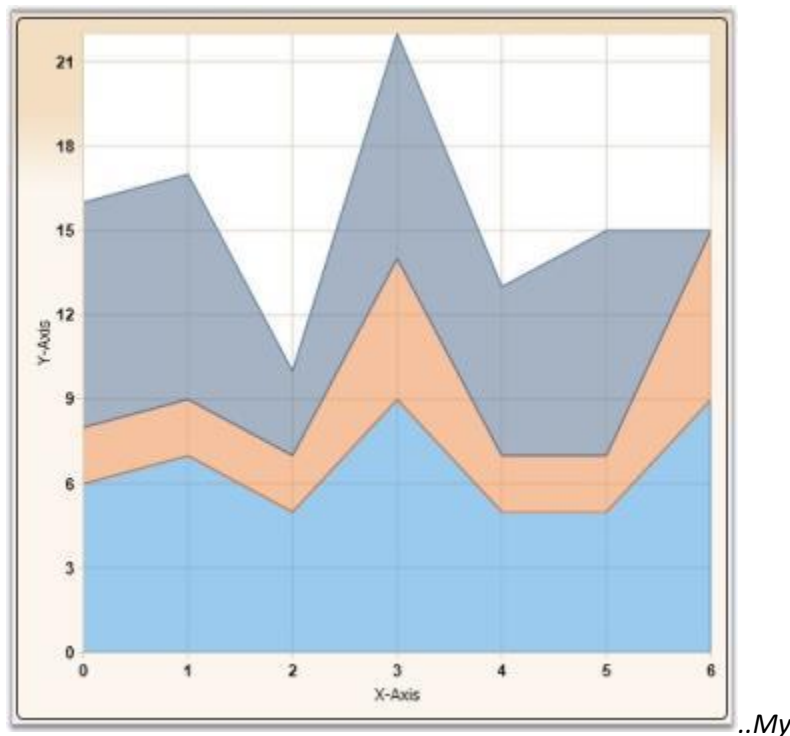
Name	Type	Container	Description
ChartSplineType.SplineCoefficient	double	ChartSeries	responsible for spline curvature

Template

While setting template the following parameters can be used:

Name	Type	Description
X1	double	x-coordinate of first point
Y1	double	y-coordinate of first point
X2	double	x-coordinate of second point
Y2	double	y-coordinate of second point
IsStartSegment	bool	<i>true</i> - if this is start segment
IsEndSegment	bool	<i>true</i> - if this is end segment
Geometry	Geometry	segment geometry
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Area Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Area Chart Demo

Stacking Area Chart

Stacking Area Charts are similar to regular area charts except that the Y values stack on top of each other in the specified series order. This helps visualize the relationship of parts to the whole.

The following image shows a sample Stacking Area Chart.

![Chart-Controlsimg89](Chart-Controlsimages/Chart-Controls_img89.jpeg)

Data Requirements

Details	
Number of Y values per point:	one
Number of points:	one or more
Number of series:	one or more

Template

While setting template the following parameters can be used.

Name	Type	Description
X1	double	x-coordinate of first point
Y1	double	y-coordinate of first point
X2	double	x-coordinate of second point
Y2	double	y-coordinate of second point
IsStartSegment	bool	<i>true</i> - if this is start segment
IsEndSegment	bool	<i>true</i> - if this is end segment
Geometry	Geometry	segment geometry
IsUpper	bool	<i>true</i> if this is upper segment
IsLower	bool	<i>true</i> if this is lower segment
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

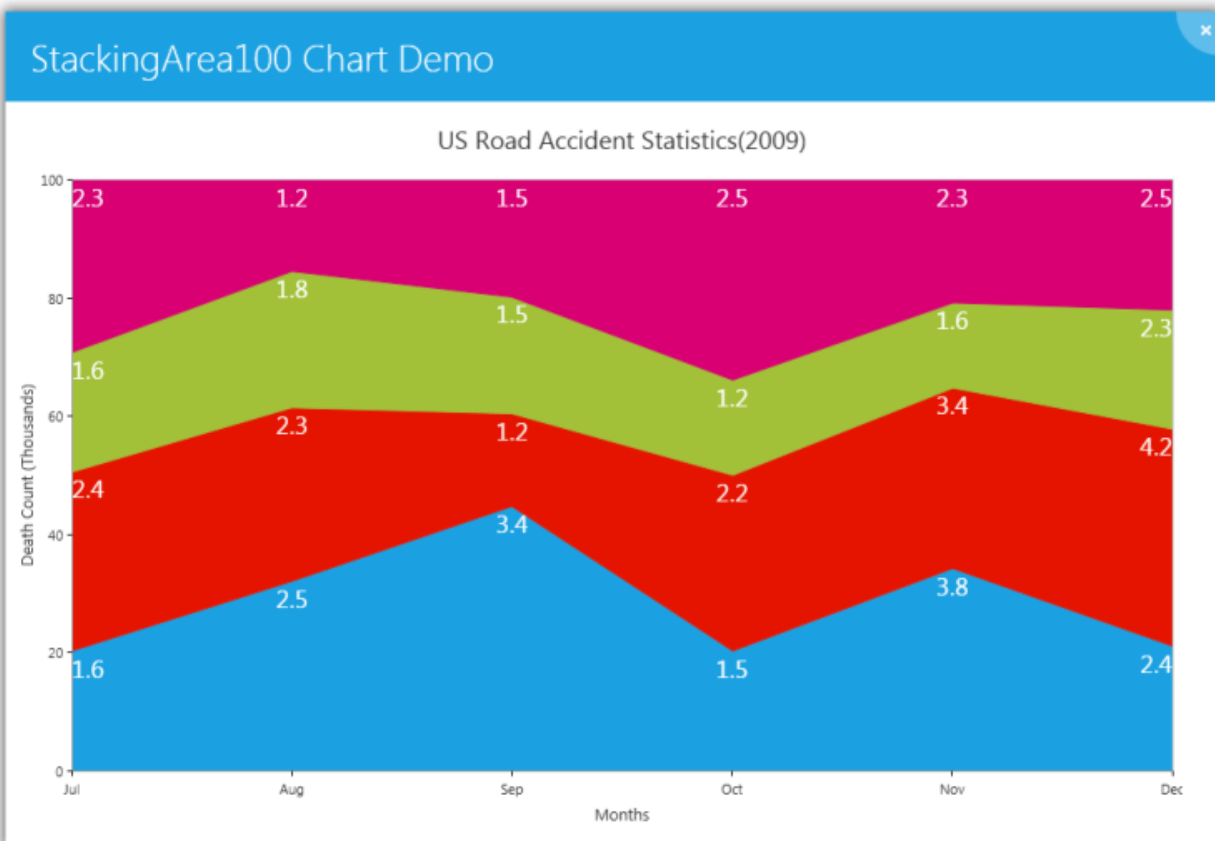
Stacking Negative Series

When negative values are added, Stacking Area chart can be made to be stacked separately in the chart area, above and below the x- axis 0.

C#

```
ChartStackingAreaType.SetRequiresNegativeSeriesStack(this.chartArea1, true);
```

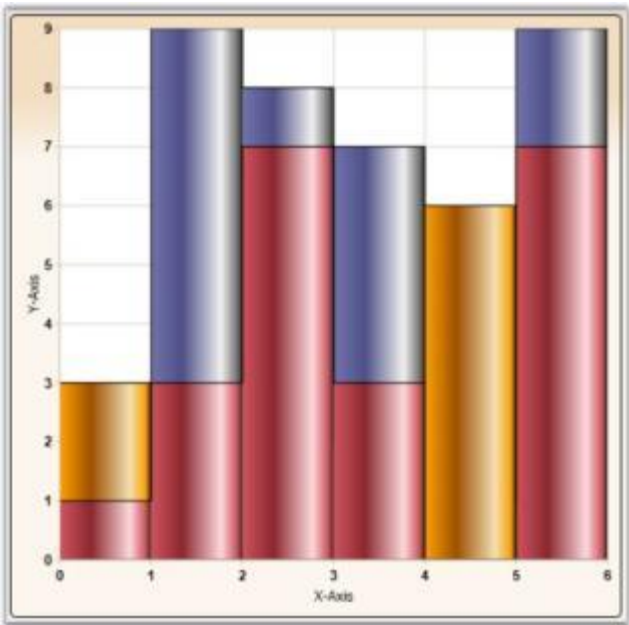
A sample which demonstrates Area Chart Types is available in the following sample installation path.



..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Area Chart Demo

[StackingArea100 Chart](#)

100% stacked area charts are similar to regular area charts except that the y values stack to 100% on top of each other in the specified series order. In the 100% stacked area charts, the cumulative proportion of each stacked element always totals 100%. This type of chart is great to visualize the relative contribution of each series' values to the whole.



![[Chart-Controlsimages/Chart-Controls_img90.png)](Chart-images90)](Chart-

[Sample Link](#)

A sample that demonstrates the StackingArea100 chart type is available in the following sample installation path:

..MyDocuments\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Basic Charts\StackingArea100 Chart Demo

[Adding StackingArea100 Chart to Chart Area](#)

The StackingArea100 chart type can be added into a chart area through the following code example:

XML

```
<syncfusion:Chart>
  <syncfusion:ChartArea>
    <syncfusion:ChartSeries Data="10,10,20,20,30,30" Type="StackingArea100"/>
    <syncfusion:ChartSeries Data="20,20,30,30,40,40" Type="StackingArea100"/>
  </syncfusion:ChartArea>
</syncfusion:Chart>
```

Data Requirement Details	
Number of Y values per point	One
Number of points	One or more
Number of Series	One or more

[Overriding Data Template for Chart Series](#)

The data template for a 100% stacked area chart can be overridden by changing the default template as shown in the following code example.

XML

```

[XAML: Default Template]
<DataTemplate x:Key="{x:Type local:ChartStackingArea100Segment}">
<Path Stroke="{Binding Stroke}" StrokeThickness="{Binding StrokeThickness}"
Fill="{Binding Interior}" Data="{Binding Geometry}" ToolTip="{Binding
ToolTip}"/>
</DataTemplate>
[Custom Template]
<DataTemplate x:Key="customTemplate">
<Path Stroke="{Binding Stroke}" StrokeThickness="4" Fill="Red"
Data="{Binding Geometry}"/>
</DataTemplate>
[Apply Template]
<syncfusion:Chart>
<syncfusion:ChartArea>
<syncfusion:ChartSeries Template="{StaticResource customTemplate}"
Data="10,10,20,20,30,30" Type="StackingArea100"/>
<syncfusion:ChartSeries Data="20,20,30,30,40,40" Type="StackingArea100"/>
</syncfusion:ChartArea>
</syncfusion:Chart>

```

When setting template, the following parameters can be used:

Name	Type	Description
Geometry	Geometry	Segment geometry
Stroke	Brush	Segment stroke
StrokeThickness	Thickness	Segment stroke thickness
Interior	Brush	Segment color
Series	ChartSeries	Reference to series owner

StepArea Chart

Step Area Charts are similar to regular area charts except that instead of a straight line tracing the shortest path between points, the values are connected by continuous vertical and horizontal lines forming a step-like progression. Also, step area charts contain only one segment for each series.

![[Chart-Controlsimg91]](Chart-Controlsimages/Chart-Controls_img91.jpeg)

Data Requirements

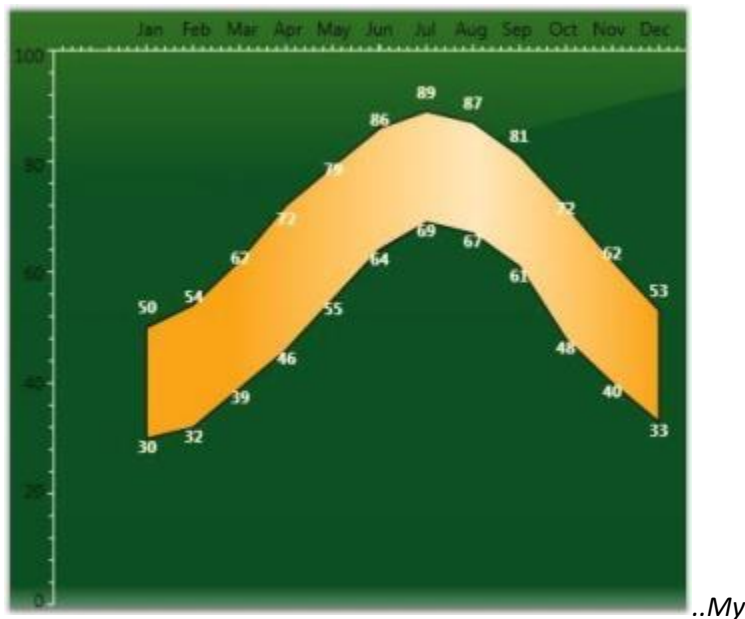
Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Template

While setting template the following parameters can be used.

Name	Type	Description
X1	double	x-coordinate of first point
Y1	double	y-coordinate of first point
X2	double	x-coordinate of second point
Y2	double	y-coordinate of second point
StepX	double	x-coordinate of transient point
StepY	double	y-coordinate of transient point
Geometry	Geometry	segment geometry
IsStartSegment	bool	<i>true</i> - if this is start segment
IsEndSegment	bool	<i>true</i> - if this is end segment
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Area Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Area Chart Demo

Range Area Chart

Range Area Chart is a variation of Area Chart type that lets you plot bands of data in a chart, like Bollinger bands, weather patterns, etc. Each point in the chart is specified by 2 Y values – the lower and higher end of the band.

![[Chart-Controlsimg92](Chart-Controlsimages/Chart-Controls_img92.jpeg)]

Data Requirements

Details	
Number of y values per point	two
Number of points	one or more
Number of series	one or more

Custom Properties

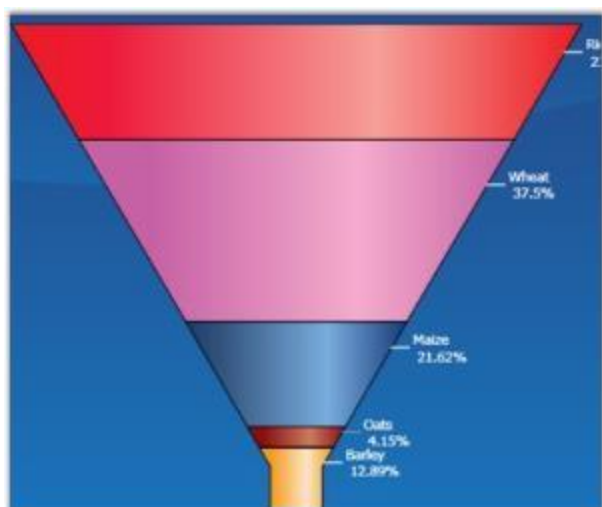
Name	Type	Container	Description
ChartRangeAreaType.HighValueInterior	Brush	ChartSeries	specifies the brush that should be used for the segment where the 2nd y value is greater than the 1st y value
ChartRangeAreaType.LowValueInterior	Brush	ChartSeries	specifies the brush that should be used for the segment where the 1st y value is greater than the 2nd y value

Template

The following parameters can be used while specifying template.

Name	Type	Description
IsHighLow	bool	<i>true</i> - if this segment is for a "high" area
Geometry	Geometry	segment geometry
FillBrush	Brush	interior for this segment
Series	ChartSeries	reference to series-owner

A sample which demonstrates Range Area Chart Type is available in the following sample installation path.



..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Range-Area Chart Demo

Accumulation Charts

Funnel Chart

The Funnel chart is a single series chart representing the data as portions of 100%, and this chart does not use any axes. Funnel chart can be viewed as 2D or 3D.

Funnel charts are often used to represent stages in a sales process and show the amount of potential revenue for each stage. This type of chart can be useful also in identifying potential problem areas in an organization's sales processes, for example. A funnel chart is similar to a stacked percent bar chart.

![[Chart-Controlsimg93]](Chart-Controlsimages/Chart-Controls_img93.jpeg)

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

FunnelType Properties

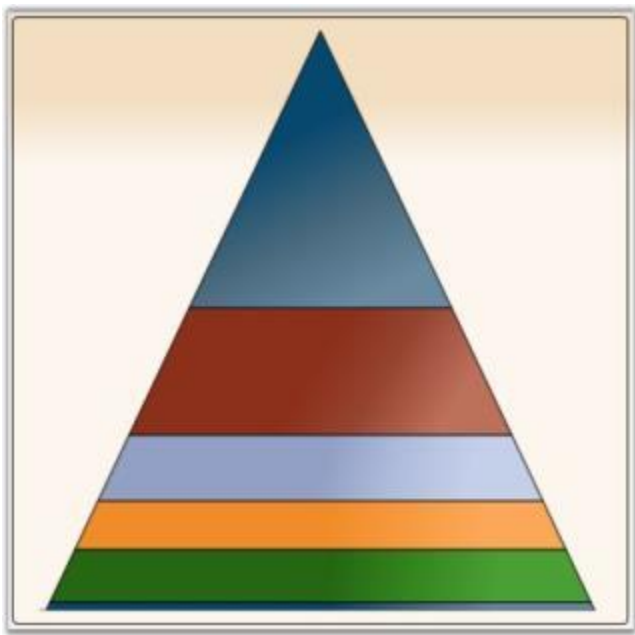
Name	Type	Container	Description
ChartFunnelType.ExplodedIndex	int	ChartSeries	index of segment which should be leant out
ChartFunnelType.GapRatio	double	ChartSeries	indicates relation of inner interval to their width
ChartFunnelType.FunnelMode	ChartFunnelMode	ChartSeries	method of data displaying

Template

While setting template the following parameters can be used:

Name	Type	Description
GapRatio	double	indicates relation of inner interval to their width
IsExploded	bool	true - if segment is leant out
ExplodedOffset	double	displacement on which segment should be leant out
MinWidth	double	minimal segment width
Geometry	Geometry	segment geometry
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Accumulation Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Accumulation Chart Demo

[Pyramid Chart](#)

Pyramid chart is similar to the funnel chart. It is often used for geographical purposes. The Pyramid Chart type displays the data which when totalled will be 100%. This type of chart is a single series chart representing the data as portions of 100%, and this chart does not use any axes. Pyramid chart can be viewed as 2D or 3D.

The following images are some sample Pyramid Charts.

![Chart-Controlsimg94](Chart-Controlsimages/Chart-Controls_img94.jpeg)

[Data Requirements](#)

Details

Number of y values per point	one
Number of points	one or more
Number of series	one or more

PyramidType Properties

Property Table

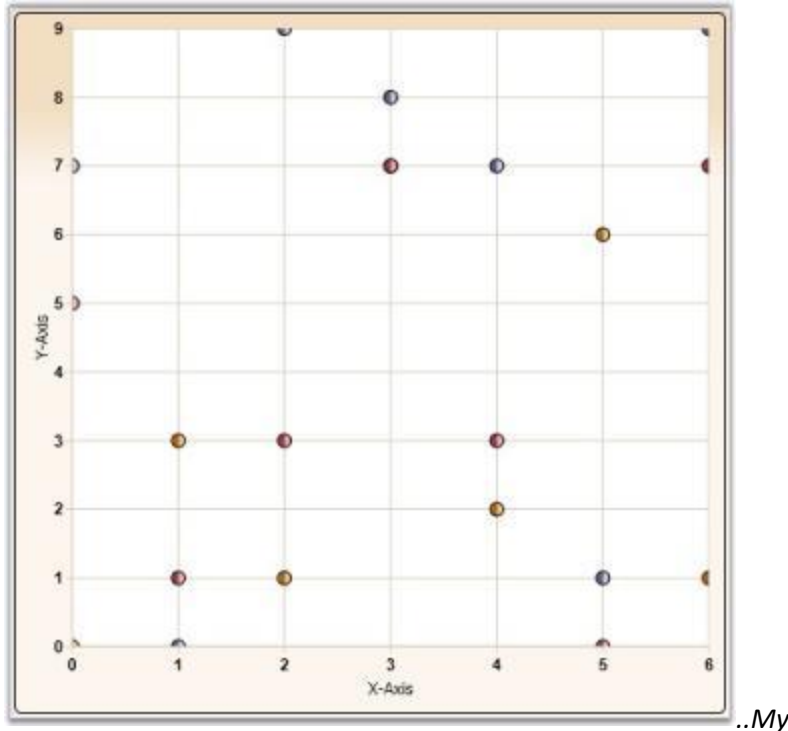
Name	Type	Container	Description
ChartPyramidType.ExplodedIndex	int	ChartSeries	index of segment which should be leant out
ChartPyramidType.GapRatio	double	ChartSeries	indicates relation of inner interval to their width
ChartPyramidType.PyramidMode	ChartPyramidMode	ChartSeries	method of data displaying

Template

While setting template the following parameters can be used.

Name	Type	Description
GapRatio	double	indicates relation of inner interval to their width
IsExploded	bool	true - if segment is leant out
ExplodedOffset	double	displacement on which segment should be leant out
Geometry	Geometry	segment geometry
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Accumulation Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Accumulation Chart Demo

XY Charts (Bubble and Scatter)

Scatter Chart

Scatter Charts, also known as XY Charts, are a plot of Y values and X values along two axes. The points are not joined together and can be customized using shapes or images to make them easily identifiable, usually independent of time.

The scatter graph lets you plot data points based on two independent variables. The variable that we seek to predict is called the dependent variable or Y-variable. The variable on which it depends is called the independent variable or the X-variable. Scatter graphs can chart multiple data sets, each represented by a different symbol and each having any number of data points.

It is used to display numerical data, either discrete or continuous. Scatter charts are commonly used for visualizing scientific data.

The following image shows a multi series Scatter Chart.

![[Chart-Controlsimg95]](Chart-Controlsimages/Chart-Controls_img95.jpeg)

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Template

While setting template the following parameters can be used:

Name	Type	Description
X	double	x point coordinate
Y	double	y point coordinate
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

Customizing the Height and Width of the Scatter Chart

You can modify the size of the Scatter chart by using the Height and Width properties of Scatter-Type, as shown in the following code sample:

XML

```
<syncfusion:ChartSeries Name="Series1" Type="Scatter"
DataSource="{Binding ExpensiveCarDetails}" BindingPathX="Position"
BindingPathsY="Price" syncfusion:ChartScatterType.ScatterHeight="30"
syncfusion:ChartScatterType.ScatterWidth="30">
</syncfusion:ChartSeries>
```

C#

```
ChartScatterType.SetScatterWidth(seriesname, 30);
ChartScatterType.SetScatterHeight(seriesname, 30);
```

A sample which demonstrates Scatter and Bubble Chart Types is available in the following sample installation path.

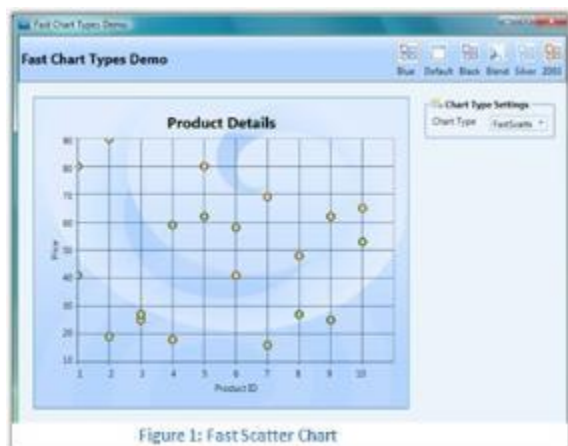


Figure 1: Fast Scatter Chart

..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Scatter And Bubble Chart Demo

Fast Scatter chart Type

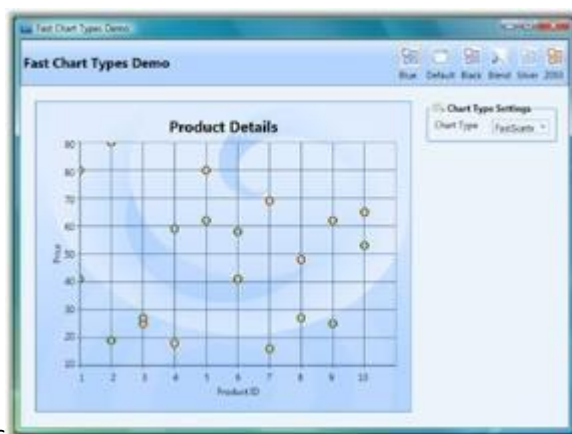
Fast Scatter charts are similar to Scatter Charts as they are a plot of Y values and X values along two axes. The points are not joined together and can be customized using shapes or images to make them easily identifiable, usually independent of time.

Fast Scatter charts can present multiple data sets, each represented by a different symbol and each having any number of data points. It is used to display numerical data, either discrete or continuous.

The following points mark the advantages of Fast Scatter Charts over Scatter Charts:

- The Fast Scatter Charts are rendered using drawing visuals.
- They load faster than the scattered charts.
- They ensure high performance for displaying data.
- They can be used as real time charts to render huge number of data points.

The Chart type Fast Scatter is added in the Enum of type ChartTypes.



![[Chart-Controls](Chart-Controlsimages/Chart-Controls_img96.jpeg)](Chart-Controlsimages/Chart-Controls_img96.jpeg)

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Template

While setting template the following parameters can be used:

Name	Type	Description
X	double	x point coordinate
Y	double	y point coordinate
Interior	Brush	column color

Series	ChartSeries	reference to series-owner
--------	-------------	---------------------------

The following code example illustrates the usage of Fast Scatter charts.

XML


```
<syncfusion:ChartSeries Type="FastScatter" Name="series1" Stroke="Black"
DataSource="{Binding}" />
```

C#

```
ChartSeries series = new ChartSeries();
series.Type = ChartTypes.FastScatter;
```

Run the sample.

A Fast Scatter chart is displayed.

![[Chart-Controls](Chart-Controlsimages/Chart-Controls_img97.jpeg)]

Customizing the Height and Width of the Fast Scatter Chart

You can modify the size of the Fast Scatter chart by using the Height and Width properties of FastScatter-Type, as shown in the following code sample:

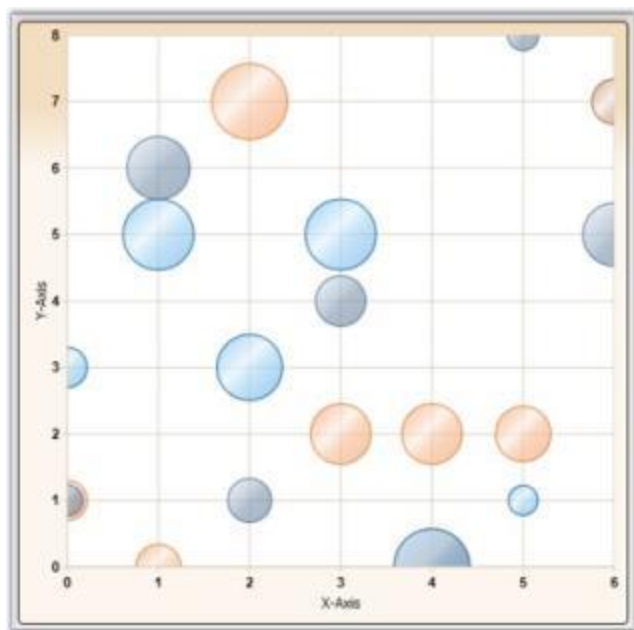
XML

```
<syncfusion:ChartSeries Name="Series1" Type="FastScatter"
DataSource="{Binding ExpensiveCarDetails}" BindingPathX="Position"
BindingPathsY="Price" syncfusion:ChartFastScatterType.FastScatterHeight="30"
syncfusion:ChartFastScatterType.FastScatterWidth="30">
</syncfusion:ChartSeries>
```

C#

```
ChartFastScatterType.SetFastScatterWidth(seriesname, 30);
ChartFastScatterType.SetFastScatterHeight(seriesname, 30);
```

A sample which demonstrates Fast Scatter Chart Types is available in the following sample installation path.



..My Documents\Syncfusion\Essential

Studio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Performance-> Fast chart types

Bubble Chart

Bubble Chart is an extension of the Scatter Chart (or XY-chart) where each data marker is represented by a circle whose dimension forms a third variable. Consequently, bubble charts allow three-variable comparisons allowing for easy visualization of complex interdependencies that are not apparent in two-variable charts. Bubble charts are frequently used in market and product comparison studies.

Though it's called a bubble chart, the data marker can be rendered as either a circle, image or square using the BubbleType property.

The following image shows a multi series Bubble Chart.

![Chart-Controlsimg98](Chart-Controlsimages/Chart-Controls_img98.jpeg)

Data Requirements

Details	
Number of y values per point	two
Number of points	one or more
Number of series	one or more

BubbleType Properties

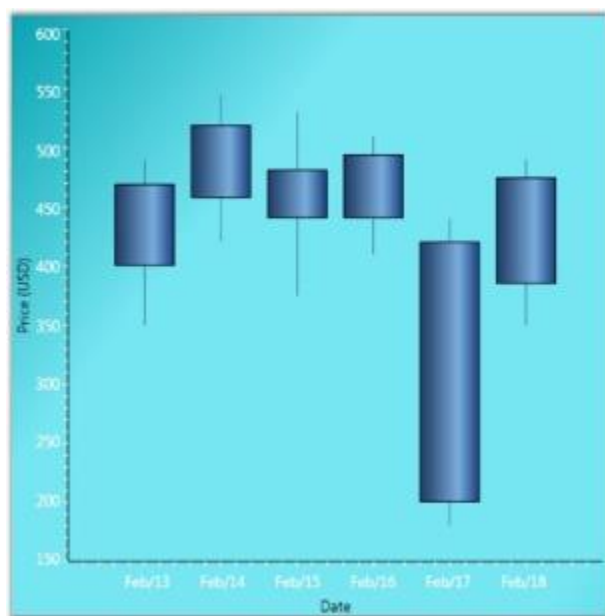
Name	Type	Container	Description
ChartBubbleType.MinRadius	double	ChartSeries	minimal figure radius
ChartBubbleType.MaxRadius	double	ChartSeries	maximal figure radius

Template

While setting template the following parameters can be used:

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Radius	double	figure radius
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Scatter and Bubble Chart Types is available in the following sample installation path.



..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Scatter And Bubble Chart Demo

Financial Charts

Candle Chart

A Candle Chart displays stock information using the High, Low, Open and Close values. The Hi and Lo values are represented by the wick of a candle. The candle represents open and close values.

The following image shows a CandleChart displaying a single series.

![Chart-Controlsimg99](Chart-Controlsimages/Chart-Controls_img99.jpeg)

Data Requirements

Details	
Number of y values per point	four (High, Low , Open and Close respectively)
Number of points	one or more
Number of series	one or more

CandleType Properties

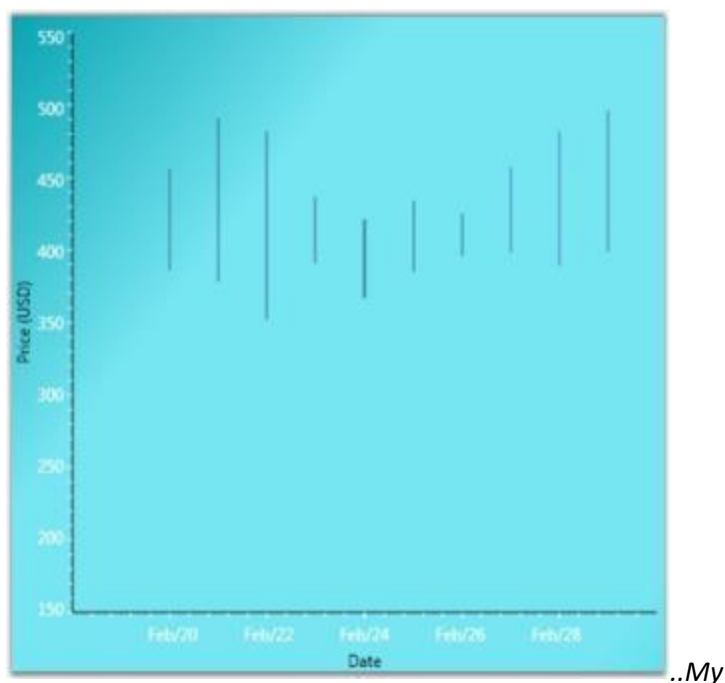
Name	Type	Container	Description
ChartType.Spacing	double	ChartArea	interval between line groupsPossible value lies between 0 and 1.

Template

While setting template the following parameters can be used:

Name	Type	Description
HiX	double	x-coordinate of upper border
HiY	double	y-coordinate of upper border
LoX	double	x-coordinate of lower border
LoY	double	y-coordinate of lower border
X	double	x-column coordinate
Y	double	y-column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Financial Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Financial Chart Demo

HiLo Chart

Hi Lo Chart is a special kind of chart that is normally used in stock analysis. They are typically used to display error bars or the trading range of a stock for each period.

The Hi Lo Chart expect 2 Y values to be specified in the series. One value should represent the high and the other value should represent the low stock price for the period. This can be specified in any order.

![Chart-Controlsimg100](Chart-Controlsimages/Chart-Controls_img100.jpeg)

Data Requirements

Details	
Number of y values per point	two
Number of points	one or more
Number of series	one or more

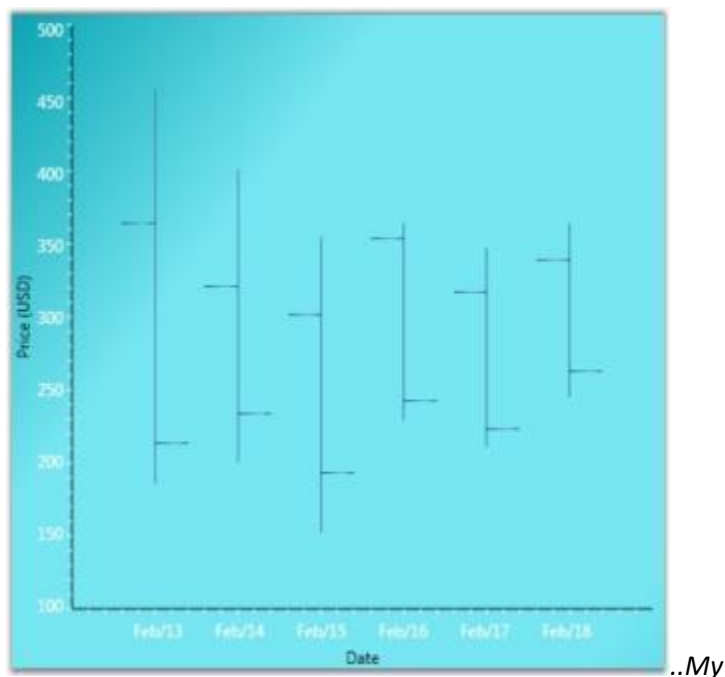
Template

While setting template the following parameters can be used:

Name	Type	Description
HiX	double	x-coordinate of upper border
HiY	double	y-coordinate of upper border
LoX	double	x-coordinate of lower border

LoY	double	y-coordinate of lower border
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Financial Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Financial Chart Demo

HiLoOpenClose Chart

Hi Lo Open Close Chart is a special kind of chart that is normally used in stock analysis. This chart type expects 4 Y values for every point in the series. Those values should represent the High, Low, Open and Close values of the stock, in that order, for that period.

![[Chart-Controlsimg101]](Chart-Controlsimages/Chart-Controls_img101.jpeg)

Data Requirements

Details	
Number of y values per point	four (Open, Close, High and Low, in that order)
Number of points	one or more
Number of series	one or more

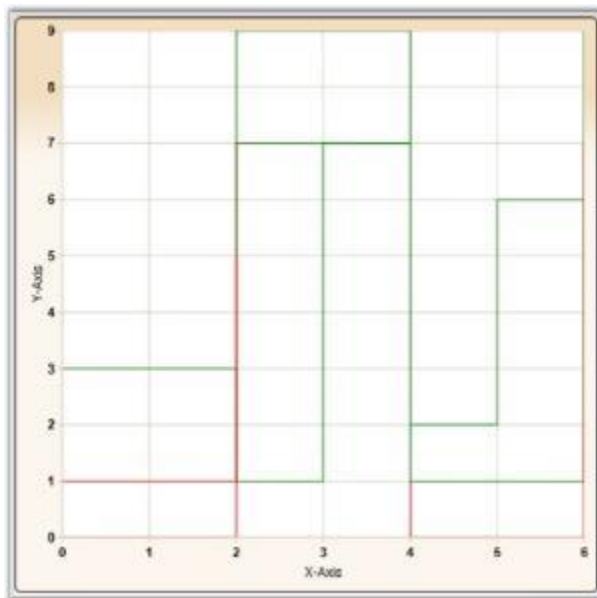
Template

While setting template the following parameters can be used:

Name	Type	Description
------	------	-------------

HiX	double	x-coordinate of upper border
HiY	double	y-coordinate of upper border
LoX	double	x-coordinate of lower border
LoY	double	y-coordinate of lower border
StartOpenX	double	x start coordinate of opening price
StartOpenY	double	y start coordinate of opening price
EndOpenX	double	x end coordinate of opening price
EndOpenY	double	y end coordinate of opening price
StartCloseX	double	x start coordinate of closing price
StartCloseY	double	y start coordinate of closing price
EndCloseX	double	x end coordinate of closing price
EndCloseY	double	y end coordinate of closing price
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Financial Chart Types is available in the following sample installation path.



..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Financial Chart Demo

Kagi Chart

Kagi Charts are a Japanese invention and date since the late 1870's, but were popularized in the western world by Steven Nison. They contain a series of connecting vertical lines where the thickness and direction of those lines depend on price. If closing prices continue to move in the direction of the prior

vertical Kagi line, then that line is extended. However, if the closing price reverses by a pre-determined "reversal" amount, a new Kagi line is drawn in the next column in the opposite direction.

The penetration of a prior column's high or low, by the latest closing price, alters the colors of the lines. These colors depict either a bullish or bearish pattern. Use the `PriceUpColor` and `PriceDownColor` properties to specify the colors for these two patterns. The wider the columns, the stronger the pattern.

![[Chart-Controlsimg102]](Chart-Controlsimages/Chart-Controls_img102.jpeg)

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

KagiType Properties

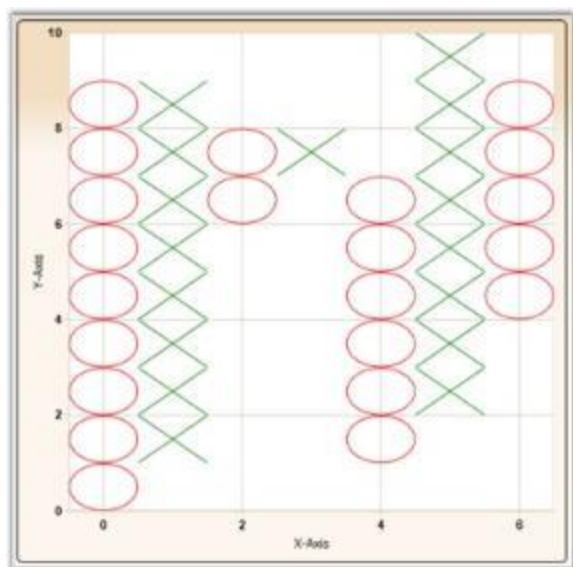
Name	Type	Container	Description
ChartKagiType.ReversalAmount	double	ChartSeries	price difference which should overcome in order to change diagram direction

Template

While setting template the following parameters can be used:

Name	Type	Description
IsPriceUp	bool	true if segment shows price rising
IsPriceDown	bool	true if segment shows price recession
Points	PointCollection	collection of segment points
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Financial Chart Types is available in the following sample installation path.



..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Financial Chart Demo

Point and Figure Chart

Point and Figure Chart is used to identify support levels, resistance levels and chart patterns. The chart ignores the time factor and concentrates solely on movements in price - a column of Xs or Os may take one day or several weeks to complete. By convention, the first X in a column is plotted one box above the last O in the previous column (and the first O in a column is plotted one box below the highest X).

This is a chart that plots the day-to-day increment and decrement in price. It uses a series of X's and O's to determine price trends where the X's represent an upward trend and the O's represent a downward trend. The default value of ReversalAmount is 1. Use the PriceUpColor to specify the color for the Xs and PriceDownColor to specify the color for the Os.

This chart requires 2 Y values, the high value and the low value for the specified period.

![Chart-Controlsimg103](Chart-Controlsimages/Chart-Controls_img103.jpeg)

Data Requirements

Details	
Number of y values per point	two
Number of points	one or more
Number of series	one or more

PointAndFigure Customization

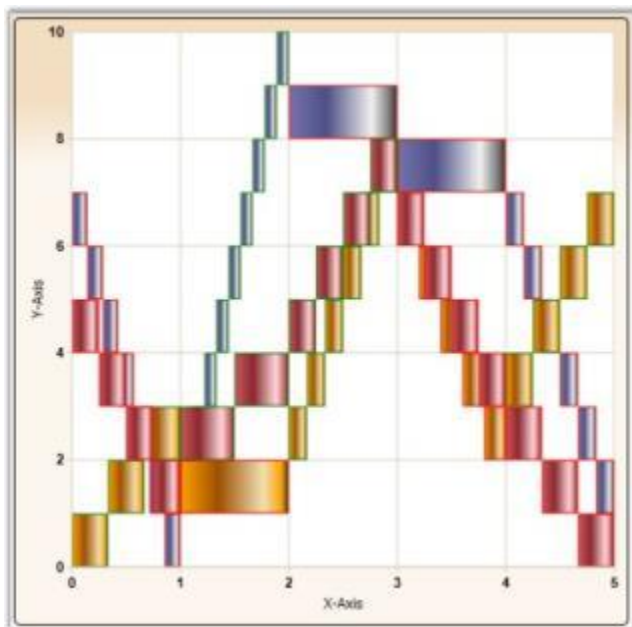
Name	Type	Container	Description
ChartPointAndFigureType.FigureCost	double	ChartSeries	price of one segment
ChartPointAndFigureType.ReversalAmount	double	ChartSeries	price difference which should be overcome in order to change diagram direction

Template

While setting template the following parameters can be used.

Name	Type	Description
X	double	x-column coordinate
Y	double	y-column coordinate
Width	double	column width
IsPoint	bool	<i>true</i> if segment is point
IsFigure	bool	<i>true</i> if segment is figure
Height	double	column height
Interior	Brush	column Color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Financial Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Financial Chart Demo

Renko Chart

Renko charting method is thought to have acquired its name from "renga" which is the Japanese word for bricks. Renko Charts were introduced by Steve Nison. Renko (Bricks) are drawn equal in size for a determined amount. A brick is drawn in the direction of the prior move only if prices move by a minimum amount. If prices change by the determined amount or more, a new brick is drawn. If prices

change by less than the determined amount (specified by ReversalAmount), the new price is ignored. The default value of ReversalAmount is 1.

If the new closing price penetrates the previous bricks closing price in the opposite direction a trend reversal highlighted by the change in color of the bricks happens. Use the PriceUpColor to indicate bullish trend and PriceDownColor to indicate bearish trend.

Since a Renko chart isolates the underlying trends by filtering out the minor ups and downs, Renko charts are excellent in determining support and resistance levels.

![[Chart-Controlsimg104]](Chart-Controlsimages/Chart-Controls_img104.jpeg)

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Renko Customization

Name	Type	Container	Description
ChartRenkoType.RenkoCost	double	ChartSeries	price of one Renko segment

Template

While setting template the following parameters can be used:

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
IsPriceUp	bool	true if segment shows price rising
IsPriceDown	bool	true if segment shows price recession
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Financial Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Financial Chart Demo

ThreeLineBreak Chart

Three Line Break Chart is similar in concept to point and figure charts. The Three Line Break charting method is so-named because of the number of lines typically used. It displays a series of vertical boxes ("lines") that are based on changes in prices. It ignores the passage of time.

The three-line break chart looks like a series of rising and falling lines of varying heights. Each new line, like the Xs and Os of a point and figure chart, occupies a new column. Based on closing prices (or highs and lows), a new rising line is drawn if the previous high is exceeded and a new falling line is drawn if the price hits a new low. Change in price trends are highlighted by changing colors. Use the PriceUpColor to indicate bullish trend and PriceDownColor to indicate bearish trend.

The ReversalAmount specifies the threshold amount by which the price should change to begin rendering a new vertical box in the appropriate direction.

![Chart-Controlsimg105](Chart-Controlsimages/Chart-Controls_img105.jpeg)

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Renko Customization

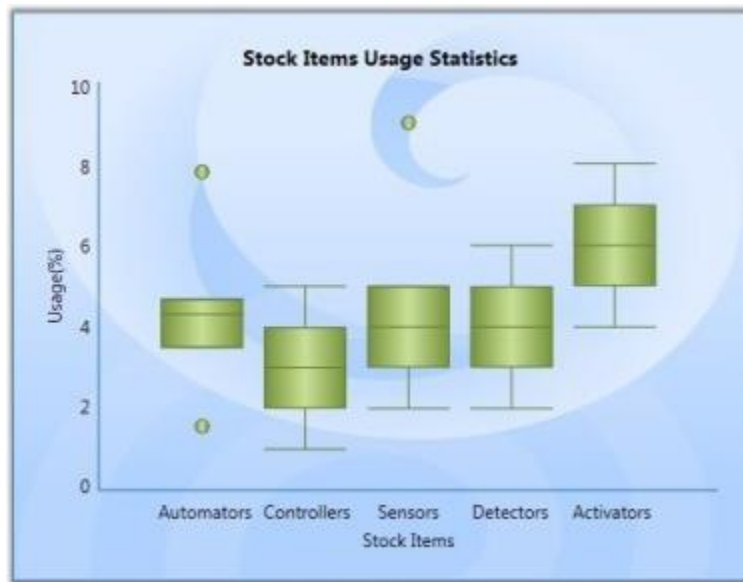
Name	Type	Container	Description
ChartThreeLineBreakType.BreakLineCount	double	ChartSeries	sets the break line count Default value is 3.

Template

While setting template the following parameters can be used.

Name	Type	Description
X	double	x-column coordinate
Y	double	y-column coordinate
Width	double	column width
IsPriceUp	bool	<i>true</i> if segment shows price rising
IsPriceDown	bool	<i>true</i> if segment shows price recession
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Financial Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Financial Chart Demo

Box and Whisker Chart

In 1977, John Tukey published an efficient method for displaying a five-number data summary. The graph is called a Box and Whisker plot (also known as BoxPlot) and summarizes the following statistical measures.

- median
- upper and lower quartiles (75 percentile to 25 percentile)
- minimum and maximum data values

The following is an example of a Box and Whisker plot.



![(Chart-Controls
Controls_img106.jpeg)img106](Chart-Controlsimages/Chart-

Custom Properties

Name	Type	Container	Description
ChartType.Spacing	double	ChartArea	interval between columnsPossible value lies between 0 and 1.

Template

While setting template the following parameters can be used.

Name	Type	Description
X	double	x-column coordinate
Y	double	y-column coordinate
Width	double	column width
TopWhiskerX1	double	x1 of upper border
TopWhiskerY1	double	y1 of upper border
TopWhiskerX2	double	x2 of upper border
TopWhiskerY2	double	y2 of upper border
BottomWhiskerX1	double	x1 of lower border
BottomWhiskerY1	double	y1 of lower border
BottomWhiskerX2	double	x2 of lower border
BottomWhiskerY2	double	y2 of lower border
MedianWhiskerX1	double	x1 of lengthwise line
MedianWhiskerY1	double	y1 of lengthwise line
MedianWhiskerX2	double	x2 of lengthwise line

MedianWhiskerY2	double	y2 of lengthwise line
Height	double	column height
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

Data Requirements

Details	
Number of y values per point	five (minimum, lower quartile, median, upper quartile, maximum)
Number of points	one or more
Number of series	one or more

Outlier Calculation

Outliers that are greater than the segment height can be rendered in the chart. Essential chart also provides option to set a difference in such outlier rendering. This difference can be set using the `SetDefaultOutlierVisible` property.

C#

```
ChartBoxAndWhiskerType.SetDefaultOutlierVisible(ser, false);
```

A sample which demonstrates Box And Whisker Chart Type is available in the following sample installation path.

..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Box-and-Whisker Chart Demo

Fast HiLo Open Close

Fast High Low (HiLo) Open Close charts are similar to HiLo Open Close Charts and are used in stock analysis. This chart type expects 4 Y values for every point in the series. Those values should represent the High, Low, Open and Close values of the stock, in that order, for a particular period.

The Fast High Low (HiLo) Open Close charts have the following advantages:

- The Fast HiLo Open Close charts are rendered using drawing visuals.
- They load faster than the HiLo Open Close charts.
- They ensure high performance for displaying data.
- They can be used as real time charts to render huge number of data points.

The Chart type HiLo Open Close is added in the Enum of type `ChartTypes`.

![[Chart-Controls*Data Requirements*

Details

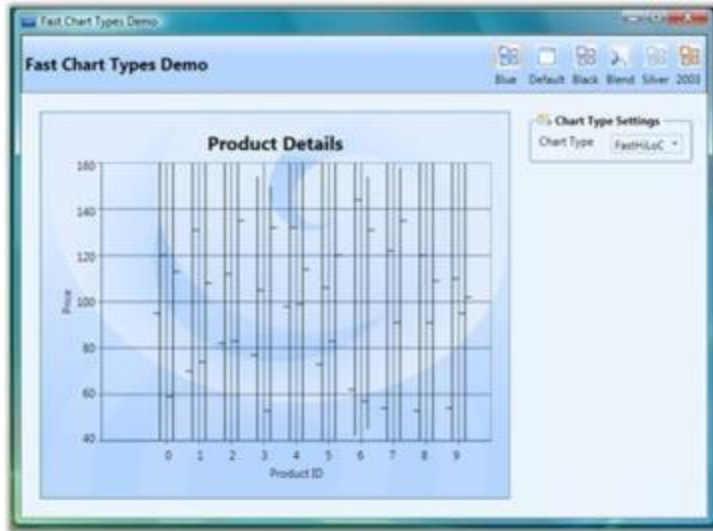
Number of y values per point	Four (open, close, high and low, in that order)
Number of points	one or more
Number of series	one or more

Template

While setting template the following parameters can be used:

Name	Type	Description
HiX	double	x-coordinate of upper border
HiY	double	y-coordinate of upper border
LoX	double	x-coordinate of lower border
LoY	double	y-coordinate of lower border
StartOpenX	double	x start coordinate of opening price
StartOpenY	double	y start coordinate of opening price
EndOpenX	double	x end coordinate of opening price
EndOpenY	double	y end coordinate of opening price
StartCloseX	double	x start coordinate of closing price
StartCloseY	double	y start coordinate of closing price
EndCloseX	double	x end coordinate of closing price
EndCloseY	double	y end coordinate of closing price
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Financial Chart Types is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version

Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Performance\Fast chart types

The following code illustrates the creation of Fast High Low (HiLo) Open Close charts.

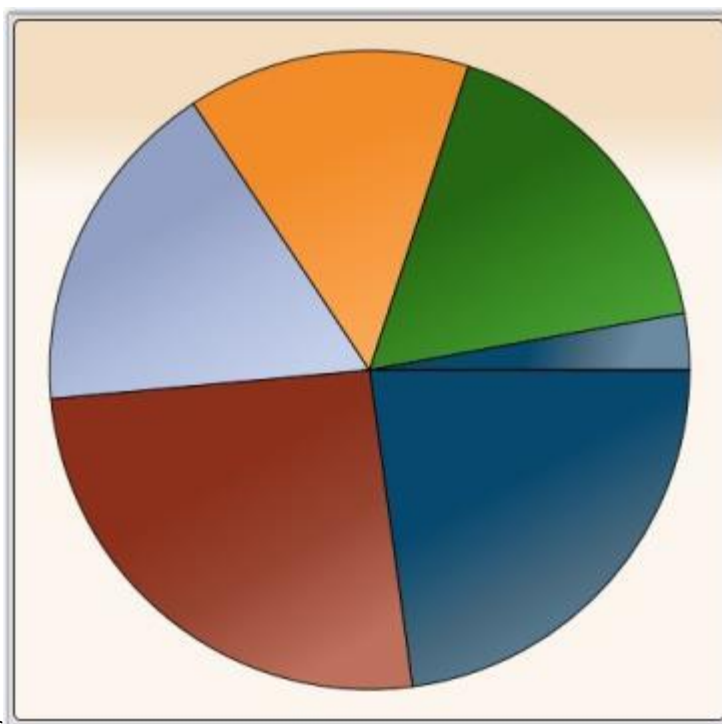
XML

```
<syncfusion:ChartSeries Type="HiLoOpenClose" Name="series1" Stroke="Black"
DataSource="{Binding}"/>
```

C#

```
ChartSeries series = new ChartSeries();
series.Type = ChartTypes.HiLoOpenClose;
```

Run the sample. The following output is provided.



![[Chart-Controlsimages/Chart-Controls_img108.jpeg]](Chart-Controlsimages/Chart-Controls_img108.jpeg)

Pie Charts

Pie Chart

A Pie Chart renders Y values as slices in a pie. These slices are rendered in proportion to the whole which is simply the sum of all the Y values in the series. Consequently, Pie Charts are used to visualize the proportional contribution (in terms of percentage or fraction) of categories of data to the whole data set. The X values in the data series will only be treated as nominal (categorical, qualitative) data. The Pie Chart can display only one *DataSet* at a time.

![[Chart-Controlsimg109]](Chart-Controlsimages/Chart-Controls_img109.jpeg)

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Pie Type Properties

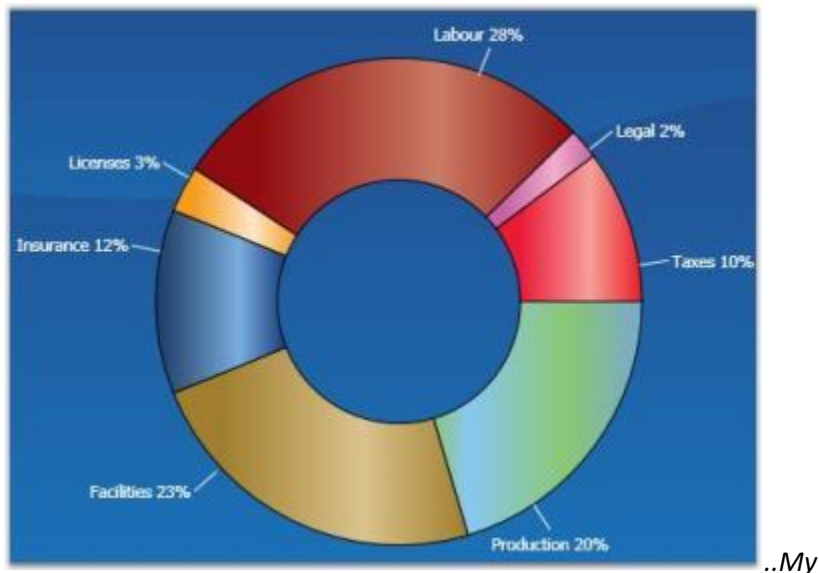
Name	Type	Container	Description
ChartPieType.ExplodedIndex	double	ChartArea	index of segment which should be leant out

Template

While setting template, the following parameters can be used.

Name	Type	Description
TickX	double	x-coordinate of sector center
TickY	double	y-coordinate of sector center
IsExploded	double	<i>true</i> if segment is leant out
ExplodRadius	double	radius to which the segment should be leant out
Geometry	Geometry	segment geometry
Interior	Brush	column color
Series	ChartSeries	reference to series-owner
AngleOfSliceRotation	double	specifies the angle (in radians) at which the segment is renderedIt is useful for creating animated templates.
StartAngle	double	specifies the angle (in radians) of one side of the pie
EndAngle	double	specifies the angle (in radians) of the other side of the pie

A sample which demonstrates Pie Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Pie Chart Demo

[Doughnut Chart](#)

Doughnut charts are pie charts with a hole, whose value is specified as the doughnut coefficient. The Doughnut Chart is best suited for presenting data in proportions.

![[Chart-Controlsimg110]](Chart-Controlsimages/Chart-Controls_img110.jpeg)

[Data Requirements](#)

Details

Number of y values per point	one
Number of points	one or more
Number of series	one or more

Doughnut Type Properties

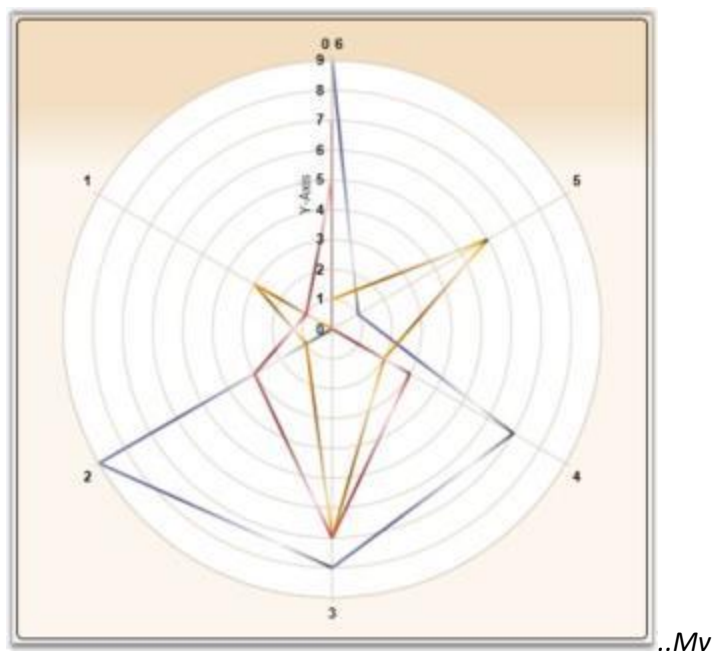
Name	Type	Container	Description
ChartDoughnutType.ExplodedIndex	int	ChartSeries	index of segment which should be leant out
ChartDoughnutType.DoughnutCoefficient	double	ChartSeries	number which shows relation of inner radius to outer radius

Template

While setting template the following parameters can be used:

Name	Type	Description
TickX	double	x-coordinate of sector center
TickY	double	y-coordinate of sector center
IsExploded	double	<i>true</i> if segment is leant out
DoughnutCoefficient	double	number which shows relation of inner radius to outer
ExplodRadius	double	radius to which the segment should be leant out
Geometry	Geometry	segment geometry
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample which demonstrates Pie Chart Types is available in the following sample installation path.



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Pie Chart Demo

Polar Chart

Essential Chart supports the implementation of Polar chart in the chart control. This chart is used to display different values and angles in the form of a graph.

A Polar Chart is a circular graph on which data is displayed in terms of values and angles. The X values define the angles at which the data points will be plotted. The Y value defines the distance of the data points from the center of the graph, with the center of the graph usually starting at 0.

It is a form of graph that allows a visual comparison between several quantitative or qualitative aspects of a situation and also allows a visual comparison between several situations that are drawn using the same axes (poles).

![[Chart-Controlsimg111]](Chart-Controlsimages/Chart-Controls_img111.jpeg)

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

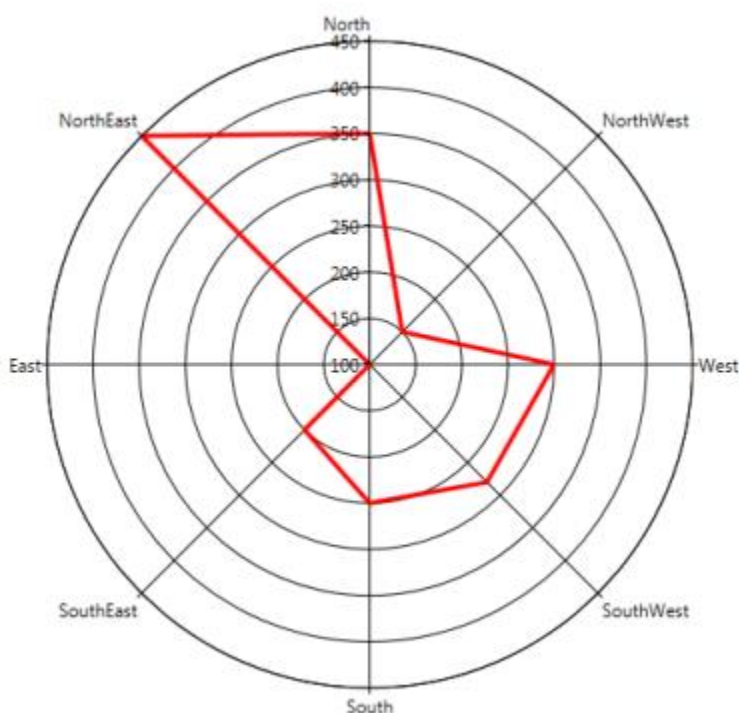
Template

While setting template the following parameters can be used.

Name	Type	Description
X1	double	x-coordinate of first point

Y1	double	y-coordinate of first point
X2	double	x-coordinate of second point
Y2	double	y-coordinate of second point
Interior	Brush	column color
Series	ChartSeries	reference to series-owner

A sample that illustrates Circular Chart Type is available in the following sample installation path.



..My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\Circular Chart Demo

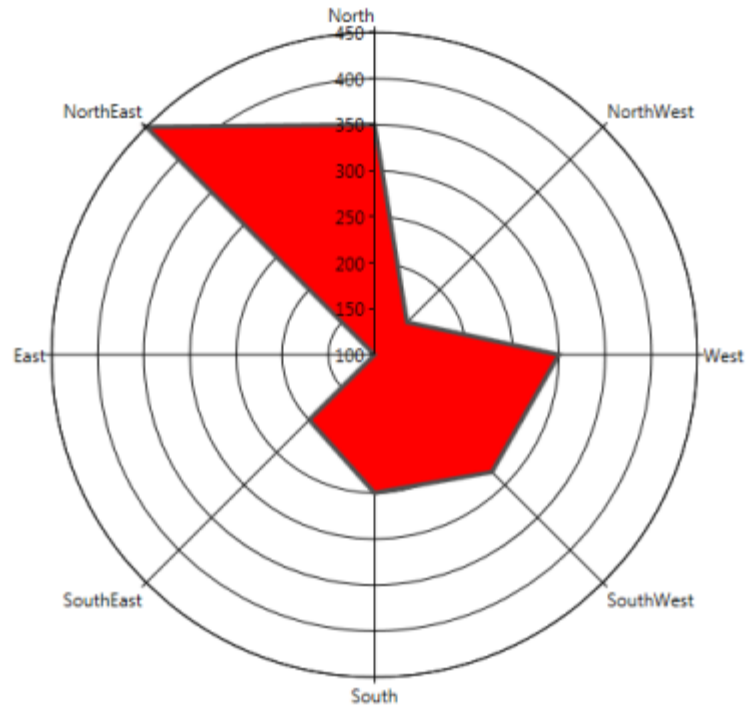
Special Support in Radar and Polar Charts

This feature provides `IsClosed` and `DrawType` support for Radar and Polar chart types. The `IsClosed` feature specifies whether the series drawn should be in closed form. This can be applied to line-type series in polar and radar charts. The `DrawType` feature specifies in which form or shape the series should be drawn.

Following are the different `DrawType` categories for radar and polar charts

Line

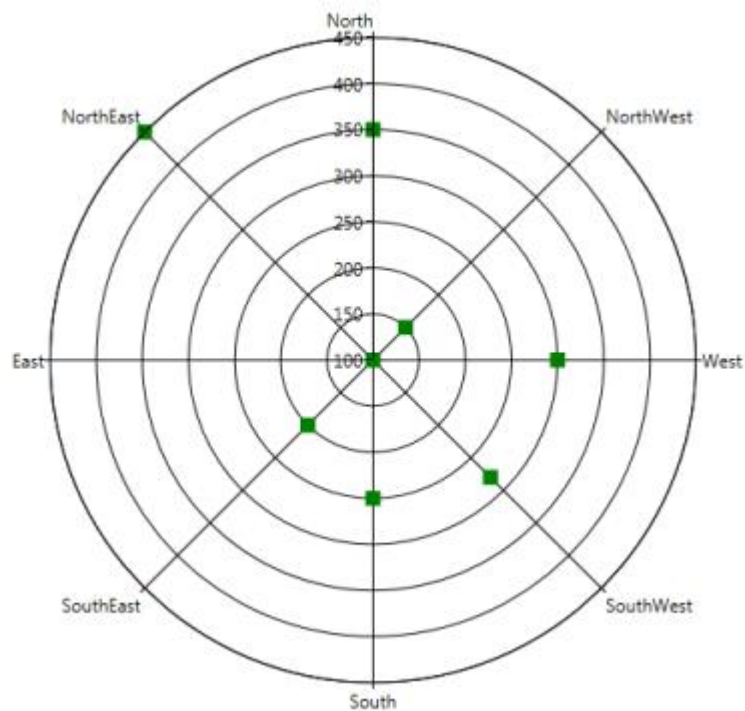
When the `DrawType` is `Line`, the series is drawn as a line segment connecting each point in the chart. The following image illustrates this.



![[Chart-Controls
img112]](Chart-Controlsimages/Chart-Controls_img112.png)

Area

When the DrawType is Area, the series is drawn as a single area segment connecting each point in the chart. The following image illustrates this.

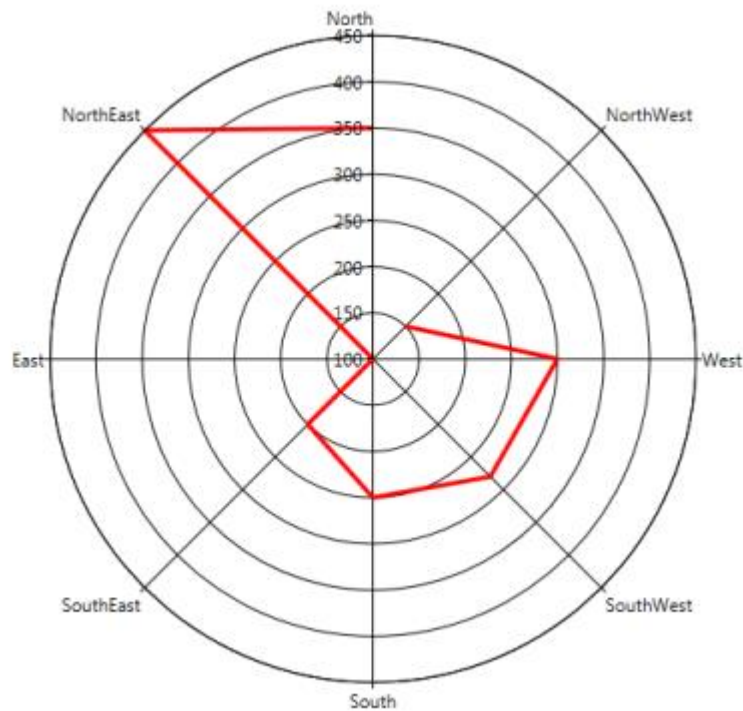


![[Chart-Controls
Controlsimages/Chart-Controls_img113.png]

img113]](Chart-

Symbol

When the DrawType is Symbol, the series is drawn as separate points as a symbol without connecting each point in the chart. The following image illustrates this.

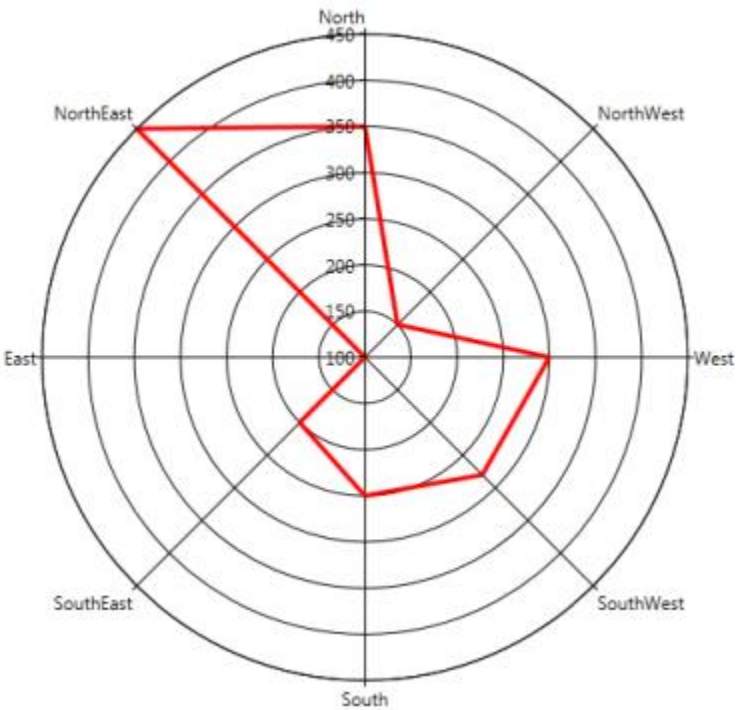


![(Chart-Controls
Controlsimages/Chart-Controls_img114.png)

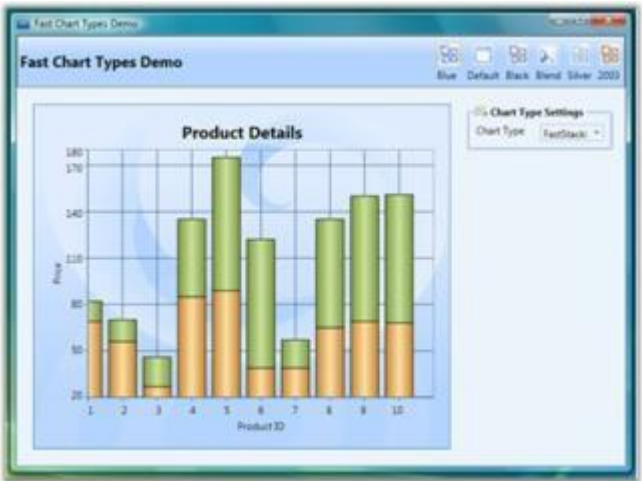
img114](Chart-

Following are the screenshots of the chart when IsClosed is set to True and False:

![[Chart-Controls



img115](Chart-Controlsimages/Chart-Controls_img115.png)



![[Chart-Controls
Controlsimages/Chart-Controls_img116.png)img116](Chart-

Properties

Property	Description	Type	Data Type
IsClosed	Used to specify how to draw the segment (closed or not closed).	Attached	Bool
DrawType	Used to specify what template is to be applied to series.	Attached	Enum (Line, Area, Symbol)

PolarSymbol	Used to assign symbol for polar charts and will be displayed when draw type is Symbol.	Attached	DataTemplate
RadarSymbol	Used to assign symbol for radar charts and will be displayed when the draw type is Symbol.	Attached	DataTemplate

Adding Support for IsClosed and DrawType in Radar and Polar Charts to an Application

XML

```
Polar Chart
<syncf:ChartArea syncf:ChartPolarType.IsClosed="True"
syncf:ChartPolarType.PolarSymbol="{StaticResource sym}"
syncf:ChartPolarType.DrawType="Line" Name="Area1">
<syncf:ChartSeries StrokeThickness="3" Interior="Red" Type="Polar"
x:Name="Series1" Data="0,100,1,150,2,300,3,50,4,214,5,166" >
</syncf:ChartSeries>
</syncf:ChartArea>
Radar Chart
<syncf:ChartArea syncf:ChartRadarType.IsClosed="True"
syncf:ChartRadarType.PolarSymbol="{StaticResource sym}"
syncf:ChartRadarType.DrawType="Line" Name="Area1">
<syncf:ChartSeries StrokeThickness="3" Interior="Red" Type="Radar"
x:Name="Series1" Data="0,100,1,150,2,300,3,50,4,214,5,166" >
</syncf:ChartSeries>
</syncf:ChartArea>
```

C#

```
Polar Chart
ChartPolarType.SetPolarSymbol(Series1, this.Resources["sym"] as
DataTemplate);
ChartPolarType.SetDrawType(Area1, ChartPolarDrawType.Line);
ChartPolarType.SetIsClosed(Area1, true);
Radar Chart
ChartRadarType.SetRadarSymbol(Series1, this.Resources["sym"] as
DataTemplate);
ChartRadarType.SetDrawType(Area1, ChartRadarDrawType.Line);
ChartRadarType.SetIsClosed(Area1, true);
```

Stacking Charts

Stacking Charts are similar to regular charts except that the Y values stack on top of each other in the specified series order. Stacking charts help visualize data that is a sum of parts, each of which is in a series.

There are different types of stacking charts.

Fast Stacking Column Charts

Fast Stacking Column charts are similar to Stacked-column charts with y-coordinate values stacked over one another, in series order allowing the chart data to be visualized as sum of series parts. The following points mark the advantages of Fast Stacking Column over Stacked-column charts:

- The Fast Stacking Column charts are rendered using drawing visuals.

- They load faster than the Stacked-column charts.
- They ensure high performance for displaying data.
- They can be used as real time charts to render huge number of data points.

The Fast Stacking Column chart is added in the Enum of type ChartTypes.

Data Requirements

Details	
Number of y values per point	one
Number of points	one or more
Number of series	one or more

Custom StackingColumn100 Properties

Name	Type	Container	Description
ChartStackingColumn100Type.ShowValueAsProbability	bool	Chart Area	The y-axis range is set between 0 and 100. If true, the y-axis range is set between 0 and 1. Default value is false.

Template

While setting template the following parameters can be used:

Name	Type	Description
X	double	x column coordinate
Y	double	y column coordinate
Width	double	column width
Height	double	column height
Interior	Brush	column color
IsUpper	boolean	true if this is upper column
IsLower	boolean	true if this is lower column
Series	ChartSeries	reference to series-owner

The following code example illustrates the usage of Fast Stacking Column charts.

XML

```
<syncfusion:ChartSeries Type="FastStackingColumn" Name="series1"
Stroke="Black" DataSource="{Binding}"/>
```

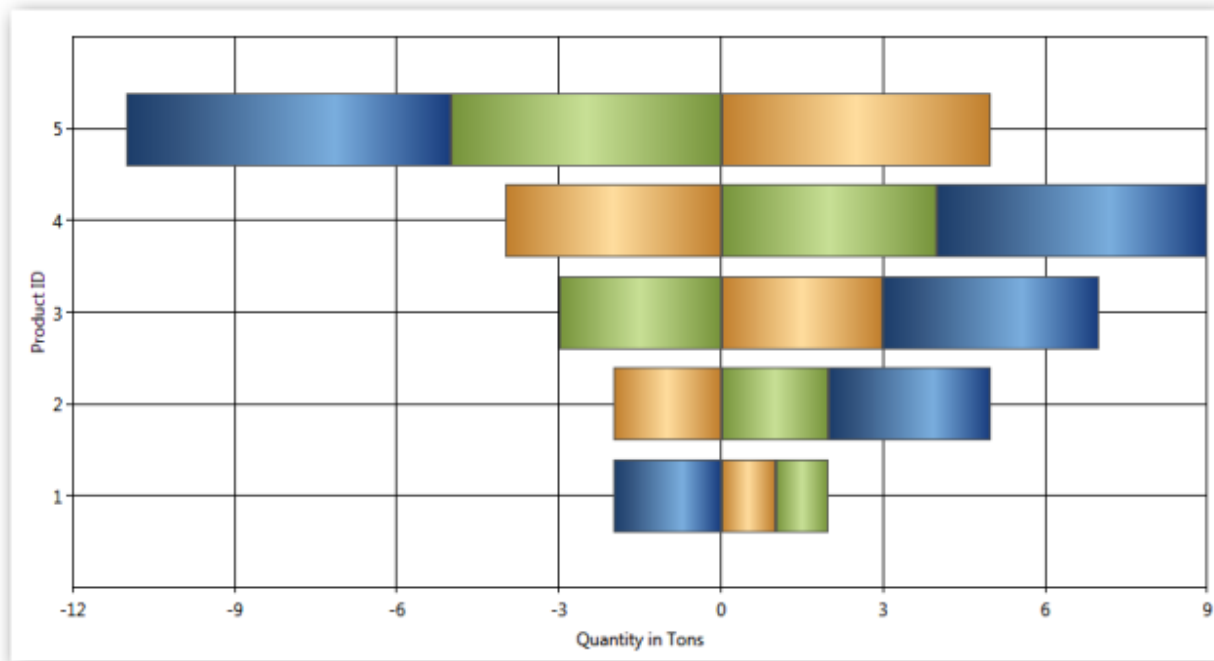

C#

```
ChartSeries series = new ChartSeries();
series.Type = ChartTypes.FastStackingColumn;
```

Run the sample.

A Fast Stacking Column chart is displayed pertaining to the data source it is bound to.

![Chart-Controls



img117](Chart-Controlsimages/Chart-Controls_img117.jpeg)

A sample which demonstrates Fast Stacking Column Chart Types is available in the following sample installation path.

..My Documents\Syncfusion\Essential Studio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Performance\Fast chart types

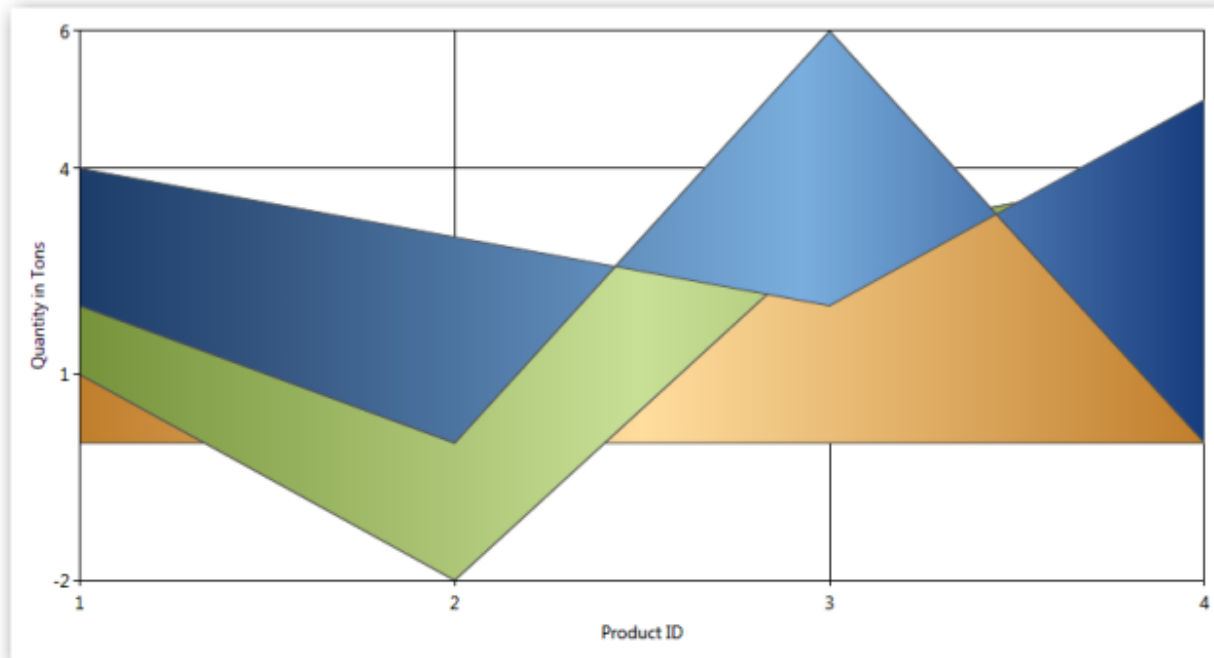
Stacking Positive and Negative Values for Stacking Chart Types

Support has been provided for stacking positive and negative values in stacking chart types. Calculation logic must be the same and it has to adhere to the general standards of stacking chart type.

Chart types are:

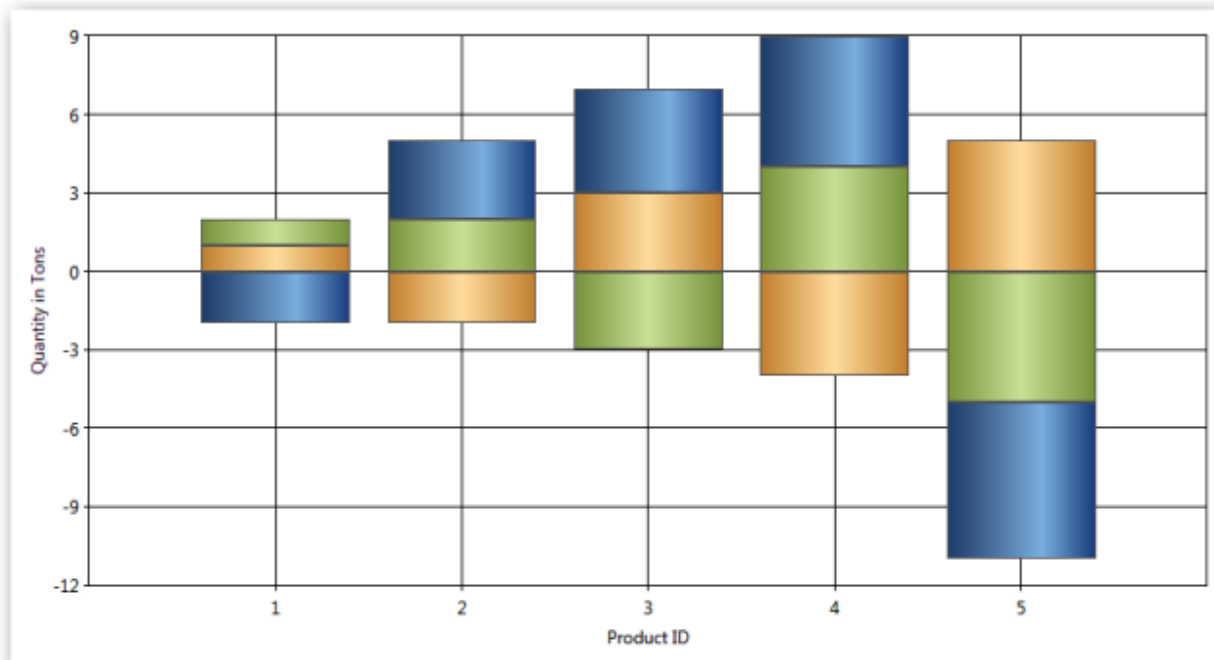
- Stacking Column
- Stacking Bar
- Stacking Area

![[Chart-Controls



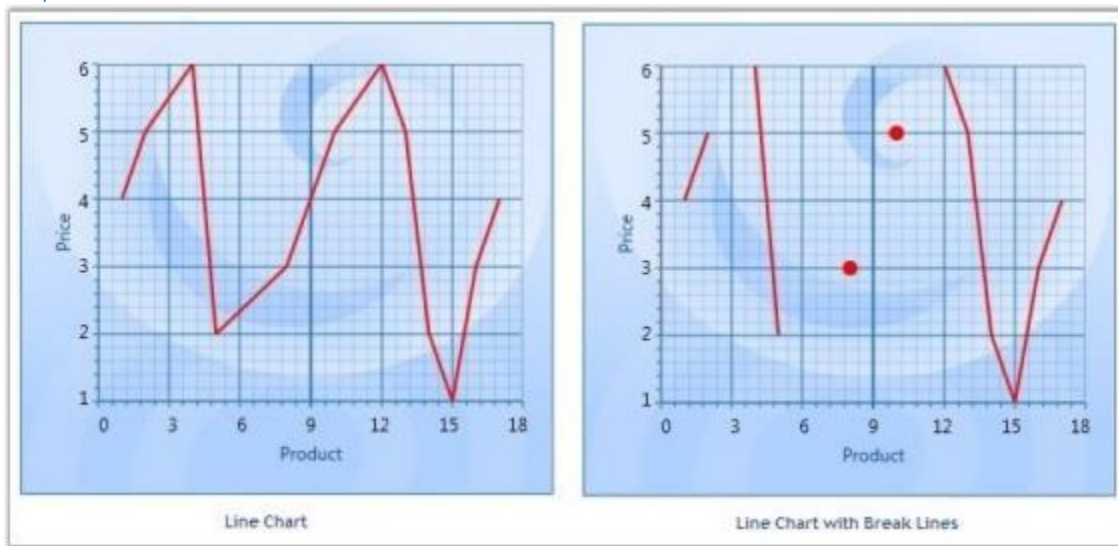
images118](Chart-Controlsimages/Chart-Controls_img118.png)

![[Chart-Controls



images119](Chart-Controlsimages/Chart-Controls_img119.png)

![[Chart-Controlsimages120](Chart-Controlsimages/Chart-Controls_img120.png)

Properties*Properties*

Property	Description	Type	Value it accepts	Any other dependencies/sub properties associated
RequiresNegativeSeriesStack	Specifies whether the positive and negative series should be stacked separately.	Bool	True/False	NA

Step Charts

Step Charts are similar to regular charts except that the values are drawn continuously, step by step without any gaps in-between.

There are two types of step charts.

Break Lines


Line charts with missing data points can be drawn with gaps for the missing points. When there is a huge gap between consecutive points, we could make the lines break for more clarity.

SetBreakLineForNonIndexedData is used to specify whether the line segments could be drawn with break lines. SetBreakLineForDoublePointsDistanceMoreThan is used to set the distance for lines that should be broken.

C#

```
ChartLineType.SetBreakLineForNonIndexedData (Chart1.Areas [0].Series [0],
true);
ChartLineType.SetBreakLineForDoublePointsDistanceMoreThan (Chart1.Areas [0].Series [0], 1);
```

If the data given are 1, 2, 4, 5, 8, 10, 12, 13, 14, 15, 16, 17 and the `SetBreakLineForDoublePointsDistanceMoreThan` is passed with a value 1, all points that don't have a point after 1 will not be drawn. Following screenshot shows the output for this data.

![[Chart-Controls

Note: This feature can be applied for both Line and Spline type charts. This can be applied for both Double and DateTime type axis values.

Break Lines for Spline Type

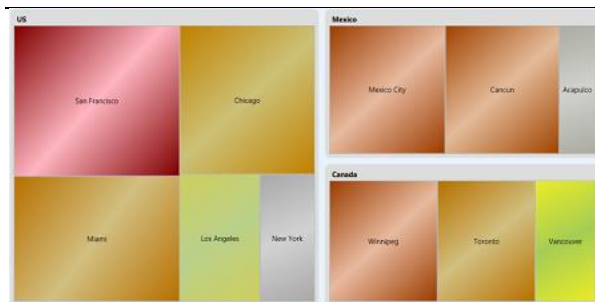
C#

```
ChartSplineType.SetBreakLineForNonIndexedData (Chart1.Areas[0].Series[0],
true);
ChartSplineType.SetBreakLineForDoublePointsDistanceMoreThan (Chart1.Areas[0].
Series[0], 1);
```

The following code could be used to specify the break distance for axis with DateTime ValueTypes.

C#

```
ChartLineType.SetBreakLineForTimeSpanPointsDistanceMoreThan (Chart2.Areas[0].
Series[0], new TimeSpan(1, 0, 0, 0));
ChartSplineType.SetBreakLineForTimeSpanPointsDistanceMoreThan (Chart2.Areas[0]
].Series[0], new TimeSpan(1, 0, 0, 0));
```



Note: This feature can be applied for non-indexed data alone. It cannot be applied for 3D charts.

Heat Map Control

HeatMapControl is a control that lays out bound child items in rectangles, whose area is based on their "weight", and whose color is based on their "color weight". It can also be bound to hierarchical data.

Property	Description
ColorCalculationLevel	specifies the items at a level (when bound to hierarchical data) for which the ColorWeight should be processedDefault value is -1, indicating this will be processed for all leaf nodes in the hierarchy. 0 indicates the top level of items in the bound hierarchy and so on. If you are adding HeatMapItems manually, make sure to set their Level property appropriately. This is a dependency property.
ColorValuePath	gets or sets a path to a value on the source object to serve as the "color weight" of the objectThis is a dependency property. This is used for items

	at all levels. It can be overridden for items at specific levels through the corresponding HeatMapItem's setting.
ColorWeightsInfo	contains computed information about the low and high colors in the bound items
HighestWeightColor	specifies the color that will be used on the item with the highest color weightThis is a dependency property. Default value is Green.
ItemsLayoutMode	specifies the mode in which items should be laid outThis setting will be applied at all levels. To customize this for specific levels, check the corresponding HeatMapItem setting. Default value is HeatMapLayoutMode.Squarified. This is a dependency property.
LowestWeightColor	specifies the color that will be used on the item with the lowest color weightThis is a dependency property. Default value is Cornsilk.
MedianWeight	specifies the "median color weight" on which the MedianWeightColor will be appliedThe valid values for this property are between 0 to 100. Default value is 50. This is a dependency property.
MedianWeightColor	specifies the color that will be used on the item with the median color weightThis is a dependency property. Default value is Yellow.
PreferredItemsPanelHeight	when bound to a grouped CollectionViewSource, this property specifies the preferred height you want to use for groupsDefault value is 300. This is a dependency property.
PreferredItemsPanelWidth	when bound to a grouped CollectionViewSource, this property specifies the preferred width you want to use for groupsDefault value is 300. This is a dependency property.
WeightValuePath	gets or sets a path to a value on the source object to serve as the "weight" of the objectThis is a dependency property. This is used for items at all levels. It can be overridden for items at specific levels, through the corresponding HeatMapItem's setting.

Gradient Support for Heat Maps

The heat map control contains the IsGradientBrush property, which supports linear gradient brushes in its interior.

Property Details

Name of Property	Description	Type of Property	Value It Accepts	Property syntax	Sub properties
IsGradientBrush	Used to set the gradient brush for a	Dependency	Bool or True/False.	IsGradientBrush="True"	LowestWeightGradient, MedianWeightGradient, HighestWeightGradient (All properties are type of brushes)

	heat map control.				
--	-------------------	--	--	--	--

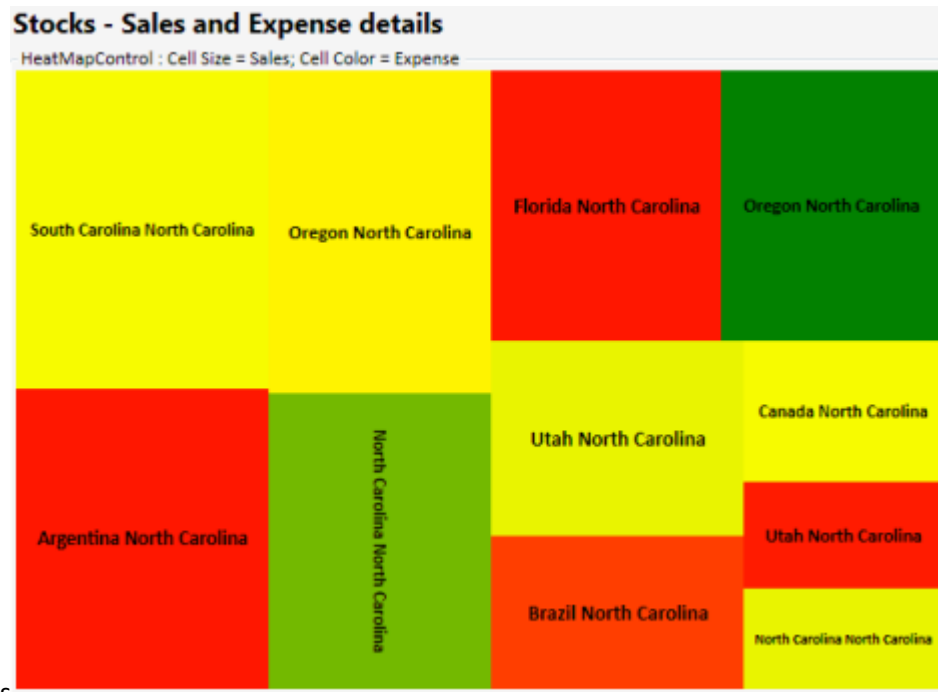
Setting Gradient Brush for a Heat Map Control

The following code is used to set a gradient brush for a heat map control.

XML

```
<syncfusion:HeatMapControl Grid.Row="0" x:Name="heatMap"
IsGradientBrush="True" LowestWeightGradient="{StaticResource
SeriesBInterior}"
MedianWeightGradient="{StaticResource SeriesAInterior}"
HighestWeightGradient="{StaticResource SeriesCInterior}" />
```

When the code runs, the following output displays.



![[Chart-Controls
img122]](Chart-Controlsimages/Chart-Controls_img122.png)

Resize the Font to Fit in the HeatMap Item

This feature automatically resizes the content to fit inside the HeatMap item, so that the font size of the content is adjusted according to the width of the Heat Map item and the orientation is changed according to the height and width of the HeatMap item.

Use Case Scenarios

It is used in the field where data are represented in a two dimensional map namely Population Survey. The data of the survey will automatically fit into the HeatMap item by resizing its font size.

![[Chart-Controlsimages123]](Chart-Controlsimages/Chart-Controls_img123.png)

Sample Link

The path to access the sample is:



<sample installation

location>\Syncfusion\EssentialStudio\Version Number\WPF\Chart.WPF\WindowSamples\3.5\Controllers\Heat Map\FlatListDemo"

Text Wrapping Behavior in HeatMapItem

The HeatMap control ships with the enhancement of customizing the text within the HeatMap item using the enum property TextIntersectAction which includes Shrink and Wrap to shrink the size of the overlapped text on resizing and to support multiline text respectively.

Use Case Scenarios

1. Avoid overlapping of text on resizing.
2. Make the text inside the HeatMapItem readable.

Properties

Property	Description	Type	Data Type
TextIntersectAction	Sets the intersect action for the text in the HeatMapItem.	Dependency	Enum

Sample Links

1. Open the Sample Browser and select the WPF platform.
2. Select the Chart product.
3. SB > Chart > Heat Map > Flat List Demo.

Adding Text Wrapping Behavior in HeatMapItem to an Application

To add the text wrapping support to the HeatMapItem:

XML

```
<syncfusion:HeatMapControl x:Name="heatMap" TextIntersectAction="Wrap"/>
```

C#

```
heatMap.TextIntersectAction = TextIntersectActions.Wrap;
```



![[Chart-Controls
img124]](Chart-Controlsimages/Chart-Controls_img124.png)

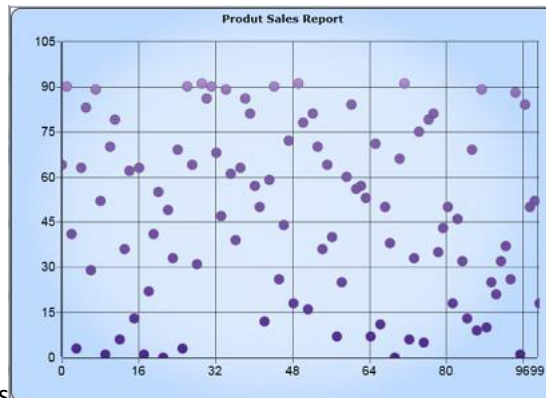
To shrink the text for the HeatMapItem.

XML

```
<syncfusion:HeatMapControl x:Name="heatMap" TextIntersectAction="Shrink"/>
```

C#

```
heatMap.TextIntersectAction = TextIntersectActions.Shrink;
```



![[Chart-Controlsimages/Chart-Controls_img125.png|img125]](Chart-Controlsimages/Chart-Controls_img125.png)

FastChart*Performance Improvements and New fast chart types*

Fast Column and Fast Scatter Charts are similar to Column and Scatter charts respectively. It uses vertical bars (called columns) and scattered circles (called ellipse) to display different values of one or more items.

The advantages of Fast Charts:

- Loads faster than other charts
- Ensures high performance for displaying data.
- They can be used as real time charts to render huge number of data points.

Use Case Scenarios

It can be used for rendering large number of points like Stock Market Analysis.

Adding FastScatter to an Application

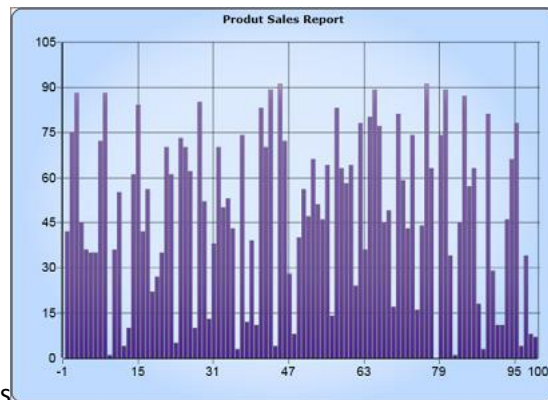
FastScatter and FastColumn Chart types can be added using the property Type in ChartSeries.

XML

```
//Add FastScatter chart type to the series.  
<sync:ChartSeries Type="FastScatter" />
```

C#

```
//Add FastScatter chart type to the series.  
Chart1.Areas[0].Series[0].Type = ChartTypes.FastScatter;
```



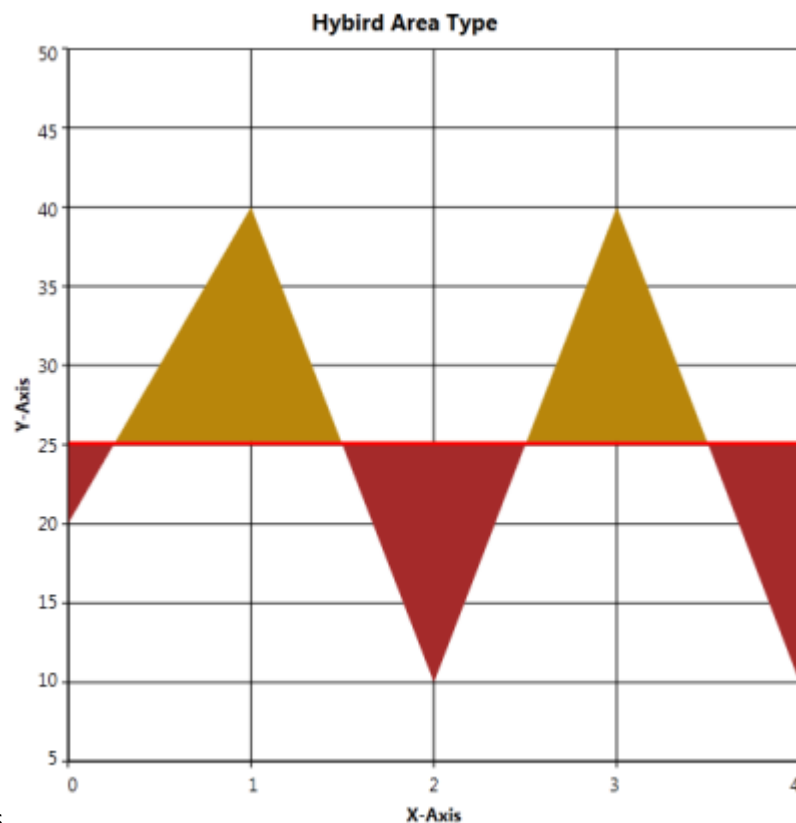
![[Chart-Controlsimages/Chart-Controls_img126.png]]images126](Chart-Controlsimages/Chart-

XML

```
//Add FastColumn chart type to the series.
<sync:ChartSeries Type="FastColumn" />
```

C#

```
//Add FastColumn chart type to the series.
Chart1.Areas[0].Series[0].Type = ChartTypes.FastColumn;
```



![[Chart-Controlsimages/Chart-Controls_img127.png]]images127](Chart-

Customizing the Height and Width of the Fast Scatter Chart

You can modify the size of the Fast Scatter chart by using the Height and Width properties of FastScatter-Type, as shown in the following code example:

XML

```
<syncfusion:ChartSeries Name="Series1" Type="FastScatter"
DataSource="{Binding ExpensiveCarDetails}" BindingPathX="Position"
BindingPathsY="Price" syncfusion:ChartFastScatterType.FastScatterHeight="30"
syncfusion:ChartFastScatterType.FastScatterWidth="30">
</syncfusion:ChartSeries>
```

C#

```
ChartFastScatterType.SetFastScatterWidth(seriesname, 30);
ChartFastScatterType.SetFastScatterHeight(seriesname, 30);
```

Custom Charts

Essential Chart WPF now comes with the Custom Chart Type that allows users to customize the appearance of the graph thereby improving the look and feel of the chart. This is achieved by initializing the "Custom" enum value to the Type property of ChartSeries control and by adding the customized chart to the ChartType property of ChartSeries.

Use Case Scenarios

Users can draw a new chart type that is not available in Essential Chart WPF. Users can create a new HybridAreaLine chart type, which is a combination of Area and Line chart types.

Adding Custom Chart Type to an Application

To set the Custom type feature in chart application:

1. Initialize the Custom Enum value to the Type property of ChartSeries control.

XML

```
<syncfusion:ChartSeries Type="Custom" />
//Initialize Chart type as Custom
chart.Areas[0].Series[0].Type = ChartTypes.Custom;
```

4. Create a new custom class, which should inherit the Segment abstract class.
5. Create another new custom class, which should be inherited by ChartType class and override the CalculateSegments method to customize the new chart type based on requirements.
6. In these CalculateSegments methods, add create new Segment object and add it to the Segments property of ChartSeries control.

CSHARP

```
public class HybridChartSegment : ChartSegment
{
    public override void Update(IChartTransformer transformer)
    {
        //transformer.TransformToVisible method has used to convert actual Chart
        points into the pixel coordinates
    }
}
```

```

.....
}
}
public class HybridChartType : ChartType
{
protected override void CalculateSegments (ChartSeries series,
ChartIndexedDataPoint[] points)
{
.....
//Add the HybirdArea segment to the Chart Series
series.Segments.Add(new HybridChartSegment(segmentPoints.ToArray(), points,
series, hybirdIntersectLine));
.....
}
}
}

```

7. Create a new object for your own custom class and initialize it to the ChartType property of ChartSeries control.

CSHARP

```

//Inititalize the customized Hybrid Area line type to ther Series.
chart.Areas[0].Series[0].ChartType = new HybridChartType();

```

8. Finally, define a Template design for your new custom chart type by initializing the Template property of ChartSeries control.

XML

```

<!--Hybird Area Line Type Template-->
<DataTemplate x:Key="HybirdAreaLineType">
<Grid>
<Grid Clip="{Binding Geometry}">
<Rectangle Fill="{Binding HighValueColor}"/>
<Rectangle Fill="{Binding LowValueColor}" Margin="{Binding HybirdMargin}"
/>
</Grid>
<Line X1="0" Y1="0" X2="{Binding Series.Area.ActualWidth}" Y2="0"
Margin="{Binding HybirdMargin}" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Stroke="{Binding LineColor}"
StrokeThickness="5"/>
</Grid>
</DataTemplate>
chart.Areas[0].Series[0].Template =
App.Current.Resources["HybirdAreaLineType"] as DataTemplate;

```

![[Chart-Controlsimages128]](Chart-Controlsimages/Chart-Controls_img128.png)

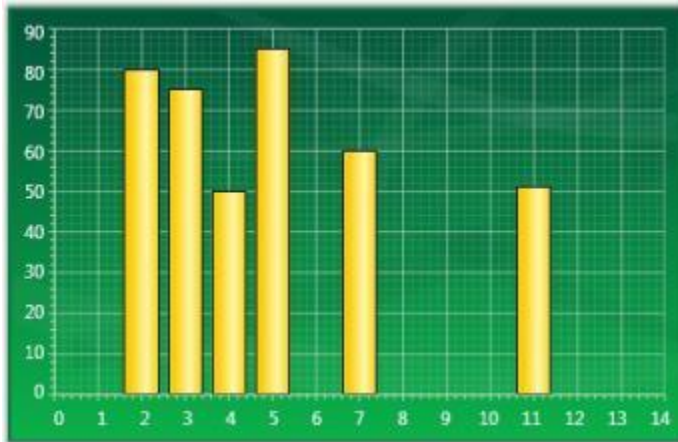
Sample Link

To run the UI Chart WPF samples:

1. On the dashboard, click the WPF Combo box and then select Run Locally Installed Samples. The WPF sample browser opens.
2. Select Chart.
3. Navigate to Chart Gallery -> CustomChart Type demo.
4. Click the Run Sample button.

Or

5. Go to



<<EssentialStudioInstalledLocation>>\Syncfusion\EssentialStudio\<Version>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Gallery\ Custom Chart Type Demo

There are two folders namely C Sharp and Visual Basic. The user can choose the required folder.

6. Run the sample by opening the project file.

Chart-Axis in WPF Chart (Classic)

Indexed X Values

By default points in a series are plotted against their X and Y values. However in some cases the X values are meaningless, they simply represent categories, and you do not want to plot the points against such X values. Such an X axis that ignores the X-values and simply uses the positional value of a point in a series is said to be indexed.

The following code example could be used to make a series as Indexed.

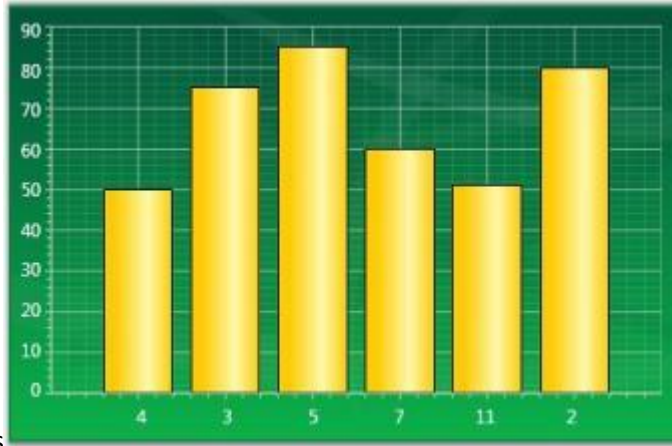
XML

```
<sfchart:ChartSeries IsIndexed="True" />
```

C#

```
//Sets the series as indexed  
series.IsIndexed = true;
```

In the following figure, the first chart shows a column chart that is not-indexed while the second chart shows a column chart whose x-axis is indexed.



![(Chart-Controlsimages/Chart-Controls_img129.jpeg)](Chart-Controlsimages/Chart-Controls_img129.jpeg)

![(Chart-Controlsimages/Chart-Controls_img130.jpeg)](Chart-Controlsimages/Chart-Controls_img130.jpeg)

ChartAxis Range

Essential Chart for WPF lets you customize the range and intervals that are displayed in the axes. This section discusses the following topics.

- Axis Range Customization
- AutoRange Customization
- Custom Range Support
- RangeCalculationMode
- VisibleRange

Axis Range Customization

You can customize the range and intervals that are displayed in the axes by using the following ChartAxis properties.

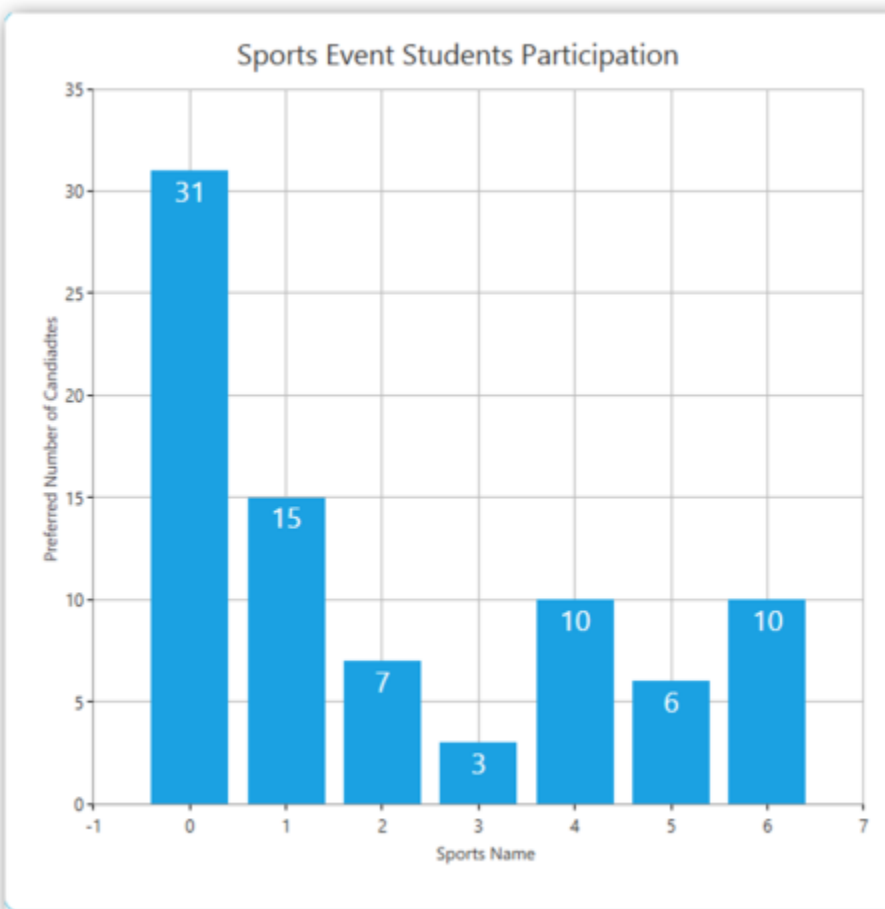
ChartAxis Properties	Description
IsAutoSetRange	A bool property specifies whether the range of the axis should be set automatically or a custom specified range will be used. Default is true (auto range).

Auto Range Customization

With the default auto range calculation setting (IsAutoSetRange=true), the following properties let you customize the automatic range calculation.

ChartAxis Properties	Description
DesiredIntervalCount	An integer property used to indicate the preferred total number of intervals to be displayed for auto range calculation.
RangePadding	An enum property, used to specify the spacing of the chart axis for auto range calculation. This property can take three values:None“ The range of the axis will be calculated from the minimum value in the data source to the maximum value in the data source.Normal“ The range of the axis will be calculated

	from the nearest multiples of the interval from the minimum and maximum values in the data source. Additional " " The range of the axis will be calculated from one interval lower than the minimum value to one interval higher than the maximum value in the data source in terms of multiples of the interval.
AdditionalRanges	This DoubleRange property is used to customize the Additional Ranges added to the Range when RangePadding is set as Additional. If Additional padding is set to (2,3), it will add 2 intervals at the start of the range and 3 intervals at the end of the range respective to its Axis ValueType.
IsSetDataValueRange	A boolean property used to calculate the axis range based on the modified data value range.



Note: The

DesiredIntervalsCount will not be taken into account when the interval is set.

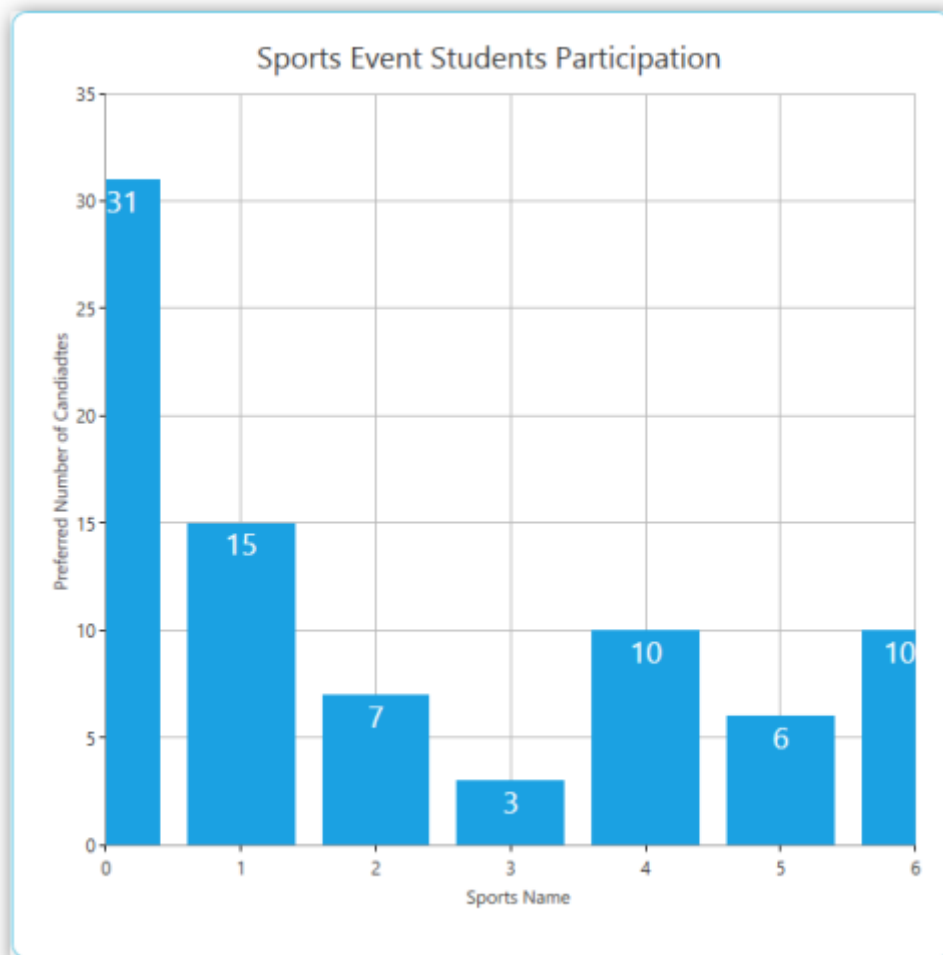
XML

```
<sfchart:Chart>
  <sfchart:ChartArea Background="LightGray" GridBackground="White">
    <sfchart:ChartArea.SecondaryAxis>
      <sfchart:ChartAxis IsAutoSetRange="True" DesiredIntervalsCount="5" RangePadding="Additional"/>
    </sfchart:ChartArea.SecondaryAxis>
  </sfchart:ChartArea>
</sfchart:Chart>
```

```
<sfchart:ChartSeries Type="Column" DataSource="{StaticResource  
SeriesData1}" BindingPathX="Year" BindingPathsY="Sales"/>  
</sfchart:ChartArea>  
</sfchart:Chart>
```

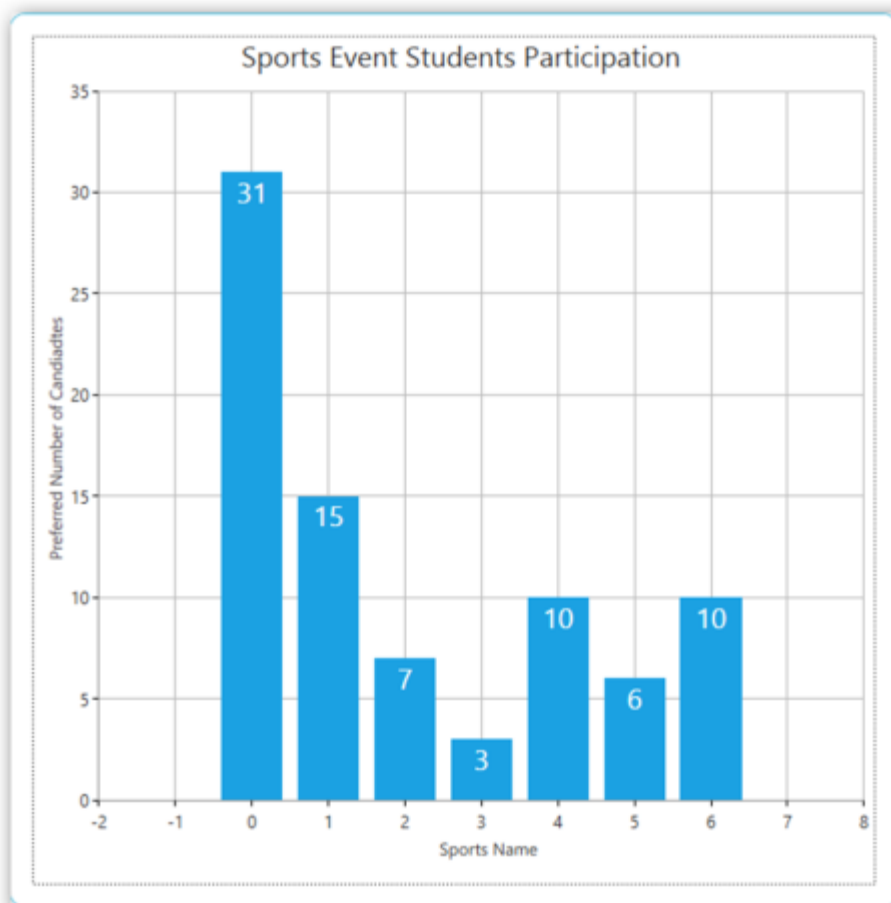
C#

```
Chart1.Areas[0].SecondaryAxis.IsAutoSetRange = true;  
Chart1.Areas[0].SecondaryAxis.RangePadding  
= ChartRangePaddingType.Additional;  
Chart1.Areas[0].SecondaryAxis.DesiredIntervalsCount = 5;
```

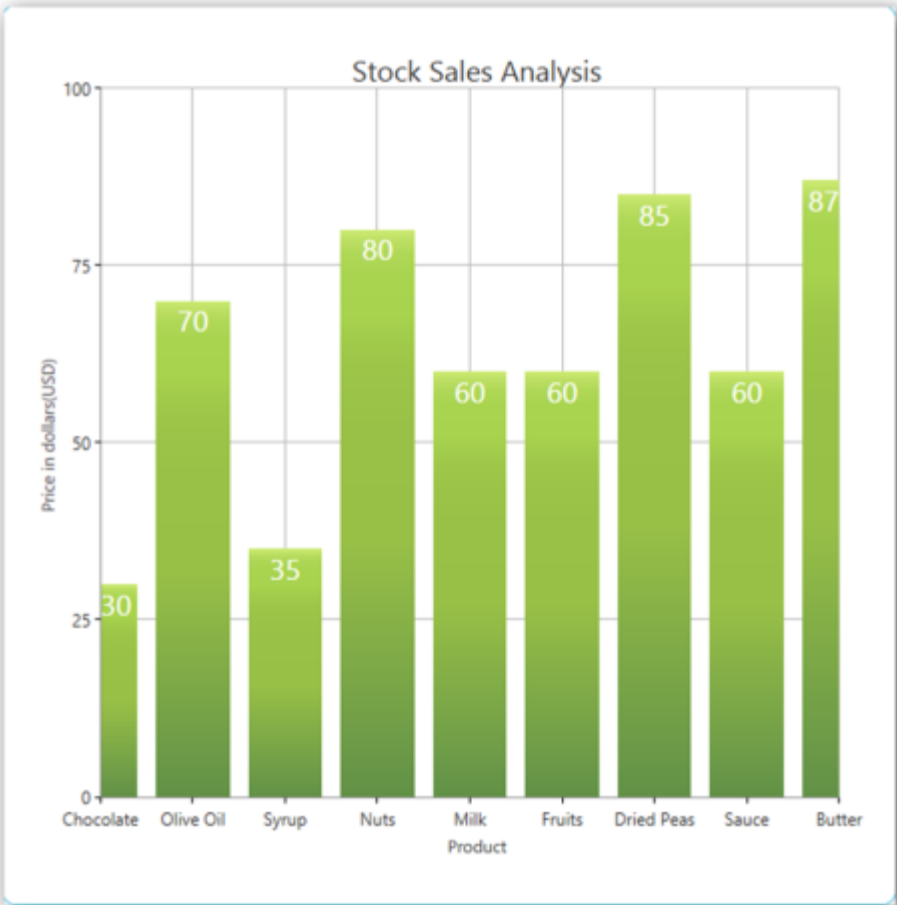


![Chart-Controls

img131](Chart-Controlsimages/Chart-Controls_img131.png)



![(Chart-Controls
img132)](Chart-Controlsimages/Chart-Controls_img132.png)



![Chart-Controls
img133](Chart-Controlsimages/Chart-Controls_img133.png)

Custom Range Support

With the auto range calculation turned off (IsAutoSetRange=false), use the following properties to format the chart on the custom range and interval length to use.

ChartAxis Properties	Description
ValueType	Specifies the metrics for the axis range. Can be Double, DateTime, or String.
Range	This DoubleRange type property specifies the custom range to use when the ValueType=Double.
Interval	An integer property that indicates the length of the intervals in the custom range specified above when the ValueType=Double.
DateTimeRange	A DateTimeRange type property that lets you specify the start and end of the axis range in DateTime when the ValueType=DateTime.
DateTimeInterval	The frequency at which intervals should be rendered. Specified in TimeSpan, when ValueType=DateTime.


MinimumInterval	An integer property that indicates the length of the MinimumInterval in the custom range specified above when the ValueType=Double. The interval will not fall below this value.
MinimumDateTimeInterval	The frequency at which the MinimumDateTimeInterval should be rendered. Specified in TimeSpan, when the ValueType=DateTime.Â The DateTime Interval will not fall below this value.

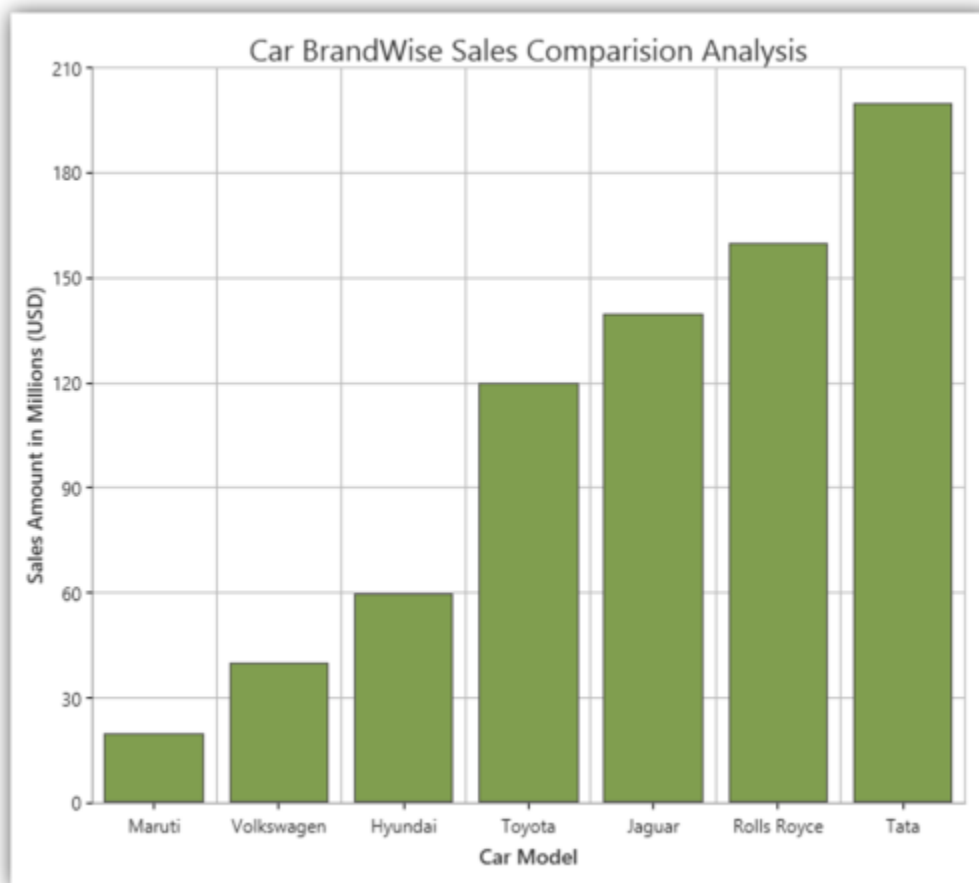
XML

```
<sfchart:Chart>
  <sfchart:ChartArea Background="LightGray" GridBackground="White">
    <sfchart:ChartArea.SecondaryAxis>
      <sfchart:ChartAxis IsAutoSetRange="False" Range="0,100" Interval="25"Minimum
Interval="25"/>
    </sfchart:ChartArea.SecondaryAxis>
    <sfchart:ChartSeries Type="Column" DataSource="{StaticResource
SeriesData1}" BindingPathX="Year" BindingPathsY="Sales"/>
  </sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
Chart1.Areas[0].SecondaryAxis.IsAutoSetRange = false;
Chart1.Areas[0].SecondaryAxis.Range = new DoubleRange(0, 100);
Chart1.Areas[0].SecondaryAxis.Interval = 25;
Chart1.Areas[0].SecondaryAxis.MinimumInterval = 25;
```

![[Chart-Controls134]](Chart-Controlsimages/Chart-Controls_img134.png)

**Note:**

The Range set for an axis with Double ValueType and DateTimeRange set for an axis with the DateTime ValueType will be taken into account only when the IsAutoSetRange property is set as false. The Interval and DateTimeInterval properties could be used to set intervals when the range calculation is done automatically or when a custom range is set. While using custom ranges, make sure that the series.IsIndexed property is set as false. This will ensure that the actual x-axis values and the range will be taken into account. Ensure that the Interval value is proportionate to the Range value. When you set a small value as the Interval to an Axis with a large Range, it results in more gridlines, generating an “Out of Memory” exception.

RangeCalculationMode

Range Calculation mode is used to position the segments between TickLines and on the TickLines. When the SegmentPosition is BetweenTicks, the segment will render between the gridlines. When the SegmentPosition is OnTicks, the segment will render on the gridlines.

Some charts need to render between TickLines so that the series segments will not be hidden. For example, Column, Bar, Stacking Column, and Stacking Bar Charts should render between the TickLine to display the whole segment. On the other hand, Line, Area, and Stacking Area charts can be rendered on the TickLines.

However, in some cases we require either of the types to be consistent. For this we could use the RangeCalculationMode property.

Chart Axis Property	Description
---------------------	-------------

RangeCalculationMode	An enum property used to specify the segment position. This property can take two values: AdjustAcrossChartTypes: Segment will render between the ticks. Segment position for this mode will be BetweenTicks. ConsistentAcrossChartTypes: Segment will render from the 1st axis. Segment position for this mode will be OnTicks. In this case, column charts will also be drawn with same range as other charts, making the first and last segments hidden partially. This is the default mode.
----------------------	---

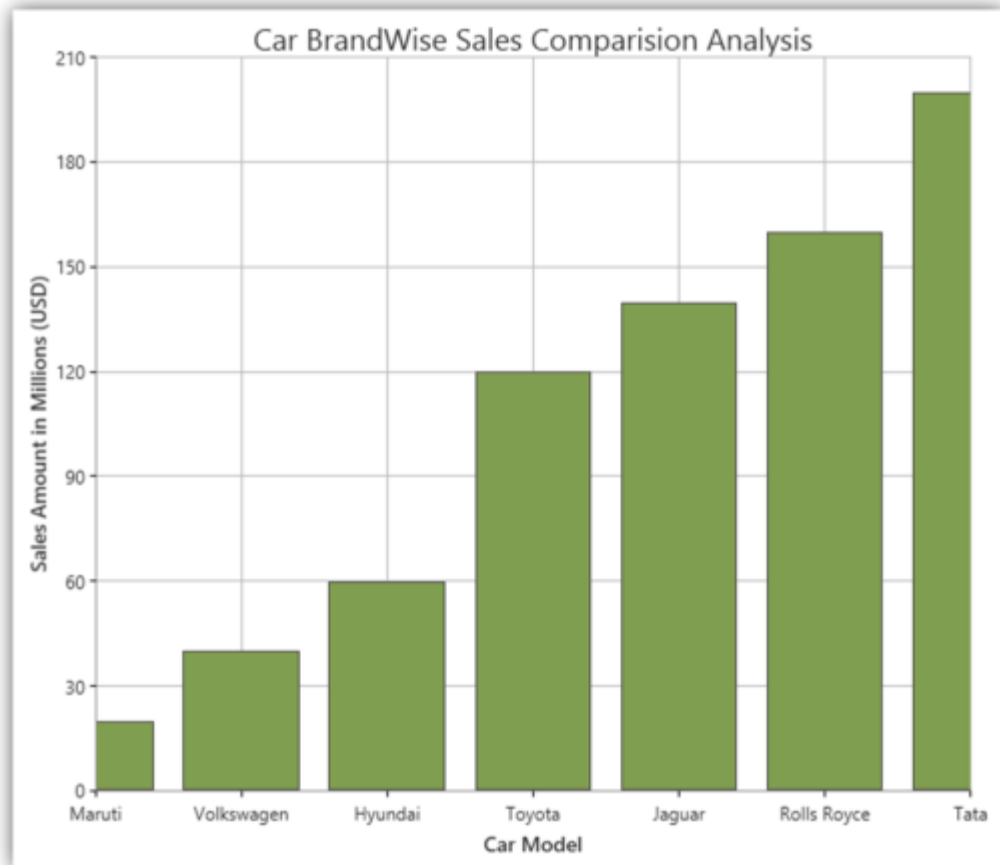
The following code example illustrates how to set the Range Calculation mode for the chart axis.

XML

```
<syncfusion:ChartArea>
  <syncfusion:ChartArea.PrimaryAxis>
    <syncfusion:ChartAxis IsAutoSetRange="True"
      RangeCalculationMode="AdjustAcrossChartTypes"/>
  </syncfusion:ChartArea.PrimaryAxis>
</syncfusion:ChartArea>
```

C#

```
// Create an instance for the Chart and Chart Area.
Chart chart = new Chart();
ChartArea area = new ChartArea();
chart.Areas.Add(area);
// Initialize the Range Calculation Mode values.
chart.Areas[0].PrimaryAxis.RangeCalculationMode =
RangeCalculationMode.ConsistentAcrossChartTypes;
```



![Chart-Controls

img135](Chart-Controlsimages/Chart-Controls_img135.png)

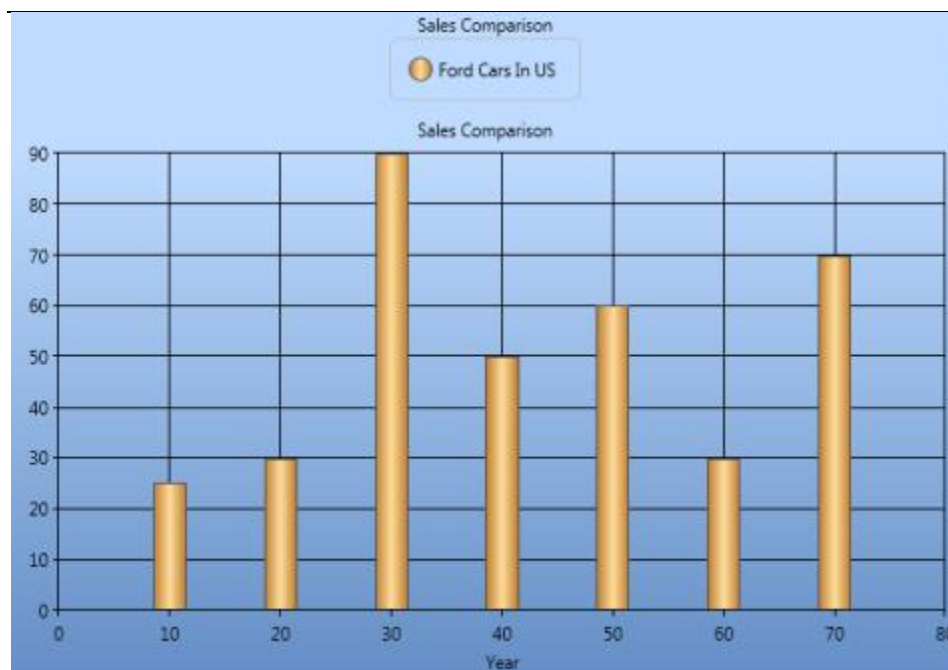
![Chart-Controlsimg136](Chart-Controlsimages/Chart-Controls_img136.png)

Visible Range

It is possible to get the Range that is visible in the ChartAxis by using the ChartAxis.VisibleRange property.

C#

```
DoubleRange range = this.Chart1.Areas[0].PrimaryAxis.VisibleRange;  
MessageBox.Show("Start " + range.Start.ToString() + ", " + "End " +  
range.End.ToString());
```



Note: Visible Range can

also be calculated for the changed value of the range by using the `Axis.RangeChanged` event. For details, see [Chart Axis Events](#)

Support to Set Axis Range Based on the Data Value

This feature implements support for sharing a data range between axes, and concurrently both axes will have appropriate visible labels rendered on them based on the modified data range at run time.

Use Case Scenarios

In a real-time data charting scenario such as a stock analysis, if the number of years to be visible in the range has been modified at run time, then the alternative y-axis range will also get modified based on the modified data range value in the x-axis.

Properties

Property	Description	Type	Data Type
<code>IsSetDataValueRange</code>	This property enables when the user wants to calculate the range based on the data point value range	Dependency Property	Binary, true/false

Sample Link

Open the Sample Browser and select the following,

1. Click User Interface > WPF and select Run Samples
2. Select the Chart product
3. Select the Chart Axis > Chart Axis Configuration demo

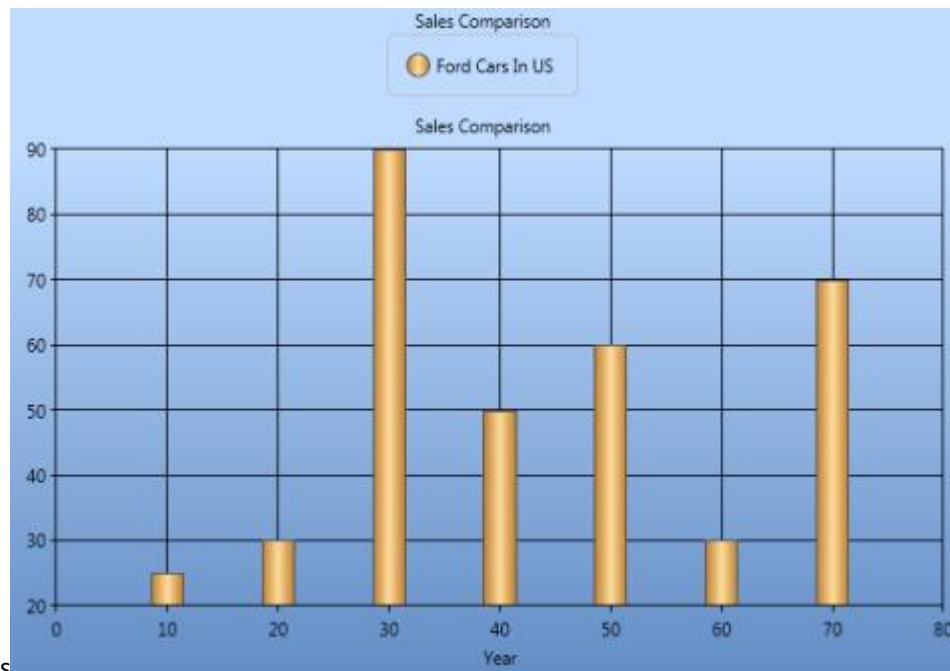
Adding `IsSetDataValueRange` to an Application

XML

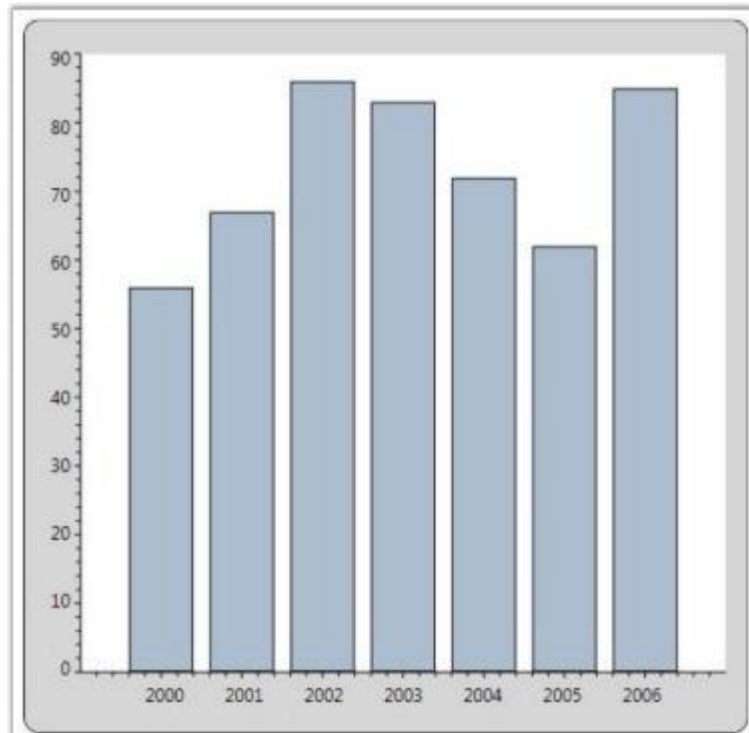
```
<syncfusion:ChartAxis x:Name="YAxis" IsAutoSetRange="True"  
RangeCalculationMode="Default" RangePadding="Normal"  
IsSetDataValueRange="True"/>
```

C#

```
this.YAxis.IsAutoSetRange = true;           this.YAxis.IsSetDataValueRange  
= true;
```



![[Chart-Controls
images137]](Chart-Controlsimages/Chart-Controls_img137.png)



!{Chart-Controls
Controlsimages/Chart-Controls_img138.png} images138}{Chart-

ChartAxis GridLines

Chart for WPF allows you to show/hide the grid lines in the Chart Area using some attached properties in ChartArea. It also allows you to customize the look and feel of the grid lines using some attached properties.

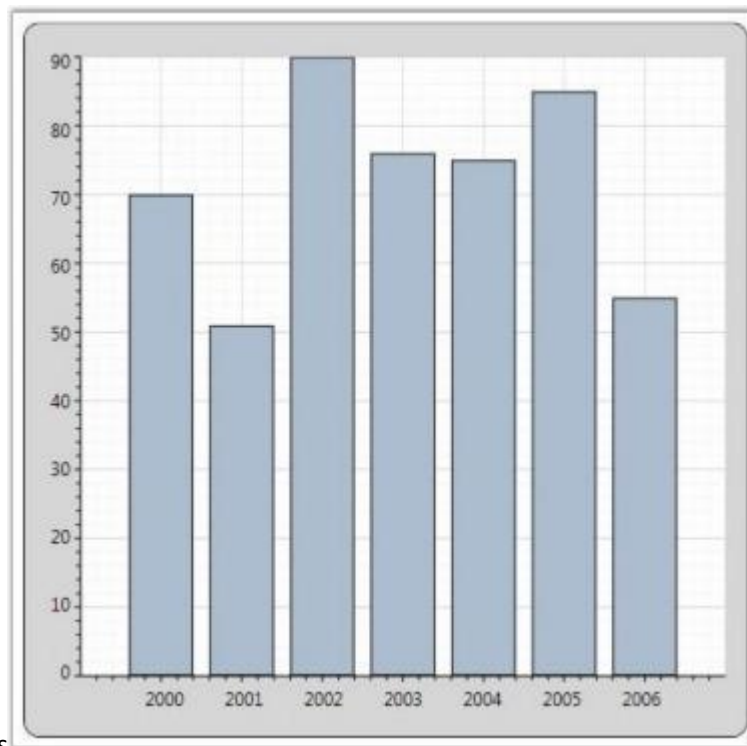
The following lines of code can be used to hide the grid lines of an axis.

XML

```
<Window.Resources>
<local:ProductSalesCollection x:Key="SeriesData1"/>
</Window.Resources>
<sfchart:Chart>
<sfchart:ChartArea Background="LightGray" GridBackground="White">
<sfchart:ChartArea.PrimaryAxis>
<sfchart:ChartAxis sfchart:ChartArea.ShowGridLines="False"/>
</sfchart:ChartArea.PrimaryAxis>
<sfchart:ChartArea.SecondaryAxis>
<sfchart:ChartAxis sfchart:ChartArea.ShowGridLines="False"/>
</sfchart:ChartArea.SecondaryAxis>
<sfchart:ChartSeries Type="Column" DataSource="{StaticResource SeriesData1}"
BindingPathX="Year" BindingPathsY="Sales"/>
</sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
ChartArea.SetShowGridLines (Chart1.Areas[0].PrimaryAxis, false);
ChartArea.SetShowGridLines (Chart1.Areas[0].SecondaryAxis, false);
```



![(Chart-Controlsimages/Chart-Controls_img139.jpeg)img139](Chart-Controlsimages/Chart-Controls_img139.jpeg)

Grid lines style can be changed using the ChartArea.GridLineStroke attached property as follows.

XML

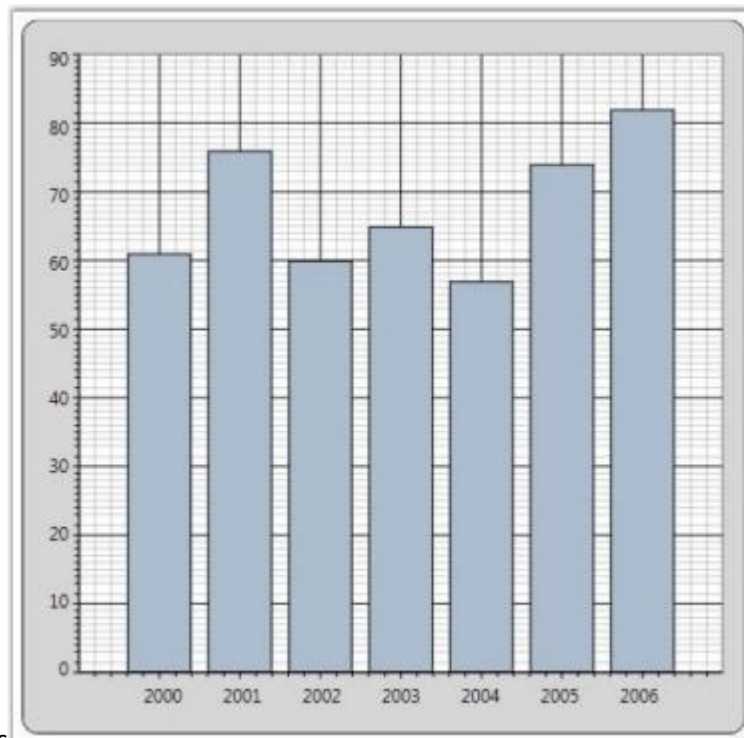
```
<sfchart:ChartArea Background="LightGray" GridBackground="White">
  <sfchart:ChartArea.PrimaryAxis>
    <sfchart:ChartAxis>
      <sfchart:ChartArea.GridLineStroke>
        <Pen Brush="#c6c6c6" Thickness="0.75">
          <Pen.DashStyle>
            <DashStyle Dashes="1,2"/>
          </Pen.DashStyle>
        </Pen>
      </sfchart:ChartArea.GridLineStroke>
    </sfchart:ChartAxis>
  </sfchart:ChartArea.PrimaryAxis>
  <sfchart:ChartArea.SecondaryAxis>
    <sfchart:ChartAxis>
      <sfchart:ChartArea.GridLineStroke>
        <Pen Brush="#C6c6c6" Thickness="0.75"/>
      </sfchart:ChartArea.GridLineStroke>
    </sfchart:ChartAxis>
  </sfchart:ChartArea.SecondaryAxis>
  <sfchart:ChartSeries Type="Column" DataSource="{StaticResource SeriesData1}"
    BindingPathX="Year" BindingPathsY="Sales"/>
</sfchart:ChartArea>
```

C#

```

Pen pen = new Pen(Brushes.LightGray, 0.75);
DashStyle style = new DashStyle();
DoubleCollection col = new DoubleCollection(2);
col.Add(1);
col.Add(2);
style.Dashes = col;
pen.DashStyle = style;
ChartArea.SetGridLineStroke(Chart1.Areas[0].PrimaryAxis, pen);
Pen pen1 = new Pen(Brushes.LightGray, 0.75);
ChartArea.SetGridLineStroke(Chart1.Areas[0].SecondaryAxis, pen1);

```



!{Chart-Controls
Controlsimages/Chart-Controls_img140.jpeg)img140}(Chart-

Small Tick Lines

The number of small ticks to be drawn per interval can be controlled using the `SmallTicksPerInterval` property. The following lines of code can be used to change the number of small ticks to draw per interval.

XML

```

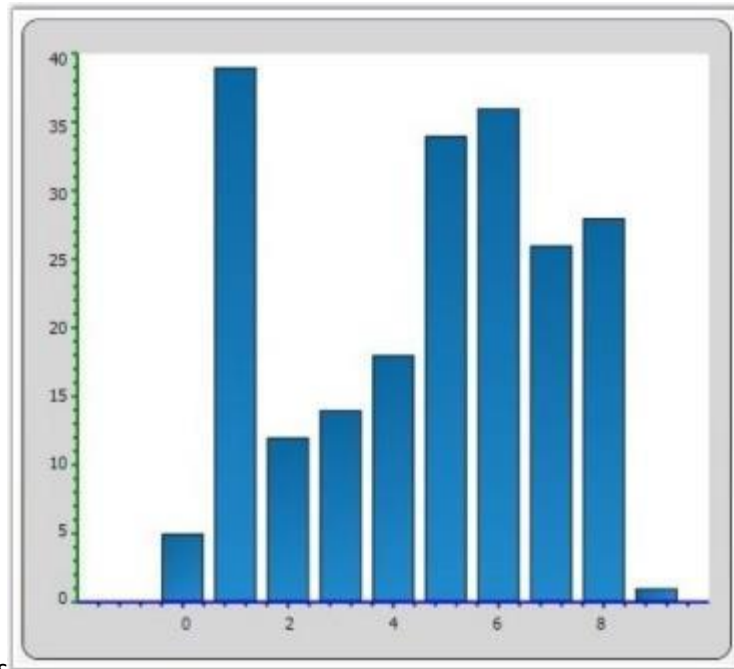
<Window.Resources>
<local:ProductSalesCollection x:Key="SeriesData1"/>
</Window.Resources>
<sfchart:Chart>
<sfchart:ChartArea Background="LightGray" GridBackground="White">
<sfchart:ChartArea.PrimaryAxis>
<sfchart:ChartAxis SmallTicksPerInterval="4"/>
</sfchart:ChartArea.PrimaryAxis>
<sfchart:ChartArea.SecondaryAxis>
<sfchart:ChartAxis SmallTicksPerInterval="6"/>
</sfchart:ChartArea.SecondaryAxis>

```

```
<sfchart:ChartSeries Type="Column" DataSource="{StaticResource SeriesData1}"
BindingPathX="Year" BindingPathsY="Sales"/>
</sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
Chart1.Areas[0].PrimaryAxis.SmallTicksPerInterval= 4;
Chart1.Areas[0].SecondaryAxis.SmallTicksPerInterval = 6;
```



![(Chart-Controlsimages/Chart-Controls_img141.jpeg)img141](Chart-

ChartAxis Lines

The ChartAxis line and tick lines can be customized using the LineStroke and TickLineStroke properties as follows.

XML

```
<sfchart:Chart>
<sfchart:ChartArea Background="LightGray" GridBackground="White">
<sfchart:ChartArea.PrimaryAxis>
<sfchart:ChartAxis sfchart:ChartArea.ShowGridLines="False">
<sfchart:ChartAxis.LineStroke>
<Pen Brush="Blue" Thickness="2"/>
</sfchart:ChartAxis.LineStroke>
<sfchart:ChartAxis.TickLineStroke>
<Pen Brush="MidnightBlue" Thickness="2"/>
</sfchart:ChartAxis.TickLineStroke>
</sfchart:ChartAxis>
</sfchart:ChartArea.PrimaryAxis>
<sfchart:ChartArea.SecondaryAxis>
<sfchart:ChartAxis sfchart:ChartArea.ShowGridLines="False">
<sfchart:ChartAxis.LineStroke>
```

```

<Pen Brush="Green" Thickness="2"/>
</sfchart:ChartAxis.LineStroke>
<sfchart:ChartAxis.TickLineStroke>
<Pen Brush="Green" Thickness="2"/>
</sfchart:ChartAxis.TickLineStroke>
</sfchart:ChartAxis>
</sfchart:ChartArea.SecondaryAxis>
<sfchart:ChartSeries Type="Column" DataSource="{StaticResource SeriesData1}"
BindingPathX="Year" BindingPathsY="Sales"/>
</sfchart:ChartArea>
</sfchart:Chart>

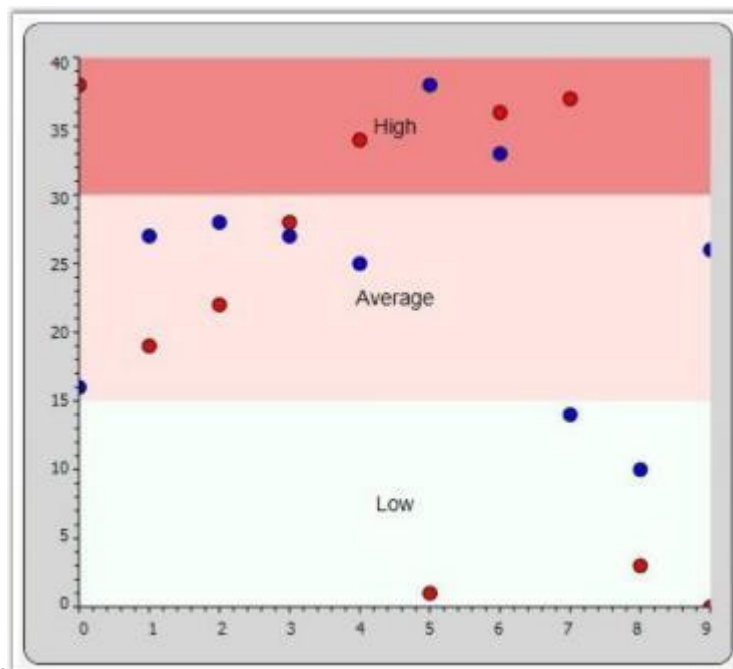
```

C#

```

Pen pen = new Pen(Brushes.Blue, 2);
Chart1.Areas[0].PrimaryAxis.LineStroke = pen;
Chart1.Areas[0].PrimaryAxis.TickLineStroke = pen;
Pen pen1 = new Pen(Brushes.Green, 2);
Chart1.Areas[0].SecondaryAxis.LineStroke = pen1;
Chart1.Areas[0].SecondaryAxis.TickLineStroke = pen1;
ChartArea.SetShowGridLines(Chart1.Areas[0].PrimaryAxis, false);
ChartArea.SetShowGridLines(Chart1.Areas[0].SecondaryAxis, false);

```



![[Chart-Controls
Controls/images/Chart-Controls_img142.jpeg)](Chart-
img142)]

Chart Striplines

Chart for WPF enables the user to highlight a specific area of the chart by adding Striplines to a ChartAxis. The strip lines length and width can be customized, a text label can be specified and the look and feel can be customized too.

Start, Frequency and Width of the line

The following properties let you customize the start, frequency and width of the lines.

ChartStripLine Properties	Description
StartFromAxis	A bool property used to specify whether the stripline starts from the beginning of the axis.
Offset	If StartFromAxis is true, this property lets you add an Offset to that starting location.
Start	If StartFromAxis is false, this property specifies where exactly the stripline should start.
Width	This double property lets you specify the width of each (optionally repeating) strip line.
RepeatEvery	Specifies how often this stripline should repeat or the frequency.
RepeatUntil	A double property that specifies until where the striplines should repeat.

Appearance

Properties	Description
Interior	A brush property used to specify the color to be filled in the stripline area.
Stroke	A Pen type used to specify the border style around the strip area.

Text Label

Properties	Description
Text	A FormattedText property that lets you specify the text and appearance of the label for this strip line.
TextAlignment	Lets you specify how this text label should be aligned.
VerticalText	Boolean property specifying if the text should be rendered vertically.
TextBackground	A Brush for the text background.

*Code Example***XML**

```

<sfchart:Chart>
  <sfchart:ChartArea Background="LightGray" GridBackground="White">
    <sfchart:ChartArea.PrimaryAxis>
      <sfchart:ChartAxis sfchart:ChartArea.ShowGridLines="False">
      </sfchart:ChartAxis>
    </sfchart:ChartArea.PrimaryAxis>
    <sfchart:ChartArea.SecondaryAxis>
      <sfchart:ChartAxis sfchart:ChartArea.ShowGridLines="False">
      <sfchart:ChartAxis.Striplines>
        <sfchart:ChartStripLine x:Name="stripl1" Interior="MintCream" Offset="0"
          StartFromAxis="True" Width="15">

```

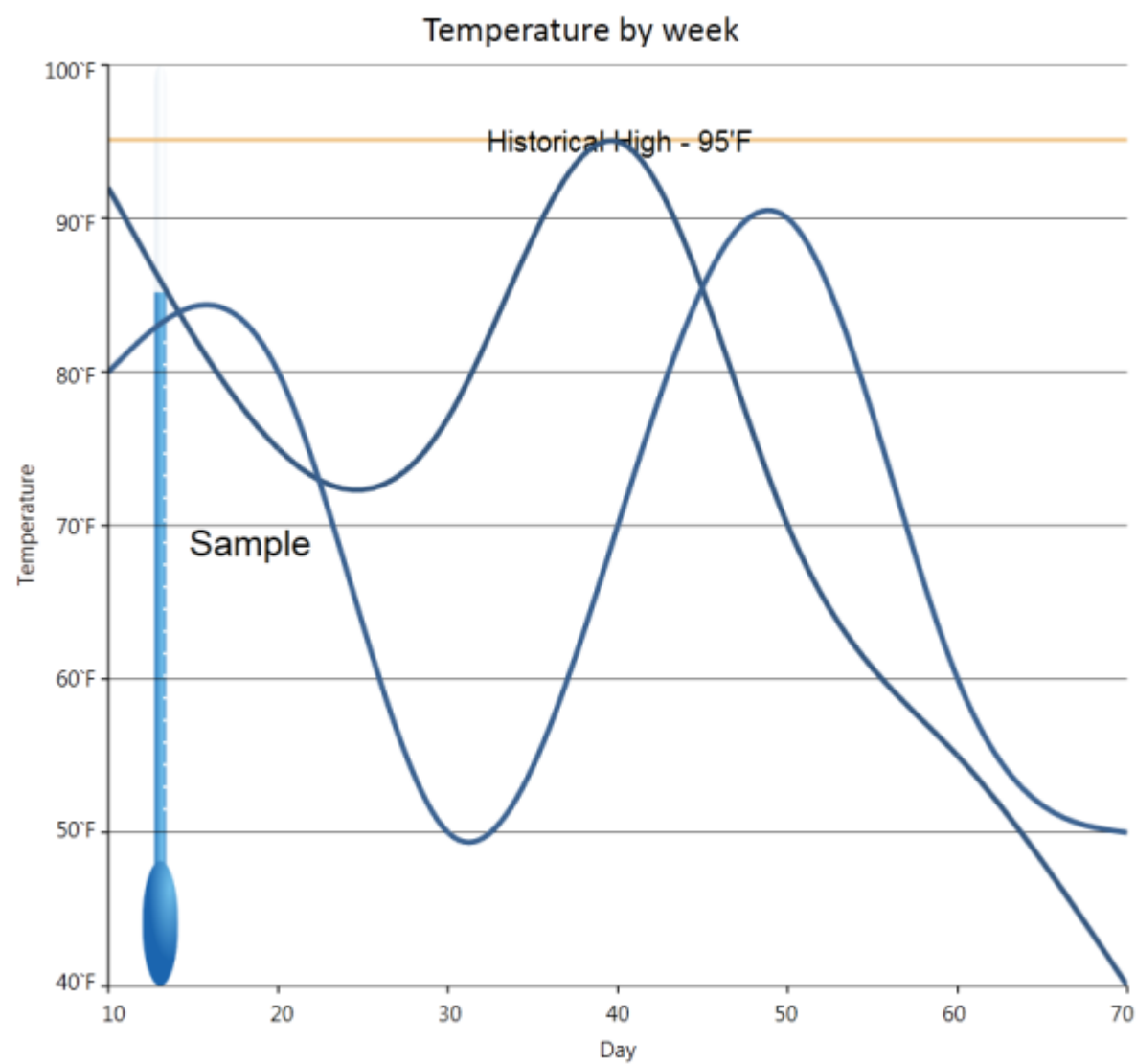
```
</sfchart:ChartStripLine>
<sfchart:ChartStripLine x:Name="strip2" Interior="MistyRose" Offset="15"
StartFromAxis="True" Width="15">
</sfchart:ChartStripLine>
<sfchart:ChartStripLine x:Name="strip3" Interior="LightCoral" Offset="30"
StartFromAxis="True" Width="10">
</sfchart:ChartStripLine>
</sfchart:ChartAxis.StripLines>
</sfchart:ChartAxis>
</sfchart:ChartArea.SecondaryAxis>
<sfchart:ChartSeries Name="Series1" Interior="Red" Type="Scatter"
DataSource="{Binding Source = {StaticResource myXmlData},
XPath=Products/Product}" BindingPathX="X" BindingPathsY="Series1Y"/>
<sfchart:ChartSeries Name="Series2" Interior="Blue" Type="Scatter"
DataSource="{Binding Source = {StaticResource myXmlData},
XPath=Products/Product}" BindingPathX="X" BindingPathsY="Series1Y1"/>
</sfchart:ChartArea>
</sfchart:Chart>
```

To set the labels for the striplines, use the following code example.

C#

```
strip1.Text = new FormattedText("Low", CultureInfo.CurrentCulture,
FlowDirection.LeftToRight, new Typeface("Times New Roman"), 20,
Brushes.Black);
strip2.Text = new FormattedText("Average", CultureInfo.CurrentCulture,
FlowDirection.LeftToRight, new Typeface("Times New Roman"), 20,
Brushes.Black);
strip3.Text = new FormattedText("High", CultureInfo.CurrentCulture,
FlowDirection.LeftToRight, new Typeface("Times New Roman"), 20,
Brushes.Black);
```

)



img143]([Chart-Controlsimages/Chart-Controls_img143.jpeg](#))

[Stripline Customization](#)

Essential Chart for WPF now supports customizing the position of strip line text. Users can set the strip line text using the `TextOffsetX` and `TextOffsetY` properties.

[Property Details](#)

Name of Property	Description	Type of Property	Value It Accepts	Property Syntax
<code>TextOffsetX</code>	Sets the horizontal position of strip line text.	Dependency	double	<code>csY.TextOffsetX = 10;</code>
<code>TextOffsetY</code>	Sets the vertical position of strip line text.	Dependency	double	<code>csY.TextOffsetY = 20;</code>

Setting the Position of Strip Line Text

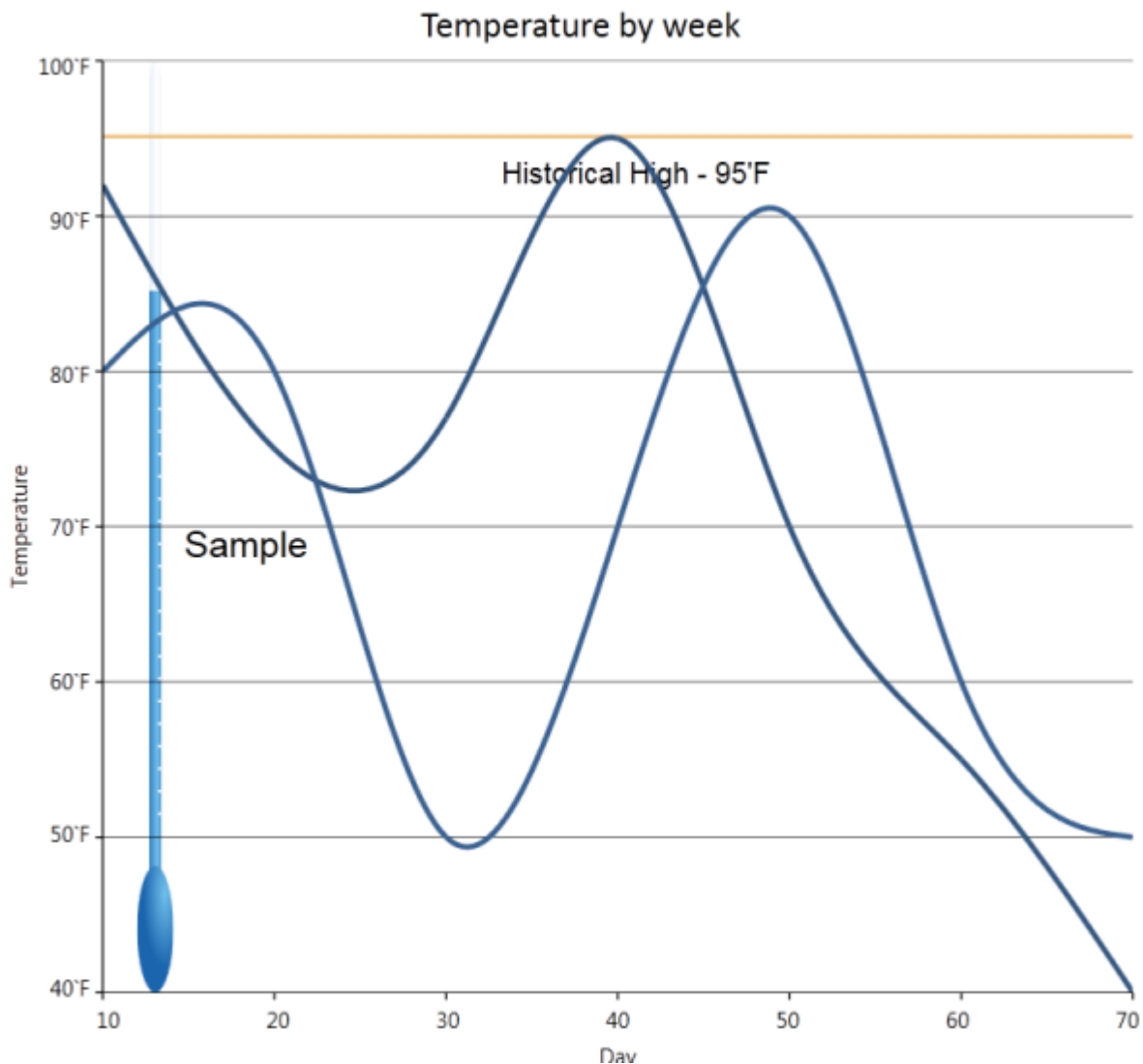
The following code is used to set the position of strip line text.

C#

```
ChartStripLine csY = new ChartStripLine();
//Set the stripline text
csY.Text = new FormattedText("Historical High - 95°F",
CultureInfo.CurrentCulture, FlowDirection.LeftToRight, new
Typeface("Arial"), 10, Brushes.Black);
//Set the stripline position
csY.TextOffsetX = 10;
csY.TextOffsetY = 20;
```

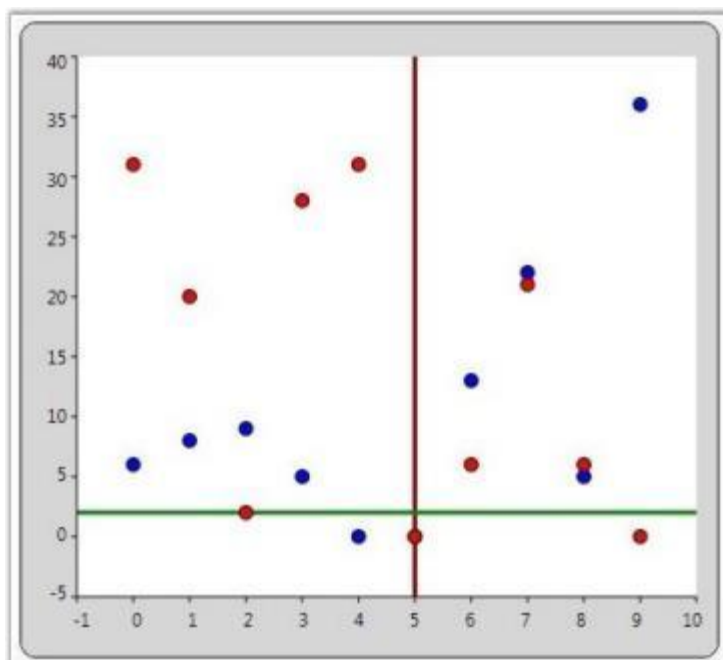
Before setting the offset:

![Chart-Controls



img144](Chart-Controlsimages/Chart-Controls_img144.png)

After setting the offset:



![[Chart-Controls
Controls/images/Chart-Controls_img145.png]](Chart-
img145)](Chart-

ChartAxis OriginLines

Custom Origin Lines can be drawn for the X and Y axis by using the ChartArea.ShowOriginLine attached property as follows. The ChartArea.OriginLineStroke can be used to customize the look and feel.

XML

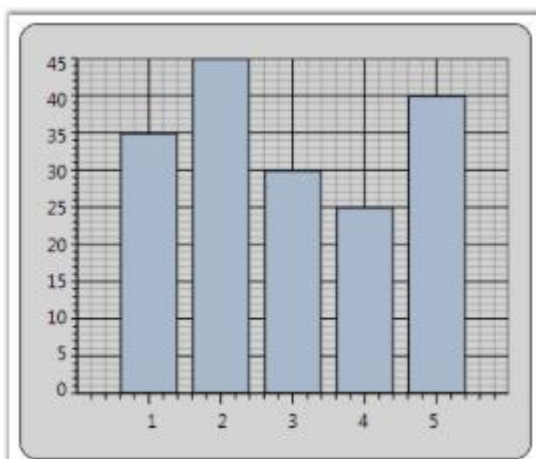
```
<sfchart:ChartArea GridBackground="White">
  <sfchart:ChartArea.PrimaryAxis>
    <sfchart:ChartAxis RangePadding="Additional"
      sfchart:ChartArea.ShowGridLines="False"
      sfchart:ChartArea.ShowOriginLine="True" Origin="5"
      SmallTicksPerInterval="0">
      <sfchart:ChartArea.OriginLineStroke>
        <Pen Brush="Maroon" Thickness="3"/>
      </sfchart:ChartArea.OriginLineStroke>
    </sfchart:ChartAxis>
  </sfchart:ChartArea.PrimaryAxis>
  <sfchart:ChartArea.SecondaryAxis>
    <sfchart:ChartAxis RangePadding="Additional"
      sfchart:ChartArea.ShowGridLines="False"
      sfchart:ChartArea.ShowOriginLine="True" Origin="2"
      SmallTicksPerInterval="0">
      <sfchart:ChartArea.OriginLineStroke>
        <Pen Brush="Green" Thickness="3"/>
      </sfchart:ChartArea.OriginLineStroke>
    </sfchart:ChartAxis>
  </sfchart:ChartArea.SecondaryAxis>
  <sfchart:ChartSeries Type="Scatter" DataSource="{StaticResource
    SeriesData2}" BindingPathX="X" BindingPathsY="Y" Interior="Blue"/>
  <sfchart:ChartSeries Type="Scatter" DataSource="{StaticResource
    SeriesData1}"
    BindingPathX="X" BindingPathsY="Y" Interior="Red"/>
</sfchart:ChartArea>
```

C#

```

ChartArea.SetShowOriginLine (Chart1.Areas[0].PrimaryAxis, true);
ChartArea.SetShowOriginLine (Chart1.Areas[0].SecondaryAxis, true);
Pen pen = new Pen (Brushes.Maroon, 3);
ChartArea.SetOriginLineStroke (Chart1.Areas[0].PrimaryAxis, pen);
Pen pen1 = new Pen (Brushes.Green, 3);
ChartArea.SetOriginLineStroke (Chart1.Areas[0].SecondaryAxis, pen1);
Chart1.Areas[0].PrimaryAxis.Origin = 5;
Chart1.Areas[0].SecondaryAxis.Origin = 2;

```



![[Chart-Controls|img146]](Chart-Controlsimages/Chart-Controls_img146.jpeg)

Chart-Axis in WPF Chart (Classic) Ticks

Chart for WPF allows to customize the length of the Axis Ticks and SmallTicks by using the ChartAxis.TickSize and ChartAxis.SmallTickSize properties.

ChartAxis Property	Description
TickSize	gets or sets the length of the axis tick
SmallTickSize	gets or sets the length of the axis small tick

XML

```

<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis TickSize="7" SmallTickSize="4" />
</syncfusion:ChartArea.PrimaryAxis>

```

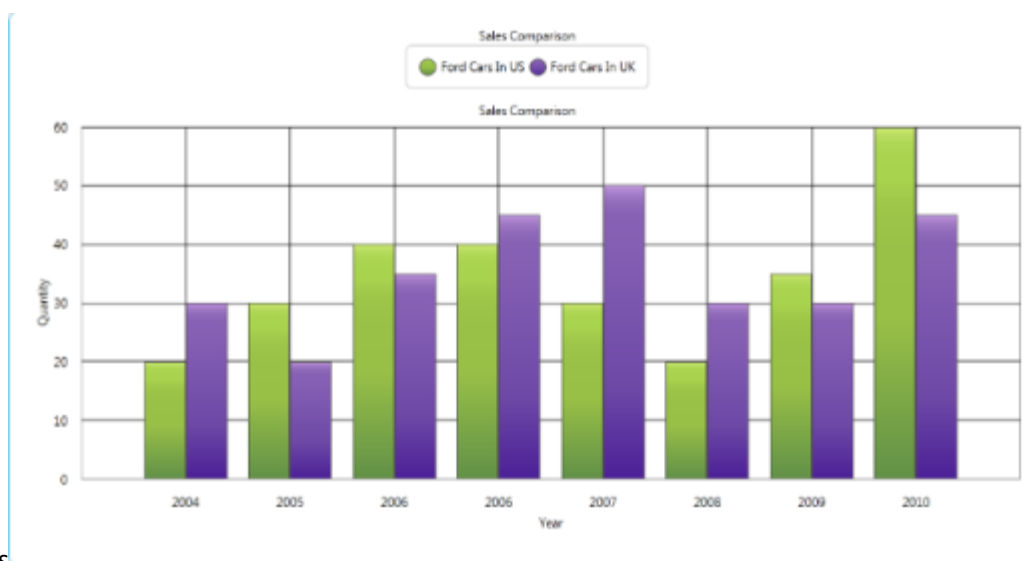
C#

```

// Sets the Axis tick size.
area.PrimaryAxis.TickSize = 7;
area.PrimaryAxis.SmallTickSize = 3;

```

The following image illustrates Chart with Axis TickSize and SmallTickSize set.



![[Chart-Controls

img147]](Chart-Controlsimages/Chart-Controls_img147.jpeg)

Support for customizing the Label position and TickLines along with the OriginAxis

This feature supports customizing the label position and TickLines along with the OriginAxis. Labels can be moved based on the AxisLabels type. Position of label will be changed based on LabelsPosition, and TickLines will be changed based on TickLinesPosition.

Properties

Property	Description	Type	Data Type	Reference links
ChartAxisLabels	Used to set the position of axis label.	Dependency Property	AxisLabels	NA
ChartTickLinesPosition	Used to set the position of tickline.	Dependency Property	TickLinesPosition	NA
ChartLabelsPoition	Used to set the position of chart label.	Dependency Property	LabelsPosition	NA
TickLinesRange	Used to set the size of TickLine to be placed inside the OriginAxis.	Dependency Property	Double	NA
SmallTickLinesRange	Used to set the size of SmallTickLine to be placed inside the OriginAxis.	Dependency Property	Double	NA

Customizing the Label position and TickLines along with the OriginAxis

You can customize the Label position and TickLines along with the OriginAxis using the properties given in the above table.

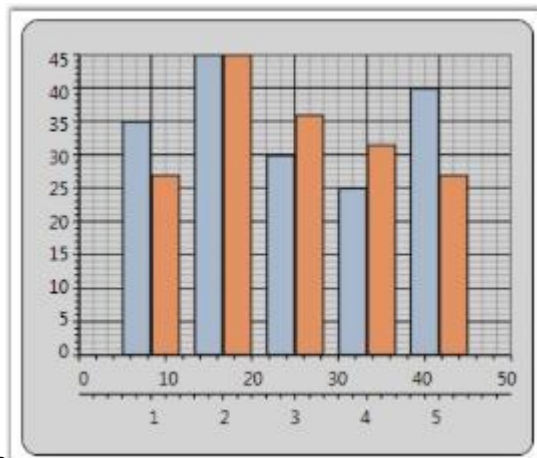
The following code illustrates this:

XML

```
<syncfusion:ChartArea.SecondaryAxis >
<syncfusion:ChartAxis x:Name="axis" ChartAxisLabels="NextToAxis"
ChartLabelPosition="Above" ChartTickLinesPosition="Inside"
TickLinesRange="0.5" SmallTickLinesRange="0.5"/>
</syncfusion:ChartArea.SecondaryAxis>
```

C#

```
XAxis.ChartAxisLabels = AxisLabels.NextToAxis;
XAxis.ChartLabelPosition = LabelsPosition.Above;
XAxis.ChartTickLinesPosition = TickLinesPosition.Inside;
```



![[Chart-Controlsimages/Chart-Controls_img148.png]]

[Sample Link](#)

To view samples:

1. Open the Syncfusion Dashboard.
2. Click the WPF drop-down list and select Run Locally Installed Samples.
3. Navigate to Chart Axis > Chart Axis Configuration > Demo.

Chart-Axis in WPF Chart (Classic) Orientation

WPF Chart enables to set the orientation of the ChartAxis. The default orientation of the ChartAxis is Horizontal. This property is mostly used in Multiple Axes Scenarios.

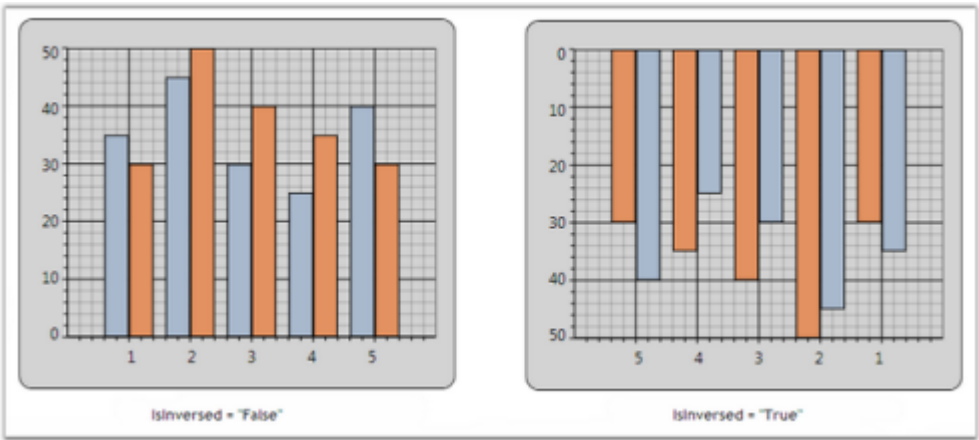
ChartAxis Property	Description
Orientation	gets / sets the orientation of the axis.

XML

```
<syncfusion:ChartArea Name="area">
<syncfusion:ChartSeries Name="series" Data=" 1 35 2 45 3 30 4 25 5 40" />
<syncfusion:ChartSeries Name="series1" Data=" 1 30 2 50 3 40 4 35 5 30" >
<syncfusion:ChartSeries.YAxis>
```

```
<syncfusion:ChartAxis Orientation="Horizontal" />
</syncfusion:ChartSeries.YAxis>
</syncfusion:ChartSeries>
</syncfusion:ChartArea>
```

The following image illustrates Chart with Y-axis orientation set as Horizontal.



![[Chart-Controls
img149]](Chart-Controlsimages/Chart-Controls_img149.jpeg)

Inverted Axis

Essential Chart provides support for inverting the values on the axis. Data on an inverted axis is plotted in the opposite direction - Top to Bottom for Y-axis and Right to Left for X-axis. To enable this behavior, set the ChartAxis.IsInversed property to True.

ChartAxis Property	Description
IsInversed	Indicates whether the axis should be reversed. When reversed, the axis will render points from right to left if horizontal, top to bottom when vertical, and clockwise if radial.

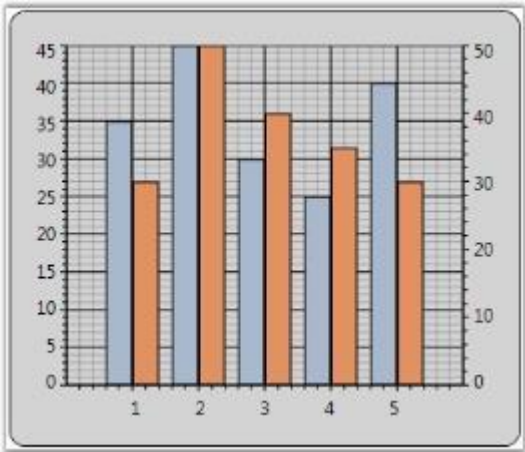
XML

```
<syncfusion:ChartArea>
<syncfusion:ChartArea.PrimaryAxisAxis>
<syncfusion:ChartAxis IsInversed="True">
</syncfusion:ChartArea.PrimaryAxisAxis>
<syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartAxis IsInversed="True">
</syncfusion:ChartArea.SecondaryAxis>
</syncfusion:ChartArea>
```

C#

```
// Sets the Axis as inverted.
chart.Areas[0].PrimaryAxis.IsInversed = true;
chart.Areas[0].SecondaryAxis.IsInversed = true;
```

The following image illustrates Chart with an Inversed Axis.



![[Chart-Controlsimages/Chart-Controls_img150.png]]

Opposed Axis

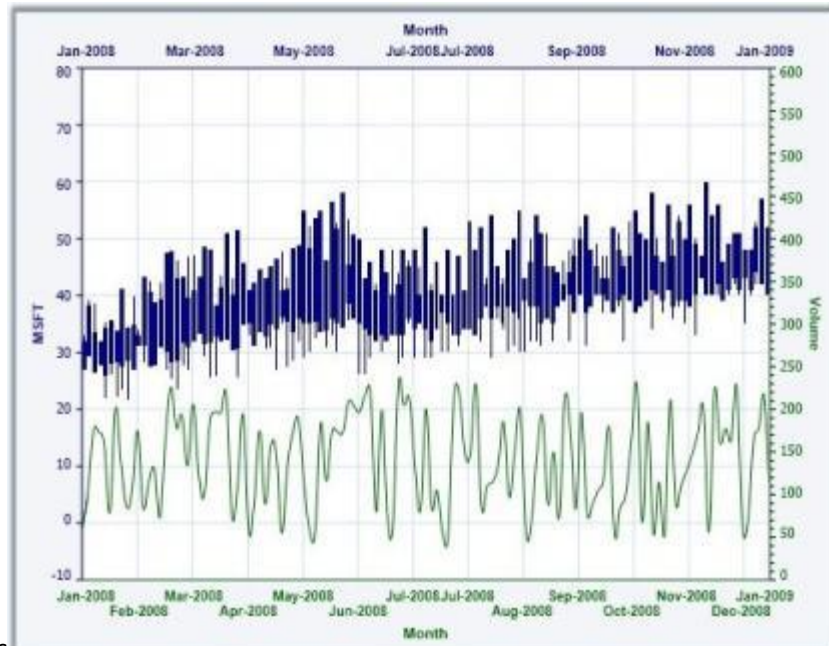
For every Chart Type, there is an implied X-axis and Y-axis position, and by default all the X-axes and Y-axes will be rendered in that corresponding position. You can override this default behavior by setting the `OpposedPosition` property to `True` for an axis, which will cause it to be rendered in the opposite side of the implied position. This feature is also used with Multiple Axes, where you can position the Secondary axis opposite to the Primary axis.

ChartAxis Property	Description
OpposedPosition	Gets / sets a value indicating whether axis should be in opposed position.

XML

```
<syncfusion:ChartArea Name="area">
  <syncfusion:ChartSeries Name="series" Data=" 1 35 2 45 3 30 4 25 5 40" />
  <syncfusion:ChartSeries Name="series1" Data=" 1 30 2 50 3 40 4 35 5 30" >
  <syncfusion:ChartSeries.YAxis>
    <syncfusion:ChartAxis Orientation="Vertical" OpposedPosition="True"/>
  </syncfusion:ChartSeries.YAxis>
</syncfusion:ChartSeries>
</syncfusion:ChartArea>
```

The following image illustrates Chart with Opposed Axis.



![[Chart-Controls/Chart-Controls_img151.jpeg]](Chart-Controls_img151.jpeg)

Multiple Axes

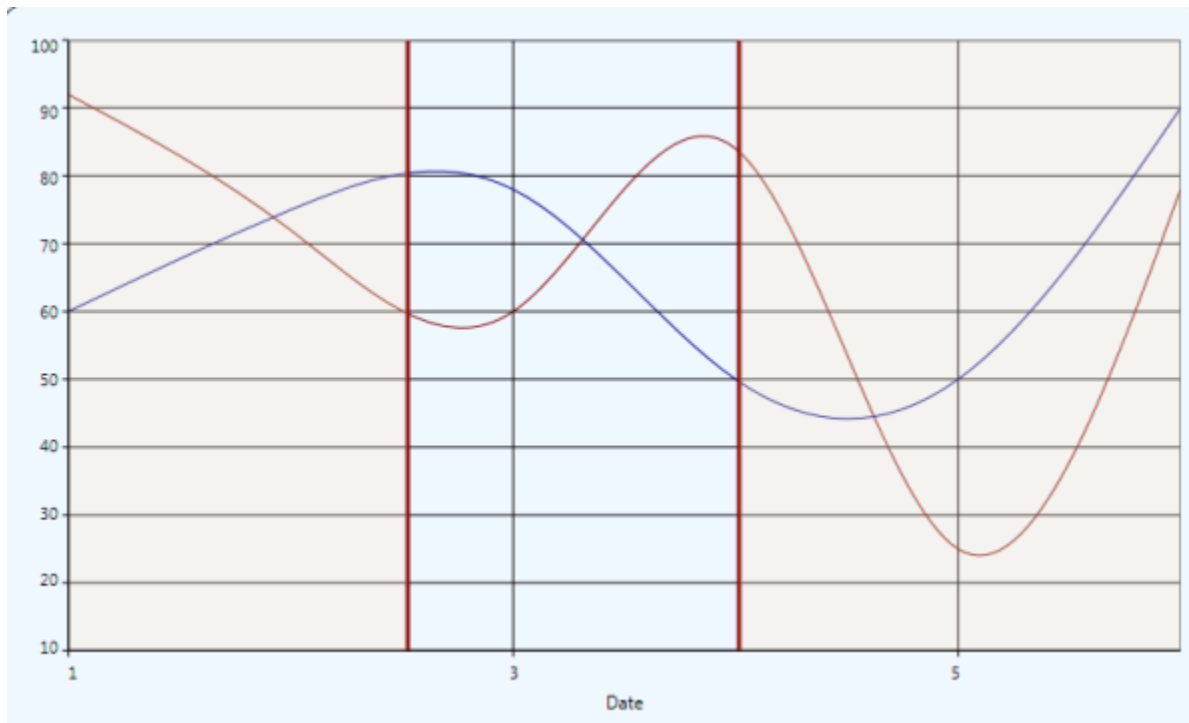
Chart is capable of rendering multiple axes in the same plot. Different series can be plotted against different axes as follows.

XML

```
<sfchart:Chart Name="Chart1">
  <sfchart:ChartArea>
    <!--This series is plotted against the default Y axis -->
    <sfchart:ChartSeries Type="Spline" x:Name="Series1" DataSource="{Binding
Source={StaticResource dataaval}}" BindingPathX="ID" BindingPathsY="Val3"
IsIndexed="False" Label="Avg Temperature">
    </sfchart:ChartSeries>
    <!--This series is plotted against a custom Y axis -->
    <sfchart:ChartSeries Type="SplineArea" x:Name="Series2" DataSource="{Binding
Source={StaticResource dataaval}}" BindingPathX="ID" BindingPathsY="Val1"
Label="Avg Rainfall">
    <sfchart:ChartSeries.YAxis>
    <sfchart:ChartAxis OpposedPosition="True" RangePadding="Normal"
IsAutoSetRange="False" Range="0,100" SmallTicksPerInterval="0" Interval="10"
Orientation="Vertical">
    <sfchart:ChartArea.ShowGridLines>false</sfchart:ChartArea.ShowGridLines>
    <sfchart:ChartAxis.Header>
    <TextBlock Text="Rainfall (mm)" FontFamily="Tahoma" FontSize="14"
Foreground="LightSteelBlue"/>
    </sfchart:ChartAxis.Header>
    </sfchart:ChartAxis>
    </sfchart:ChartSeries.YAxis>
    </sfchart:ChartSeries>
    </sfchart:ChartArea>
  </sfchart:Chart>
```


![[Chart-Controlsimg152]](Chart-Controlsimages/Chart-Controls_img152.jpeg)

For more details, refer to the sample in the following location:



...\\My Documents\\Syncfusion\\EssentialStudio\\<Version Number>\\WPF\\Chart.WPF\\Samples\\3.5\\WindowsSamples\\Chart Axis\\Chart Multiple Opposed Axes Demo

Axis Range Selection

Essential Chart for WPF now supports axis range selection. This enables the user to select a particular range of a primary axis by using two cursors.

Adding an Axis Range Selection

To add axis range selection, set the `EnableRangeSelection` to `True`. The following code illustrates this.

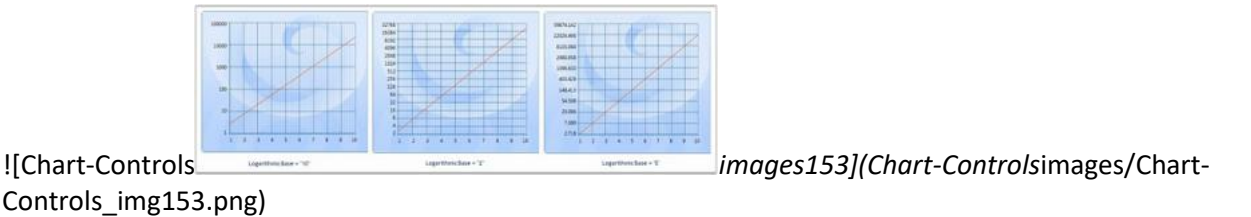
XML

```
<syncfusion:ChartArea Name="area" EnableRangeSelection="True"
LineStroke="Maroon" SelectionStroke="LightPink" />
```

C#

```
chart1.Areas[0].EnableRangeSelection = true;
double start = chart1.Areas[0].StartRange;
double end = chart1.Areas[0].EndRange;
chart1.Areas[0].LineStroke = Brushes.Maroon;
chart1.Areas[0].SelectionStroke = Brushes.LightPink;
```

When the code runs, the following output displays.



Property Details

Name of Property	Description	Type of Property	Value It Accepts	Sub Properties
EnableRangeSelection	Used to set the range selection.	Dependency	Binary or True/False	StartValueType: double EndValueType: double LineStrokeType: Brush SelectionStroke Type: Brush

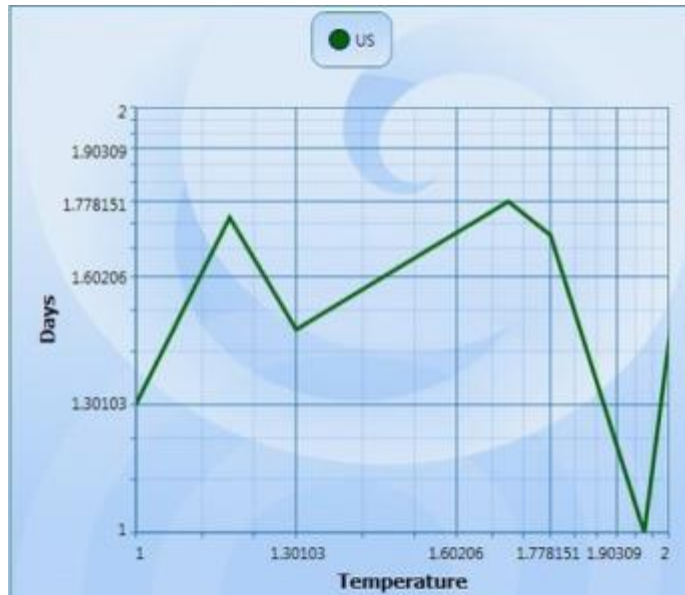
Logarithmic Axes

Logarithmic values can be applied to the Chart. This is facilitated by the IsLogarithmic and LogarithmicBase properties. On setting the Axis.IsLogarithmic property, the Axis range, interval and padding will be plotted as per the log values provided. The LogarithmicBase value allows to set the base values for Logarithmic Axis.

C#

```
// Add Data points to Chart
ChartListData points = new ChartListData();
for (int i = 1; i < 11; i++)
points.Add(new ChartPoint(i, Math.Exp(i)));
series.Data = points;
// Set the IsLogarithmic property of the Axis as true
area.SecondaryAxis.IsLogarithmic = true;
// Set the Logarithmic Base value as 10
area.SecondaryAxis.LogarithmicBase = 10;
// Set the Logarithmic Base value as 2
area.SecondaryAxis.LogarithmicBase = 2;
// Set the Logarithmic Base value as e
area.SecondaryAxis.LogarithmicBase = Math.E;
```

The following image illustrates Log chart with various LogarithmicBase values.



![[Chart-Controls
Controls/images/Chart-Controls_img154.png]img154](Chart-

Show Minor Grid Lines When the Axis Is Logarithmic

Essential Chart WPF is enhanced with minor grid lines and ticks when the axis is set as logarithmic.

[Adding Show Minor Grid Lines](#)

Add Show Minor Grid Lines, by using the following code.

XML

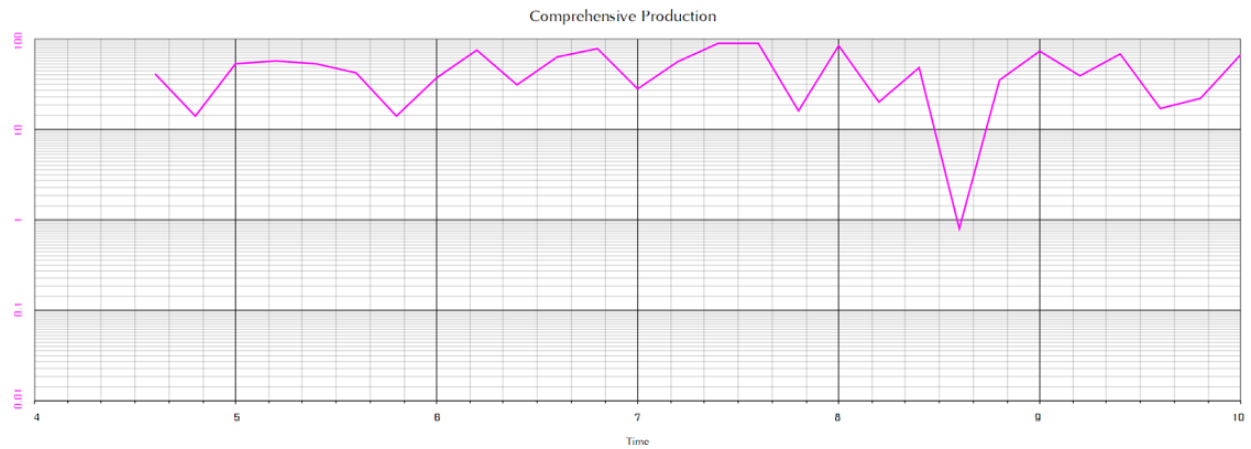
```
<syncfusion:Chart Name="Chart1" Grid.Row="1" Margin="10">
  <syncfusion:ChartArea Name="area" >
    <syncfusion:ChartArea.PrimaryAxis>
      <!--X axis declaration with required property settings-->
      <syncfusion:ChartAxis Header="Year" IsLogarithmic="True"
        EnableLogLabels="True" >
    </syncfusion:ChartAxis>
    </syncfusion:ChartArea.PrimaryAxis>
    <syncfusion:ChartArea.SecondaryAxis>
      <!--Y axis declaration with required property settings-->
      <syncfusion:ChartAxis IsLogarithmic="True"
        EnableLogLabels="True">
    </syncfusion:ChartAxis>
    </syncfusion:ChartArea.SecondaryAxis>
  </syncfusion:ChartArea>
</ syncfusion:Chart>
```

C#

```
Chart1.Areas[0].PrimaryAxis.IsLogarithmic = true;
Chart1.Areas[0].PrimaryAxis.EnableLogLabels = true;
Chart1.Areas[0].SecondaryAxis.IsLogarithmic = true;
Chart1.Areas[0].SecondaryAxis.EnableLogLabels = true;
```

When the code runs, the following output displays.

![[Chart-Controls



img155](Chart-Controlsimages/Chart-Controls_img155.jpeg)

The following table contains the property Details.

Name of the Property	Description	Type of the Property	Value It Accepts
EnabelLogLabels	Set /unset the log labels	Dependency Property	Bool (true or false)

See Also

[Multiple Axes](#)

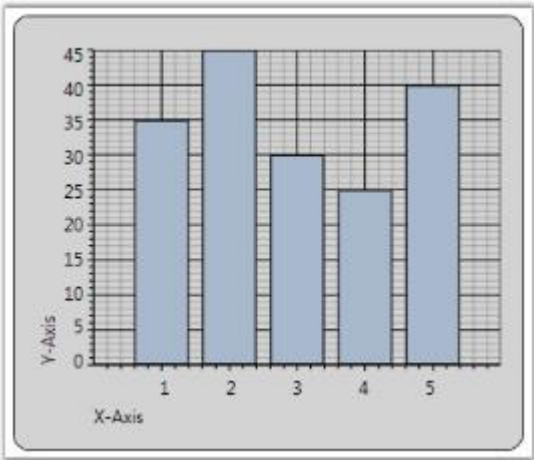
Small tick for Logarithmic axis

Essential chart allows user to set the minor grid lines for log axis.

Name of the Property	Type of the property	Value it accepts	Property syntax	Any other dependencies/ sub properties associated
SmallTicksPerInterval	Dependency property	Integer and any whole number	SmallTicksPerInterval = 5	NA

XML

```
<syncfusion:ChartAxis SmallTicksPerInterval="20" IsLogarithmic="True" >
```



![[Chart-Controls](Chart-Controlsimages/Chart-Controls_img156.png)]

Chart-Axis in WPF Chart (Classic) Header

Chart for WPF provides support for Chart Axis Titles with the help of the attached property `ChartAxis.Header`. Its position can be adjusted by using the `HeaderAlignment` property of the `ChartAxis` class.

ChartAxis Property	Description
Header	Gets/sets the title of the axis.
HeaderAlignment	Gets/sets the header alignment. The options included are as follows.FarCenterNear

XML

```
<syncfusion:ChartArea Name="area">
  <syncfusion:ChartArea.PrimaryAxis>
    <syncfusion:ChartAxis Header="X-Axis" HeaderAlignment="Near" />
  </syncfusion:ChartArea.PrimaryAxis>
  <syncfusion:ChartArea.SecondaryAxis>
    <syncfusion:ChartAxis Header="Y-Axis" HeaderAlignment="Far" />
  </syncfusion:ChartArea.SecondaryAxis>
  <syncfusion:ChartSeries Name="series" Data=" 1 35 2 45 3 30 4 25 5 40" />
</syncfusion:ChartArea>
```

C#

```
area.PrimaryAxis.Header = "X-Axis";
area.PrimaryAxis.HeaderAlignment = ChartAlignment.Near;
area.SecondaryAxis.Header = "Y-Axis";
area.SecondaryAxis.HeaderAlignment = ChartAlignment.Far;
```

The following image illustrates Chart with Axis `HeaderAlignment` set.



![(Chart-Controlsimages/Chart-Controls_img157.jpeg)](Chart-Controlsimages/Chart-Controls_img157.jpeg)

Chart Striplines

Strip lines are horizontal or vertical bands marked in the background of a chart at regular or custom intervals. These strip lines help you to distinguish data points on a chart, highlight specific ranges of data, define threshold points etc.

Segmented strip lines are created when the vertical and horizontal strip lines intersect each other. The segment created by this intersection for each strip line is called a Segmented Strip Line.

The following code examples illustrate the creation of Segmented Strip Line at required locations.

XML

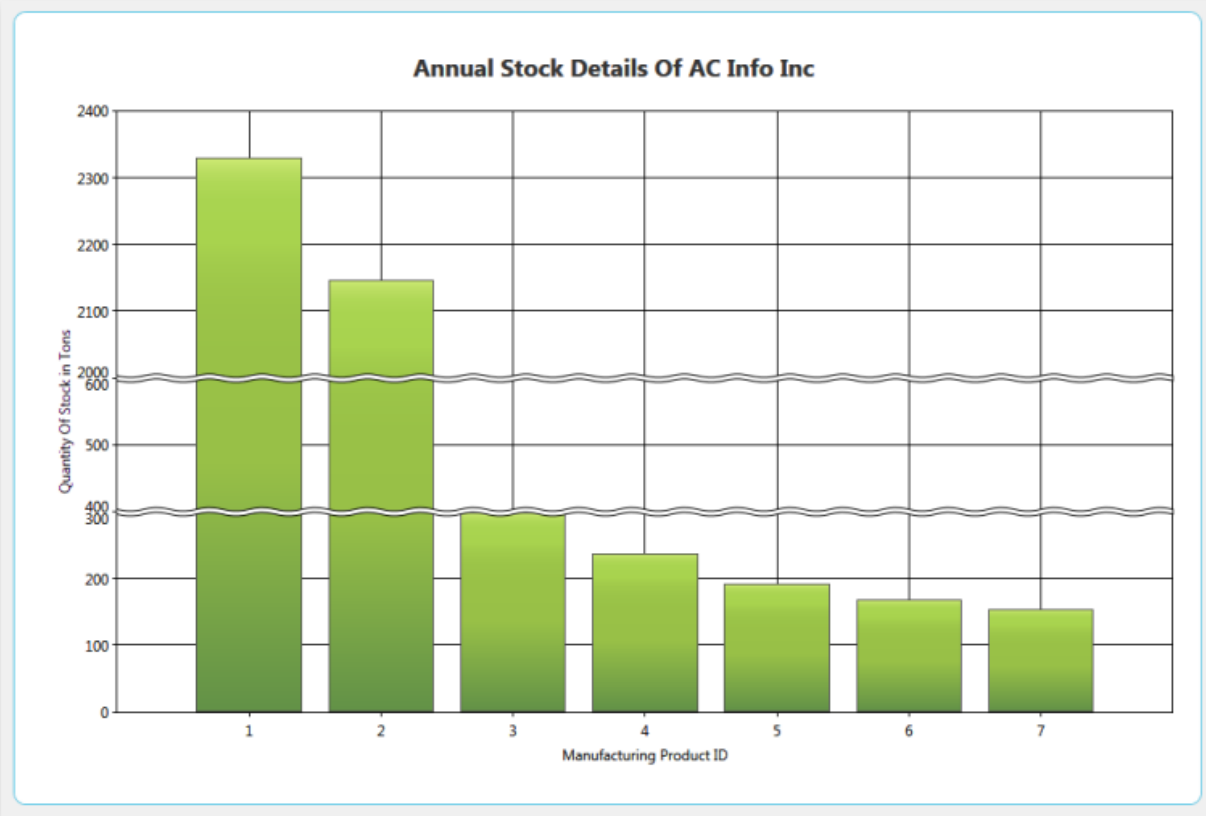
```
<syncfusion:ChartAxis.StripLines>
<syncfusion:ChartStripLine IsSegmented="True" Start="1" Width="1"
SegmentStartValue="10" Interior="Pink" Repeat="3" SegmentEndValue="60">
</syncfusion:ChartStripLine>
</syncfusion:ChartAxis.StripLines>
```

C#

```
ChartStripLine sp = new ChartStripLine();
sp = new ChartStripLine();
sp.IsSegmented = true;
sp.Start = 1;
sp.Width = 1;
sp.Repeat=3;
sp.Interior = Brushes.Pink;
sp.StartFromAxis = false;
sp.SegmentStartValue = 10;
sp.SegmentEndValue = 60;
sp.Text = new FormattedText(item.Production.ToString(),
CultureInfo.CurrentCulture, FlowDirection.LeftToRight, new
Typeface("Arial"), 10, Brushes.Black);
```

Run the code. The following output is displayed.

![[Chart-Controls



img158](Chart-Controlsimages/Chart-Controls_img158.jpeg)

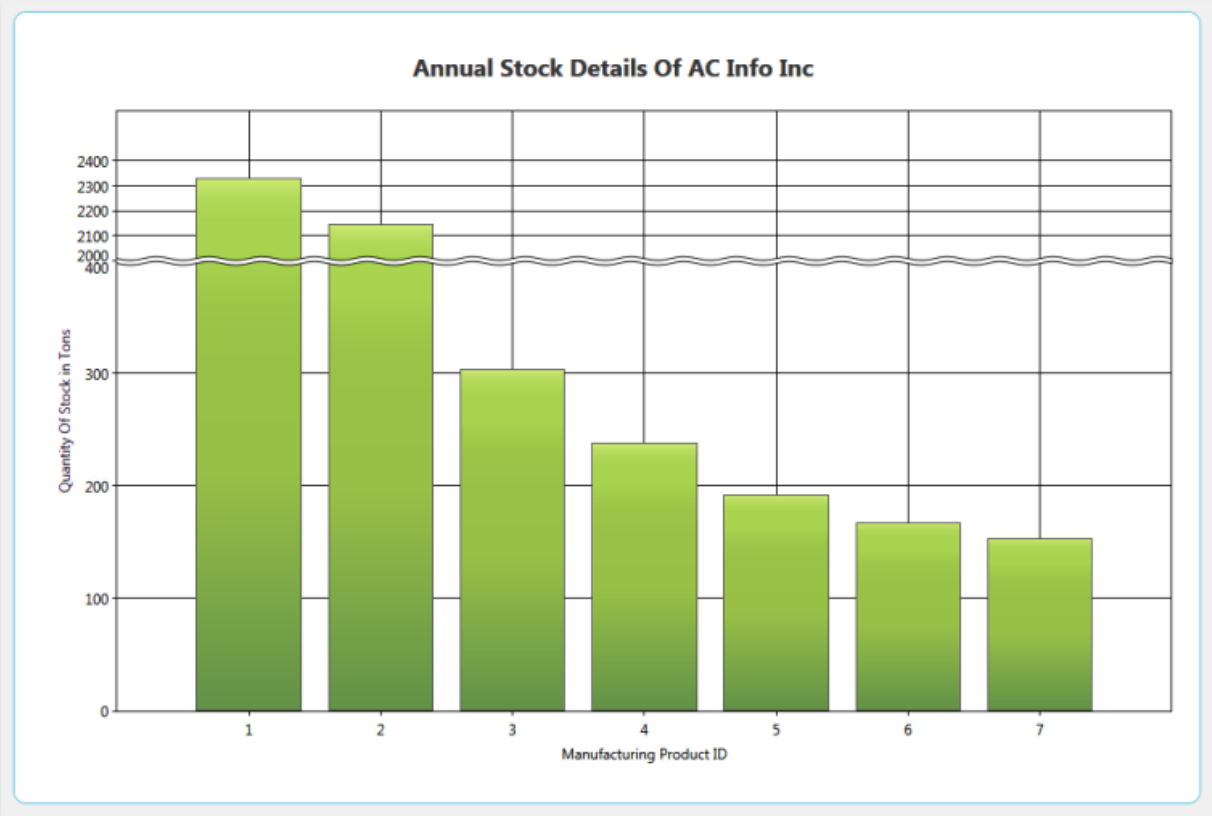
Scale Break Support

Breaks are very useful if you add points with too large difference in values. To enable breaks, you need to set the `EnableBreaks` property to true and set the break mode (`BreaksMode` property). A scale break is a line drawn across the plotting area of a chart to denote a break in continuity between the highest and lowest values on a value axis. A Scale Break displays two distinct ranges in the same chart area.

There are three possible modes. They are,

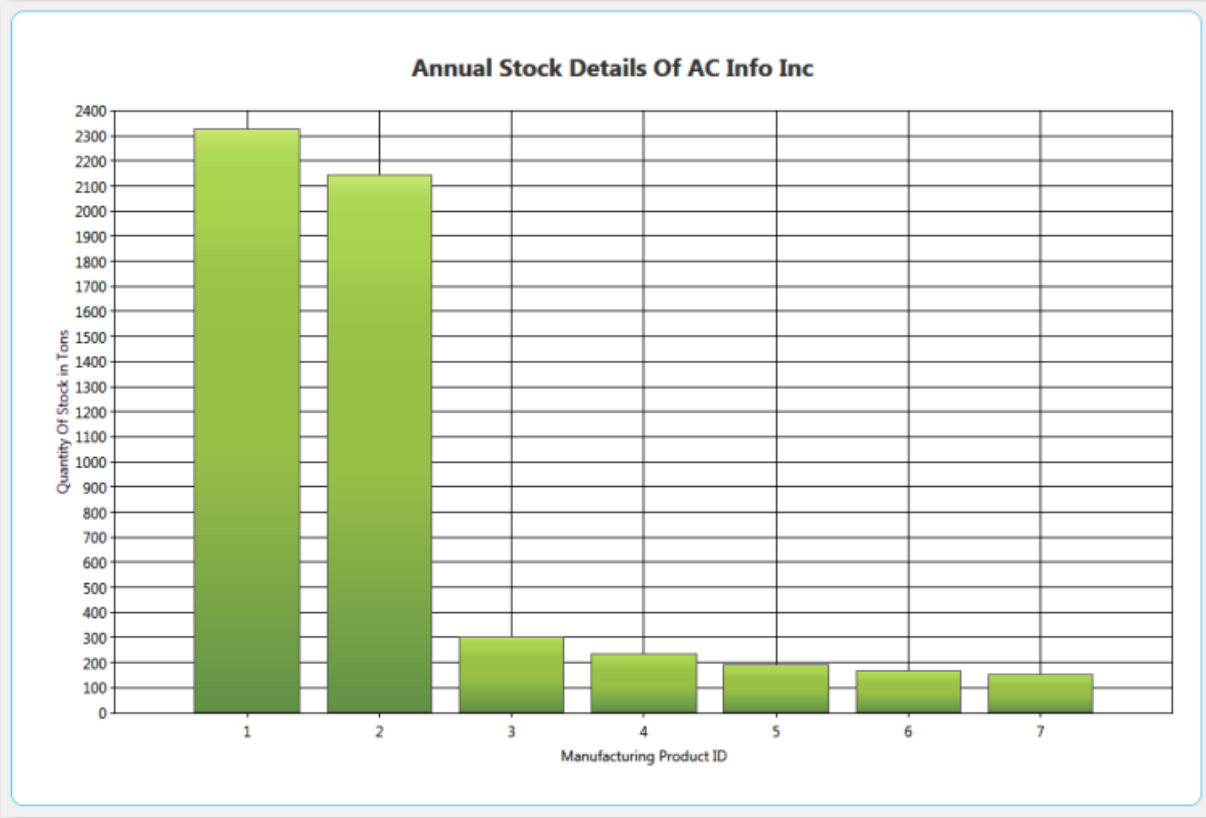
- `ChartBreaksModes.Manual` (default): If this value is set, you can manually set the breaks ranges.
- `ChartAxis.BreakRanges.Union`: to add a new break range
- `ChartAxis.BreakRanges.Exclude`: to remove the break range
- `ChartAxis.BreakRanges.Clear`: to remove all break ranges
- `ChartBreaksMode.Auto`: If this mode is enabled, chart will compute the breaks ranges automatically.
- This mode has several exclusions
 - Breaks are computed only for actual y-axis of series
 - Breaks don't work with zooming
 - Breaks don't work with stacking
 - All breaks work only with decart axes
- `ChartBreaksModes.None`: If this value is set, breaks are not used.

![[Chart-Controls

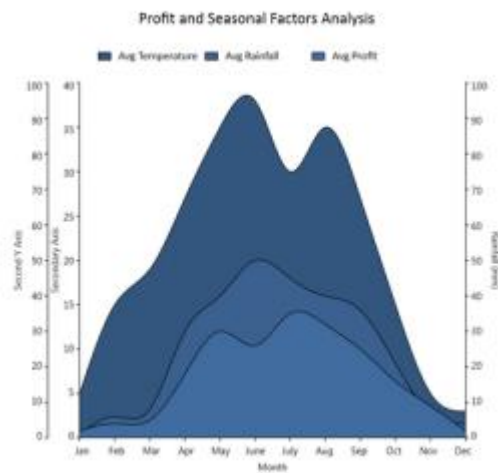


images159](Chart-Controlsimages/Chart-Controls_img159.png)

![[Chart-Controls



images160](Chart-Controlsimages/Chart-Controls_img160.png)

![[Chart-Controls
Controls_img161.png)

images161](Chart-Controlsimages/Chart-

Properties and Methods Tables

Properties

Property	Description	Type	Reference links
EnableBreaks	To enable break ranges	bool	NA

BreaksMode	3 types: Manual, Auto and None	ChartBreaksModes	NA
LineType	Straight line, wave and randomize	ChartBreakLineTypes	NA
LineColor	Different types of colors for the line	Brush	NA
LineWidth	Values from 1 to 10	Double	NA
Linestyle	Dash Line, DashDodDot Line ,DashDot Line, Dot Line, Solid Line	DashStyle	NA
SpacingColor	Different types of colors for the line	Brush	NA
SpacingWidth	Values from 1 to 10	Double	NA

Methods

Method	Description	Parameters	Return Type	Reference links
Union	To add a particular break range	DoubleRangeChartBreakRangeInfo	void	NA
Exclude	To delete the specified break range	DoubleRange	void	NA
Clear	Clear all existing break ranges	NA	void	NA

Retaining Axis Position

Essential Chart for WPF enables users to retain the axis position of the primary and secondary axis when multiple axes are added in a chart area.

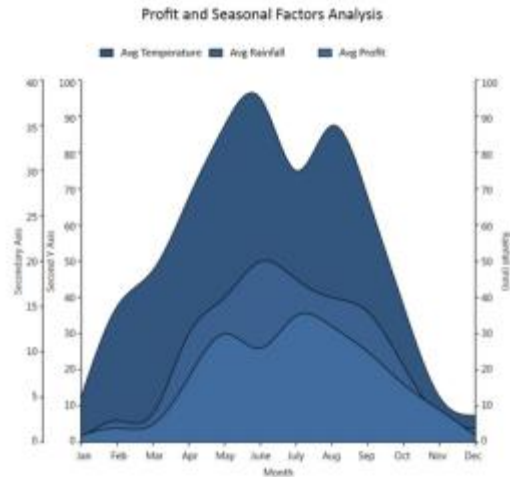
Adding Support for Retaining Axis Position

Set the `IsRetainAxisPosition` property to `True` to add axes in the order they are added to the chart area. The following code illustrates this.

XML

<code><syncfusion:ChartArea IsRetainAxisPosition="True"></code>

When the code runs, the following output displays.

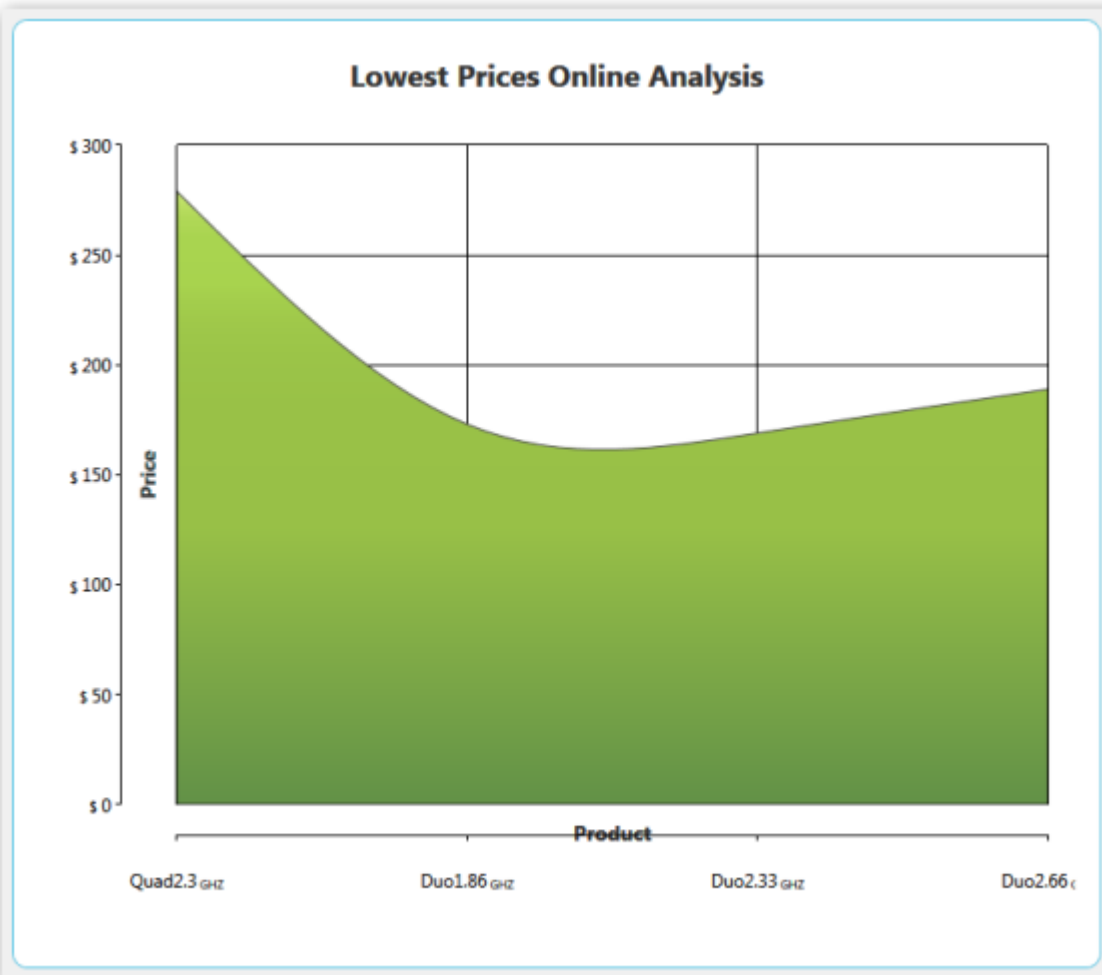


![(Chart-Controls
Controls_img162.png)

img162](Chart-Controlsimages/Chart-

Set IsRetainAxisPosition is False to add axes in reverse order.

![(Chart-Controls



img163](Chart-Controlsimages/Chart-Controls_img163.png)

Name of Property	Description	Type of Property	Value It Accepts	Property Syntax
IsRetainAxisPosition	Determines the order of arranging the multiple axes.	Dependency	Bool or True/False.	IsRetainAxisPosition="True"

Chart-Axis in WPF Chart (Classic) Improvements

Essential Chart provides support for some improvements in the existing chart axis by implementing the following features.

- Axis headers can be positioned inside, outside, or across the chart axes.
- Edge labels can be adjusted by setting `EdgeLabelsDrawingMode` to `Fit` to avoid the partial appearance of edge axis labels.
- Prefixes and suffixes can be added to chart axes labels to mention the units along with the labels.
- Labels can be aligned horizontally and vertically with respect to the specified width and height of chart axis labels.

Properties

Property	Description	Type	Data Type
HeaderPosition	Specifies the position of the header with respect to the chart axis. Insideâ€”Header is positioned inside the chart axis. Outsideâ€”Header is positioned outside the chart axis. Crossâ€”Header is positioned across the chart axis.	Dependency	AxisPositions
LabelHeight	Specifies the height of axis labels	Dependency	double
LabelWidth	Specifies the width of axis labels	Dependency	double
LabelHorizontalAlignment	Aligns the labels horizontally within the specified width of the labels.	Dependency	HorizontalAlignment
LabelVerticalAlignment	Aligns the labels horizontally within the specified width of the labels.	Dependency	VerticalAlignment
LabelsPrefix	Specifies the prefix for chart axis labels.	Dependency	DataTemplate
LabelsPostfix	Specifies the suffix for chart axis labels.	Dependency	DataTemplate

EdgeLabelsDrawingMode	Fit"Draws the edge labels to fit within the chart area	Dependency	EdgeLabelsDrawingMode
-----------------------	--	------------	-----------------------

[Sample Link](#)

To access the chart axis improvement demo:

1. Open the Syncfusion Dashboard.
2. Select User Interface.
3. Click the WPF drop-down list and select Explore Samples.
4. Browse to the path Chart.WPF\Samples\3.5\ WindowsSamples\Chart Axis\Chart Axis Improvement Demo.

[Adding Axis Improvement properties to an Application](#)

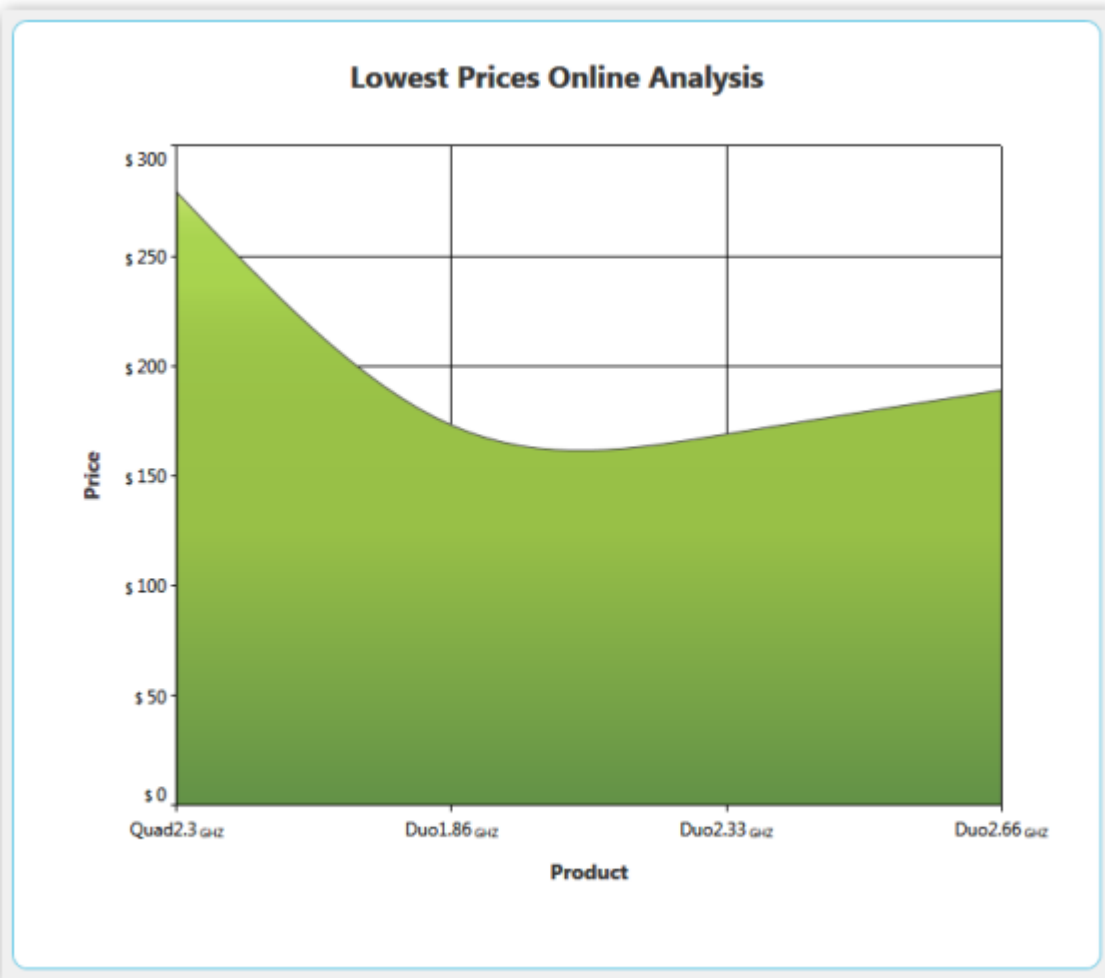
XML

```
<syncfusion:ChartAxis HeaderPosition="Cross"
LabelHeight="40" LabelWidth="120"
LabelsPrefix="{StaticResource yPrefix}"
LabelsPostfix="{StaticResource yPostfix}"
LabelHorizontalAlignment="Left" LabelVerticalAlignment="Top"
EdgeLabelsDrawingMode="Fit">
</syncfusion:ChartAxis>
```

C#

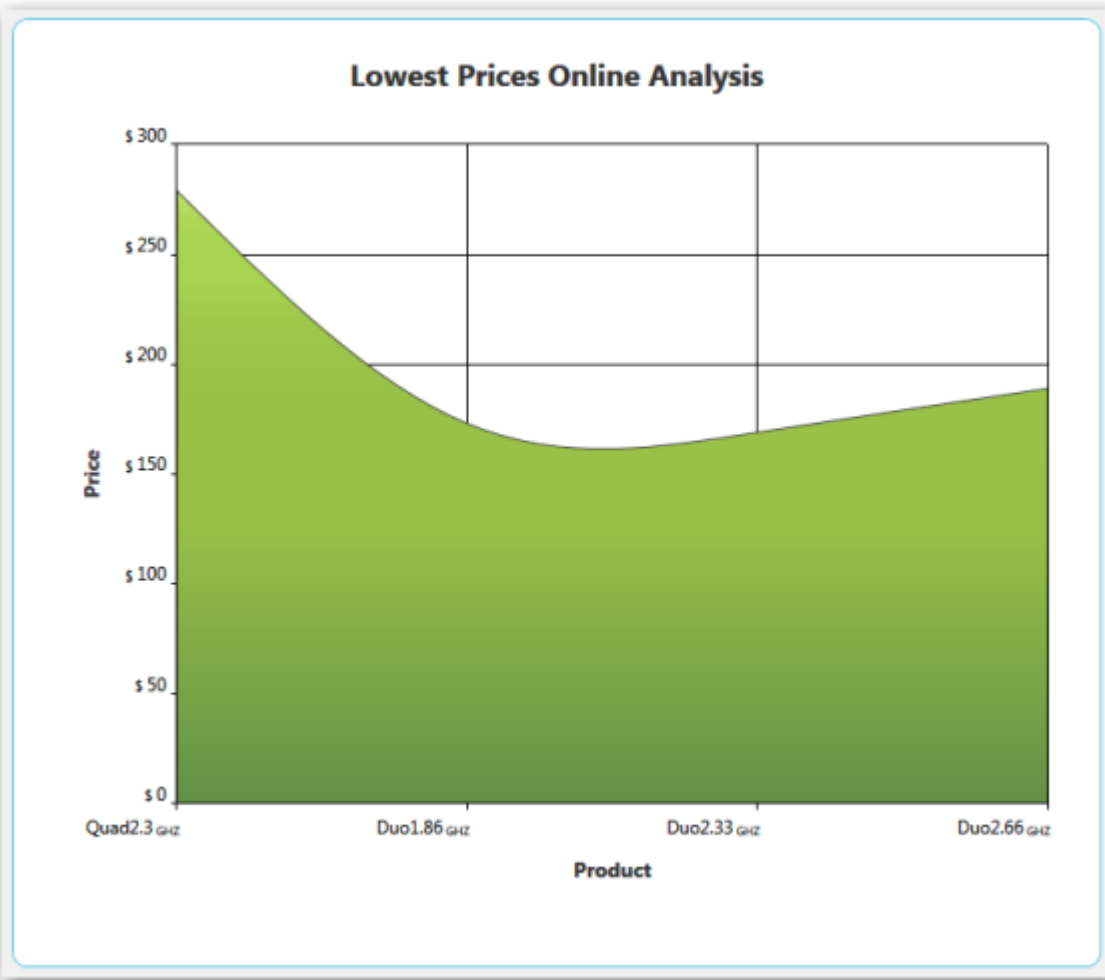
```
this.primaryAxis.HeaderPosition = AxisPositions.Cross;
this.primaryAxis.LabelHeight = 40; this.primaryAxis.LabelWidth =
120; this.primaryAxis.LabelHorizontalAlignment =
HorizontalAlignment.Left; this.primaryAxis.LabelVerticalAlignment
= VerticalAlignment.Top; this.primaryAxis.LabelsPrefix = xPrefix;
this.primaryAxis.LabelsPostfix = xPostfix;
this.primaryAxis.EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Fit;
```

![[Chart-Controlsimages164]](Chart-Controlsimages/Chart-Controls_img164.png)

*Primary*

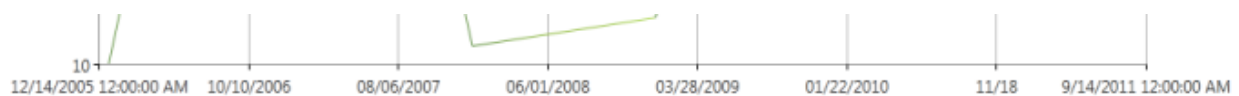
Axis Labels with Postfix (GHZ) and Secondary Axis Labels with Prefix (\$)

![[Chart-Controls]



Secondary Axis EdgeLabelsDrawingMode [Shift]

![[Chart-Controlsimages166]](Chart-Controlsimages/Chart-Controls_img166.png)



Secondary Axis LabelHeight [30], LabelVerticalAlignment [Top]

Smart Axis Labels

Essential Chart for WPF supports smart axis labels which help to display labels in a smart and effective manner. This support is available for the following axis types:

- Double
- DateTime
- TimeSpan

Use Case Scenarios

Smart axis labels are useful in the following scenarios:

Axis Type Double: In cases where axis ranges have values that are so large (e.g., millions or trillions) that the labels use too much room on the axis. Smart axis labels are rendered with suffixes based on the data range value. For example, “M” will be appended to all labels with million values—1,000,000 will be displayed as 1M.

Axis Type DateTime: In cases where LabelDateTimeFormat is set to Default, the labels will appear too large displaying “MM/dd/yyyy hh:mm tt”. Smart labels calculate the data range and reduce the labels. For example, only “MM/dd” would be displayed depending upon the interval between two labels.

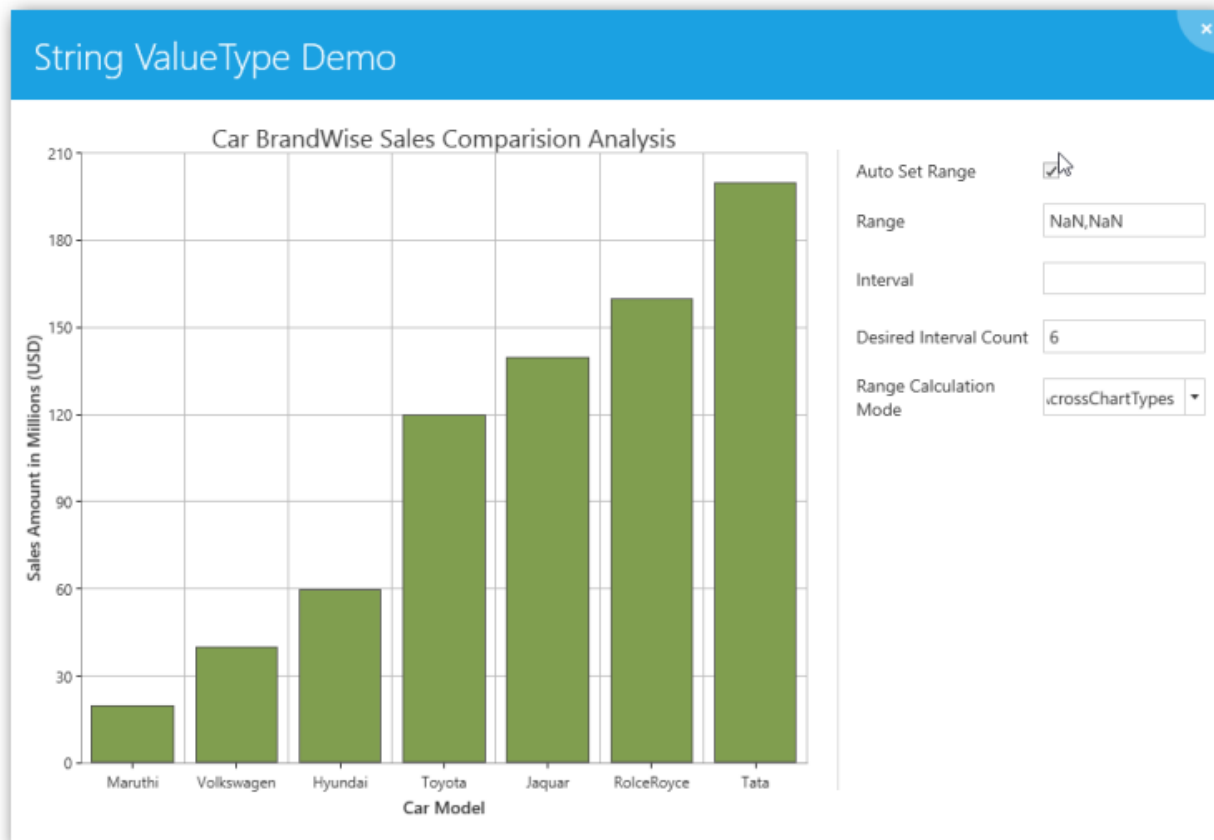
![[Chart-Controls



images167](Chart-Controlsimages/Chart-Controls_img167.png)

Axis Type TimeSpan: In cases where the time span is set with intervals of hours, smart labels can convert and display when there is a change with the hours. For example, 12:00 to 15:00 will be displayed as 12:00:00. ... 1 ... 2 12 ... 13 ... 14 ... 15:00:00

![[Chart-Controls



images168](Chart-Controlsimages/Chart-Controls_img168.png)

Properties

Property	Description	Type	Data Type
----------	-------------	------	-----------

EnableSmartAxisLabels	Used to enable or disable SmartAxisLabels	Dependency property	Bool
DoubleDisplayUnit	Used to set double display units such as millions, trillions, etc.	Dependency property	Enum DisplayUnit
DoubleDisplayUnitAlignment	Used to set the alignment for double display units.	Dependency property	Enum ChartAlignment

[Sample Link](#)

C:\Documents and Settings\<user name>\My documents\Syncfusion\Essential Studio\Samples\WPF\Chart.WPF\Samples\3.5\WindowsSamples\ChartAxis\Smart Axis Label

Enabling Smart Axis Labels in Chart Axis

The following code example shows how to enable smart axis labels:

C#

```
ChartAxis axis=new ChartAxis();
axis.EnableSmartAxisLabel= True;
Chart1.PrimaryAxis=axis;
```

XML

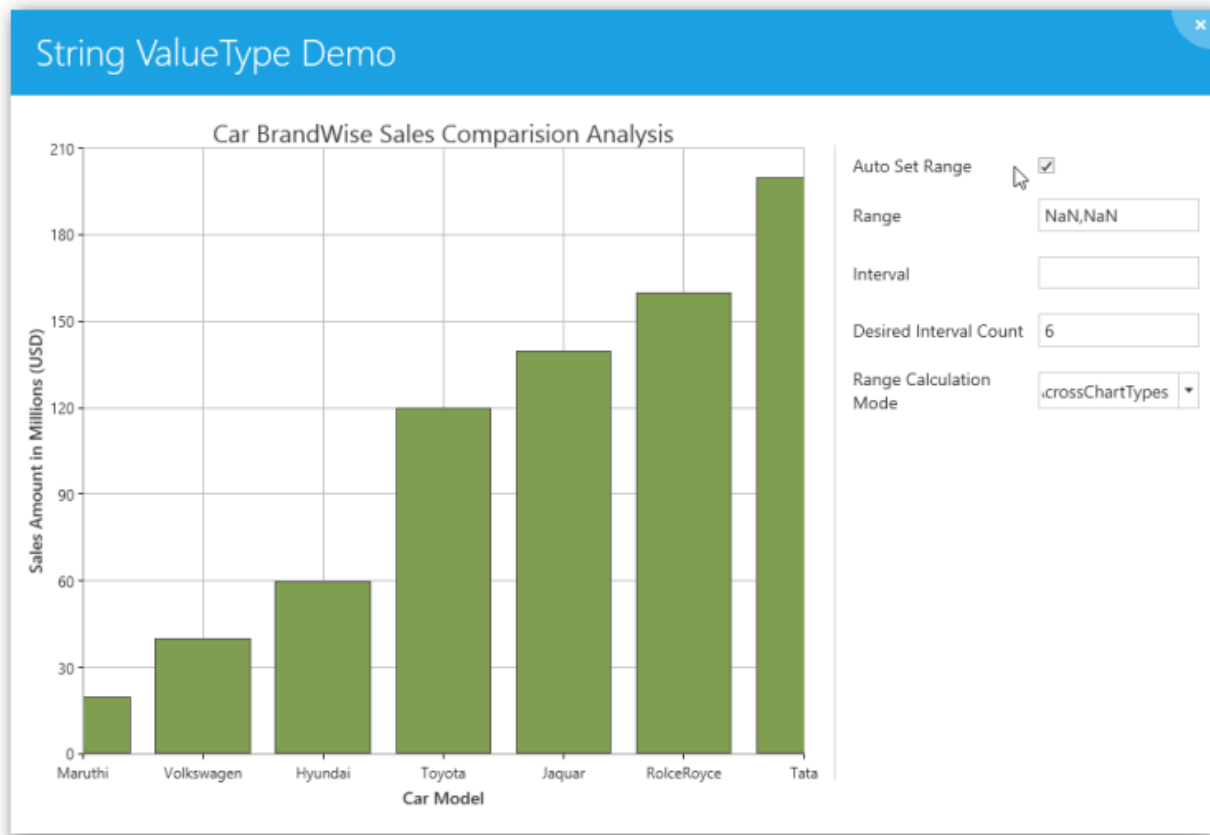
```
<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis EnableSmartAxisLabel="True"/>
</syncfusion:ChartArea.PrimaryAxis>
```

[Placing axis labels and series segment in between Ticklines](#)

Provide the option to position the segments between axis lines and on the axis line. For 'AdjustAcrossChartType', the segments is visible between the axis lines and for ConsistentAcrossChartType, the segment is visible on the axis lines.

The following screenshot depicts the segments between TickLine

![[Chart-Controls



img169][Chart-Controlsimages/Chart-Controls_img169.png)

The following screenshot depicts the segments on TickLine

![[Chart-Controlsimg170][Chart-Controlsimages/Chart-Controls_img170.png)

Properties

Property	Description
RangeCalculationMode	AdjustAcrossChartType - The segment starts from 1st axis and the constant behavior is BetweenTicks.ConsistentAcrossChartType - Similar to AdjustAcrossChartType and the constant behavior is OnTicks.

Adding Segment Position to an Application

For placing axis label and series segment in between Ticklines do the following:

1. Create a new WPF Application in VS2012.
2. Create a chart sample with ChartArea and ChartAxis.
3. Change the RangeCalculationMode property in primary axis to *AdjustAcrossChartType* or *ConsistentAcrossChartType* for changing the segment position.

Code Example

The following code example illustrates the usage of placing axis label and series segment between the Ticklines

XML

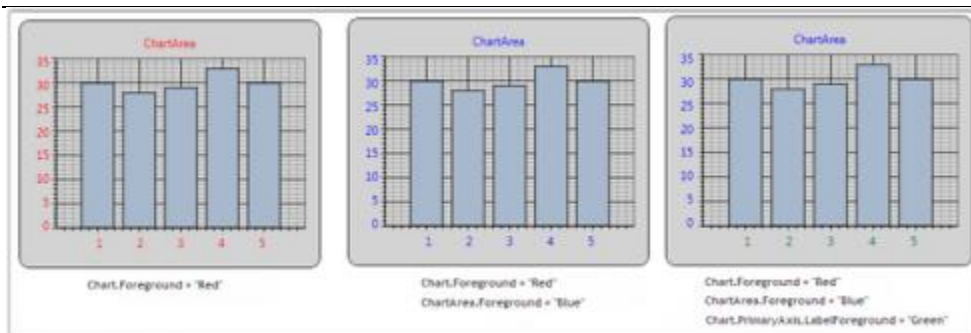
```
<sfchart:Chart>
  <sfchart:ChartArea Background="LightGray" GridBackground="White">
    <sfchart:ChartArea.PrimaryAxis>
      <sfchart:ChartAxis IsAutoSetRange="False" Range="0,100" Interval="25"
        MinimumInterval="25" RangeCalculationMode="AdjustAcrossChartType"
        RangePadding="Additional"/>
    </sfchart:ChartArea.PrimaryAxis>
    <sfchart:ChartSeries Template="{StaticResource Template1}"
      DataSource="{Binding Source={StaticResource
        myXmlData},XPath=Products/Product}" BindingPathX="Month"
      BindingPathsY="Sales" Interior="Red" Type="FastSpline"/>
    </sfchart:ChartArea>
  </sfchart:Chart>
```

Chart-Labels in WPF Chart (Classic)

Chart Font Settings

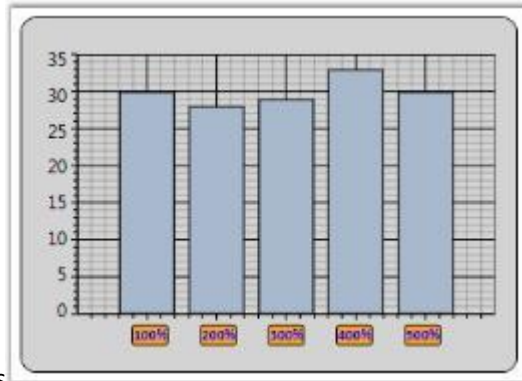
Font settings of each and every part of Chart such as Chart, ChartArea, ChartLegend and ChartAreaLegend can be customized with the following given properties.

- FontFamily
- FontSize
- FontStretch
- FontStyle
- FontWeight
- Foreground



Note: Font settings

set to the Chart will be applied to all the units of chart. However, properties set separately for those units such as ChartArea, ChartAxis, ChartLegend will override, these settings.



![(Chart-Controls_images/Chart-Controls_img171.png)](Chart-Controls_images/Chart-Controls_img171.png)

Chart Axis Label

ChartAxis Labels can be customized by using the following given properties.

ChartAxis Property	Description
LabelBackground	gets or sets the Background of the ChartAxis Label
LabelForeground	gets or sets the Foreground of the ChartAxis Label
LabelBorderBrush	gets or sets the Border brush of the ChartAxis Label
LabelBorderThickness	gets or sets the Border thickness of the ChartAxis Label
LabelCornerRadius	gets or sets the Corner Radius of the ChartAxis Label
LabelFontFamily	gets or sets the FontFamily of the ChartAxis Label
LabelFontSize	gets or sets the FontSize of the ChartAxis Label
LabelFontWeight	gets or sets the FontWeight of the ChartAxis Label
LabelFormat	gets or sets the Format of the ChartAxis Label such as 0.00 or 0 percent
LabelDateTimeFormat	gets or sets the DateTime Format of the ChartAxis LabelApplicable when Axis.ValueType is DateTime

The following code example could be used to customize the Chart Axis Labels.

XML

```
<sfchart:ChartArea.PrimaryAxis>
  <sfchart:ChartAxis LabelForeground="Blue" LabelBackground="Orange"
    LabelBorderBrush="Black" LabelBorderThickness="1" LabelCornerRadius="2"
    LabelFontFamily="Calibri" LabelFontSize="10" LabelFontWeight="Bold"
    LabelFormat="0%"/>
</sfchart:ChartArea.PrimaryAxis>
```

C#

```
//Sets the Label settings of PrimaryAxis
Area.PrimaryAxis.LabelForeground = Brushes.Blue;
```

```
Area.PrimaryAxis.LabelBackground = Brushes.Orange;
Area.PrimaryAxis.LabelBorderBrush = Brushes.Black;
Area.PrimaryAxis.LabelBorderThickness = new Thickness(1);
Area.PrimaryAxis.LabelCornerRadius = new CornerRadius(2);
Area.PrimaryAxis.LabelFontFamily = new FontFamily("Calibri");
Area.PrimaryAxis.LabelFontSize = 10;
Area.PrimaryAxis.LabelFontWeight = FontWeights.Bold;
Area.PrimaryAxis.LabelFormat = "0%";
```

Following given figure illustrates Chart with customized Primary Axis labels.

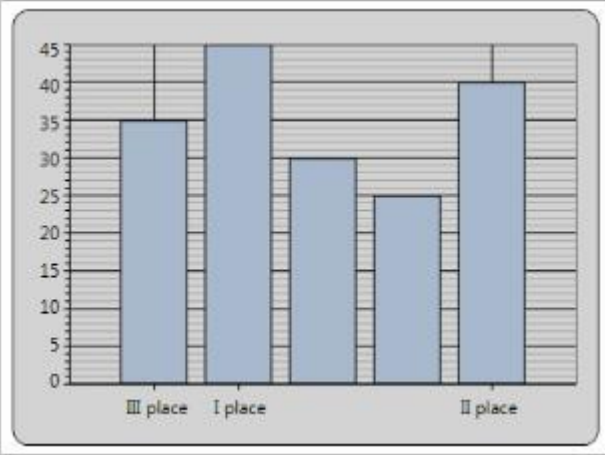
![[Chart-Controlsimg172]](Chart-Controlsimages/Chart-Controls_img172.jpeg)

Customizing Label Text

Apart from the default Labels displayed, you can also add Custom Labels to be displayed in the Chart. This section discusses the following topics.

Properties Used to Customize Label Text

The following properties are used to determine the data source for the text of the Labels.

ChartAxis Property	Description
LabelsMode	<p><i>Auto</i>: content is determined automatically<i>Custom</i>: custom values are used for labels content representation<i>DataSource</i>: external datasource is used for labels content<i>Default</i>: content for labels is either determined automatically, taken from external dataSource or being set with custom values</p> <div></div> <p><i>None</i>: labels values are taken from point's X-coordinate</p>
LabelsSource	gets or sets the Labels Source
PositionPath	gets or sets the position path
ContentPath	gets or sets the content path

By assigning the LabelsMode property to ChartAxis.CustomLabels, you can add Custom Labels to the ChartAxis.

XML

```

<syncfusion:Chart Name="Chart1" >
<syncfusion:ChartArea Name="area" >
<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis LabelsMode="Custom"
RangeCalculationMode="AdjustAcrossChartTypes">
<syncfusion:ChartAxis.CustomLabels>
<syncfusion:ChartAxisLabel Content="III place" Position="0" />
<syncfusion:ChartAxisLabel Content="I place" Position="1" />
<syncfusion:ChartAxisLabel Content="II place" Position="4" />
</syncfusion:ChartAxis.CustomLabels>
</syncfusion:ChartAxis>
</syncfusion:ChartArea.PrimaryAxis>
</syncfusion:ChartArea>
</syncfusion:Chart>

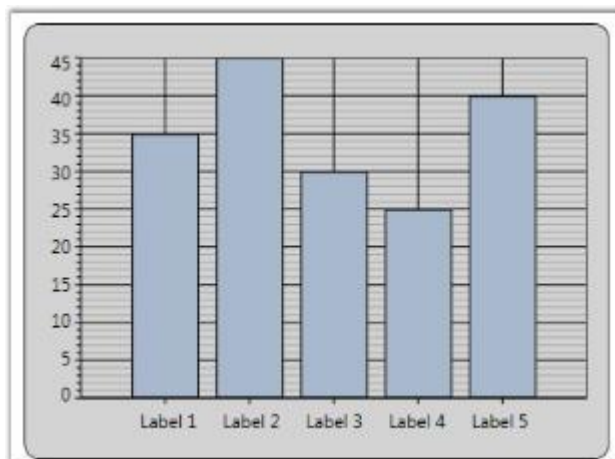
```

C#

```

// Indicates that the axis labels need to be taken from a custom source.
area.PrimaryAxis.LabelsMode = ChartAxisLabelsMode.Custom;
ChartAxisLabel customLabel1 = new ChartAxisLabel();
customLabel1.Content = "III place";
customLabel1.Position = 0;
ChartAxisLabel customLabel2 = new ChartAxisLabel();
customLabel2.Content = "I place";
customLabel2.Position = 1;
ChartAxisLabel customLabel3 = new ChartAxisLabel();
customLabel3.Content = "II place";
customLabel3.Position = 4;
// Adding custom label to labels collection.
area.PrimaryAxis.CustomLabels.Add(customLabel1);
area.PrimaryAxis.CustomLabels.Add(customLabel2);
area.PrimaryAxis.CustomLabels.Add(customLabel3);
The following screenshot illustrates Chart PrimaryAxis with Custom Labels.

```



![[Chart-Controls
Controls/images/Chart-Controls_img173.jpeg)](Chart-Controls/images/Chart-Controls_img173.jpeg)

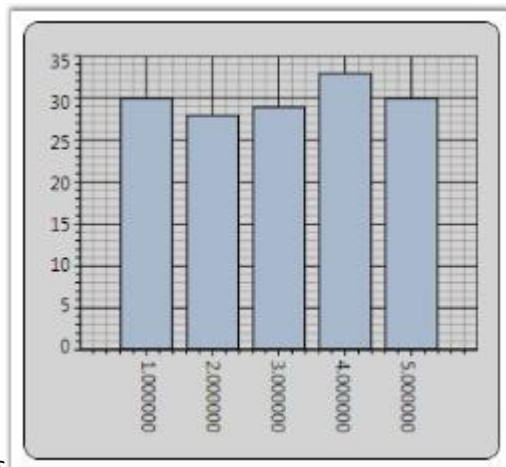
Labels from Data Source

The following code example illustrates how Custom Labels can be used in the ChartAxis.

C#

```
// Indicates that the axis labels need to be taken from a data source.
area.PrimaryAxis.LabelsMode = ChartAxisLabelsMode.DataSource;
// Creates DataSource with desired labels.
List<object> labels = new List<object>();
for (double i = 1; i < 6; i++)
{
    labels.Add(new { Position = i, Content = "Label " + i.ToString() });
}
// Associates DataSource with the Chart Labels.
area.PrimaryAxis.LabelsSource = labels;
// Set the position path where the labels should be placed.
area.PrimaryAxis.PositionPath = "Position";
// Set the content path from which labels are to be taken.
area.PrimaryAxis.ContentPath = "Content";
```

The following screenshot illustrates Chart PrimaryAxis with Labels from Data Source.



![[Chart-Controls_images/Chart-Controls_img174.jpeg]](Chart-Controlsimages/Chart-Controls_img174.jpeg)

Chart Axis Label Rotate

ChartAxis labels could be rotated with custom angles. Axis.LabelRotateAngle property is used to define the angle in which the Axis Labels need to be rotated.

The following code example could be used to customize the labels to be rotated with 90°.

XML

```
<sfchart:ChartArea.PrimaryAxis>
  <sfchart:ChartAxis LabelRotateAngle="90" LabelFormat="0.000000" />
</sfchart:ChartArea.PrimaryAxis>
```

C#

```
//Sets the Label to be rotated with 90° angle
Area.PrimaryAxis.LabelRotateAngle = 90;
```

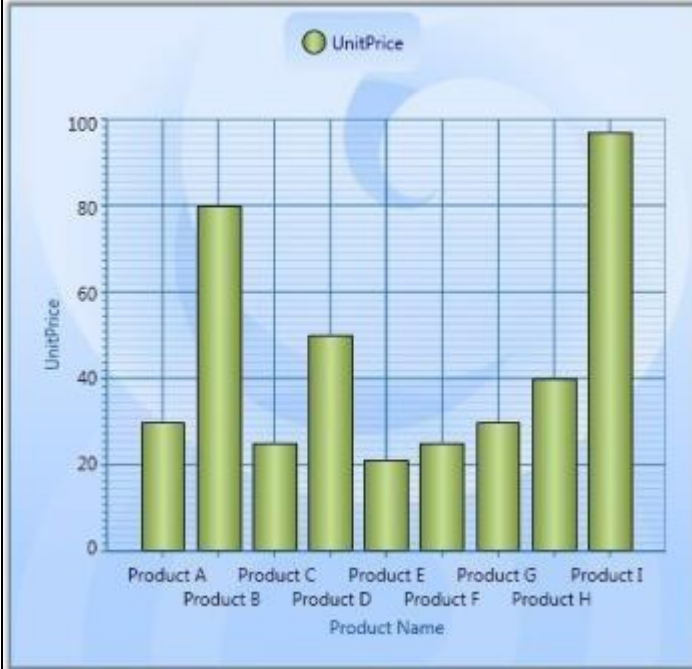
The following figure illustrates Chart with Primary Axis labels rotated with 90° angle

![[Chart-Controls_img175.jpeg]](Chart-Controlsimages/Chart-Controls_img175.jpeg)

Note: LabelRotateAngle property will not have effect when the Axis.IntersectAction property is set as Rotate.

Intersecting Labels

Sometimes the chart dimensions could cause the labels to intersect. The chart will, by default, render those texts one over the other. But, it also has some built-in capabilities to work around this overlap and lets you dictate the technique to follow. Refer to the properties as follows.

ChartAxis Property	Description
IntersectAction	<i>Hide</i> – labels are hidden to avoid intersection <i>MultipleRows</i> – labels are wrapped into multiple rows to avoid intersection <i>None</i> – no special action Labels may intersect <i>Rotate</i> – labels are rotated to avoid intersection <i>Wrap</i> – labels are wrapped to avoid intersection
HidePartialLabel	<i>True</i> – hides the labels that appear partially Usually the labels in the edges will be affected. <i>False</i> – labels are drawn as such No action will be taken.
EdgeLabelsDrawingMode	<i>Center</i> – draws the edge labels at the center of the GridLines  <i>Shift</i> – value indicating that edge label should be shifted to either left or right so that it comes within the Chart Area

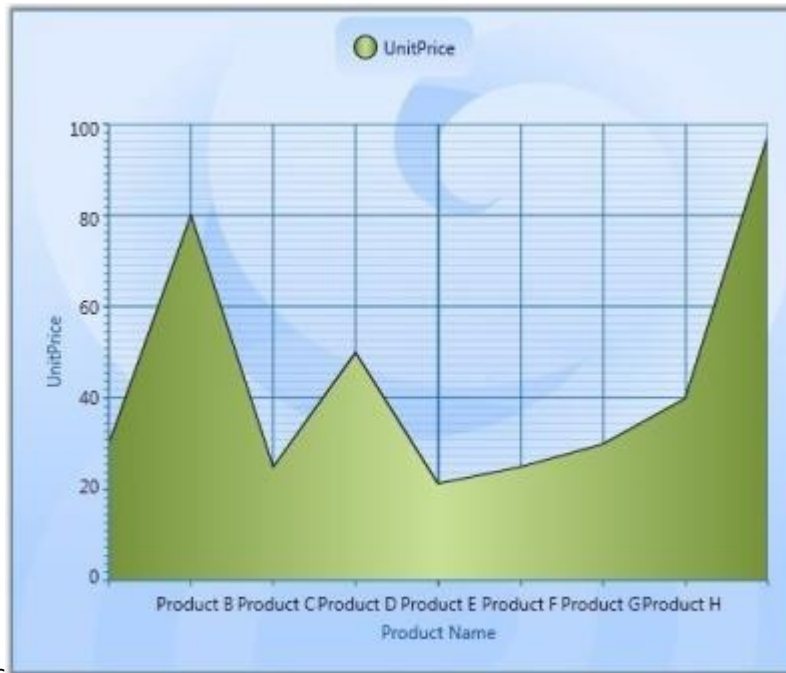
C#

```
Chart1.Areas[0].PrimaryAxis.IntersectAction =
ChartLabelIntersectAction.MultipleRows;
Chart1.Areas[0].PrimaryAxis.HidePartialLabel = true;
Chart1.Areas[0].PrimaryAxis.EdgeLabelsDrawingMode =
EdgeLabelsDrawingMode.Shift;
```

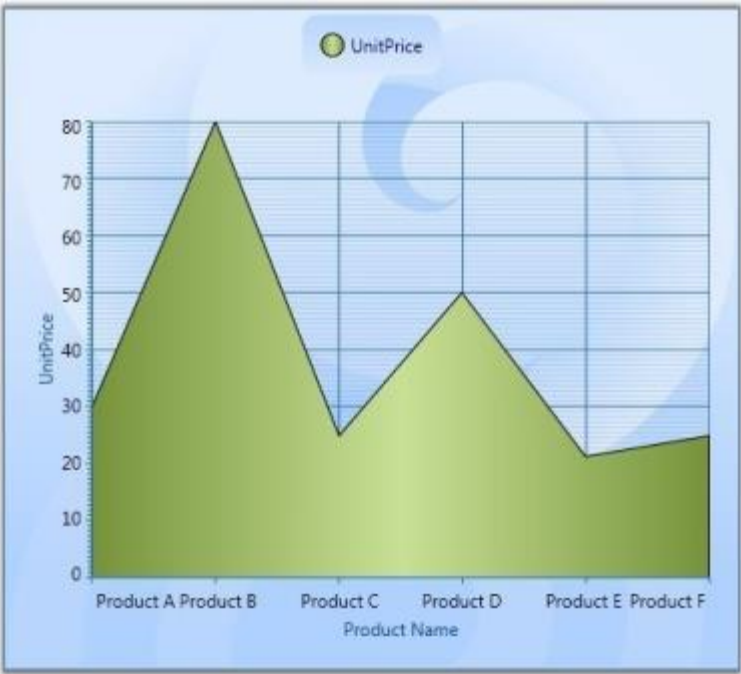

XML

```
<syncfusion:ChartArea.PrimaryAxis>  
<syncfusion:ChartAxis Header="Product Name" IntersectAction="MultipleRows"  
HidePartialLabel="True" EdgeLabelsDrawingMode="Shift" />  
</syncfusion:ChartArea.PrimaryAxis>
```

The following screenshot illustrates various techniques for avoiding the Label intersection.



![[Chart-Controls
Controlsimages/Chart-Controls_img176.jpeg)img176]](Chart-



![(Chart-Controlsimages/Chart-Controls_img177.jpeg)](Chart-Controlsimages/Chart-Controls_img177.jpeg)

![(Chart-Controlsimages/Chart-Controls_img178.jpeg)](Chart-Controlsimages/Chart-Controls_img178.jpeg)

Legend Panel Customization

Essential Chart enables you to arrange the legend items in any panel. You can arrange them in common panels like Grid, StackPanel, Canvas and WrapPanel. You can also create custom panel and arrange them.

Property

Property	Description	Type	Data Type	Reference links
LegendItemsPanel	Sets a template panel for arranging legend items.	Dependency Property	ItemsPanelTemplate	ItemsPanelTemplate Class

Customizing legend Panel



LegendItemsPanel

You can customize the legend panel using the property of ChartLegend. Following code illustrates this:

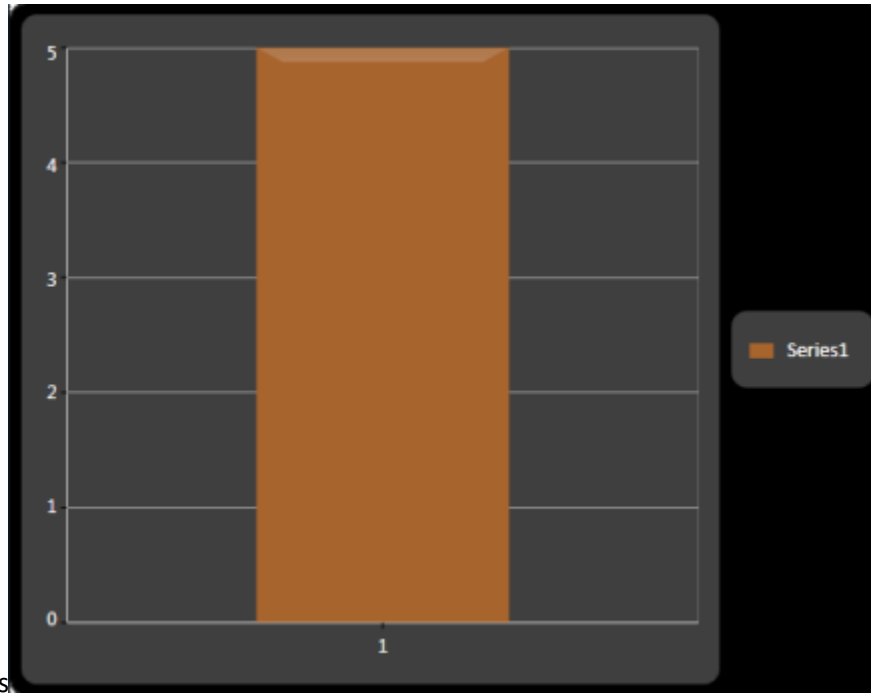
XML

```
<syncfusion:ChartLegend IconVisibility="Visible" BorderThickness="0"
LegendPanel="Custom">
<syncfusion:ChartLegend.LegendItemsPanel>
<ItemsPanelTemplate>                                <StackPanel
```

```

Orientation="Vertical"/>
</ItemsPanelTemplate>
</syncfusion:ChartLegend.LegendItemsPanel>
</syncfusion:ChartLegend>

```



![[Chart-Controls
img179]](Chart-Controlsimages/Chart-Controls_img179.png)

[Sample Link](#)

To view sample:

1. Open the Syncfusion Dashboard.
2. Select User Interface.
3. Click the WPF drop-down list and select Explore Samples.
4. Navigate to WPF\Chart.WPF\Samples\3.5\WindowsSamples\Chart Customization\Custom Legend Items Panel Demo\

Chart-Appearance in WPF Chart (Classic)

Chart Styles

Essential Chart for WPF supports styles. Users can customize the styles for a chart area, series, legend and axis using the ChartVisualStyle property.

Name of Property	Description	Type of Property	Value It Accepts	Property Syntax	Sub properties
ChartVisualStyle	Used to set the style for the Chart.	Dependency property	ChartStyles or any value of the	ChartVisualStyle="Style48"	ChartAreaStyleLegendStyleSeriesStylePrimaryAxisStyleNote: Type - Style

			chart style enum.		
--	--	--	-------------------------	--	--

Customizing Chart Style

The styles for a chart area, series, legend, and axis can be customized by using the following code.

XML

```
<sync:Chart x:Name="Chart1" ChartVisualStyle="ChocolateBlend">
//Insert the ChartLegend
<sync:Chart.Legends>
<sync:ChartLegend/>
</sync:Chart.Legends>
//Insert the ChartArea
<sync:ChartArea IsContextMenuEnabled="True">
//Inserts the ChartSeries
<sync:ChartSeries Data="1,5" Label="Series1"/>
</sync:ChartArea>
</sync:Chart>
```

C#

```
Chart1.ChartVisualStyle = ChartStyles.ChocolateBlend;
```

When the code runs, the following output displays.

![[sshot-1](Chart-Controlsimages/Chart-Controlsimg180.png)]

Chart Skins

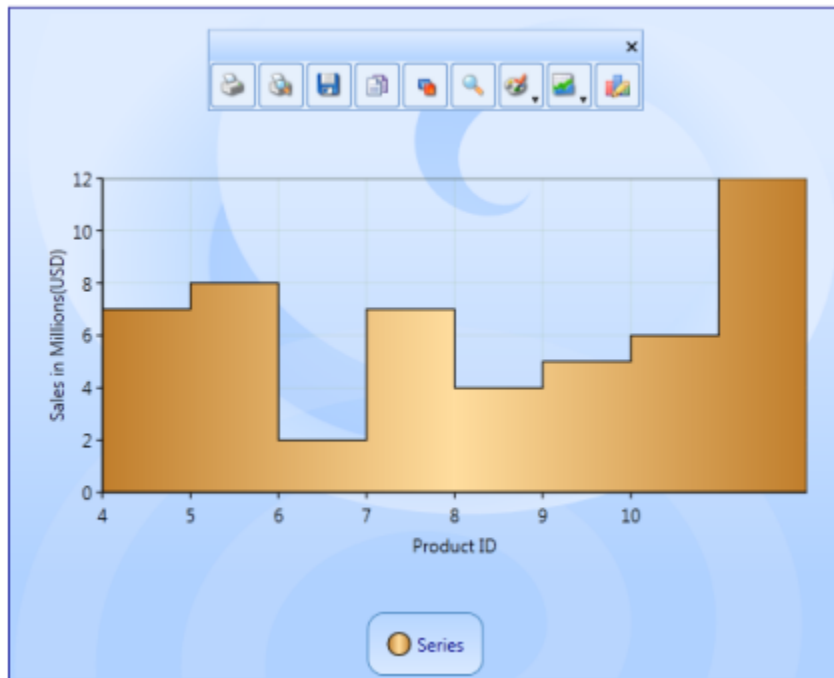
Essential Chart for WPF provides a number of built-in skins that delivers the chart with appealing look and feel with just one property, the VisualStyle property of the class SkinStorage from the Shared.WPF assembly. In addition for the skins getting applied to the window and Window title Bar, the skins will also be applied to all parts of the chart such as Chart Area and Chart Legend.

XML

```
<syncfusion:Chart Grid.Column="0"
syncfusion:SkinStorage.VisualStyle="Office2007Blue" >
</syncfusion:Chart>
```

C#

```
using Syncfusion.Windows.Shared;
SkinStorage.SetVisualStyle(Chart1, "Office2007Blue");
```



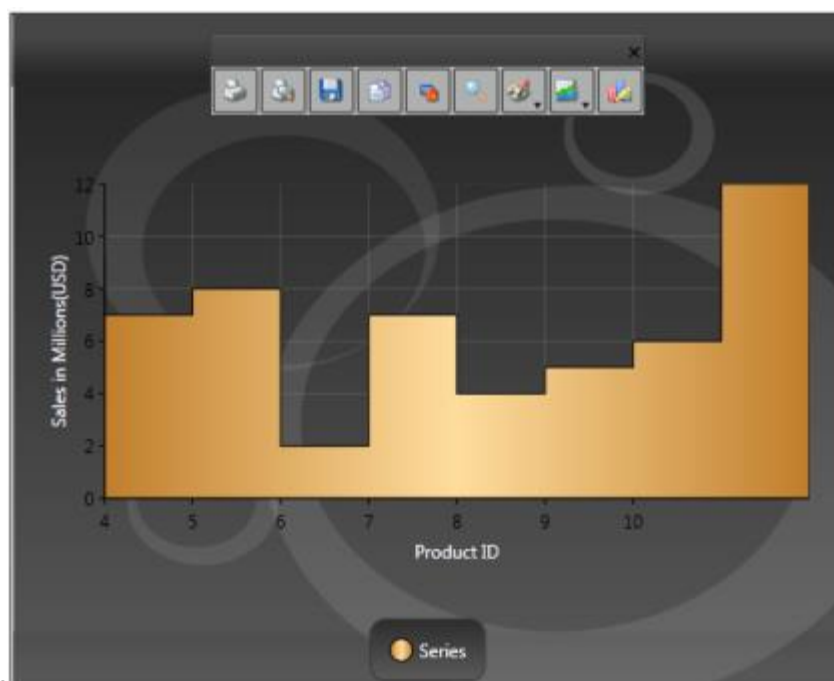
Note: Shared.WPF assembly

should be referenced in the project to make use of this settings.

Various Built-In skins supported are:

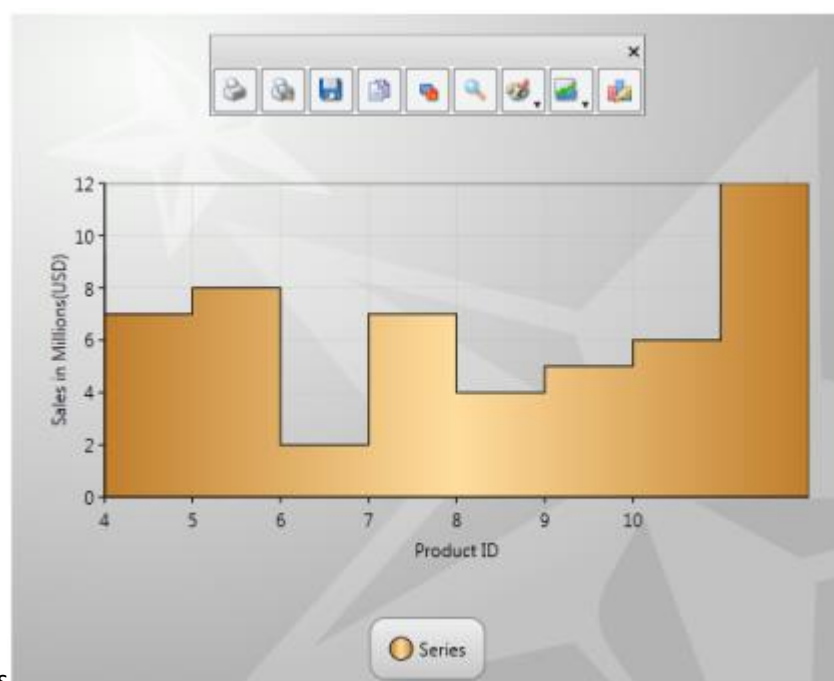
- Default
- Blend
- Office2003
- Office2007Blue
- Office2007Silver
- Office2007Black
- CoolBlue
- BlueWave
- BrightGray
- ChocolateYellow
- ForestGreen
- LawnGreen
- MixedGreen
- SpringGreen
- OrangeRed
- VS2010

The following images illustrate the various skins applied to the Chart.



![[Chart-Controls
Controlsimages/Chart-Controls_img181.png)

img181]](Chart-

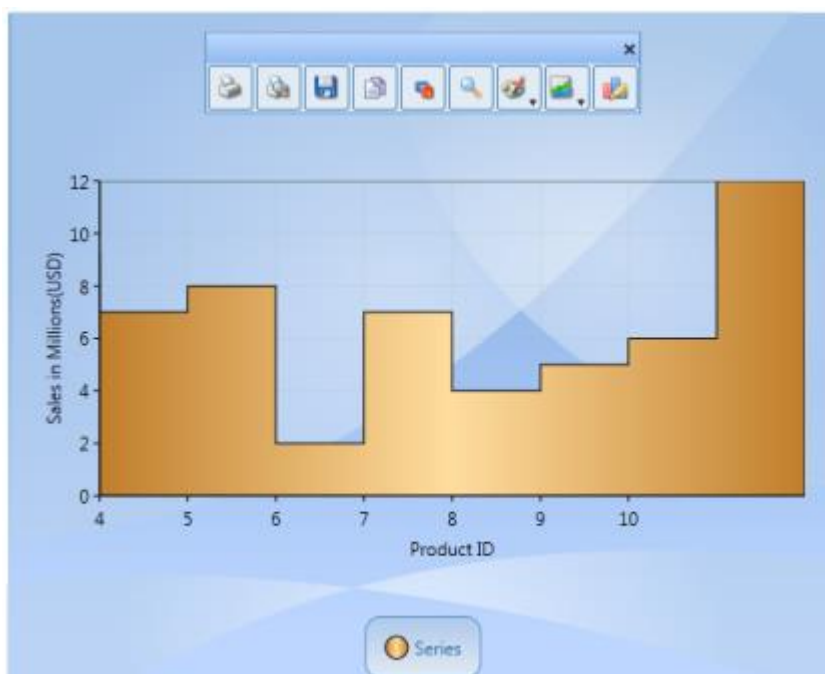


![[Chart-Controls
Controlsimages/Chart-Controls_img182.png)

img182]](Chart-

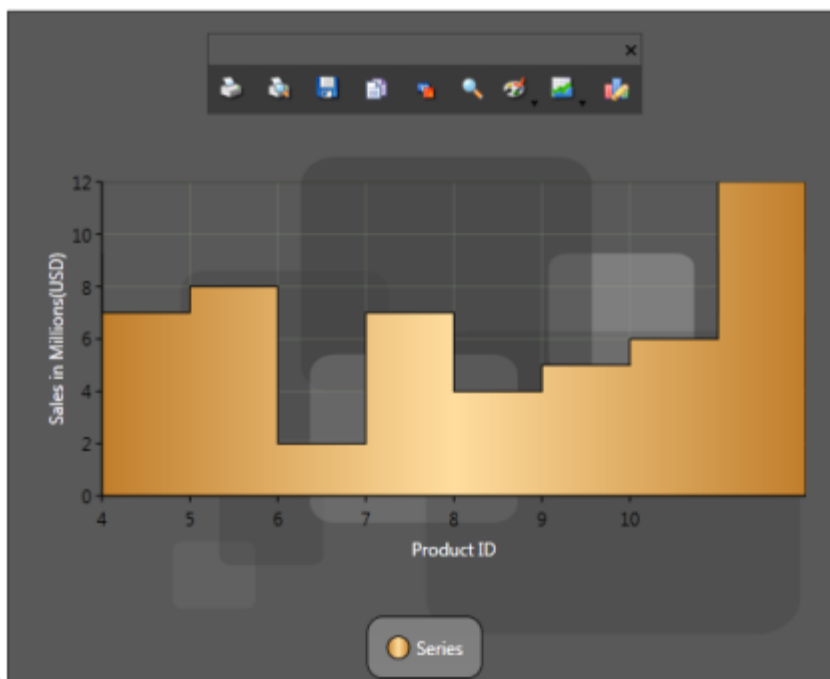


![[Chart-Controls
img183]](Chart-Controlsimages/Chart-Controls_img183.png)



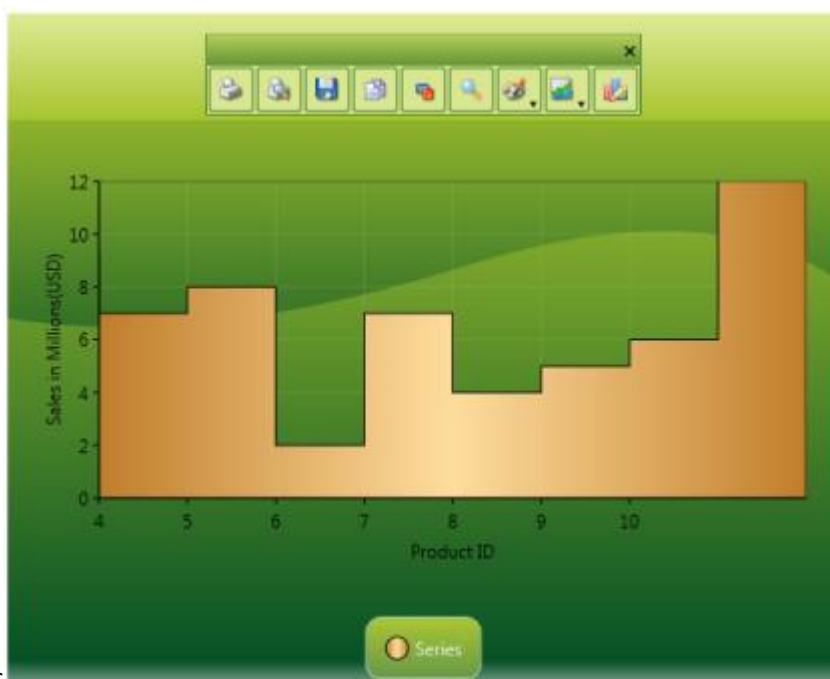
![[Chart-Controls
Controlsimages/Chart-Controls_img184.png]

img184]](Chart-



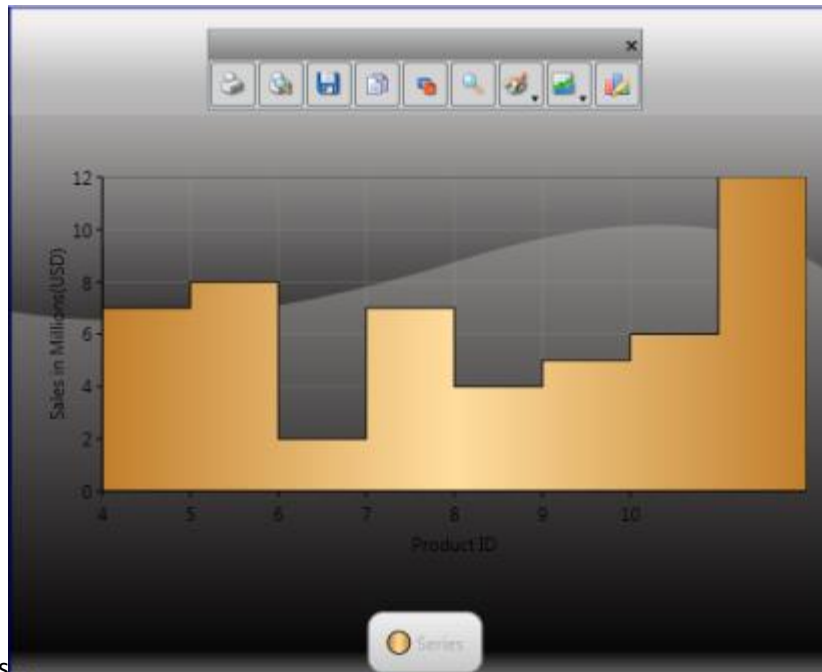
![[Chart-Controls
Controlsimages/Chart-Controls_img185.png)

img185](Chart-



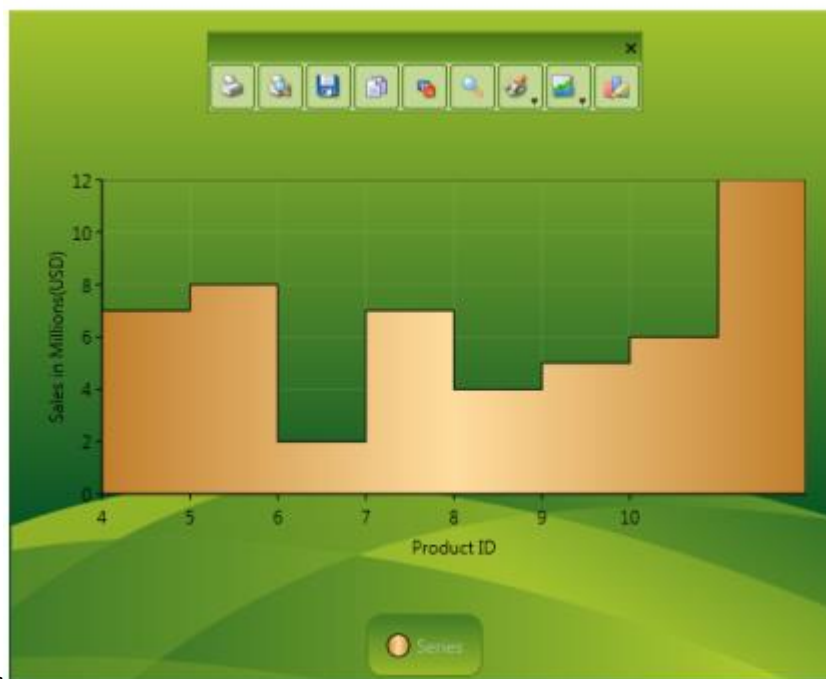
![[Chart-Controls
Controlsimages/Chart-Controls_img186.png)

img186](Chart-



![[Chart-Controls
Controlsimages/Chart-Controls_img187.png)

img187](Chart-



![[Chart-Controls
Controlsimages/Chart-Controls_img188.png)

img188](Chart-

![[Chart-Controlsimg189](Chart-Controlsimages/Chart-Controls_img189.png)

Chart Animation

Essential Chart WPF now comes with animation support. Charts can be animated by using animation options available. The state-of-the-art animation lets you to create lively charts that can be used for marketing, attractive data presentation, and so on.

It allows you to:

- Set the animation option for a chart.
- Set the type of animation.
- Set the animation by each series.
- Set the duration for the animation.

Properties

The following table lists the properties and their usage in chart animation.

Property	Description	Type	Value Returned
EnableAnimation	Sets the Animation option for the Chart.	Dependency Property	Boolean
AnimateOption	Sets the Type of Animation.	Dependency Property	Enum(AnimationOptions)
AnimateOneByOne	Sets the Animation by each series.	Dependency Property	Boolean
AnimationDuration	Sets the duration for the Animation.	Dependency Property	Enum(TimeSpan)

Events

The following table lists the events and their usage in chart animation.

Event	Event Trigger	Event Args	Purpose
OnEaseAnimationChanged	Whenever the properties AnimationDuration, AnimateOneByOne, AnimateOption and EnableAnimation change.	-	To set the Animation option selected by the user.
OnEnableAnimationChanged	Whenever the properties AnimationDuration, AnimateOneByOne, AnimateOption and EnableAnimation change.	-	To select or unselect the Animation of chart.

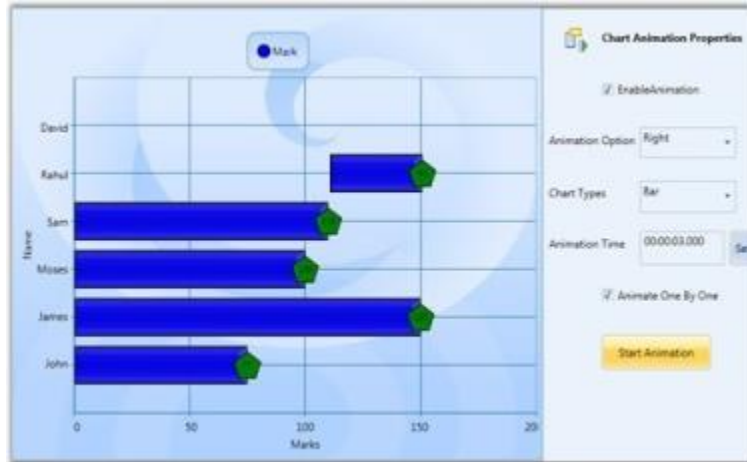
Methods

The following table lists the methods and their usage in chart animation.

Method	Return Type	Purpose
StartAnimation	Void	This method is called when the user starts animation by using the options specified.

Enabling and Customizing Chart Animation

The chart animation can be enabled by setting EnableAnimation property to



true.

XML

```
<syncfusion:ChartSeries Type="Column" EnableEffects="True" Label="Mark"
EnableAnimation="{Binding ElementName=enableanimation,
Path=IsChecked}" Interior="Blue" StrokeThickness="2"
DataSource="{StaticResource data}" BindingPathX="Name" BindingPathsY="Mark,
MinMark,MaxMark, Low, High" AnimateOneByOne="{Binding
ElementName=animateind, Path=IsChecked}">
</syncfusion:ChartSeries>
```

C#

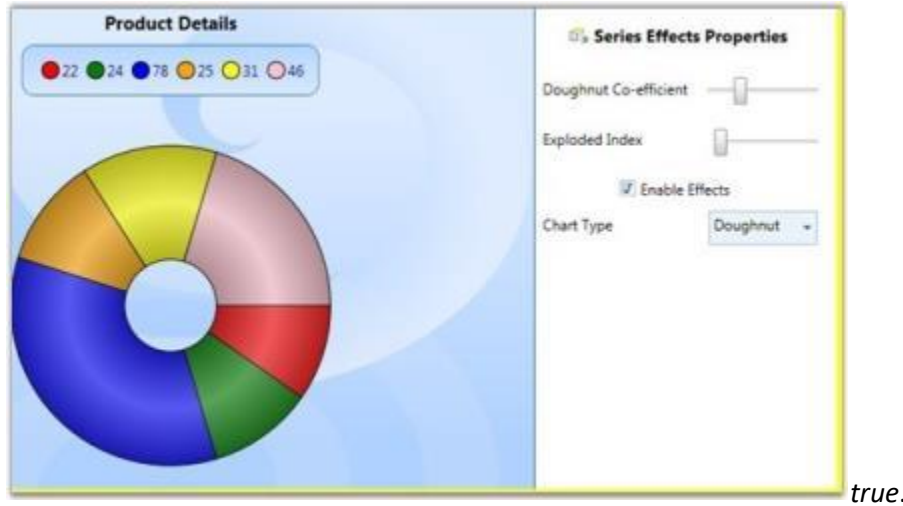
```
chart.Areas[0].Series[0].StartAnimation();
chart.Areas[0].Series[0].AnimationDuration = ts;
chart.Areas[0].Series[0].AnimateOption = AnimationOptions.Rotate;
```

Run the code. The following output is displayed.

![[Chart-Controlsimg190]](Chart-Controlsimages/Chart-Controls_img190.jpeg)

Chart Series Effects

Essential Chart lets you customize the series effects for all chart types. The look and feel can be enhanced by using the effects available. This can be achieved by setting the `EnableEffects` property to



true.

Properties

The following table provides more information on the property used.

Property	Description	Type	Value Returned
<code>EnableEffects</code>	Dependency	Boolean	Sets the effects on chart.

Events

The following table provides more information on the event used.

Event	Event Trigger	Event Args	Purpose
<code>EnableEffectsChanged</code>	This event is triggered whenever the value of the <code>EnableEffects</code> property changes.	<code>OnEnableEffectsChanged</code>	Sets the effects to the chart.

Customization of Chart Effects

The chart series effects can be applied to the chart by using the following code examples.

1. Using XAML

XML

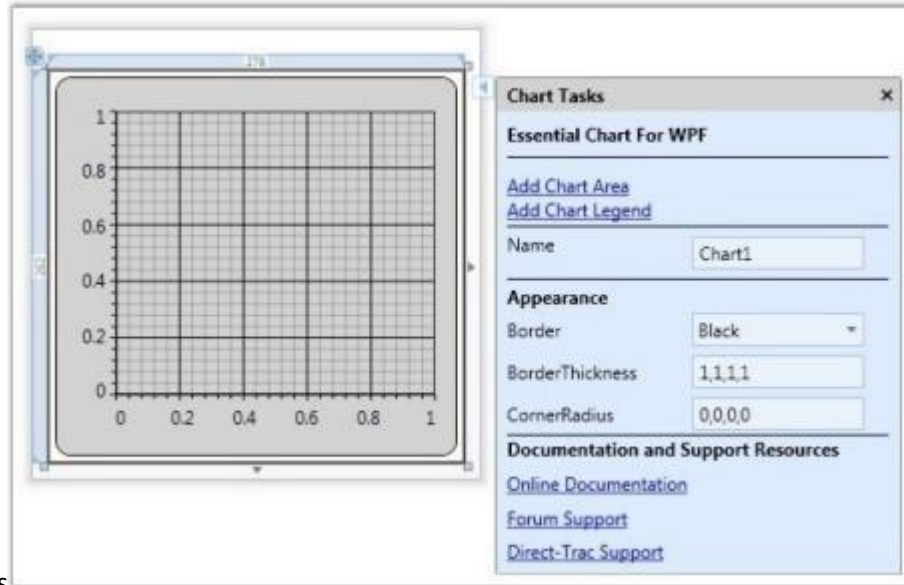
```
<syncfusion:ChartSeries Type="Column" EnableEffects="True" Label="Mark"
DataSource="{StaticResource data}" BindingPathX="Name" BindingPathsY="Mark,
MinMark,MaxMark, Low, High">
</syncfusion:ChartSeries>
```

2. Using C#

CSHARP

```
chart.Areas[0].Series[0].EnableEffects = True;
```

Run the code. The following output is displayed.



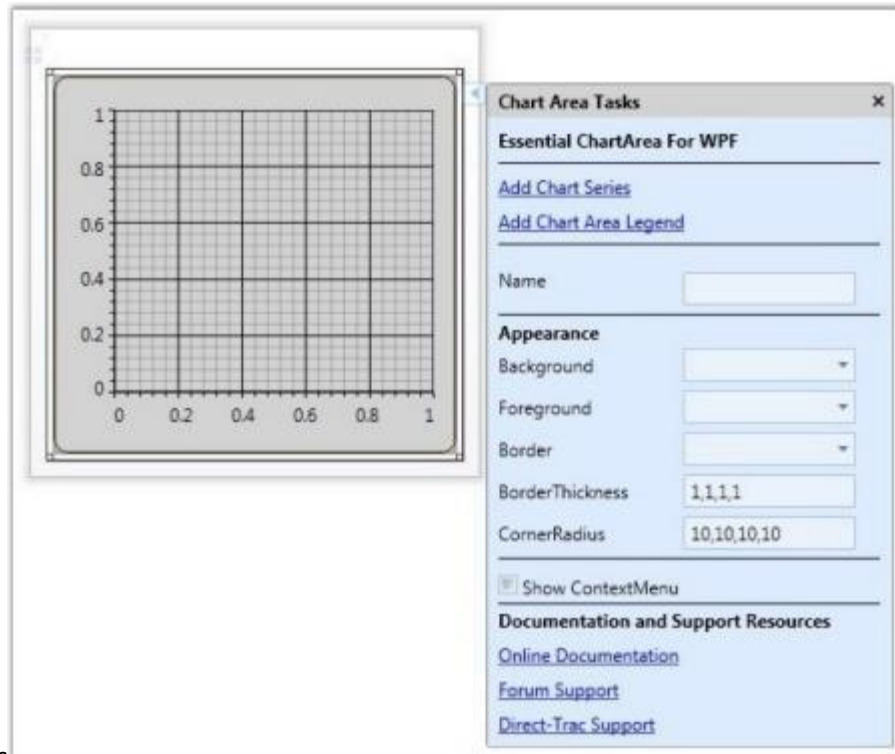
![[Chart-Controls
images191]](Chart-Controlsimages/Chart-Controls_img191.jpeg)

Design Time Support in WPF Chart (Classic)

Chart Control Smart Tag Support

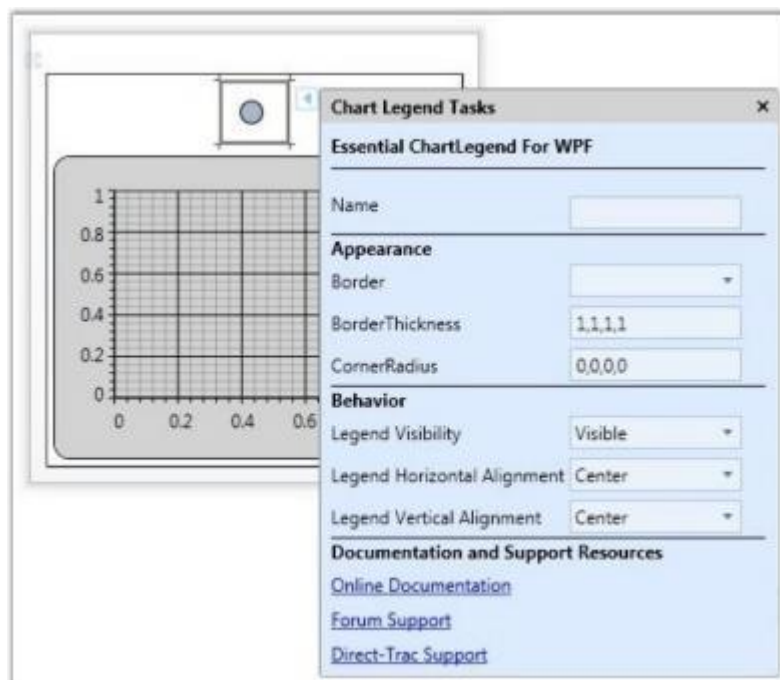
Smart Tag appears for the Chart, Chart Area and Chart Legend. By using the Smart Tag, you can add Chart Areas, Chart Legends and Chart Series. In addition, you can customize the Chart appearance by using the Smart Tag in the designer.

The following screenshot illustrates Smart Tag support in Chart.



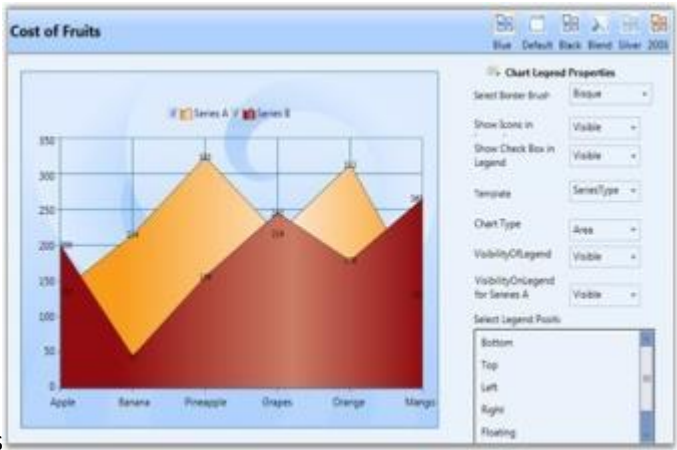
![[Chart-Controls
img192]](Chart-Controlsimages/Chart-Controls_img192.jpeg)

The following screenshot illustrates Smart Tag support in Chart Area.



![[Chart-Controls
img193]](Chart-Controlsimages/Chart-Controls_img193.jpeg)

The following screenshot illustrates Smart Tag support in Chart Legend.



![[Chart-Controls
Controlsimages/Chart-Controls_img194.jpeg)img194]](Chart-

Chart and ChartArea Legends in WPF Chart (Classic)

Legend Label

Essential Chart allows you to customize the legend icon of a chart. You can change the legend Icon of a chart legend by using the options provided. The legend icons can be represented in two ways:

- It can be set as a symbol.
- It can be set as the chart icon (in case of chart series).

The LegendIcon property is used to set the required icon.

- If LegendIcon property is set to a symbol, it will be displayed as the selected legend icon.
- If LegendIcon property is set to the SeriesType, the corresponding chart icon will be displayed as legend icon.

Properties

The following table provides more information on the property used.

Property	Description	Type	Value Returned
LegendIcon	The legend icon is displayed according to the option selected.Â	Dependency	Enum(CharLegendIcon)

Events

The following table provides more information on the event used.

Event	Event Trigger	Event Args	Purpose
LegendIconChanged	The event is triggered by calling a method when the value of LegendIcon is changed.	OnLegendIconChanged	The Icon is changed whenever the value for LegendIcon property changes.

Methods

The following table provides more information on the method used.

Method	Parameters	Return Type	Description
UpdateLegendIcon	ChartSeries	Void	Updation of the legend icon with a new icon.

Customizing Legend Icon

The legend icon can be customized by using the following code exaples.

1. Using XAML

XML

```
<syncfusion:ChartLegend Name="chrtlgnd" BorderThickness="0.5"
></syncfusion:ChartLegend>
<syncfusion:ChartSeries Name="SeriesA" LegendIcon="Circle" Type="Bar"
BindingPathX="FruitName"
BindingPathsY="Price,NumberOfFruits,FruitID,Year" Label="Series A"
Stroke="#FF000000" StrokeThickness="0.5" ></syncfusion:ChartSeries>
```

2. Using C#

CSHARP

```
Chart1.Areas[0].Series[0].LegendIcon = ChartLegendIcon.Circle;
```

Run the code. The following output is displayed.



![[Chart-Controls|images195]](Chart-Controlsimages/Chart-Controls_img195.jpeg)

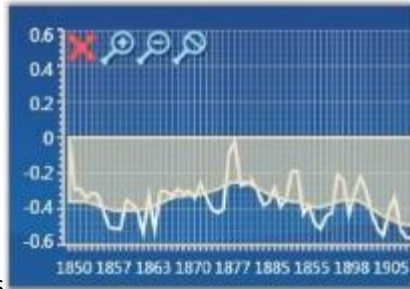
User-Interaction in WPF Chart (Classic)

Zooming

Chart for WPF lets you zoom into a narrower range within the chart area. This section discusses the following topics.

Zooming Using Mouse

You can switch to the zooming mode in the Chart by using the built-in context menu. Users can choose to zoom a specific series, if they do so, the rest of the series will be rendered semi-transparently, based on the `InactiveSeriesOpacityOnZoom` property (discussed later in this section).



![(Chart-Controls
Controls_img196.jpeg)](Chart-Controlsimages/Chart-Controls_img196.jpeg)

Zoom Using Zooming Toolkit

In the Zooming mode, a Zooming toolkit is displayed at the top-left corner of the ChartArea. Using the buttons in the Zooming toolkit, ChartSeries can be zoomed in, out, reset or closed (to exit zoom mode).



![(Chart-Controls
Controls_img197.jpeg)](Chart-Controlsimages/Chart-Controls_img197.jpeg)

Display/Hide Buttons in Zooming Toolkit

The visibility of the Zooming Toolkit or the individual buttons in the toolkit can be controlled by using the following properties.

Property	Description
ZoomInButtonVisibility	gets or sets zoom in button visibility
ZoomOutButtonVisibility	gets or sets zoom out button visibility
ZoomCloseButtonVisibility	gets or sets zoom close button visibility
ZoomResetButtonVisibility	gets or sets zoom reset button visibility
ZoomingToolkitVisibility	gets or sets zooming toolkit visibility

XML

```
<sfchart:ChartArea
  sfchart:ChartZoomingToolkit.ZoomInButtonVisibility="Collapsed"
  sch:ChartZoomingToolkit.ZoomOutButtonVisibility="Hidden"
  sch:ChartZoomingToolkit.ZoomResetButtonVisibility="Collapsed"
  sch:ChartZoomingToolkit.ZoomingToolkitVisibility="Visible">
  <!--your code here-->
</sfchart:ChartArea>
```

CSHARP

```

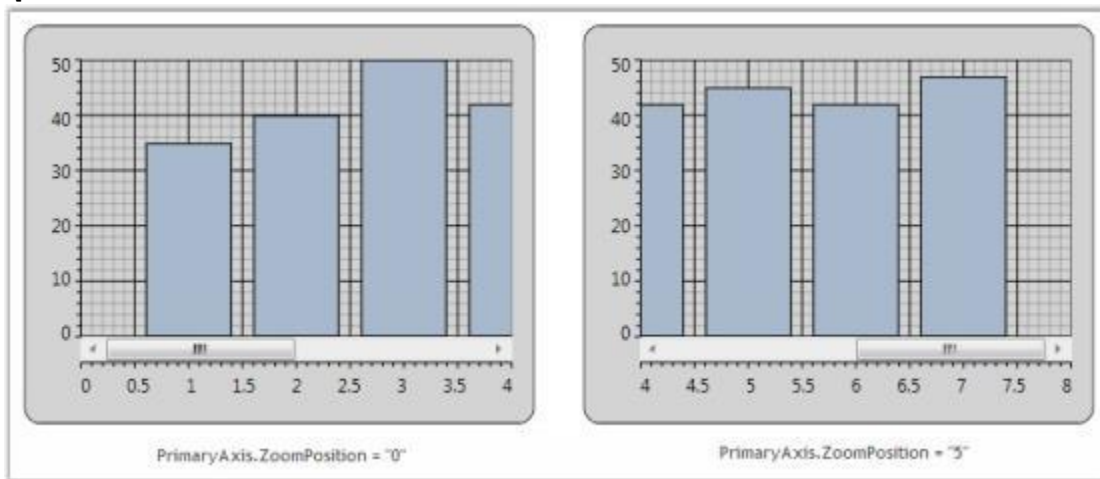
ChartZoomingToolkit.SetZoomInButtonVisibility(chartArea,
Visibility.Collapsed);
ChartZoomingToolkit.SetZoomOutButtonVisibility(chartArea,
Visibility.Hidden);
ChartZoomingToolkit.SetZoomResetButtonVisibility(chartArea,
Visibility.Collapsed);
ChartZoomingToolkit.SetZoomingToolkitVisibility(chartArea,
Visibility.Visible);

```

Zooming by Manual Drag

You can also manually drag-select an area to perform the zoom operation.

![[Chart-Controls



img198](Chart-Controlsimages/Chart-Controls_img198.jpeg)

Zooming Through Code

Chart can be zoomed programmatically by using Chart Commands, Keyboard Keys and Pre-defined Properties.

Zooming Using Chart Commands

Zooming mode can be enabled by calling the appropriate Chart Commands.

CSHARP

```

// To enter Zooming mode.
ChartAreaCommands.SwitchZooming.Execute(null, chart1.Areas[0]);
// To perform Zoom In.
ChartAreaCommands.ZoomIn.Execute(null, Chart1.Areas[0]);
// To perform Zoom Out.
ChartAreaCommands.ZoomOut.Execute(null, Chart1.Areas[0]);
// To Perform Zoom Reset.
ChartAreaCommands.ZoomReset.Execute(null, Chart1.Areas[0]);
// To Close the Zooming Toolkit again, use CancelZooming (exit zooming
mode).
ChartAreaCommands.CancelZooming.Execute(null, Chart1.Areas[0]);

```

Enabling Zooming Without Using Zooming Toolkit

Drag-select an area in the Chart to enable zooming when the zooming toolkit is invisible. The following code example illustrates this.

CSHARP

```
// Enable zoom mode.
ChartAreaCommands.SwitchZooming.Execute(null, Chart1.Areas[0]);
// Hide the zooming toolkit.
ChartZoomingToolkit.SetZoomingToolkitVisibility(Chart1.Areas[0],
Visibility.Collapsed);
```

Zoom Using Keyboard Keys

You can enable zooming by using the keyboard keys. The following code example shows how the ChartArea can be Zoomed by using the

ALT+I key combination.

CSHARP

```
// Adding Key Gesture Alt + I keys to Zoom In the Chart Area.
KeyGesture gesture = new KeyGesture(Key.I, ModifierKeys.Alt);
KeyBinding zoomInGesture = new KeyBinding(ChartAreaCommands.ZoomIn, gesture)
{ CommandTarget = this.Chart1.Areas[0] };
this.InputBindings.Add(zoomInGesture);
```

*Zooming using Pre-defined Properties**ZoomFactor*

The Chart can be zoomed by using the Axis.ZoomFactor property. The ZoomFactor is usually between 0 and 1. When set to 1, the chart will not be zoomed. When set to 0.5, the size of the chart will be doubled. Scrollbars will be automatically displayed to allow any section of the hidden range to be viewed. The default value is 1.

CSHARP

```
// Zoom the Chart with Zoom factor 0.5.
this.Chart1.Areas[0].PrimaryAxis.ZoomFactor = 0.5;
```

ZoomPosition

You can also programmatically specify the scrollbar position of the zoomed-in axes by using the Axis.ZoomPosition property.

The following code example shows how the chart with values 0 to 5 will have the scrollbar positioned near point 5, when the ChartAxis is zoomed, with the ZoomFactor set to 0.5.

CSHARP

```
// Zoom the Chart with Zoom factor 0.5.
this.Chart1.Areas[0].PrimaryAxis.ZoomFactor = 0.5;
// Position the ScrollBar to point 5.
this.Chart1.Areas[0].PrimaryAxis.ZoomPosition = 5;
```

![[Chart-Controlsimg199]](Chart-Controlsimg199.jpeg)

Disable Zooming

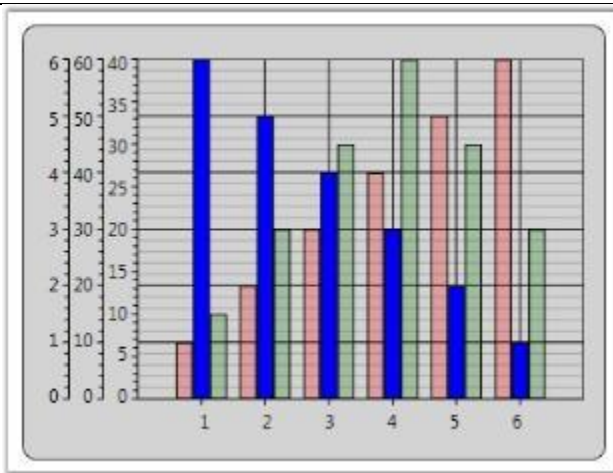
Zooming using mouse can be disabled for a particular axis by using the EnableZooming property. The following code example illustrates how zooming is disabled only for the Primary axis of the ChartArea while it is enabled for the other axes.

XML

```
<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis EnableZooming="False"/>
</syncfusion:ChartArea.PrimaryAxis>
```

CSHARP

```
this.Chart1.Areas[0].PrimaryAxis.EnableZooming = false;
```



Note: EnableZooming property is useful to disable zooming using mouse and keyboard keys. But it will not take effect when the ZoomFactor property is set. When ZoomFactor property is set between 0 to 1, and EnableZooming property is set to False, the axis will still be zoomed.

Zooming Specific ChartAxis Associated with a ChartSeries Through Code

The Axis associated with a particular ChartSeries can be enabled/disabled through the Zooming functionality. This is achieved by using the series.IsZoomable property.

The following code illustrates how zooming can be disabled for all the axis in Chart and enabled only for the axis associated with Chart Series2.

CSHARP

```
<syncfusion:Chart>
<syncfusion:ChartArea IsContextMenuEnabled="True" ZoomAllAxes="False">
<syncfusion:ChartSeries Label="Series1" Data="1 1 2 2 3 3 4 4 5 5 6 6"
Interior="Red" />
<syncfusion:ChartSeries Label="Series2" Data="1 60 2 50 3 40 4 30 5 20 6 10"
IsZoomable="True" Interior="Blue">
<syncfusion:ChartSeries.YAxis>
<syncfusion:ChartAxis Orientation="Vertical"/>
</syncfusion:ChartSeries.YAxis>
</syncfusion:ChartSeries>
<syncfusion:ChartSeries Label="Series3" Data="1 10 2 20 3 30 4 40 5 30 6 20"
Interior="Green">
<syncfusion:ChartSeries.YAxis>
<syncfusion:ChartAxis Orientation="Vertical"/>
</syncfusion:ChartSeries.YAxis>
</syncfusion:ChartSeries>
</syncfusion:ChartArea>
</syncfusion:Chart>
```

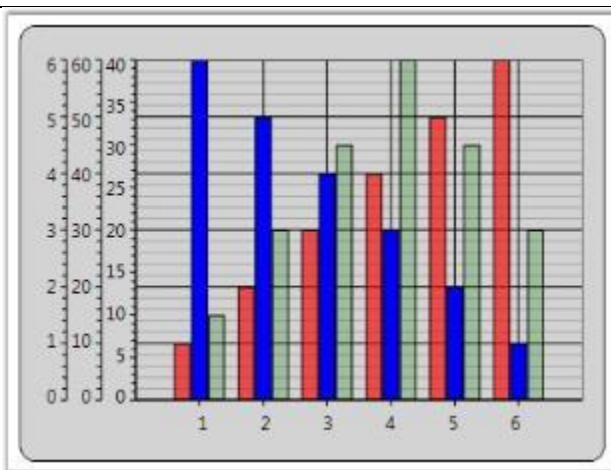
```
</syncfusion:ChartArea>
</syncfusion:Chart>
```

CSHARP

```
void Window1_Loaded(object sender, RoutedEventArgs e)
{
    Chart1.Areas[0].ZoomAllAxes = false;
    Chart1.Areas[0].Series[1].IsZoomable = true;
}
```

The following screenshot illustrates this.

![[Chart-Controlsimg200]](Chart-Controlsimages/Chart-Controls_img200.jpeg)



Note: Series.IsZoomable is used to enable/disable the zooming of the Axis associated with it. Hence this property is used only in Multiple Axes Charts. This property takes effect only when zooming is performed through code. (Refer to Zooming using Keyboard Keys in this page for zooming through keyboard keys). When zooming is enabled / disabled through the context menu, this property will not take effect.

Inactive Series Opacity

When an individual series in the chart is zoomed, the opacity of the series which have not been zoomed can be controlled, so that the series which is zoomed is clearly visible. This can be done by using the InactiveSeriesOpacityOnZoom property of the ChartSeries.

The following lines of code can be used to change the opacity of an inactive series while zooming.

XML

```
<syncfusion:ChartSeries Name="Series1" Label="Series1" Interior="Red"
    InactiveSeriesOpacityOnZoom="0.65" Data="1 1 2 2 3 3 4 4 5 5 6 6"/>
```

![[Chart-Controlsimg201]](Chart-Controlsimages/Chart-Controls_img201.jpeg)

Fractional Values in Axis while Zooming

On zooming the chart, the ChartAxis Labels appear with fractional values. This can be restricted by using the Axis.IsFractionEnabledOnZoom property.

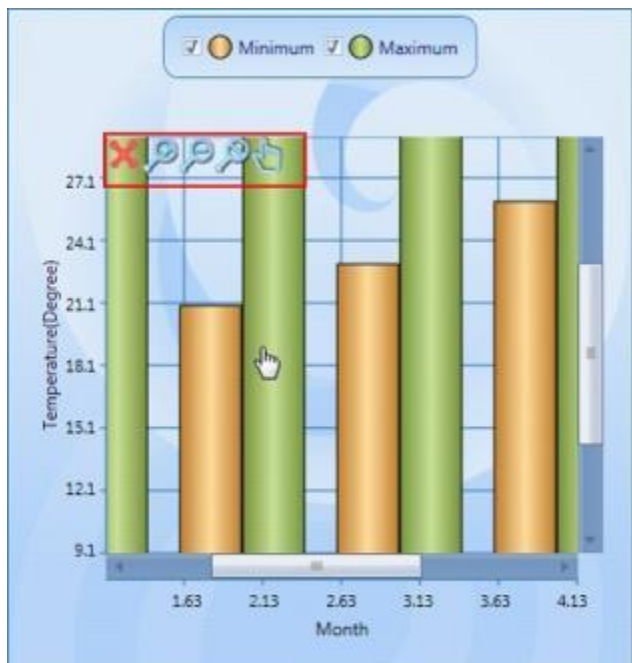
XML

```
<syncfusion:Chart Name="Chart1" Height="300" Width="400">
  <syncfusion:ChartArea IsContextMenuEnabled="True" >
    <syncfusion:ChartArea.SecondaryAxis>
      <syncfusion:ChartAxis IsFractionEnabledOnZoom="False" />
    </syncfusion:ChartArea.SecondaryAxis>
    <syncfusion:ChartSeries Name="Series1" Label="Series1" Interior="Red"
      Data="1 1 2 2 3 3" />
  </syncfusion:ChartArea>
</syncfusion:Chart>
```

CSHARP

```
// Hides fractional values in Secondary Axis labels.
Chart1.Areas[0].SecondaryAxis.IsFractionEnabledOnZoom = false;
```

For more details, refer to the sample in the following location:



... \My

Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\User Interaction\Zoom And Scrolling Demo
[Zooming and Panning support for Chart WPF](#)

Essential Chart WPF is now enhanced with Zooming and Panning. This feature is used to drag the Zoomed chart area from one point to the other.

Adding Zooming and Panning

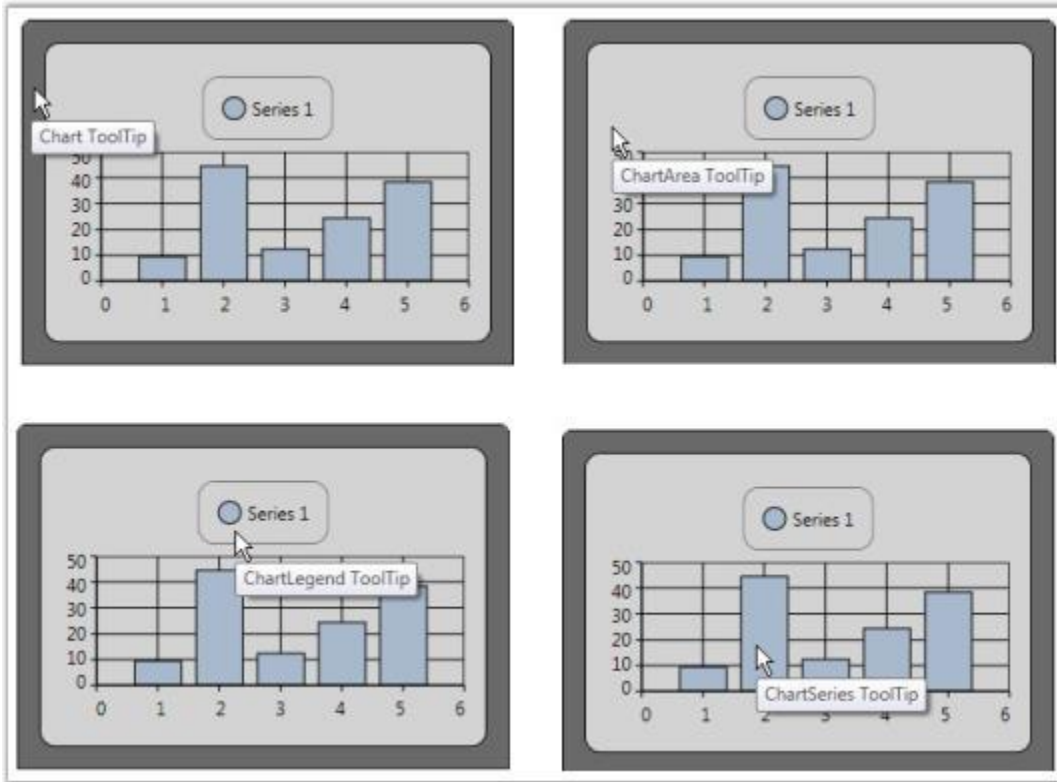
Add Zooming and Panning to the chart, by using the following code.

XML

```
<syncfusion:Chart Name="Chart1" Grid.Row="1" Margin="10">
  <syncfusion:ChartArea Name="area" IsContextMenuEnabled="True"/>
</syncfusion:Chart/>
```

CSHARP

```
ChartArea area = this.Chart1.Areas[0];
area.IsContextMenuEnabled = true;
```

![Chart-Controls

img202](Chart-Controlsimages/Chart-Controls_img202.jpeg)

Highlighting And Selection

The following topics discuss highlight data points and selecting data points:

ToolTip

ToolTip can be shown on various sections of a chart control such as chart, chart area, chart legend and chart series. Refer to the following code example for creating and assigning the tooltip to the chart.

XML

```
<!-- Sets Tooltips for a chart-->
<sf:Chart ToolTip="Chart Tooltip">
  <syncfusion:ChartArea Margin="10" ToolTip="ChartArea Tooltip">
    <syncfusion:ChartArea.Legend>
      <syncfusion:ChartLegend ToolTip="ChartLegend Tooltip" />
    </syncfusion:ChartArea.Legend>
    <syncfusion:ChartSeries Label="Series 1" Data="1 10 2 45 3 13 4 25 5 39"
      ToolTip="ChartSeries Tooltip" IsIndexed="False"/>
  </syncfusion:ChartArea>
</syncfusion:Chart>
```

The following image illustrates the tooltip feature in various sections of the chart.

![[Chart-Controls



img203](Chart-Controlsimages/Chart-Controls_img203.jpeg)

Default ToolTips for a Chart

Essential Chart for WPF is now enhanced with the ShowTooltip property, which allows ToolTips to be displayed.

Name of property	Description	Type of property	Value it accepts	Property syntax
ShowTooltip	Controls the visibility of ToolTips	Dependency Property	Bool or True/False.	ShowToolTip="True"

Setting a Tooltip for a Chart

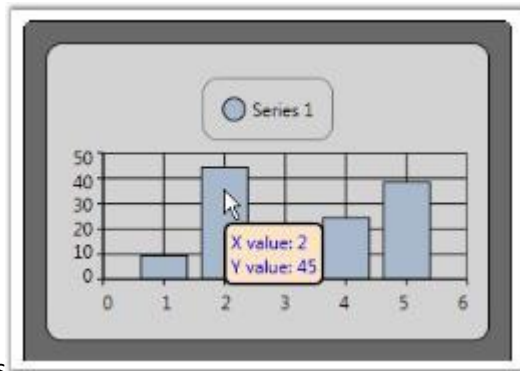
To enable ToolTips for a chart, set the ShowTooltip property to True. The following code illustrates setting the ShowTooltip property.

XML

```
<syncfusion:ChartSeries ShowToolTip="True">
```


C#

```
Series.ShowToolTip = true;
```



![[Chart-Controlsimg204]](Chart-Controlsimages/Chart-Controls_img204.png)

Custom Tooltip for a Series

You can set custom tooltips for the chart series. Associate a custom tooltip text with appealing appearance to the chart series using the following code example.

XML

```
<Window.Resources>
  <!--Defining custom Tooltip in Resources-->
  <ToolTip x:Key="CustomToolTip">
    <ToolTip.Template>
      <ControlTemplate>
        <Border CornerRadius="5" Background="Bisque" TextBlock.Foreground="Blue"
          Padding="3"
          BorderBrush="Black" BorderThickness="2">
          <Grid>
            <Grid.RowDefinitions>
              <RowDefinition/>
              <RowDefinition/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
              <ColumnDefinition/>
              <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <TextBlock Text="X value: " />
            <TextBlock Text="{Binding CorrespondingPoints[0].DataPoint.X}"
              Grid.Column="1" />
            <TextBlock Text="Y value: " Grid.Row="1" />
            <TextBlock Text="{Binding CorrespondingPoints[0].DataPoint.Y}"
              Grid.Column="1" Grid.Row="1" />
          </Grid>
        </Border>
      </ControlTemplate>
    </ToolTip.Template>
  </ToolTip>

  <!--Note: To have custom text alone a simple tooltip as below could be used
  <ToolTip x:Key="SimpleToolTip" Content="{Binding
    CorrespondingPoints[0].DataPoint.X}" />-->
</Window.Resources>
```

```
<!--Associate Custom Tooltip with ChartSeries-->
<syncfusion:ChartSeries Label="Series 1" Data="1 10 2 45 3 13 4 25 5 39"
ToolTip="{StaticResource CustomToolTip}" IsIndexed="False"/>
```



![[Chart-Controlsimg205]](Chart-Controlsimages/Chart-Controls_img205.jpeg)

Toolbar

A built-in toolbar is available for the Chart control, which can be made visible to enable the user to do the following functionalities:

- Print the chart
- Print preview the chart
- Save Chart as an image
- Copy chart image to clipboard
- Toggle Legend Appearance
- Toggle Zoom mode
- Change the color palette of the chart
- Change Chart Type

The code as follows illustrates how a ToolBar could be added to the Chart control.

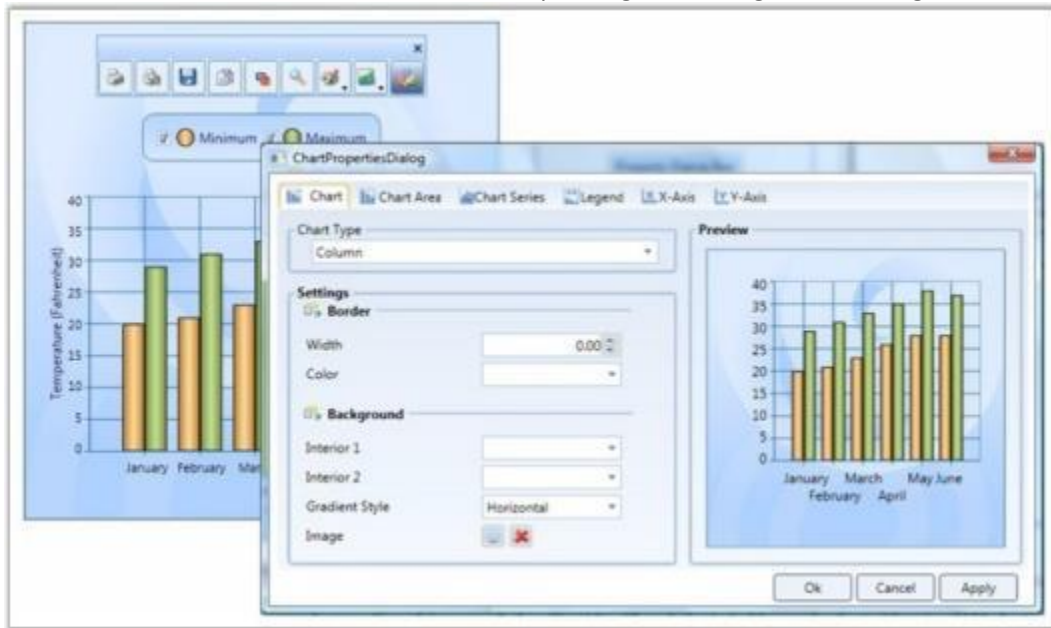
XML

```
<syncfusion:Chart.ToolBar >
<syncfusion:ChartToolBar Name="chartToolbar"
SelectedItemChanged="ChartToolBar_SelectedItemChanged"
CloseButtonVisibility="Visible" TitleBarVisibility="Visible" >
</syncfusion:ChartToolBar>
</syncfusion:Chart.ToolBar>
```

![[Chart-Controlsimg206]](Chart-Controlsimages/Chart-Controls_img206.jpeg)

Excluding or including the Toolbar While Printing and Saving a Chart

You can exclude or include the toolbar while printing and saving a chart using the



ShowToolBarOnPrintAndSave property. To include the toolbar, set this to true. To exclude it set this to false. By default this is set to true.

The following code illustrates how to exclude the toolbar while printing and saving:

XML

```
<syncfusion:Chart ShowToolBarOnPrintAndSave="False">
  <syncfusion:Chart.ToolBar>
    <syncfusion:ChartToolBar/>
  </syncfusion:Chart.ToolBar>
  <syncfusion:ChartArea>
    </syncfusion:ChartArea>
  </syncfusion:Chart>
```

C#

```
Chart1.ShowToolBarOnPrintAndSave = false;
The following code illustrates how to include the toolbar while printing and saving:
```

XML

```
<syncfusion:Chart ShowToolBarOnPrintAndSave="True">
  <syncfusion:Chart.ToolBar>
    <syncfusion:ChartToolBar/>
  </syncfusion:Chart.ToolBar>
  <syncfusion:ChartArea>
    </syncfusion:ChartArea>
  </syncfusion:Chart>
```

C#

```
Chart1.ShowToolBarOnPrintAndSave = true;
```

Property Settings Dialog

This topic provides an introduction to Property Settings Dialog. Property Settings Dialog lets you to change the properties of the chart during runtime.

This topic encapsulates the following details:

- Property Settings Dialog
- What are all the properties that can be set?
- Advantages Of Property Settings Dialog
- Special Features of Property Settings Dialog

Property Settings Dialog

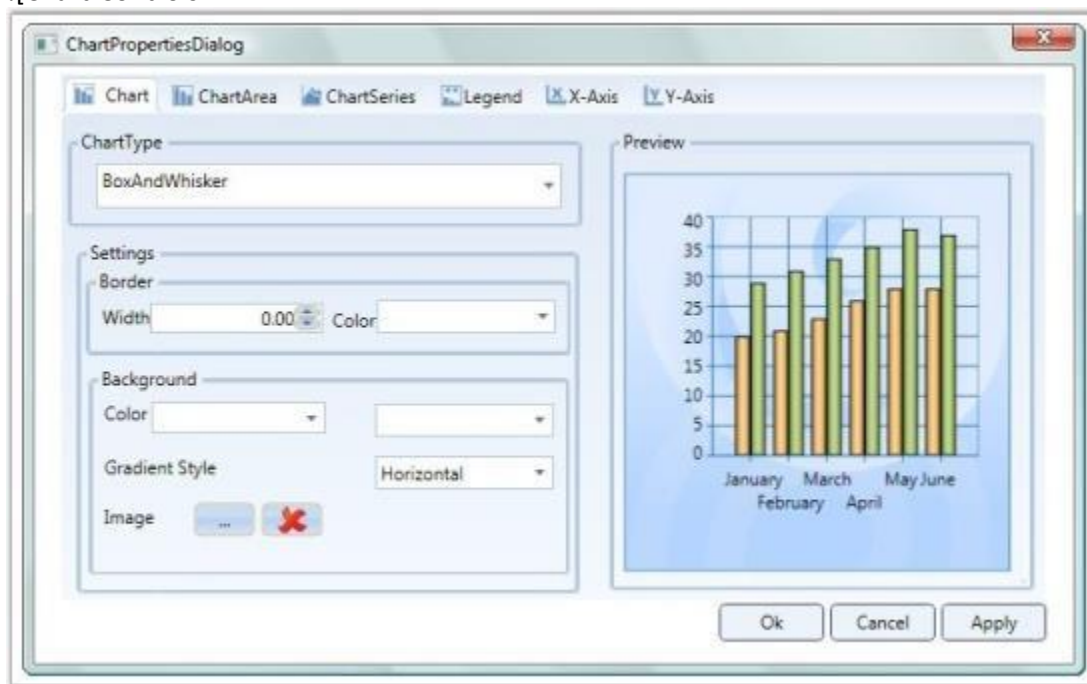
Essential Chart for WPF provides the Property Settings Dialog, to change the properties of the chart during runtime. This is to provide the developer, the ability to change the chart properties without the using the code.

There are two ways to invoke this dialog. They are, Using Tool Bar and ShowPropertyDialog. The following section briefs these two options.

Using ToolBar

By clicking the Properties Tool Item in the Toolbar, the property settings dialog can be invoked.

![Chart-Controls



img207](Chart-Controlsimages/Chart-Controls_img207.jpeg)

Using ShowPropertyDialog

The API, Show Property dialog, helps you to invoke the property settings dialog.

C#

```
// Invokes the property dialog box.
```

```
private void PropertyDialogbtn_Click(object sender, RoutedEventArgs e)
{
    if (Chart1 != null)
    {
        Chart1.ShowPropertyDialog();
    }
}
```

VB.NET

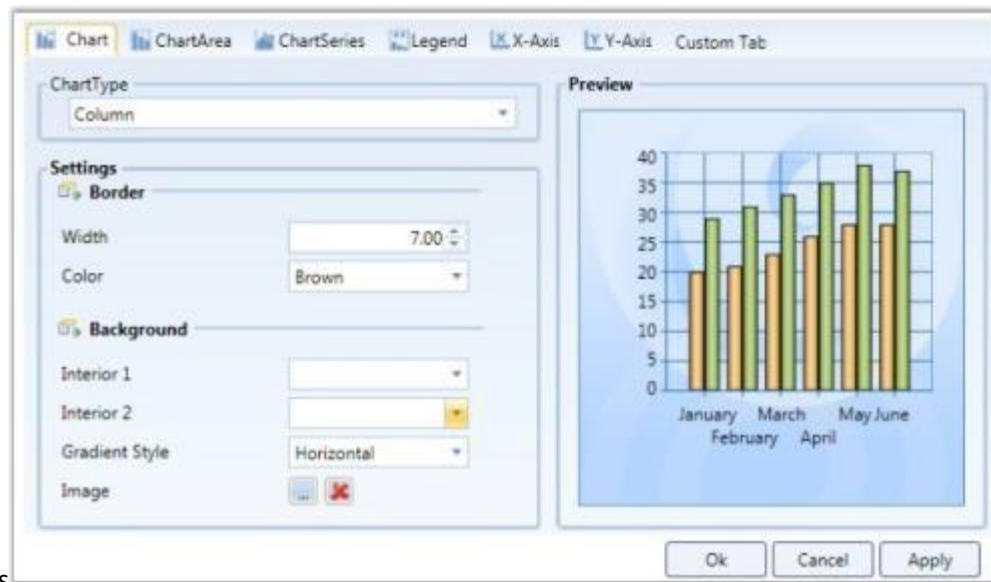
```
' Invokes the property dialog box.
Private Sub PropertyDialogbtn_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
If Chart1 IsNot Nothing Then
Chart1.ShowPropertyDialog()
End If
End Sub
```

Setting Properties

The property settings dialog allows you to set various properties of the chart. The chart properties are grouped together according to their functionality. The Property Settings Dialog contains the following Tab items.

- Chart
- Chart Area
- Chart Series
- Legend
- x-axis
- y-axis

Tab Item	Description
Chart	contains the properties specifically related to ChartFor example, Chart Background property, Chart Border property, and so on.
Chart Area	contains the properties specifically related to Chart AreaFor example, Chart Area Background property.
Chart Series	contains the properties specifically related to Chart SeriesFor example, SeriesNumber, AdornmentInfo, and so on
Legend	contains the properties specifically related to LegendFor example, RowCount, ColumnCount, and so on.
X-Axis	contains the properties specific to Primary axis of the ChartFor example, AxisLineWidth, AxisTitle, and so on.
Y-Axis	contains the properties specific to Secondary axis of the Chart, which is similar to x-axis tab



![[Chart-Controls
img208]](Chart-Controlsimages/Chart-Controls_img208.jpeg)

The modified settings can be applied to the Main Chart when you click Apply or Ok.

Methods

Custom Tab

The WPF Chart property settings dialog allows you to add your own custom tabs in the dialog. The added custom tab also has features through other tabs, like initializing the properties, changing the settings of the properties, Applying settings to the main chart. The following segment illustrates this special feature.

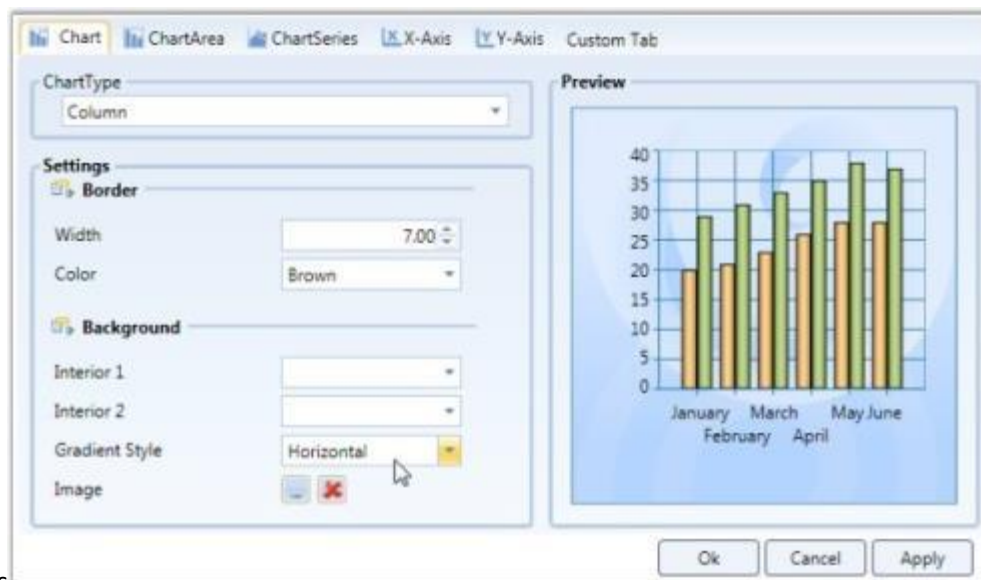
This option is to allow the developer to add his/her own tab into the property settings dialog. To add custom tab, AddCustomTabs API is used.

CSHARP

```
// Adding the Custom Tab
private void AddTab_Click(object sender, RoutedEventArgs e)
{
    tbi.Header = "Custom Tab";
    Chart1.AddCustomTabs(tbi);
}
```

VB.NET

```
'Adding the Custom Tab
Private Sub AddTab_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    tbi.Header = "Custom Tab"
    Chart1.AddCustomTabs(tbi)
End Sub
```



![[Chart-Controls
img209]](Chart-Controlsimages/Chart-Controls_img209.jpeg)

CSHARP

```
//Adding the custom tab in the proptdialog
Chart1.AddCustomTabs(tabItem);
```

Hiding the Tab

You can hide the tab that is not required using the Chart.HideTabItem(tabIndex) method.

![[Chart-Controlsimg210]](Chart-Controlsimages/Chart-Controls_img210.jpeg)

CSHARP

```
//To hide the ChartSeries Tab.
Chart1.HideTabItem(3);
```

Events

Various events that can be used while invoking a property Dialog are listed as follows:

Initialize CustomTab Page

The InitializeCustomTabPage event is used to initialize the Custom tab that is created using the above code.

CSHARP

```
//event raised
Chart1.InitializeCustomTabPage += new RoutedEventHandler(Chart1_ItemAdded);
//Initialize the custom tab:
private void AddTab_Click(object sender, RoutedEventArgs e)
{
    //Tab item Initialization
}
```

Apply Custom Tab Page

The Custom Tab page properties can be applied to the Main chart, similar to applying property settings available in the other tabs. This can be achieved by raising the ApplyCustomTabPages event.

CSHARP

```
// Event raising
Chart1.ApplyCustomTabPages += new RoutedEventHandler(Chart1_ItemApplied);
//Applying the Custom tab item
void Chart1_ItemApplied(object sender, RoutedEventArgs e)
{
    //Code for applying the custom tab item.
}
```

For more details, refer to the sample in the following location:



Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\User Interaction\Property Dialog Demo

Zoomed Area Selection

Essential Chart WPF lets the user to select the zoomed area in a chart. This can be achieved using the ZoomedXRange and ZoomedYRange properties. Under a zoomed state, you can select:

- The range of zoomed area in horizontal axis.
- The range of zoomed area in vertical axis.

Properties

The following table provides more information on the property used.

Property	Description	Type	Value Returned
ZoomedXRange	Dependency	DoubleRange	Selects the zoomed area for horizontal axis.
ZoomedYRange	Dependency	DoubleRange	Selects the zoomed area for vertical axis.

Methods

The following table provides more information on the method used.

Method	Parameters	Return Type	Description
VisibleRangeForZoomAllAxis	ChartArea,ChartAxis	Void	Enables zoom in all axes.
VisibleRangeForZoomHorizontalAxis	ChartArea,ChartAxis	Void	Enables zoom in x-axis.
VisibleRangeForZoomVerticalAxis	ChartArea,ChartAxis	Void	Enables zoom in y-axis.

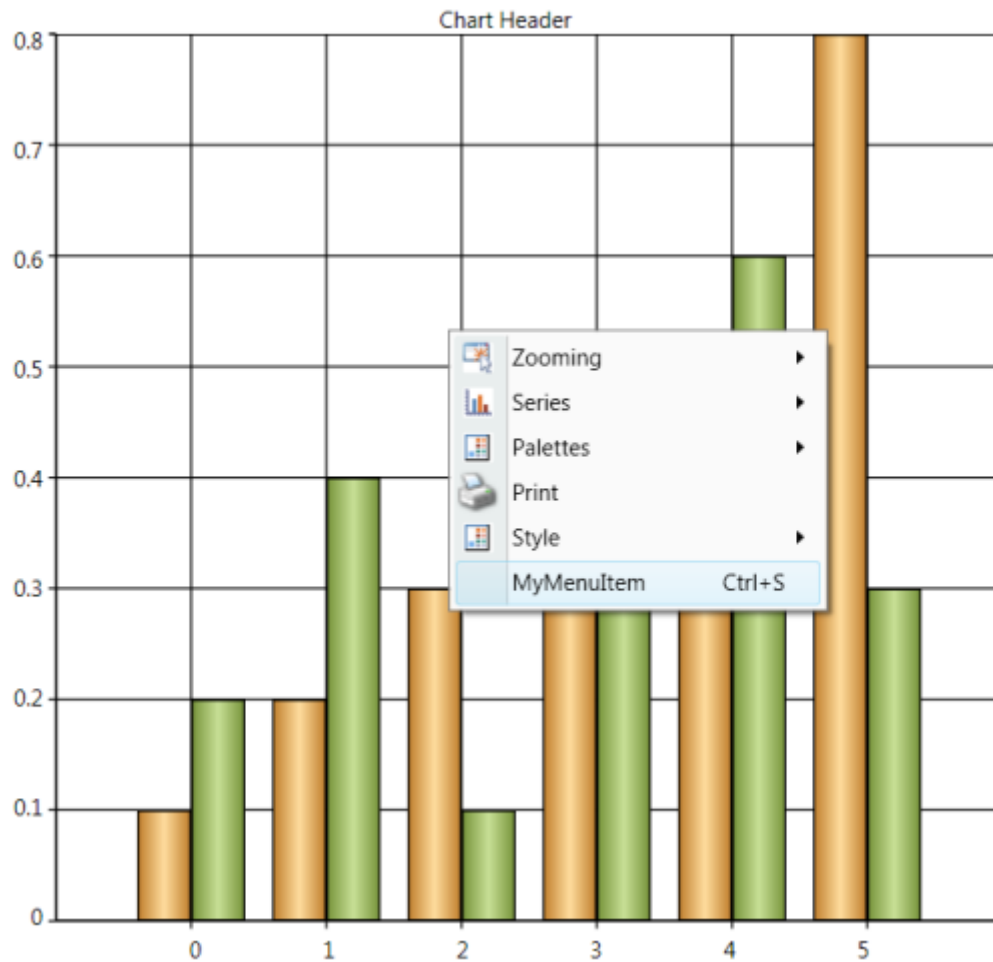
Selecting a Zoomed Area

The following code example illustrates selection of zoomed area in a chart.

1. Using XAML

XML

```
<TextBox Margin="0,6,12,2" Name="textBox1" Grid.Column="2" Grid.Row="1"
Text="{Binding ElementName=area,Path=ZoomedXRange,
Converter={StaticResource rangeConverter}}" />
<TextBox Margin="0,6,11.999,2" Name="textBox2" Grid.Column="4" Grid.Row="1"
Text="{Binding ElementName=area,Path=ZoomedYRange,
Converter={StaticResource rangeConverter}}" />
```



![[Chart-Controls
images211]](Chart-Controlsimages/Chart-Controls_img211.jpeg)

Customizing Context Menu

Essential Chart for WPF enables users to add custom menu items to the default context menu.

Adding a Menu Item to the Context Menu

Menu items can be added by using two methods: adding with the default menu, or adding as a new list.

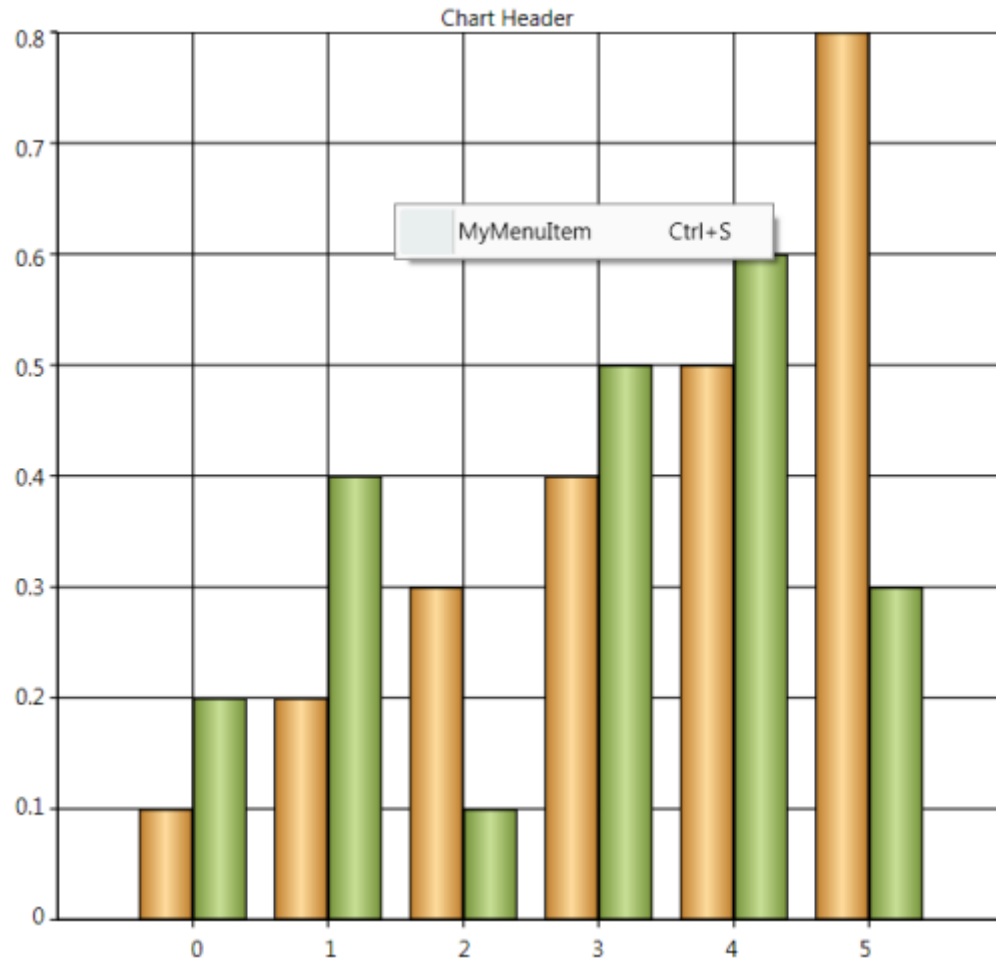
Adding with the Default Menu

By setting `ContextMenuType` to `DefaultWithCustom`, the menu item is added along with the default list.

The following code illustrates this.

XML

```
<syncfusion:ChartArea IsContextMenuEnabled="True"
  ContextMenuType="DefaultWithCustom">
  <syncfusion:ChartArea.CustomContextMenuItems>
    <MenuItem Header="MyMenuItem" />
  </syncfusion:ChartArea.CustomContextMenuItems>
</syncfusion:ChartArea>
```



![[Chart-Controls

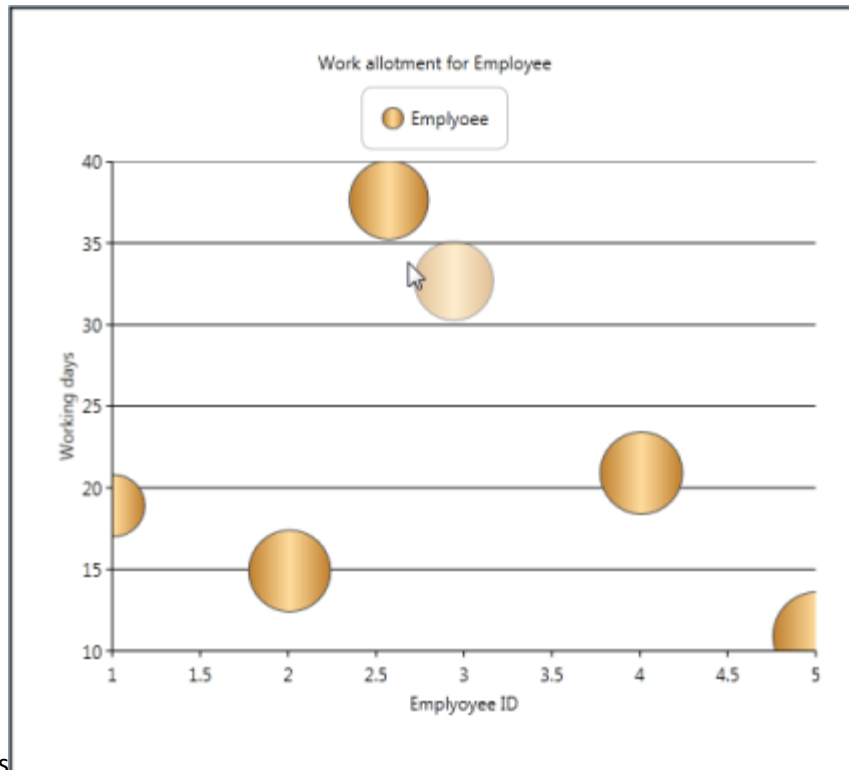
img212](Chart-Controlsimages/Chart-Controls_img212.png)

[Adding Menu Items as a New List](#)

By setting `ContextMenuType` to `Custom`, menu items are added as a new list.

XML

```
<syncfusion:ChartArea IsContextMenuEnabled="True" ContextMenuType="Custom">
  <syncfusion:ChartArea.CustomContextMenuItems>
    <MenuItemHeader="MyMenuItem" Command="ApplicationCommands.Save"/>
  </syncfusion:ChartArea.CustomContextMenuItems>
</syncfusion:ChartArea>
```



![[Chart-Controlsimages/Chart-Controls_img213.png]](Chart-Controlsimages/Chart-Controls_img213.png)

Name of Property	Description	Type of Property	Value It Accepts	Property Syntax	Sub Properties
CustomContextMenuItems	Used to customize the context menu.	Normal	ObservableCollection<MenuItem>	<pre><syncfusion:ChartArea.CustomContextMenuItems> <MenuItem Header="MyMenuItem"/> </syncfusion:ChartArea.CustomContextMenuItems></pre>	Property Name : ContextMenuType Type:DependencyProperty /ContextMenuTypes /Ex: ContextMenuTypes.Custom

Name of the Event	Description	Method	Event Args
ChartContextMenuEventHandler	Used to handle adding items to the context	ChartContextMenuEventHandler	ChartContextMenuEventArgs

	menu and removing items from the context menu.		
--	--	--	--

Built-in Drag-and-Drop Support for Chart Series

Features

This feature helps the user to drag the Chart point from one location to another location within the Chart area and improves the user interaction by editing the under bound model's object at run time. This feature also maps the mouse coordinates to the coordinates of the ChartAxis and positions the data point. Hence, all the relevant properties like tooltip and series annotation are changed according to the new position.

Use Case Scenarios

- Inbuilt Drag and Drop support for the Chart Series can be used in rescheduling the number of working days for any task assigned to employee.
- Reschedule the task start and end time using typical Gantt Chart type.
- It allows the user to modify the under bound data of segment dynamically by clicking and dragging the segment to the new position inside the Chart Area.

![Chart-Controlsimages214](Chart-Controlsimages/Chart-Controls_img214.png)

Note: The AllowSegmentDragDrop property does not work when you apply a custom data template for the Chart Series, because the Chart Series' default template is overridden by the custom template that affects the existing behavior.

Tables for Properties, Methods, and Events

Properties

Property Table

Property	Description	Type	Data Type	Reference links
AllowSegmentDragDrop	To set the Drag and Drop Support for the Chart Series	Dependency Property	Boolean	NA

Events

ChartSegmentDragging Table

Event	Description	Arguments	Type	Reference links
ChartSegmentDragging	Triggered before dragging is started	ChartSegment	Routed Event	NA

ChartSegmentDragged Table

Event	Description	Arguments	Type	Reference links
ChartSegmentDragged	Event is triggered immediately after dragging is started	ChartSegment	Routed Event	NA

ChartSegmentDropping Table

Event	Description	Arguments	Type	Reference links
ChartSegmentDropping	Triggered before dropping the segment	ChartSegment	Routed Event	NA

Event	Description	Arguments	Type	Reference links
ChartSegmentDropped	Event is triggered immediately after the segment is dropped.	ChartSegment	Routed Event	NA

[Sample Link](#)

To view sample:

1. Open the WPF sample browser from the dashboard.
2. Navigate to WPF Chart -> User Interaction->Drag and Drop support demo

Adding Inbuilt Drag and Drop Support for Chart Series to an Application

Inbuilt Drag and Drop Support can be added to an Application using the following code example:

CSHARP

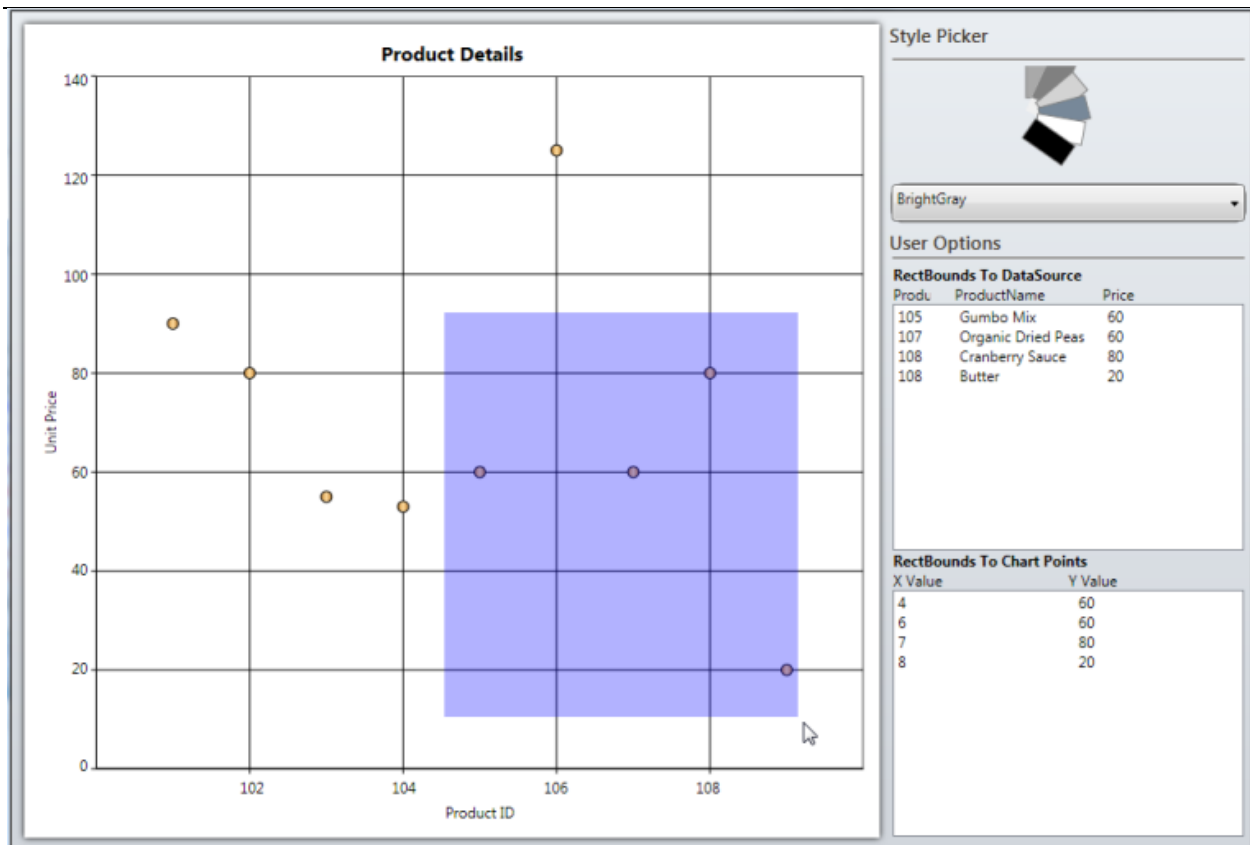
```
Chart1.Areas[0].AllowSegmentDragDrop = true;
```

XML

```
<sync:Chart x:Name="Chart1">
  <sync:ChartArea AllowSegmentDragDrop="True">
    <sync:ChartSeries x:Name="series1" Type="Bubble"/>
  </sync:ChartArea>
</sync:Chart>
```

[ChartAreaBounds](#)

Essential Chart is now enhanced to obtain the list of points and the DataSource present in the user-given rectangle bounds region, using BoundsToPoints and BoundsToDataSource methods of ChartArea control. The rectangle region given by the user can also convert to the corresponding axes range values using the ConvertBoundsToAxesRangeValues method of ChartArea control.

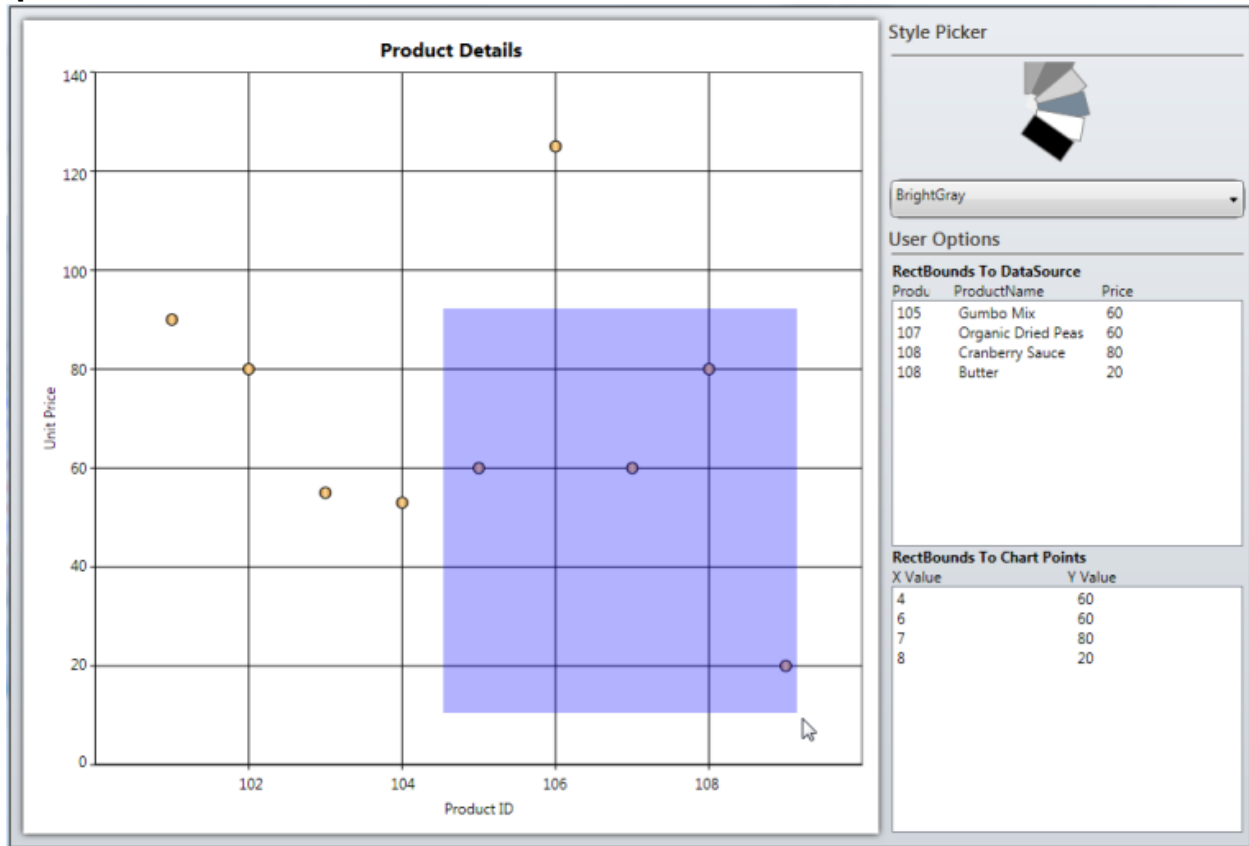


Note: The given points of the rectangle region can internally convert to the relative position of the ChartArea control.

Use Case Scenario

ChartAreaBounds is used in obtaining the list of points bound to a user-given rectangle .i.e. viewing the survey details in the area selected by the user.

![[Chart-Controls



images215]](Chart-Controlsimages/Chart-Controls_img215.png)

Adding ChartAreaBounds to an Application

BoundsToDataSource: You can use the `BoundsToDataSource` method of `ChartArea` control to get the list of the data that exist in the given rectangle region. The following code example illustrates the invoke of this method:

CSHARP

```
//Get the List of underlying objects available in the given "rect" region
chartdatasource.ItemsSource = chart.Areas[0].BoundsToDataSource(rect,
chart.Areas[0].Series[0]);
```

BoundsToPoints: You can use the `BoundsToPoints` method of `ChartArea` control to get the list of the chart points that exist in the given rectangle region. The following code example illustrates how to invoke this method.

CSHARP

```
//Get the list of chart points existing in the given "rect" region
chartpoints.ItemsSource = chart.Areas[0].BoundsToPoints(rect,
chart.Areas[0].Series[0]);
```

![[Chart-Controlsimages216]](Chart-Controlsimages/Chart-Controls_img216.png)

Return Type	Method Name	Description
-------------	-------------	-------------

ObservableCollection<object>	BoundsToDataSource(Rect rect, ChartSeries series)	Returns the collection of underlying data objects (existing in the user-given rectangle region) from the user bound data on the DataSource property of ChartSeries. rect:Â The given rectangle points can internally convert relative to the ChartArea control. series: Initializes the series, which contains the actual DataSource in Chart Area.
ObservableCollection<IChartDataPoint>	BoundsToPoints(Rect rect, ChartSeries series)	Returns the Chart points objects (existing in the given Rect region) from the specific ChartSeries. rect: The given rectangle points can internally convert relative to the ChartArea control. series: Initializes the series, which contains the actual DataSource in the Chart area.
Rect	ConvertBoundsToAxesRangeValues(ChartAxis xAxis, ChartAxis yAxis, Rect actualRect)	Converts the given actual rectangle region coordinates to the

		corresponding chart axes range values. xAxis: Initializes the X-Axis of the particular Chart series. yAxis: Initializes the Y-Axis of the particular Chart Series. actualRect: The given rectangle points can internally convert relative to the ChartArea control.
--	--	--

[Sample Link](#)

To run the sample:

1. Open Essential Studio Dashboard by selecting Start -> Program -> Syncfusion-> Essential Studio <> -> Dashboard.
2. To run the UI Chart WPF samples, select Run locally installed samples from the WPF drop-down list in the User Interface pane.
3. Select Chart in the sample browser.
4. Select User Interaction from the Essential Chart menu and choose Bounds To Rectangle Demo.
5. Click the Run Sample button.

To open the sample project:

1. Go to the following location in your system:
2. "<sample installation location>\Syncfusion\EssentialStudio\Version Number \WPF\Chart.WPF\Samples\3.5\WindowsSamples\User Interaction\Bounds To Rectangle Demo".
3. This location contains two sub folders CS and VB. You can open the sample projects from the respective folders based on your application developing language.

Serialization

Serialization is the process of converting the state of an object into a form that can be persisted or transported. The complement of serialization is deserialization, which converts a stream into an object. Together, these processes allow data to be easily stored and transferred.

[Serialization in Essential Chart WPF](#)

Essential Chart WPF has built-in support to serialize itself into stream of XAML string. Essential Chart works on the basis of XamlWriter.Save. The API Chart.Serialize serializes the Chart control into XAML string. API Chart.Deserialize does complement for Chart.Serialize, it converts the XAML string passed as parameter to this method into Chart object.

Following are some useful behaviors of Essential Chart Serialization feature.

- Essential Chart WPF serialization is able to save Bindings to the Chart control that is being serialized.
- Essential Chart WPF serialization can serialize Templates and Styles. For instance, ChartAxis.LabelTemplate can be serialized. Hence you can save and restore the template that is defined for ChartAxis Labels.
- Essential Chart WPF can serialize almost all types of data bound to it except XML data.
- The Chart control accesses the data set to the ChartSeries and saves them. If Binding is set on the ChartSeries.DataSource, then the serialized string contain Binding code in the XAML string. If ChartSeries.DataSource is initialized in XAML or C#, then the serialized string contain the run time value of the collection.
- Essential Chart Serialization feature enables to save and restore almost all interactive features available in Chart WPF. Interactive features namely, the current value of Zooming position, the interactive cursor position and so on can be saved and restored.

Use Case Scenarios

- Serialization feature can be used to clone Chart object.
- Chart can be saved into a XML / text file and restored.
- Chart can be stored into database and retrieved from application.
- Chart can be shipped across network. Chart can be serialized to XML and transfer it across network through web requests and responses.
- Apply Styles to the Chart and try to serialize the Chart, and reload the chart from saved XAML.
- Save and load Chart with templates applied using StaticResources and Key References.
- Save and Load the DataBound Chart Series.

Methods

Method	Description	Parameters	Type	Return Type	Reference links
Serialize	Serializes the Chart control into XAML string.	Public void Serialize()	N/A	string	Serialize
Deserialize	Deserializes the string that is being passed as parameter into Chart object. Returns null if the string parameter does not represent Chart object.	Public object Deserialize(string)	N/A	object	Deserialize

Sample Link

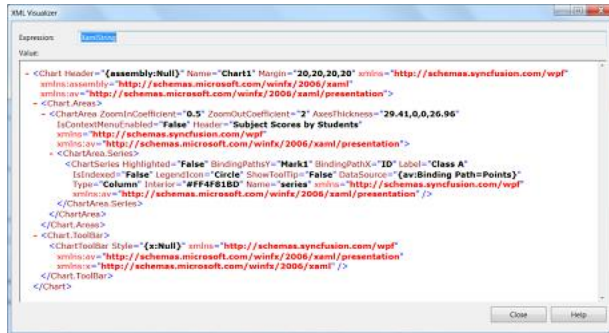
To run the UI WPF sample:

1. Open Essential Studio Dashboard by selecting Start -> Program -> Syncfusion-> Essential Studio <> -> Dashboard.

2. Select Run locally installed samples, from the WPF drop-down list on the User Interface pane.
3. Select Chart in the sample browser.
4. Select User Interaction -> Serialization Demo on the Essential Chart pane and click the Run Sample button.

To open the sample project:

Go to the following sample location in your system:



“sample installation

location>\Syncfusion\EssentialStudio\Version Number
 \WPF\Chart.WPF\Samples\3.5\WindowsSamples\User Interaction\Serialization Demo”

This location contains two sub folders CS and VB. You can open the sample projects from the respective folders based on your application developing language.

Adding Serialization to an Application

Serialize Chart control

The following code example explains the serialization of the Chart control.

CSHARP

```
string Serializedstring = Chart1.Serialize();
```

The Chart1 in above code example represents the Chart object, and Serializedstring in above code example has the serialized string (XAML string) while execution.

![Chart-Controlsimages217](Chart-Controlsimages/Chart-Controls_img217.png)

Serialized XAML string representing Essential Chart WPF

Deserialize Chart control

The following code example explains the deserialization of the Chart control.

CSHARP

```
Chart DeserializedChartControl = new Chart();
DeserializedChartControl =
DeserializedChartControl.Deserialize(Serializedstring) as Chart;
```

The Chart1 in above code example represents the Chart object, and Serialized string in the above code example needs to be XAML representation of a Chart object. If the string passed as a parameter to this method does not represent the Chart object, then the method returns Null value.

Property Dialog for Chart

The property dialog or property window lets you to modify the values of properties of the Chart control at run time. The property dialog helps you to accurately modify the look and feel of the control.

The property dialog has tabbed structure. There are five tabs available by default, namely Chart, ChartArea, ChartSeries, ChartAxis and ChartLegend. You can also add custom tabs to property window, which would help you to access the properties that are not available in these default tabs (For example, you can add custom tab which would modify the Chart Annotation points).

The property window has [localization support](#). The properties and their options can be viewed in local languages just by adding .resx file with local string for corresponding key values in the application.

Note: Any changes done in the property window directly affects the Chart control from Essential studio version 9.1 onwards whereas earlier versions have a preview chart in the property window to view the modifications.

Use Case Scenarios

- When the Chart control is used in an application and deployed, the end user or client can customize the look and feel of the Chart at run time. End user can set their own brushes, headers for Chart, Annotation Labels of Chart and etc. using the Chart property window.
- If you need to change the Annotation labels of the Chart Control at run time, then you can add a custom tab, in which you can define controls to change the Chart Annotation label's properties. The property window has this addition tab added to it and hence you can modify the Annotation labels of Chart at run time.

Tables for Properties, Methods, and Events

Properties

Property	Description	Type	Data Type	Reference links
PropertyWindowTabs	Represents the collection of tabs that are displayed in the property window.	Dependency property	TabItemCollection	NA

Methods

Method	Description	Parameters	Type	Return Type	Reference links
ShowPropertyWindow	Displays the Chart Property window	public void ShowPropertyDialog()	N/A	Void	NA
ClosePropertyWindow	Hides the Chart Property window	public void ClosePropertyDialog()	N/A	Void	NA

Events

Event	Description	Arguments	Type	Reference links
-------	-------------	-----------	------	-----------------

ChartPropertyWindowOpening	tde property is triggered before tde property window is displayed.tdis event is cancellable.	ChartPropertyWindowCancelEventArgs e.e.PropertyWindow gives tde entire property window	Routed Event	-
ChartPropertyWindowOpened	tde property is triggered after tde property window is displayed.	ChartPropertyWindowCancelEventArgs e.e.PropertyWindow gives tde entire property window	Routed Event	-
ChartPropertyWindowClosing	tde property is triggered before tde property window is displayed.tdis event is cancellable.	ChartPropertyWindowCancelEventArgs e.e.PropertyWindow gives tde entire property window	Routed Event	-
ChartPropertyWindowClosed	tde property is triggered before tde property window is displayed.	ChartPropertyWindowCancelEventArgs e.e.PropertyWindow gives tde entire property window	Routed Event	-

[Sample Link](#)

To run the UI WPF sample:

1. Open Essential Studio Dashboard by selecting Start -> Program -> Syncfusion-> Essential Studio <> -> Dashboard.
2. Select Run locally installed samples, from the WPF drop-down list on the User Interface pane.
3. Select Chart in the sample browser.
4. Select User Interaction -> Property Dialog Demo on the Essential Chart pane and click the Run Sample button.

To open the sample project:

Go to the following sample location in your system:

"<sample installation location>\Syncfusion\EssentialStudio\Version Number
\WPF\Chart.WPF\Samples\3.5\WindowsSamples\User Interaction\Property Dialog Demo"

This location contains two sub folders CS and VB. You can open the sample projects from the respective folders based on your application developing language.

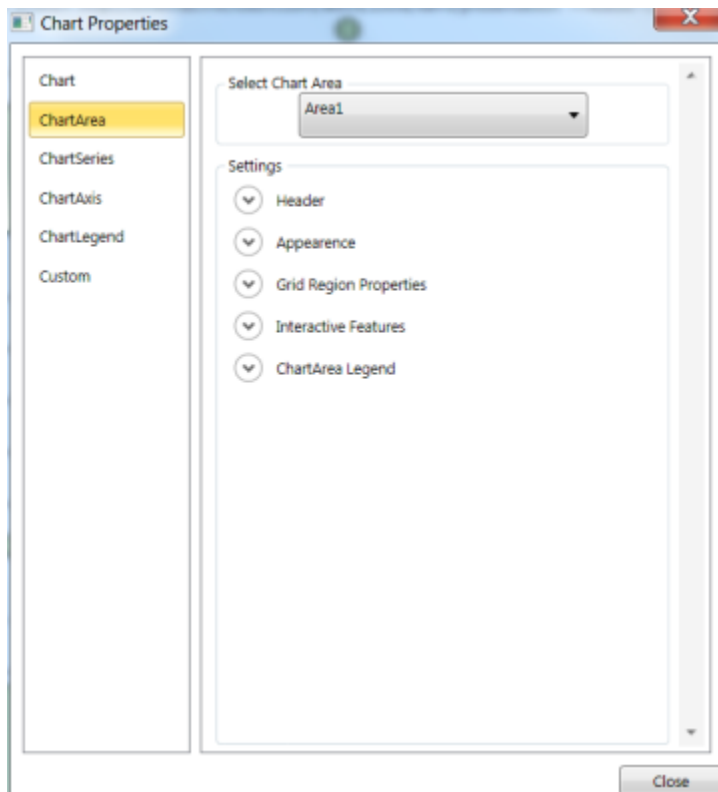
Adding Property Dialog to an Application

There are two ways to invoke the Property dialog. They are, using Toolbar and ShowPropertyDialog. The following section briefs these two options.

Opening Property Dialog through code

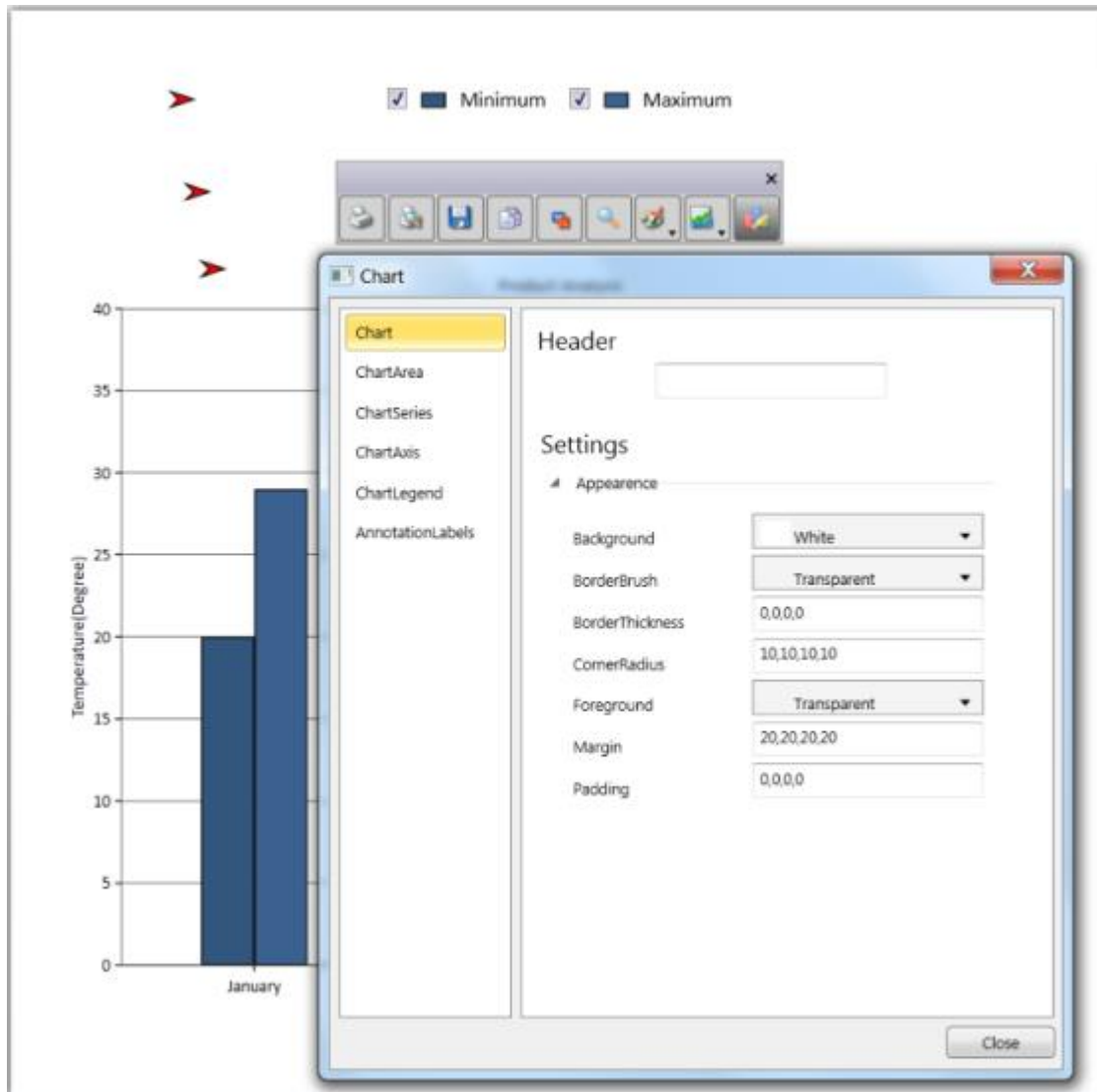
CSHARP

```
Chart1.ShowPropertyDialog();
```



Using Toolbar

By clicking the Properties Tool Item in the Toolbar, the property settings dialog can be invoked.



Adding custom tabs to property window of Chart

You can include a custom tab to the property window in your application easily by adding a tab to the `PropertyWindowTabs` property. The following code example explains adding custom tab to the property window through XAML and in C#.

XAML

```
<syncfusion:Chart Margin="20" Grid.Row="1" Grid.Column="0" Name="Chart1">
  <syncfusion:Chart.PropertyWindowTabs>
    <TabItem Header="AnnotationLabels">
      <TabItem.Content>
      </TabItem.Content>
    </TabItem>
  </syncfusion:Chart.PropertyWindowTabs>
</syncfusion:Chart>
```

C#

```
TabItem CustomTab = new TabItem();
```



```
Chart1.PropertyWindowTabs.Add(CustomTab);
```

Listen to opening and closing of the property window

You can listen to opening of the property window by adding a delegate method to the event `ChartPropertyWindowOpening`. Similarly, you can listen to closing of the property window by adding a delegate method to the event `ChartPropertyWindowClosing`.

C#

```
Chart1.ChartPropertyWindowOpening += new  
ChartPropertyWindowCancelEventHandler(Chart1_ChartPropertyWindowOpening);  
Chart1.ChartPropertyWindowClosing += new  
ChartPropertyWindowCancelEventHandler(Chart1_ChartPropertyWindowClosing);
```

XML

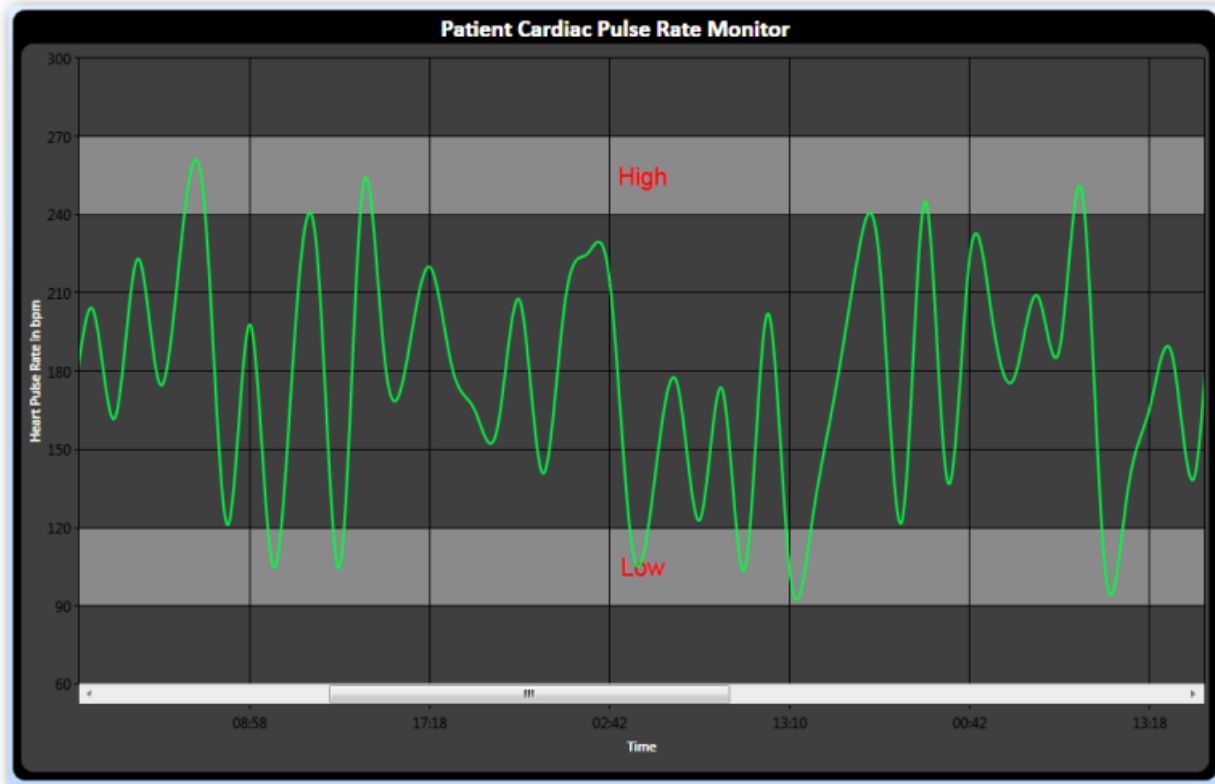
```
<syncfusion:Chart Margin="20" Grid.Row="1" Grid.Column="0" Name="Chart1"  
ChartPropertyWindowOpening="Chart1_ChartPropertyWindowOpening"  
ChartPropertyWindowClosing="Chart1_ChartPropertyWindowClosing" />
```

Adding Scroll Bar to a Chart

Automatic scrolling ensures that the specified data always remains visible in the chart window. If autoscrolling is enabled on an axis, then a scroll bar appears along the particular axis and the scroll bar displays a particular set of data for which the “`AutoScrollingDelta`” value is specified.

Use Case Scenario

While adding huge amount of data to the chart in real time, the autoscrolling functionality helps us to view a particular set of data in the chart at a given time. This makes the scroll bar to display the recently added data and the set of newly added data to be viewed clearly, according to the `AutoScrollingDelta` specified for the axis in the chart.



Property

Property	Description	Type of the property	Value it accepts	Reference links
EnableAutoScrolling	It enables auto scrolling	bool	True/False	NA
AutoScrollingDelta	Data is displayed based on the value specified	double	NA	NA

Adding Scroll Bar to a Chart

To enable AutoScrolling

XML

```
<syncfusion:ChartArea.PrimaryAxis>
  <!--Y axis declaration with required property settings-->
  <syncfusion:ChartAxis x:Name="XAxis" EnableAutoScrolling="True"
    AutoScrollingDelta="50" >
  </syncfusion:ChartAxis>
</syncfusion:ChartArea.PrimaryAxis>
```

CSHARP

```
this.XAxis.EnableAutoScrolling = true;
this.XAxis.AutoScrollingDelta = 50;
```

VB.NET

```
Me.XAxis.EnableAutoScrolling = True
Me.XAxis.AutoScrollingDelta = 50
```

Chart-Events in WPF Chart (Classic)

Chart Axis Events

ChartAxis events that could be used to track the Axis changes are as follows.

Axis.Changed

This event is triggered whenever any properties of the axis are changed.

C#

```
Area.PrimaryAxis.Changed += new EventHandler(PrimaryAxis_Changed);
// PrimaryAxis.Changed Event
void PrimaryAxis_Changed(object sender, EventArgs e)
{
    MessageBox.Show(Area.PrimaryAxis.ToString());
}
Axis.RangeChanged
```

Both Primary and Secondary Axis comes with Rangechanged event. This event occurs when the Range of the axis is changed. We could get the old and new range from this event.

C#

```
///

```

Chart Series Mouse Events

The following are the mouse events and their corresponding descriptions:

Event	Description
MouseClick	This event is handled when any mouse button is clicked, while mouse pointer is over the series.
MouseDown	This event is handled when any mouse button is pressed, while mouse pointer is over the series.
MouseEnter	This event is handled when mouse pointer enters the bounds of the series.
MouseLeave	This event is handled when mouse pointer leaves the bounds of the series.

MouseUp	This event is handled when any mouse button is released over the series.
MouseLeftButtonUp	This event is handled when left mouse button is released over the series.
MouseLeftButtonDown	This event is handled when left mouse button is pressed over the series.
MouseRightButtonUp	This event is handled when right mouse button is released over the series.
MouseRightButtonDown	This event is handled when right mouse button is pressed over the series.

These events can be initialized using the following lines of code.

XML

```
<sfchart:ChartSeries Data="0 3 1 4 2 5 3 9 6 4 7 3 8 5 9 11" Type="Pie"
  MouseClick="ChartSeries_MouseClick"
  MouseHover="ChartSeries_MouseHover"/>
```

C#

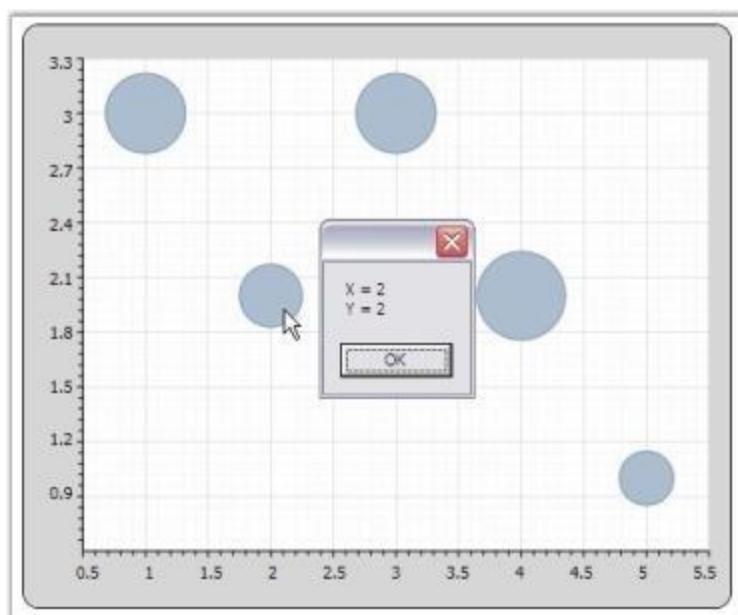
```
ChartSeries chartSeries = new ChartSeries();
this.MouseClick += new EventHandler(ChartSeries_MouseClick);
private void ChartSeries_MouseClick(object sender, EventArgs e)
{
    // Your code here
}
```

Chart MouseEventArgs

ChartMouseEventArgs are the arguments returned when the mouse events are triggered by ChartSeries. ChartMouseEventArgs returns the segment on which the mouse events are triggered along with default mouse event args. This event args can be used to perform customization of a segment when a mouse event is encountered. The segment returns different values that can be used to perform calculations or operations. The following lines of code demonstrates how ChartMouseEventArgs can be used to retrieve information about the ChartSeries segment.

C#

```
series.MouseClick += new ChartMouseEventHandler(series_MouseClick);
static 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());
}
```



3D-Charts in WPF Chart (Classic)

Enabling 3D Mode

3D mode can be easily enabled on a ChartArea using the View3DMode property as follows.

Annotations in WPF Chart (Classic)

Annotations in WPF Chart (Classic) at X-Y Coordinates

Annotations at specific X-Y coordinates can be added to the chart programmatically. The ChartSeriesAnnotation type used to define an annotation provides these properties:

ChartSeriesAnnotation Property	Description
Description	String description of the annotation.
X	Specifies the x coordinate in the plot.
Y	Specifies the y coordinate in the plot.
OffsetX	X offset for locating the annotation.
OffsetY	Y offset for locating the annotation.
Template	Apecifies the look and feel of a specific annotation instance.

The AnnotationsCollection type, which is the container for the above ChartSeriesAnnotation instances, provides some properties that are applied on all the annotations:

AnnotationsCollection Property	Description
AnnotationsTemplate	Specifies the look and feel for all the annotations in the ChartSeries.
LineColor	Specifies the color of the line drawn from the location to the annotation, in case OffsetX and OffsetY are specified.

IsRelative	Z and y values above are x-y coordinates of the plot. If <i>true</i> , they are in Chart Area coordinates. Default value is <i>false</i> .
------------	--

Here is a code example that adds a few annotations to a chart.

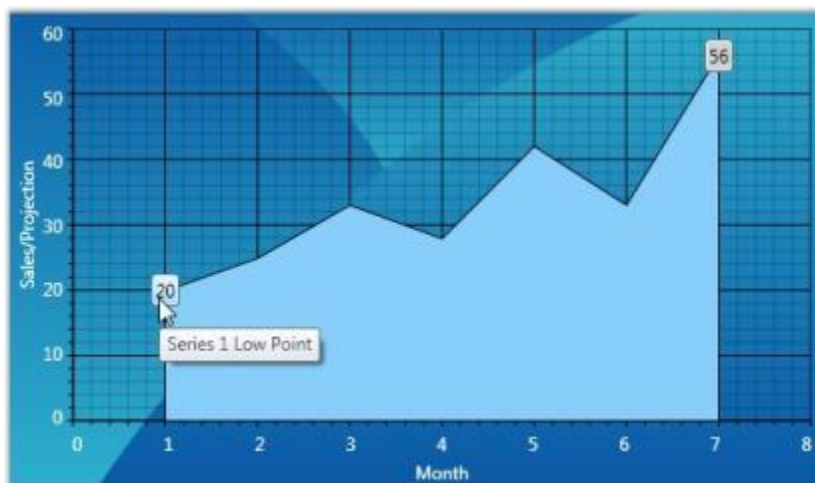
XML

```
<sfchart:ChartSeries Name="series1" Label="Series1" Type="Area"
Interior="LightSkyBlue">
<sfchart:ChartSeries.Annotations>
<sfchart:AnnotationsCollection LineColor="White" x:Uid="Annot">
<!--Define the look and feel of the annotation.-->
<sfchart:AnnotationsCollection.AnnotationsTemplate>
<DataTemplate>
<Button Content="{Binding Y}" ToolTip="{Binding Description}"
Background="LightGray" Name="Button1"
Click="Button_Click" />
</DataTemplate>
</sfchart:AnnotationsCollection.AnnotationsTemplate>
</sfchart:AnnotationsCollection>
<!--The annotations are added to this collection in code-behind.-->
</sfchart:ChartSeries.Annotations>
</sfchart:ChartSeries>
```

C#

```
// Series1 Annotations
ChartSeriesAnnotation ser1LowPoint = new ChartSeriesAnnotation() { X = 1, Y
= 20, Description = "Series 1 Low Point" };
ChartSeriesAnnotation ser1HighPoint = new ChartSeriesAnnotation() { X = 7, Y
= 56, Description = "Series 1 High Point" };
this.Chart1.Areas[0].Series[0].Annotations.Items.Add(ser1LowPoint);
this.Chart1.Areas[0].Series[0].Annotations.Items.Add(ser1HighPoint);
```

The resultant annotations look like this.



Annotations in WPF Chart (Classic) At Control Coordinates

Chart for WPF also lets you add some annotations to the chart at specific control coordinates. By default, these annotations appear as simple text labels. But, their look and feel can be fully customized using custom templates.

Here are some Chart control properties that let you add annotations at Chart control coordinates:

Property	Description
AnnotationLabels	Collection into which you add the ChartAnnotationLabel instances representing the annotations.
AnnotationLabelTemplate	Defines a custom look and feel for the annotation.

The ChartAnnotationLabel type used to define an annotation exposes these properties.

ChartAnnotationLabel Property	Description
Content	Content that needs to be displayed in the annotation (usually a string).
OffsetX	X offset from the top-left of the control used to determine the x-location of the annotation.
OffsetY	Y offset from the top-left of the control used to determine the y-location of the annotation.

Here is some code example that shows how to add annotations at Chart coordinates and how to customize their look and feel.

XML

```
<syncfusion:Chart Name="chart1">
  <!--Template defining the custom look and feel of the annotations.-->
  <syncfusion:Chart.AnnotationLabelTemplate>
    <DataTemplate>
      <Border Background="MintCream" BorderBrush="Black" BorderThickness="1">
        <TextBlock Text="{Binding}" Foreground="Black" FontFamily="Tahoma"
          FontSize="12" Margin="5"/>
      </Border>
    </DataTemplate>
  </syncfusion:Chart.AnnotationLabelTemplate>
  <syncfusion:Chart.AnnotationLabels>
    <syncfusion:ChartAnnotationLabelsCollection>
      <!--ChartAnnotationLabel instance representing the location and content of
      the annotation.-->
      <syncfusion:ChartAnnotationLabel x:Name="label1" Content="Top 6 Products"
        OffsetX="50" OffsetY="60">
      </syncfusion:ChartAnnotationLabel>
    </syncfusion:ChartAnnotationLabelsCollection>
  </syncfusion:Chart.AnnotationLabels>
</syncfusion:Chart>
```



Annotation Shapes

Predefined shapes for annotation objects are used to point at specific information about a position in the chart. For example: Circle, Arrow etc.

The following table describes more about the annotation shapes:

Name of the Property	Type of Property	Values It accepts
Content	Dependency Property	String
AnnotationShape	Dependency Property	Enum of type AnnotationShapes
Fill	Dependency Property	Colors from brushes
Offset X	Dependency Property	Double value
Offset Y	Dependency Property	Double Value

The Content property helps represent the required content in an annotation shape.

The AnnotationShape property helps create the required shape for an annotation object and the Fill property helps fill the selected shape with required color.

The following code example illustrates the creation of predefined annotation shape for a Chart.

XML

```
<syncfusion:Chart.AnnotationsLabel>
  <syncfusion:ChartAnnotationLabelsCollection>
    <syncfusion:ChartAnnotationLabel x:Name="Annotlabel" Content="Target
Reached" OffsetY="200" OffsetX="200" AnnotationShape="Circle" Fill="Red"/>
  </syncfusion:ChartAnnotationLabelsCollection>
</syncfusion:Chart.AnnotationsLabel>
<syncfusion:ChartArea >
```

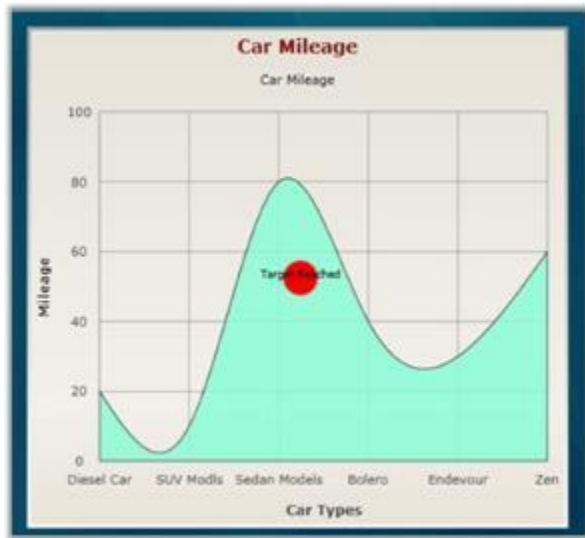
C#

```
Chart1.AnnotationsLabel[0].OffsetX = 200;
```



```
Chart1.AnnotationsLabel[0].OffsetY = 200;  
Chart1.AnnotationsLabel[0].Content = "TargetReached";  
Chart1.AnnotationsLabel[0].AnnotationShape = AnnotationShapes.Circle;  
Chart1.AnnotationsLabel[0].AnnotationShape = Red;
```

Run the code. The following output is displayed.



A sample which demonstrates the various predefined annotation shapes in Essential Chart, is available in the following install location:

C:\Documents and Settings\<user name>\My Documents\Syncfusion\Essential Studio\Samples\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Annotations

Localization Support in WPF Chart (Classic)

Localization is the process of making your application multi-lingual, by formatting content according to cultures. This involves configuring the application for a specific language. Culture is the combination of language and location (e.g. En-US is the culture for English spoken in United States; En-GB is the culture for English spoken in Great Britain). Syncfusion Chart allows you to set custom resource through the Resx file. You can simply give the string values in the resource file for a specific culture and set the culture in the application. The given string values are set to the Chart that does not affect the Code Block of the chart.

Use Case Scenario

The Essential Chart WPF can be localized according to the native language. It thus helps you to use the Chart more effectively.

Adding Localization to an Application

The following steps explain the implementation of Localization support in applications.

Creating an Application

Create a WPF application and add Chart control to it.

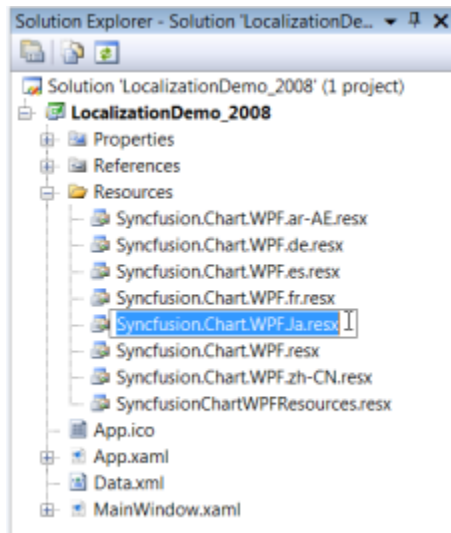
Creating a Resource File

To create a Resource file:

1. Create a folder named Resources in the application.
2. Create a resource file (Resx file) and name it Syncfusion.Chart.Wpf.resx E.g. Syncfusion.Chart.WPF.Ja.resx

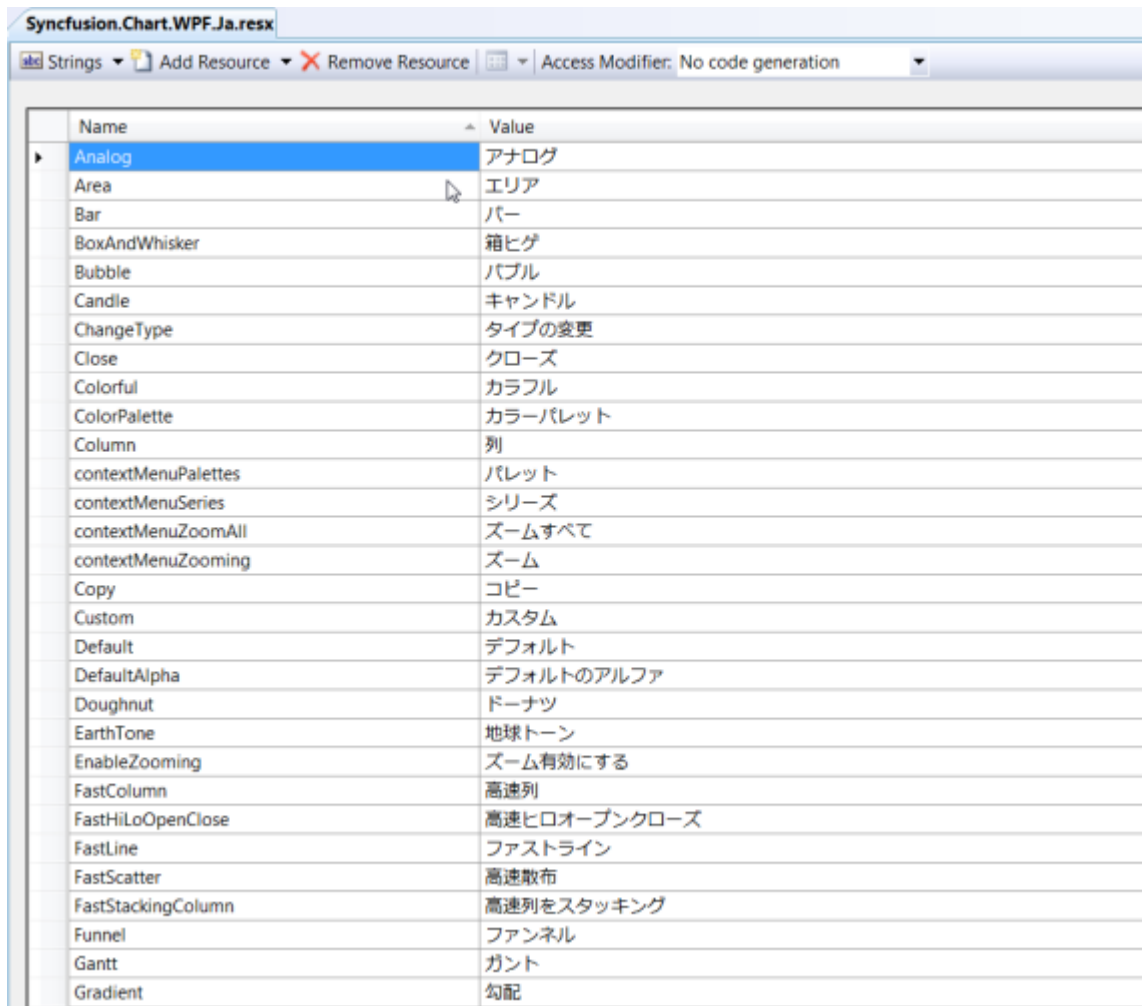
Use the prescribed naming convention as it is mandatory.

The following screenshot explains the addition of a Resource file to the application.



Enter the "Name" and "Value" in the Resource file.

The String Property names used in the Chart are given in the Properties table. This is explained in the following screenshot.



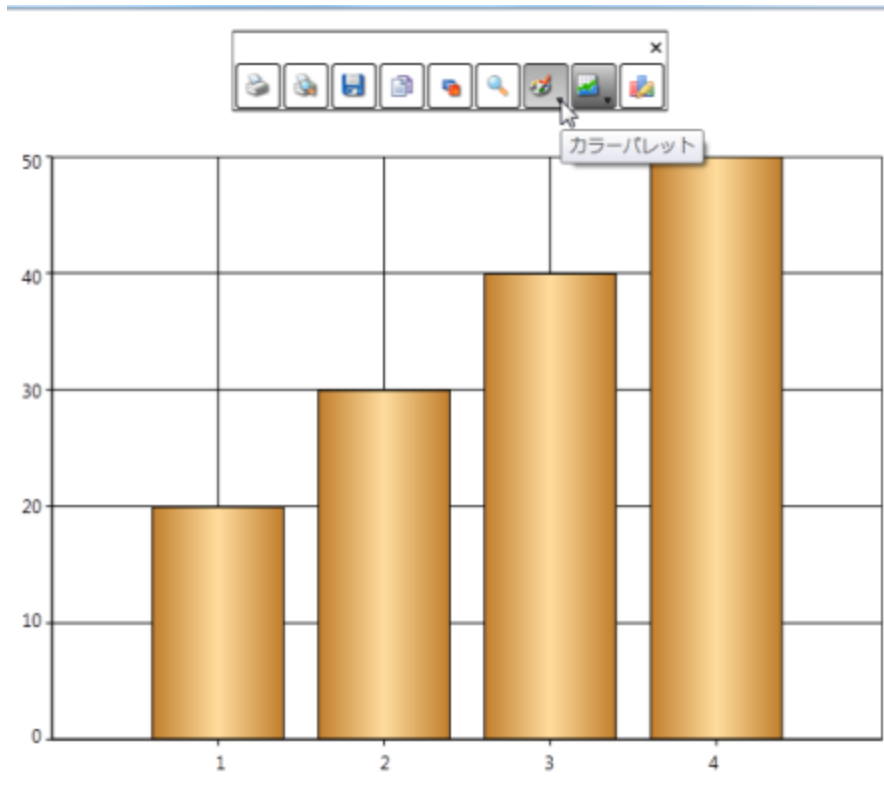
Name	Value
Analog	アナログ
Area	エリア
Bar	バー
BoxAndWhisker	箱ヒゲ
Bubble	バブル
Candle	キャンドル
ChangeType	タイプの変更
Close	クローズ
Colorful	カラフル
ColorPalette	カラーパレット
Column	列
contextMenuPalettes	パレット
contextMenuSeries	シリーズ
contextMenuZoomAll	ズームすべて
contextMenuZooming	ズーム
Copy	コピー
Custom	カスタム
Default	デフォルト
DefaultAlpha	デフォルトのアルファ
Doughnut	ドーナツ
EarthTone	地球トーン
EnableZooming	ズーム有効にする
FastColumn	高速列
FastHiLoOpenClose	高速ヒロオープンクローズ
FastLine	ファストライン
FastScatter	高速散布
FastStackingColumn	高速列をスタッキング
Funnel	ファンネル
Gantt	ガント
Gradient	勾配

Setting the Culture Information in the Application

The culture information should be set in the application before the `InitializeComponent()` method is called. Now, the application is set to Japanese Culture info. The following code example explains the implementation of this.

C#

```
System.Threading.Thread.CurrentThread.CurrentUICulture = new
System.Globalization.CultureInfo("ja");
```



Property	Description	Type	Data Type
Analog	Sets the string for the Analog property	static	string
Area	Sets the string for the Area property	static	string
Bar	Sets the string for the Bar property	static	string
BoxAndWhisker	Sets the string for the Box and Whisker property	static	string
Bubble	Sets the string for the Bubble property	static	string
Candle	Sets the string for the Candle property	static	string
ChangeType	Sets the string for the ChangeType property	static	string
Close	Sets the string for the Close property	static	string
Colorful	Sets the string for the Colorful property	static	string
ColorPalette	Sets the string for the ColorPalette property	static	string
Column	Sets the string for the Column property	static	string
contextMenuPalettes	Sets the string for the ContextMenuPalette property	static	string
contextMenuSeries	Sets the string for the ContextMenuSeries property	static	string
contextMenuZoomAll	Sets the string for the ContextMenuZoomAll property	static	string

contextMenuZooming	Sets the string for the ContextmenuZooming property	static	string
Copy	Sets the string for the Copy property	static	string
Custom	Sets the string for the Custom property	static	string
Default	Sets the string for the Default property	static	string
DefaultAlpha	Sets the string for the DefaultAlpha property	static	string
Doughnut	Sets the string for the Dhoughnut property	static	string
EarthTone	Sets the string for the EarthTone property	static	string
EnableZooming	Sets the string for the EnableZooming property	static	string
FastColumn	Sets the string for the FastColumn property	static	string
FastHiLoOpenClose	Sets the string for the FastHiLoOpenClose property	static	string
FastLine	Sets the string for the FastLine property	static	string
FastScatter	Sets the string for the FastScatter property	static	string
FastStackingColumn	Sets the string for the FastStackingColumn property	static	string
Funnel	Sets the string for the Funnel property	static	string
Gantt	Sets the string for the Gantt property	static	string
Gradient	Sets the string for the Gradient property	static	string
Grayscale	Sets the string for the GrayScale property	static	string
HiLo	Sets the string for the HiLo property	static	string
HiLoArea	Sets the string for the HiLoArea property	static	string
HiLoOpenClose	Sets the string for the HiLoOpenClose property	static	string
Histogram	Sets the string for the Histogram property	static	string
Kagi	Sets the string for Kagi property	static	string
LegendWindowCancel	Sets the string for the LegendWindowControl property	static	string
LegendWindowCheckBox	Sets the string for the LegendWindowCheckBox property	static	string
LegendWindowIcon	Sets the string for the LegendWindowIcon property	static	string
LegendWindowOK	Sets the string for the legendWindowOk property	static	string
LegendWindowTitle	Sets the string for the LegendWindowTiltle property	static	string
Line	Sets the string for the Line property	static	string
Nature	Sets the string for the Nature property	static	string

Office2007Black	Sets the string for the Office2007Black property	static	string
Office2007Blue	Sets the string for the Office2007Blue property	static	string
Office2007Silver	Sets the string for the Office2007Silver property	static	string
Palette1	Sets the string for the Palette1 property	static	string
Palette2	Sets the string for the Palette2 property	static	string
Palette3	Sets the string for the Palette3 property	static	string
Palette4	Sets the string for the Palette4 property	static	string
Palette5	Sets the string for the Palette5 property	static	string
Palette6	Sets the string for the Palette6 property	static	string
Palette7	Sets the string for the Palette7 property	static	string
Palette8	Sets the string for the Palette8 property	static	string
Panning	Sets the string for the Panning property	static	string
Pastel	Sets the string for the Pastel property	static	string
Pie	Sets the string for the Pie property	static	string
PointAndFigure	Sets the string for the PointAndFigure property	static	string
Polar	Sets the string for the Polar property	static	string
printDialogAdvanced	Sets the string for the PrintDialogAdvanced property	static	string

[Sample Link](#)

To run the UI WPF Chart samples:

1. Open Essential Studio Dashboard by selecting Start -> Program -> Syncfusion-> Essential Studio <-> Dashboard.
2. Click Run locally installed samples from the WPF drop-down list on the User Interface pane.
3. Select Chart on the sample browser.
4. Select User Interaction -> Localization Demo on the Essential Chart pane and click the Run Sample button.

To open the sample projects:

1. Go to the following sample location in your system:
2. “\Syncfusion\EssentialStudio\Version Number \WPF\Chart.WPF\Samples\3.5\WindowsSamples\User Interaction\Localization Demo”
3. This location contains two sub folders CS and VB. You can open the sample projects from the respective folders based on your application developing language.

Export Chart To PDF in WPF Chart (Classic)

Essential Chart for WPF now comes with the support to export the chart to a PDF file; this conversion can be done using the Syncfusion.ChartConversion.WPF assembly.

Methods

Method	Description	Parameters	Type	Return Type
ChartPdfConverter	Converts chart to PDF file.	Chart, Filename	ChartString	Void

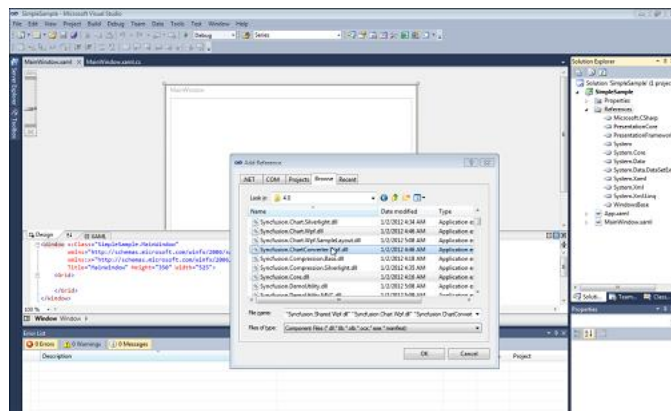
Sample Link

1. Open the WPF sample browser
2. Select the Chart product
3. Select Chart > Export and Import > Chart to PDF

Adding Support to Convert a Chart to PDF to an Application

The following steps explain how to convert a chart to PDF.

1. Create a new Visual Studio 2010 or 2008 project.
2. Add the following assemblies to the project:
3. Syncfusion.Chart.WPF.dll
4. Syncfusion.ChartConverter.WPF.dll

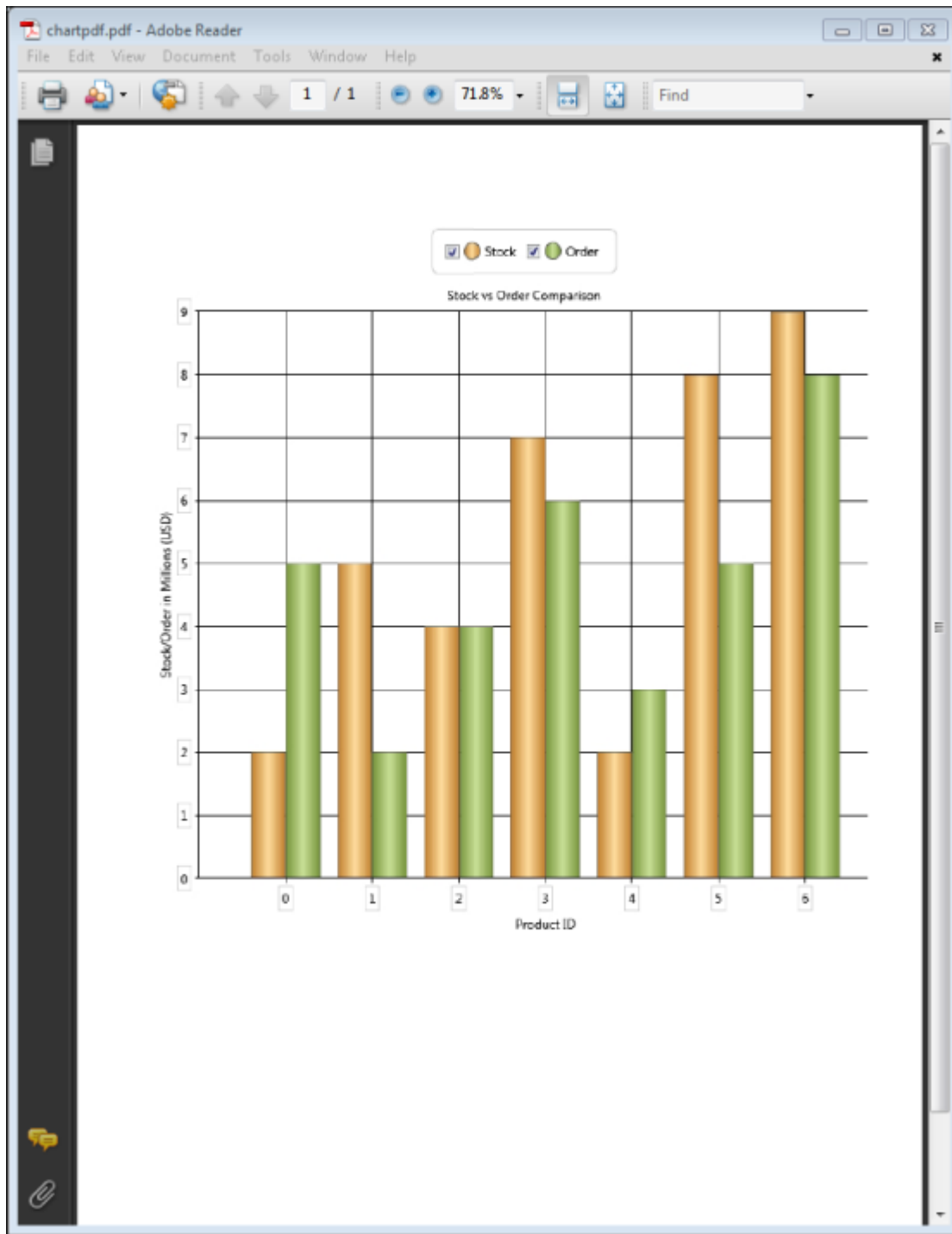


3. Create a chart to be exported to PDF. Use the following code to convert the chart to a PDF file.

C#

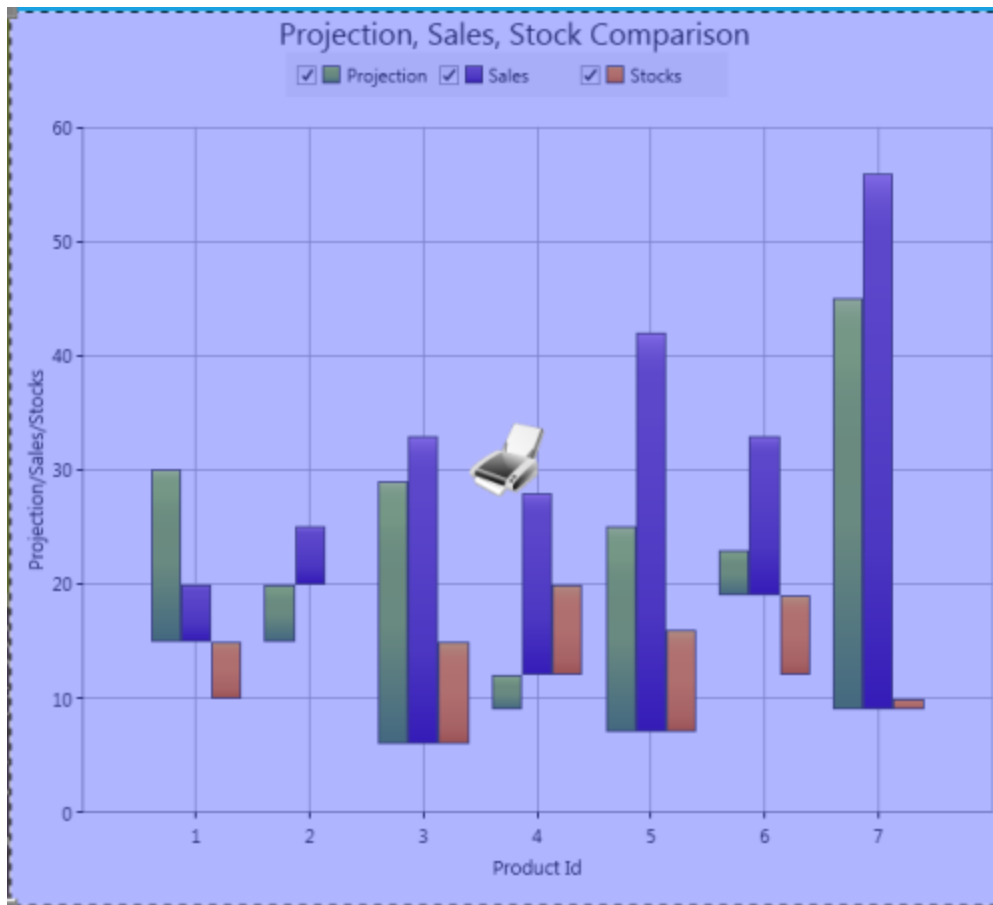
```
ChartPdfConverterControl control = new ChartPdfConverterControl();
control.ChartPdfConverter (Chart1, "chartpdf.pdf");
```

4. The PDF file is generated as follows.



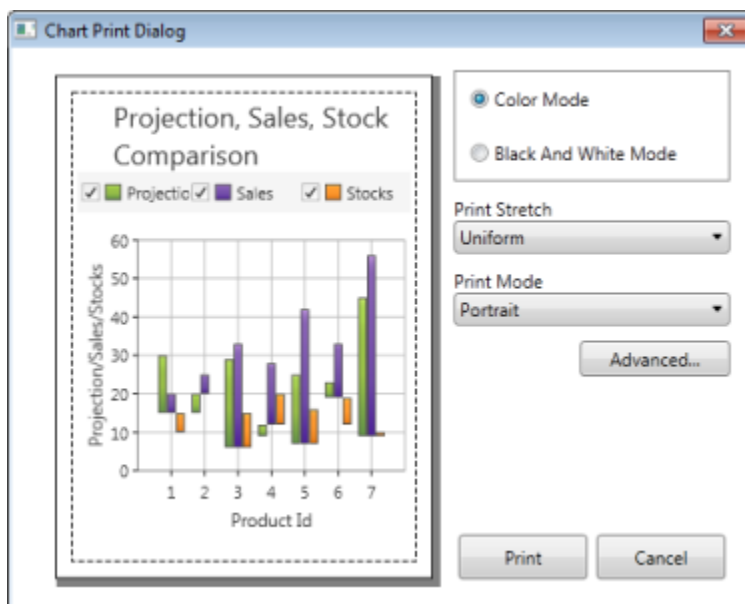
Print in WPF Chart (Classic)

Essential Chart for WPF now comes with support to print the chart and printing options such as page orientation, print preview, color mode, and more.



Use Case Scenarios

Printing the chart is useful for visual representation in organizational meetings.



Methods

Method	Description	Parameters	Type	Return Type
--------	-------------	------------	------	-------------

Print()	Used to print the Chart control. This opens the printing dialog box. This method returns a bool value after printing.	Overloads: (Rect printArea)	N/A	Bool
PrintSwitchMode()	To select a particular area for printing.	N/A	N/A	Void

Sample Link

..My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Chart.WPF\Samples\3.5\WindowsSamples\Export and Print\Printing Chart Demo

Print in WPF Chart (Classic)ing a Chart

Charts can be printed by using the following code example.

XML

```
<!--Button when clicked executes the Print command-->
<Button Grid.Row="0" Content="Print"
        Command="{x:Static ApplicationCommands.Print}"
        CommandTarget="{Binding ElementName=Chart1}"
x:Name="button"/>
<!--Button when clicked executes the
SwitchPrinting command-->
<Button Grid.Row="0" Grid.Column="1"
Content="Printing Mode"
        Command="{x:Static syncfusion:ChartCommands.SwitchPrinting}"
        CommandTarget="{Binding
ElementName=Chart1}" x:Name="button1" />
```

C#

```
Chart1.Print();
Chart1.PrintSwitchMode();
```

Statistical Formula and Utility Functions in WPF Chart (Classic)

This feature allows the user to calculate basic statistical functions that include mean, median, standard deviation, variance, variance based estimator, variance unbiased estimator, correlation coefficient and covariance, ANOVA, T-test, Z-test, and F-test performed based on sample series, and utility functions like normal distribution, T-cumulative distribution and F-cumulative distribution.

Use Case Scenarios

1. This feature supports built-in statistical formulas.
2. The curve in the graph can be drawn based on normal distribution, T-cumulative distribution, and F-cumulative distribution.
3. Basic functions like mean, median, and standard deviation can be obtained from the series drawn.

Methods

Method	Description	Parameters	Return Type
--------	-------------	------------	-------------

Mean	Returns the mean value of the x values of the series.	Series	Double
Mean	Returns the mean of y values.	Series, Yindex(int)	Double
VarianceUnbiasedEstimator	Estimates the variance for the x values of the series.	Series	Double
VarianceUnbiasedEstimator	Estimates the variance for the y values of the series.	Series,Yindex(int)	Double
VarianceBiasedEstimator	Estimates the variance of the sample.	Series	Double
VarianceBiasedEstimator	Estimates the variance for the y value of the series.	Series, Yindex(int)	Double
Variance	The following code samples demonstrate how to get the variance of the data points in a series.	Series,sampleVariance(bool)	Double
Variance	The following code samples demonstrate how to get the variance of the y-value data points in a series.	Series,Yindex(int),sampleVariance(bool)	Double
StandardDeviation	This method determines the standard Deviation for x values of the series.	Series,sampleVariance(bool)	Double
StandardDeviation	This method determines the standard deviation for y values of the series.	SeriesYindex(double)sampleVariance(bool)	Double
Covariance	Returns the average of the	Series1,Series 2	Double

	product of deviations of the data points from their respective means.		
Covariance	Returns the average of the product of deviations of the data points from their respective means based on y values.	Series1, Series 2, Yindex(int)	Double
Correlation	Measures the relationship between two data sets that are scaled to be independent of the unit of measurement. This correlation method returns the covariance of two data sets divided by the product of their standard deviations, and always ranges from -1 to 1.	Series1, Series2	Double
Correlation	Measures the relationship between two data sets that are scaled to be independent of the unit of measurement. This correlation method returns the covariance of two data sets divided by the product of their standard deviations, and always ranges from -1 to 1.	Series1, Series2, yIndex(double)	Double

Median	Calculates the median of the points stored in a series.	Series	Double
Median	Calculates the median of the points stored in a series.	Series,yIndex(int)	Double
ZTest	This method performs a Z-test for two groups of data and returns the results using a ZTestResult object.	(double) hypothesizedMeanDifference, (double) varianceFirstGroup, (double) varianceSecondGroup, (double) probability, (ChartSeries) firstInputSeries, (ChartSeries) secondInputSeries	ZTestResult
ZTest	This method performs a Z-test for two groups of data and returns the results using a ZTestResult object, for Y values.	(double)hypothesizedMeanDifference, (double) varianceFirstGroup, (double) varianceSecondGroup, (double) probability, (ChartSeries)firstInputSeries, (ChartSeries)secondInputSeries,(Int) yIndex	ZTestResult
TTestEqualVariances	This method performs a T-test for two groups of data and assumes equal variances between the two groups (i.e. series) for the x values.	(Double) hypothesizedMeanDifference, (double) probability, (ChartSeries)firstInputSeries, (ChartSeries) secondInputSeries	TTestResult
TTestEqualVariances	This method performs a T-test for two groups of data and assumes equal variances between the two groups (i.e. series) for the y values.	(Double) hypothesizedMeanDifference, (double) probability, (ChartSeries)firstInputSeries, (ChartSeries) secondInputSeries,(int)yIndex	TTestResult
TTestUnEqualVariances	This method performs a T-test for two groups of data and assumes unequal variances	(Double) hypothesizedMeanDifference, (double) probability, (ChartSeries) firstInputSeries, (ChartSeries) secondInputSeries	TTestResult

	between the two groups (i.e. series).		
FTest	This method returns the results of the F-test using an FTestResult object.	(Double) probability, (ChartSeries) firstInputSeries, (ChartSeries) secondInputSeries	FTestResult
FTest	This method returns the results of the F-test for the y values using an FTestResult object.	(Double) probability, (ChartSeries) firstInputSeries, (ChartSeries) secondInputSeries,(Int) yIndex	FTestResult
Anova	An ANOVA test is used to test the difference between the means of two or more groups of data.	double probability, ChartSeries[] inputSeries	AnovaResult
Anova	An ANOVA test is used to test the difference between the means of two or more groups of data for y values.	double probability, ChartSeries[] inputSeries,int Yindex	AnovaResult
GammaLn	Natural logarithm of gamma function (for $y > 0$).	Double	Double
Factorial	Factorial $n!$ (for $n \geq 0$).	Double	Double
FactorialLn	Logarithm of factorial $n!$ (for $n \geq 0$).	Int	Double
BetaLn	Logarithm of beta function.	Double a, double b	Double
Beta	Beta function.	double a, double b	Double

Sample Link

1. Open the Sample Browser samples
2. Select the Chart control
3. Statistical Analysis > Statistical Formula

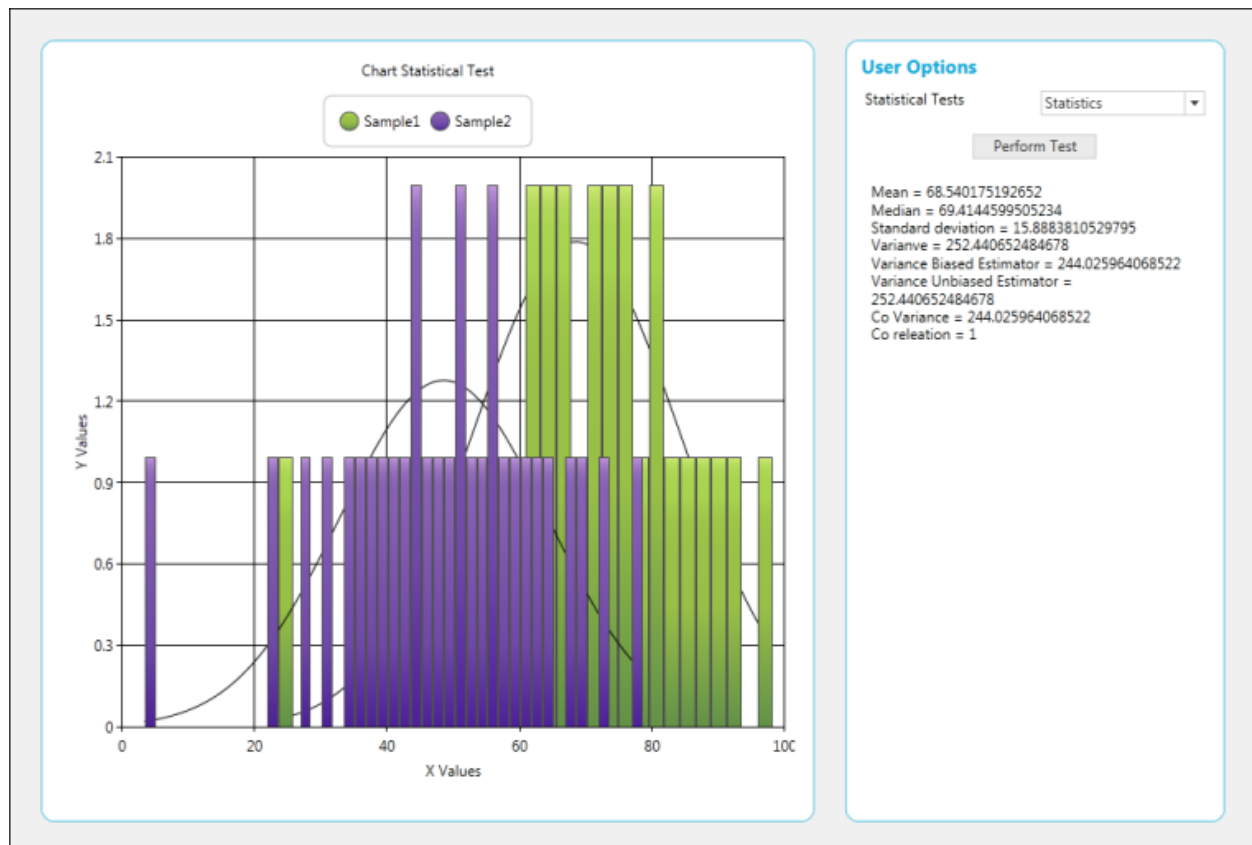
4. Statistical Analysis > Utility Functions

Adding Statistical Formula and Utility Functions to an Application

Statistical Formulas

C#

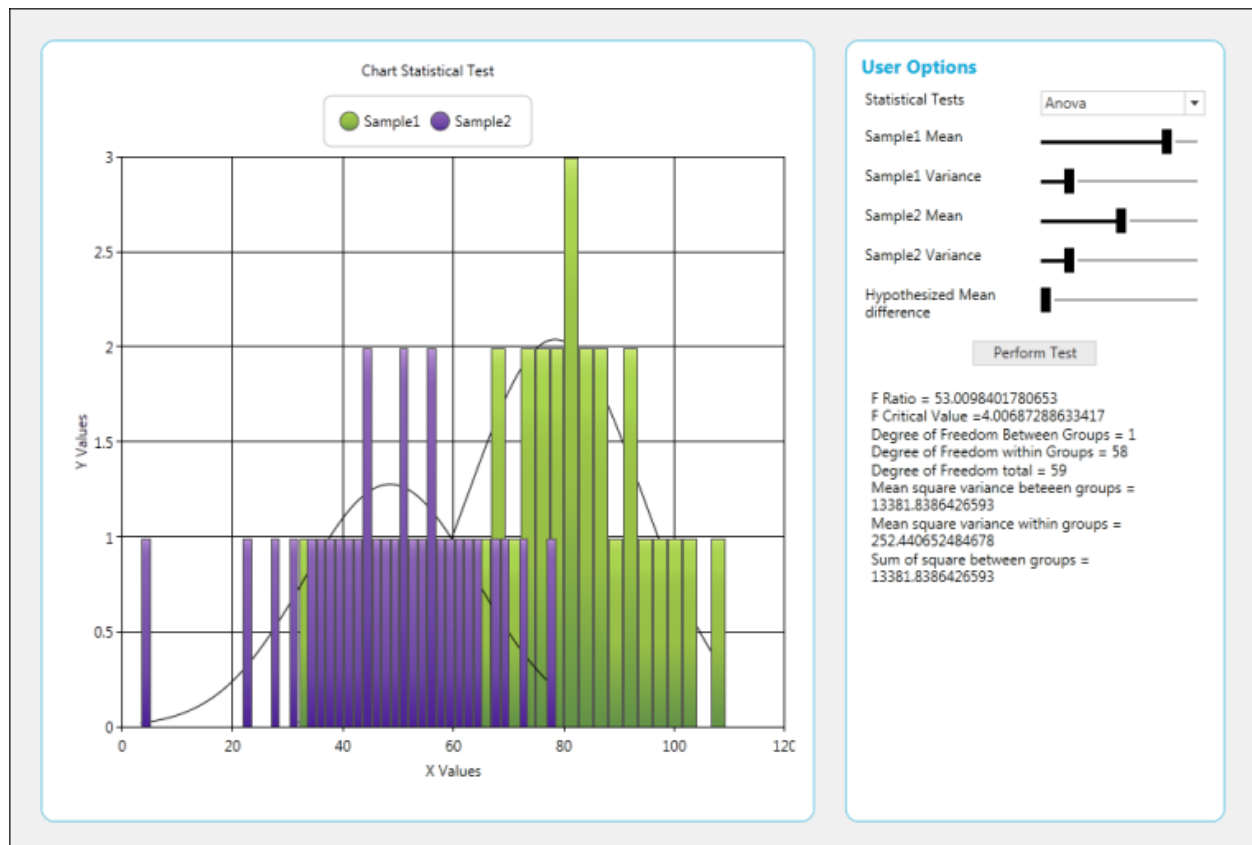
```
string val = "";
val += "Mean = " + BasicStatisticalFormulas.Mean(series).ToString() +
"\r\n";
val += "Median = " + BasicStatisticalFormulas.Median(series).ToString() +
"\r\n";
val += "Standard Deviation = " +
BasicStatisticalFormulas.StandardDeviation(series, true).ToString() +
"\r\n";
val += "Variance = " + BasicStatisticalFormulas.Variance(series,
true).ToString() + "\r\n";
val += "Variance Based Estimator = " +
BasicStatisticalFormulas.VarianceBiasedEstimator(series).ToString() +
"\r\n";
val += "Variance UnBased Estimator = " +
BasicStatisticalFormulas.VarianceUnbiasedEstimator(series).ToString() +
"\r\n";
val += "Correlation Co-efficient = " +
BasicStatisticalFormulas.Correlation(this.chartControl1.Series[0],
this.chartControl1.Series[1]).ToString() + "\r\n";
val += "Covariance = " +
BasicStatisticalFormulas.Covariance(this.chartControl1.Series[0],
this.chartControl1.Series[1]).ToString() + "\r\n";
this.richTextBox1.Text = val;
```



Perform ANOVA Test

C#

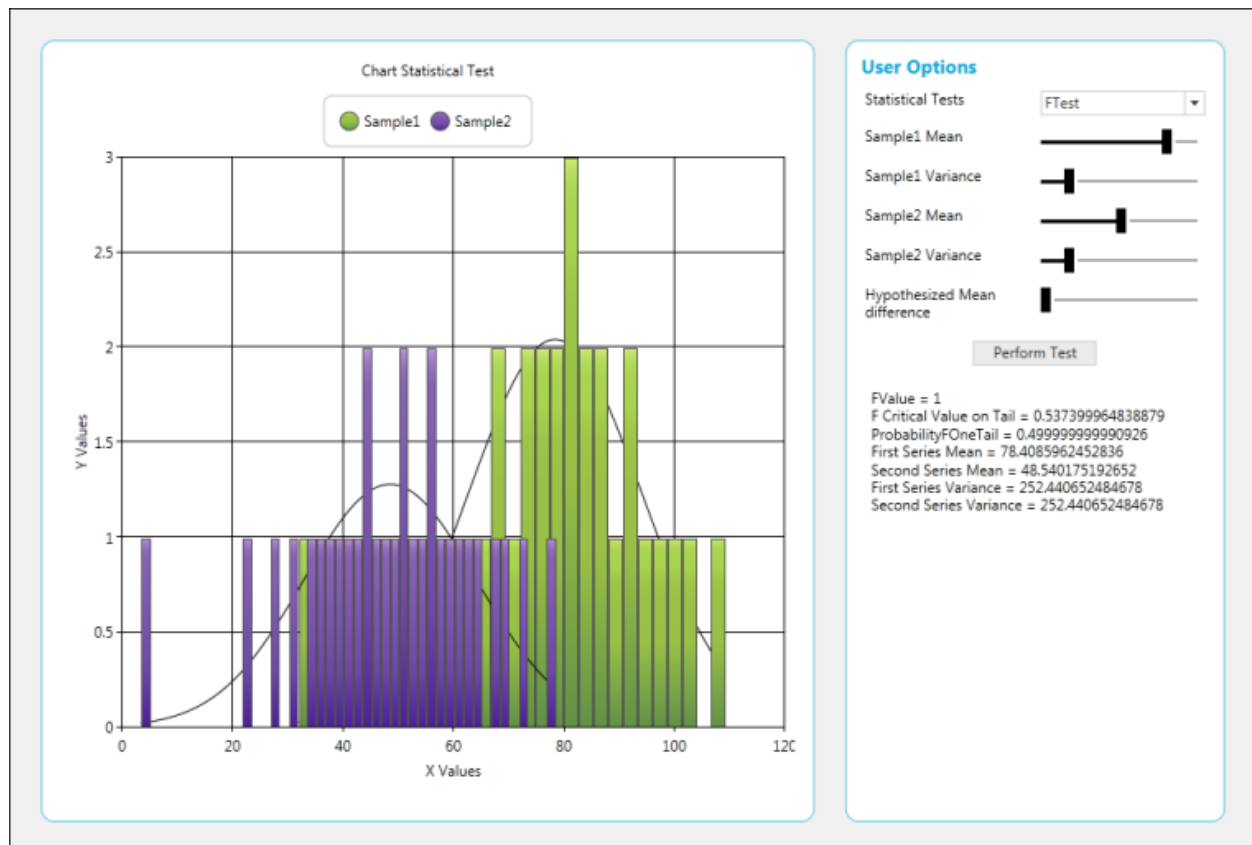
```
AnovaResult anova = BasicStatisticalFormulas.Anova(0.05, new ChartSeries[] {
    series1, series2 });
result.Text = "F Ratio = " + anova.FRatio + "\n" +
"F Critical Value =" + anova.FCriticalValue + "\n" +
"Degree of Freedom Between Groups = " + anova.DegreeOfFreedomBetweenGroups +
"\n" +
"Degree of Freedom within Groups = " + anova.DegreeOfFreedomWithinGroups +
"\n" +
"Degree of Freedom total = " + anova.DegreeOfFreedomTotal + "\n" +
"Mean square variance beteeen groups = " +
anova.MeanSquareVarianceBetweenGroups + "\n" +
"Mean square variance within groups = " +
anova.MeanSquareVarianceWithinGroups + "\n" +
"Sum of square between groups = " + anova.SumOfSquaresBetweenGroups + "\n";
```

Perfrom F-Test

C#

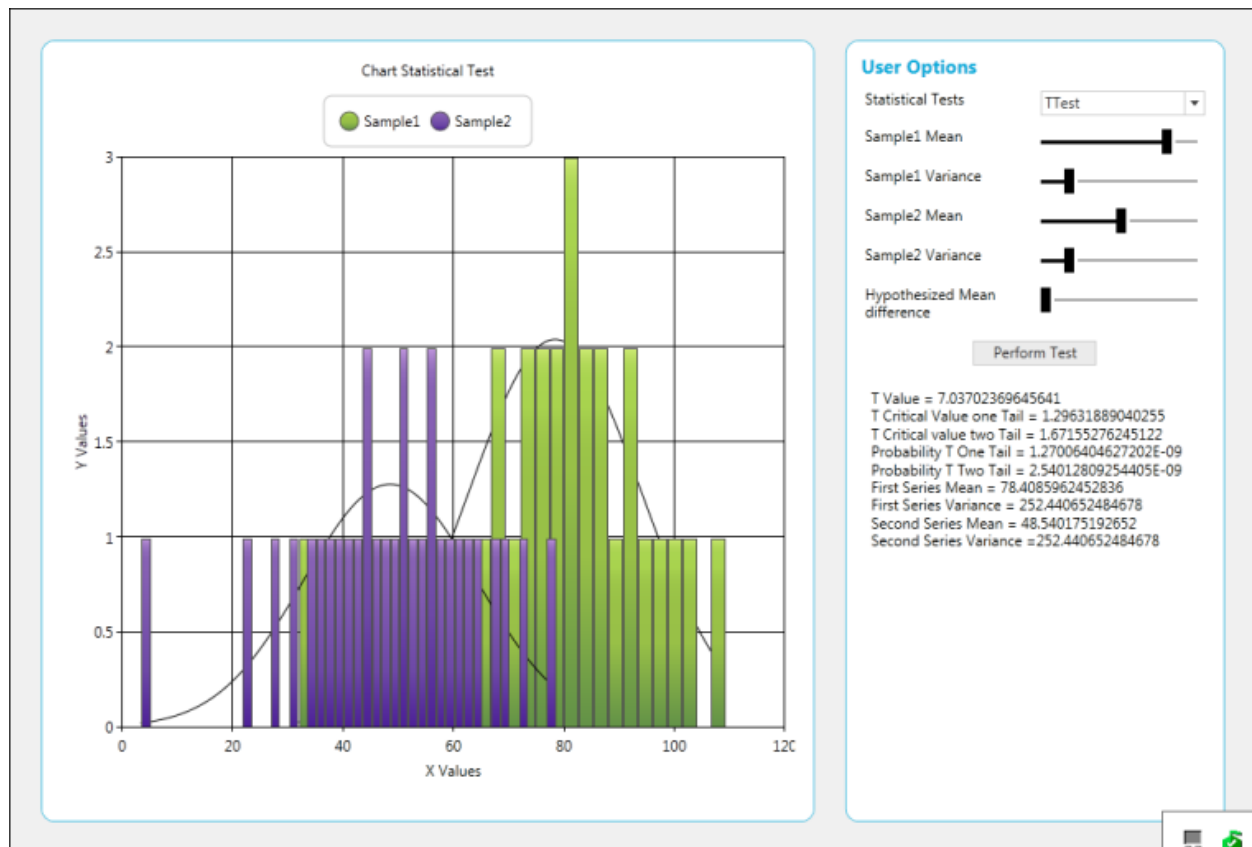
```
FTestResult ftest = BasicStatisticalFormulas.FTest(0.05, series1, series2);
result.Text = "FValue = " + ftest.FValue.ToString() + "\n" +
"F Critical Value on Tail = " + ftest.FCriticalValueOneTail.ToString() +
"\n" +
"ProbabilityFOneTail = " + ftest.ProbabilityFOneTail.ToString() + "\n" +
"First Series Mean = " + ftest.FirstSeriesMean.ToString() + "\n" +
"Second Series Mean = " + ftest.SecondSeriesMean.ToString() + "\n" +
"First Series Variance = " + ftest.FirstSeriesVariance.ToString() + "\n" +
"Second Series Variance = " + ftest.SecondSeriesVariance.ToString() + "\n";
```



Perform T-Test

C#

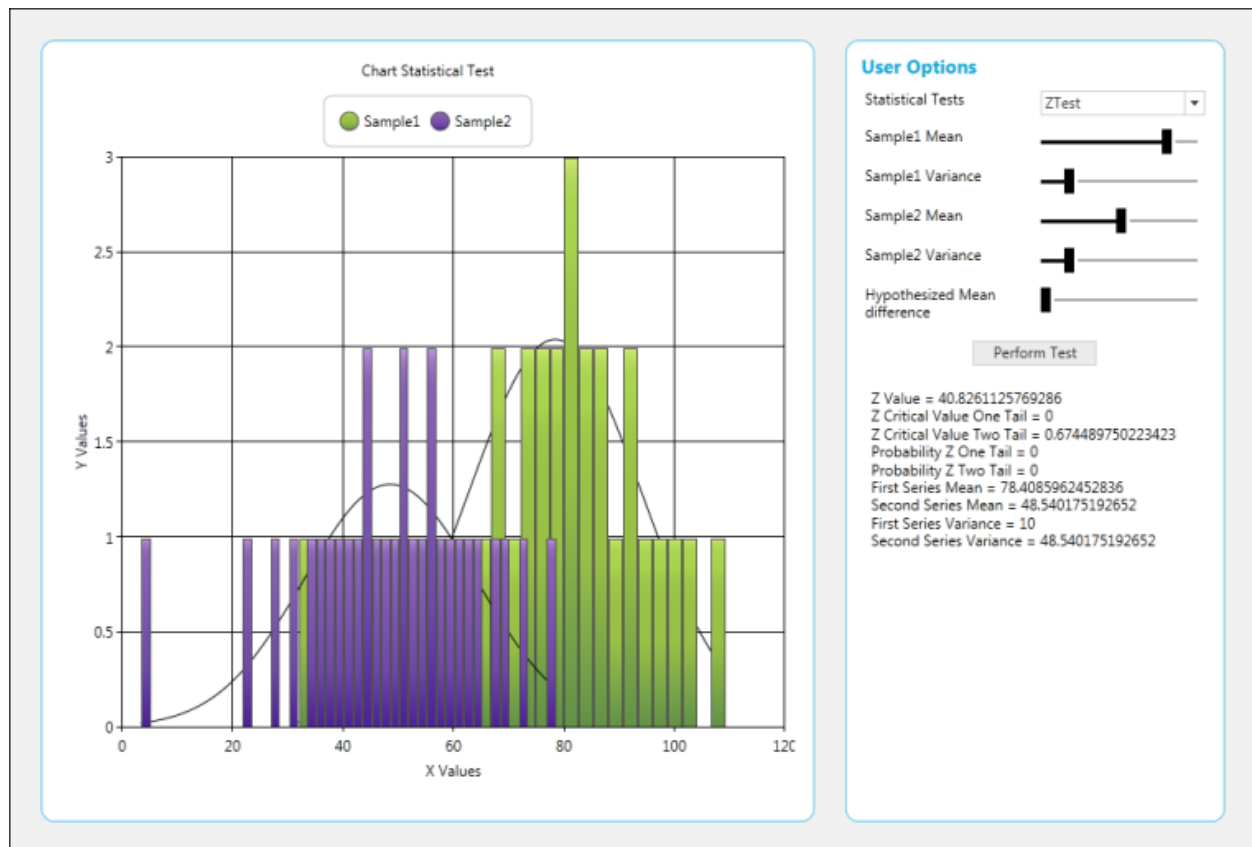
```
TTestResult ttest =
BasicStatisticalFormulas.TTestEqualVariances(meandiff.Value, 0.1, series1,
series2);
result.Text = "T Value = " + ttest.TValue.ToString() + "\n" +
"T Critical Value one Tail = " + ttest.TCriticalValueOneTail.ToString() +
"\n" +
"T Critical value two Tail = " + ttest.TCriticalValueTwoTail.ToString() +
"\n" +
"Probability T One Tail = " + ttest.ProbabilityTOneTail.ToString() + "\n" +
"Probability T Two Tail = " + ttest.ProbabilityTTwoTail.ToString() + "\n" +
"First Series Mean = " + ttest.FirstSeriesMean.ToString() + "\n" +
"First Series Variance = " + ttest.FirstSeriesVariance.ToString() + "\n" +
"Second Series Mean = " + ttest.SecondSeriesMean.ToString() + "\n" +
"Second Series Variance = " + ttest.SecondSeriesVariance.ToString() + "\n";
```



Perform Z-Test

C#

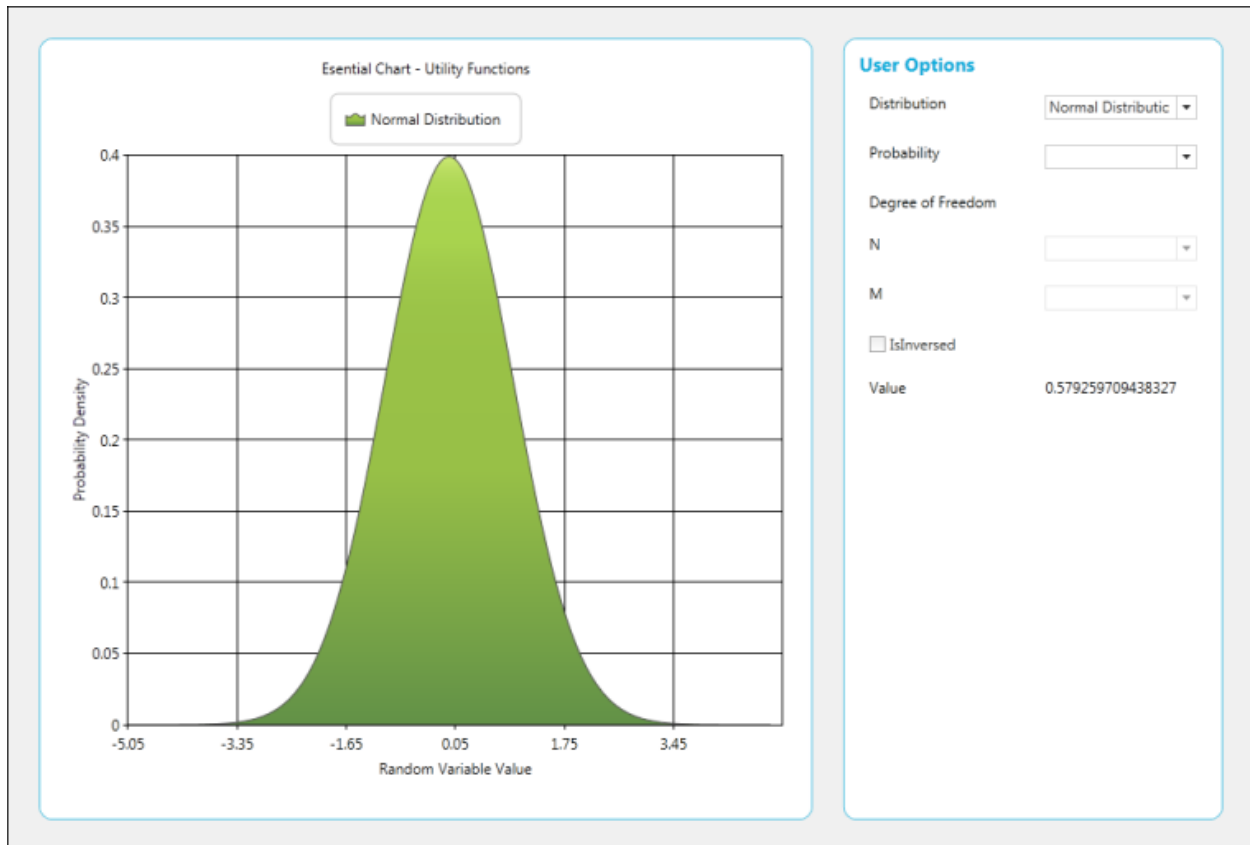
```
ZTestResult ztest = BasicStatisticalFormulas.ZTest(meandiff.Value, 10, 5,
0.5, series1, series2);
result.Text = "Z Value = " + ztest.ZValue.ToString() + "\n" +
"Z Critical Value One Tail = " + ztest.ZCriticalValueOneTail.ToString() +
"\n" +
"Z Critical Value Two Tail = " + ztest.ZCriticalValueTwoTail.ToString() +
"\n" +
"Probability Z One Tail = " + ztest.ProbabilityZOneTail.ToString() + "\n" +
"Probability Z Two Tail = " + ztest.ProbabilityZTwoTail.ToString() + "\n" +
"First Series Mean = " + ztest.FirstSeriesMean.ToString() + "\n" +
"Second Series Mean = " + ztest.SecondSeriesMean.ToString() + "\n" +
"First Series Variance = " + ztest.FirstSeriesVariance.ToString() + "\n" +
"Second Series Variance = " + ztest.SecondSeriesMean.ToString() + "\n";
```



Normal Distribution

C#

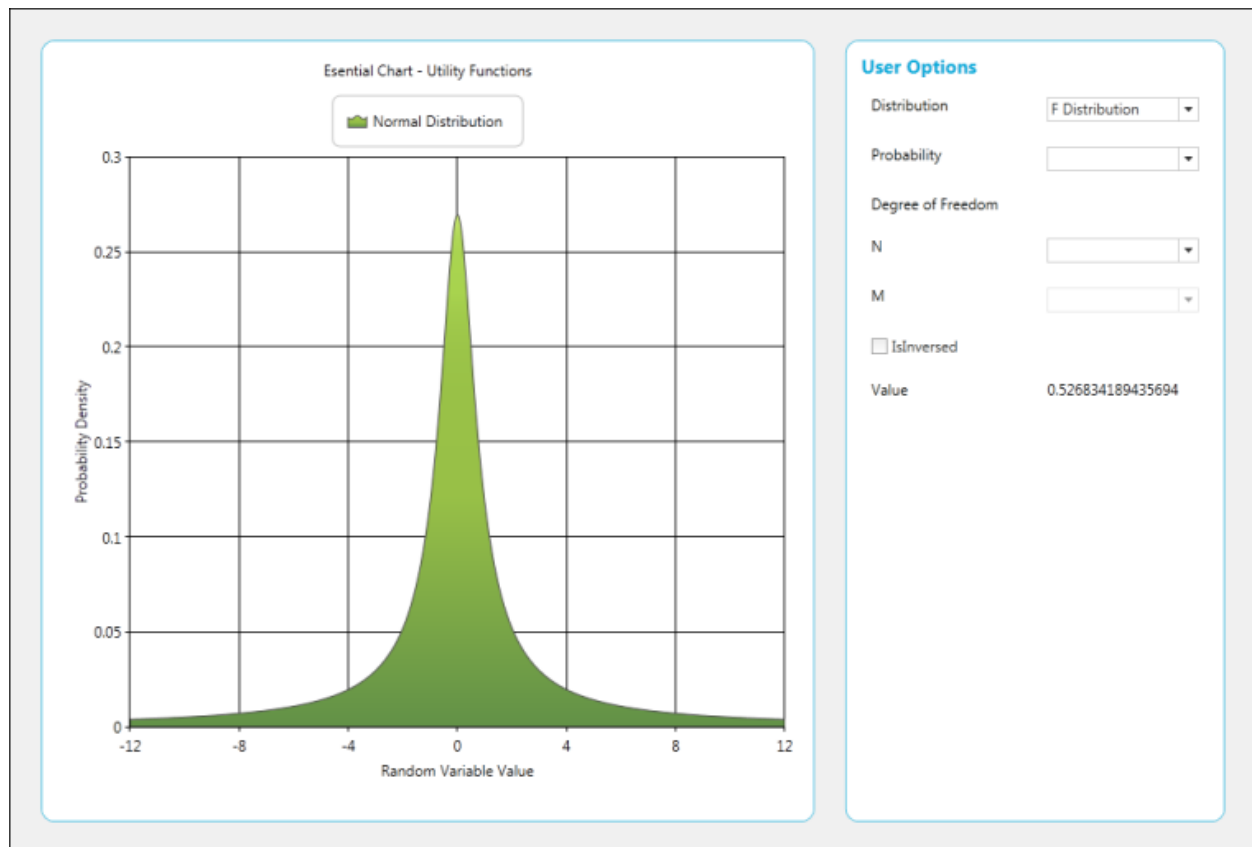
```
value.Text =
UtilityFunctions.NormalDistribution((double)probability.SelectedItem).ToString();
```



F-Cumulative Distribution

C#

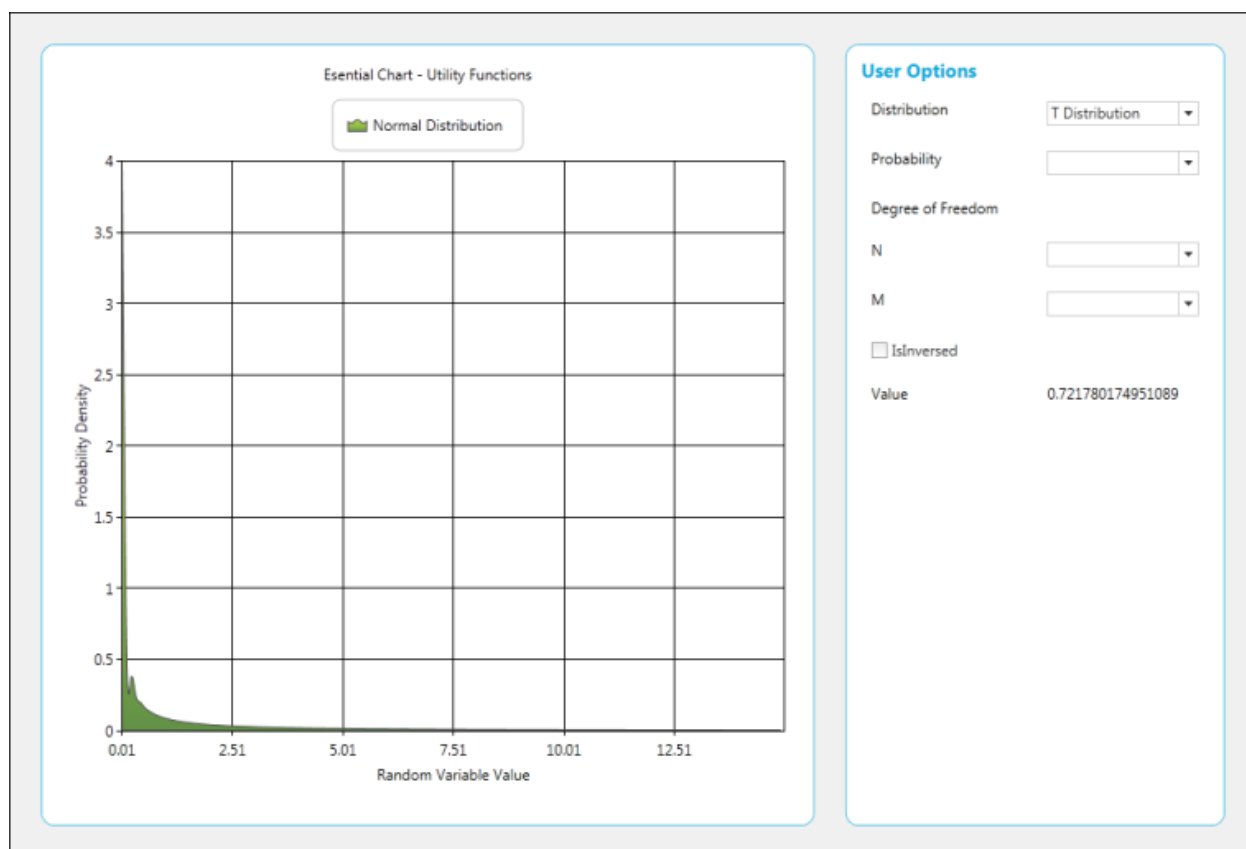
```
value.Text =  
UtilityFunctions.FCumulativeDistribution((double)probability.SelectedItem,  
(double)n.SelectedItem, (double)m.SelectedItem).ToString();
```



T-Cumulative Distribution

C#

```
value.Text =  
UtilityFunctions.TCumulativeDistribution((double)probability.SelectedItem,  
(double)n.SelectedItem, true).ToString();
```



Sparkline in WPF Chart (Classic)

A Sparklinecontrol is a type of information graphic characterized by its small size, high data density and lightweight. It presents trends and variations in a very condensed fashion. The Sparkline does not contain an axis scale and is intended to give a high level overview of what happened to the data over time.

Use Case Scenarios

A sparkline can display a trend based on adjacent data in a clear and compact graphical representation. The purpose of sparkline is to quickly see the data range difference with high density data and it is represented in lightweight graphical representation. You can use it as per your requirement.

The following screenshot shows three types of sparklines, which are drawn inside the grid control cell, based on row values.



Properties

Property	Description	Type	Data Type	Reference links
SparkLineType	Get or set the type of spark lines.By default, it is set to Line type.	Dependency property	SparkLineTypes â€“Enum {Line, Column, WinLoss}	NA
ItemSource	Gets or sets the data source for sparkline data points	Dependency Property	IEnumerable	NA
DisplayMemberPath	Gets or sets the property name that has to be taken as data for displaying points	Dependency Property	String	NA
FirstPointHighlightBrush	Gets or sets the brush used to highlight first data point in spark line	Dependency Property	Brush	NA
IsFirstPointHighlighted	Helps to enable or disable highlighting of first data point in spark line	Dependency Property	Bool	NA
LastPointHighlightBrush	Gets or sets the brush used to highlight last data point in spark line	Dependency Property	Brush	NA
IsLastPointHighlighted	Helps to enable or disable highlighting of last data point in spark line	Dependency Property	Bool	NA
HighPointHighlightBrush	Gets or sets the brush used to highlight highest data point in spark line	Dependency Property	Brush	NA
IsHighPointHighlighted	Helps to enable or disable highlighting of highest data point in spark line	Dependency Property	Bool	NA

LowPointHighlightBrush	Gets or sets the brush used to highlight lowest data point in spark line	Dependency Property	Brush	NA
IsLowPointHighlighted	Helps to enable or disable highlighting of lowest data point in spark line	Dependency Property	Bool	NA
NegativePointHighlightBrush	Gets or sets the brush used to highlight negative data points in spark line	Dependency Property	Brush	NA
IsNegativePointHighlighted	Helps to enable or disable highlighting of negative data point in spark line	Dependency Property	Bool	NA
MarkerColor	Gets or sets the brush of the markers in spark line. This property has effect over Line type spark line only	Dependency Property	Brush	NA
IsMarkerEnabled	Helps to enable or disable markers in line type spark line	Dependency Property	Bool	NA

Types of Sparklines

Presently, Syncfusion SparkLine control supports three types of Sparklines and the sparkline control must be bound to a data source. It supports a variety of datasource such as DataTable and any component that implements the interface IEnumerable, ICollection, IList.

- Line
- Column
- WinLoss

Drawing Sparkline in an Application

Drawing Line Sparkline in an Application

The line type of spark line represents a set of data points, connected by a line.

Refer to the following code examples to draw the line sparkline.

C#

```
//Set Sparkline points to source property
this.sparkLine1.ItemsSource = new double[] { 30, -20, 80, 20, 40, -50, -30,
70, -40, 50 };
//Set line type sparkline
this.sparkLine1.SparkLineType = SparkLine.SparkLineType.Line;
```

VB.NET

```
'Set Sparkline points to source property
Me.sparkLine1.ItemsSource = New Double() {30, -20, 80, 20, 40, -50, -30, 70, -
40, 50}
'Set line type sparkline
Me.sparkLine1.SparkLineType = SparkLine.SparkLineType.Line
```

*Drawing Column Sparkline in an Application*

The column type of spark line represents each data point by a column. The vertical column direction represents the negative or positive value.

Refer to the following code examples to draw the column sparkline:

C#

```
//Set Sparkline points to source property
this.sparkLine1.ItemsSource = new double[] { 30, -20, 80, 20, 40, -50, -30,
70, -40, 50 };
//Set line type sparkline
this.sparkLine1.SparkLineType = SparkLine.SparkLineType.Column;
```

VB.NET

```
'Set Sparkline points to source property
Me.sparkLine1.ItemsSource = New Double() {30, -20, 80, 20, 40, -50, -30, 70, -
40, 50}
'Set line type sparkline
Me.sparkLine1.SparkLineType = SparkLine.SparkLineType.Column
```

*Drawing WinLoss Sparkline in an Application*

The Winloss type of spark line is similar to column type but all columns have equal length for data points. The vertical column direction represents the negative or positive value.

Refer to the following code examples to draw the WinLoss sparkline:

C#

```
//Set Sparkline points to source property
```

```
this.sparkLine1.ItemSource = new double[] { 30, -20, 80, 20, 40, -50, -30, 70, -40, 50 };
//Set line type sparkline
this.sparkLine1.SparkLineType = SparkLine.SparkLineType.WinLoss;
```

VB.NET

```
'Set Sparkline points to source property
Me.sparkLine1.ItemSource = New Double() {30, -20, 80, 20, 40, -50, -30, 70, -40, 50}
'Set line type sparkline
Me.sparkLine1.SparkLineType = SparkLine.SparkLineType. WinLoss
```

**Marker Support****Markers Support for Line**

This marker feature supports data points of line sparkline. You can choose the marker color for data points.

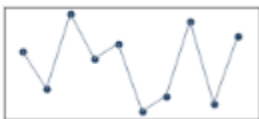
Refer to the following code examples to enable the marker in line sparkline.

C#

```
//To enable marker to sparkline for all data points
this.sparkLine1.Markers.ShowMarker = true;
```

VB.NET

```
'To enable marker to sparkline for all data points
Me.sparkLine1.Markers.ShowMarker = True
```



You can choose the highlight color for data points.

Refer to the following code examples to enable the marker in column sparkline.

C#

```
//To enable marker to sparkline high,low,start,end,negative data points
this.sparkLine1.IsHighPointHighlighted = true;
this.sparkLine1.IsLowPointHighlighted = true;
this.sparkLine1.IsFirstPointHighlighted = true;
this.sparkLine1.IsLastPointHighlighted = true;
this.sparkLine1.IsNegativePointHighlighted = true;
```

VB.NET

```
//To enable marker to sparkline high,low,start,end,negative data points
Me.sparkLine1.IsHighPointHighlighted = true;
Me.sparkLine1.IsLowPointHighlighted = true;
Me.sparkLine1.IsFirstPointHighlighted = true;
Me.sparkLine1.IsLastPointHighlighted = true;
Me.sparkLine1.IsNegativePointHighlighted = true;
```

Markers Support for Column

This marker feature supports high point, low point, start point, end point and negative points of column sparkline. You can choose the highlight color for data points.

Refer to the following code examples to enable the marker in column sparkline.

C#

```
//To enable marker to sparkline high,low,start,end,negative data points
this.sparkLine1.IsHighPointHighlighted = true;
this.sparkLine1.IsLowPointHighlighted = true;
this.sparkLine1.IsFirstPointHighlighted = true;
this.sparkLine1.IsLastPointHighlighted = true;
this.sparkLine1.IsNegativePointHighlighted = true;
```

VB.NET

```
//To enable marker to sparkline high,low,start,end,negative data points
Me.sparkLine1.IsHighPointHighlighted = true;
Me.sparkLine1.IsLowPointHighlighted = true;
Me.sparkLine1.IsFirstPointHighlighted = true;
Me.sparkLine1.IsLastPointHighlighted = true;
Me.sparkLine1.IsNegativePointHighlighted = true;
```



Markers Support for WinLoss

This marker feature supports High Points, Low Points, Start Point, End point and Negative Point of WinLoss Sparkline. You can choose the highlight color for data points. The markers feature of WinLoss is the same as Column markers.

Refer to the following code examples to enable the marker in column sparkline.

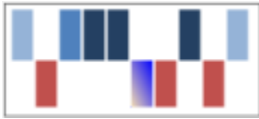
C#

```
//To enable marker to sparkline high,low,start,end,negative data points
this.sparkLine1.IsHighPointHighlighted = true;
this.sparkLine1.IsLowPointHighlighted = true;
this.sparkLine1.IsFirstPointHighlighted = true;
this.sparkLine1.IsLastPointHighlighted = true;
this.sparkLine1.IsNegativePointHighlighted = true;
```

VB.NET

```
//To enable marker to sparkline high,low,start,end,negative data points
```

```
Me.sparkLine1.IsHighPointHighlighted = true;
Me.sparkLine1.IsLowPointHighlighted = true;
Me.sparkLine1.IsFirstPointHighlighted = true;
Me.sparkLine1.IsLastPointHighlighted = true;
Me.sparkLine1.IsNegativePointHighlighted = true;
```



Range Band for Sparkline Chart

Range Band is useful for displaying the normal range of a variable in a Sparkline Chart. By default, the band is defined as a grey rectangle.

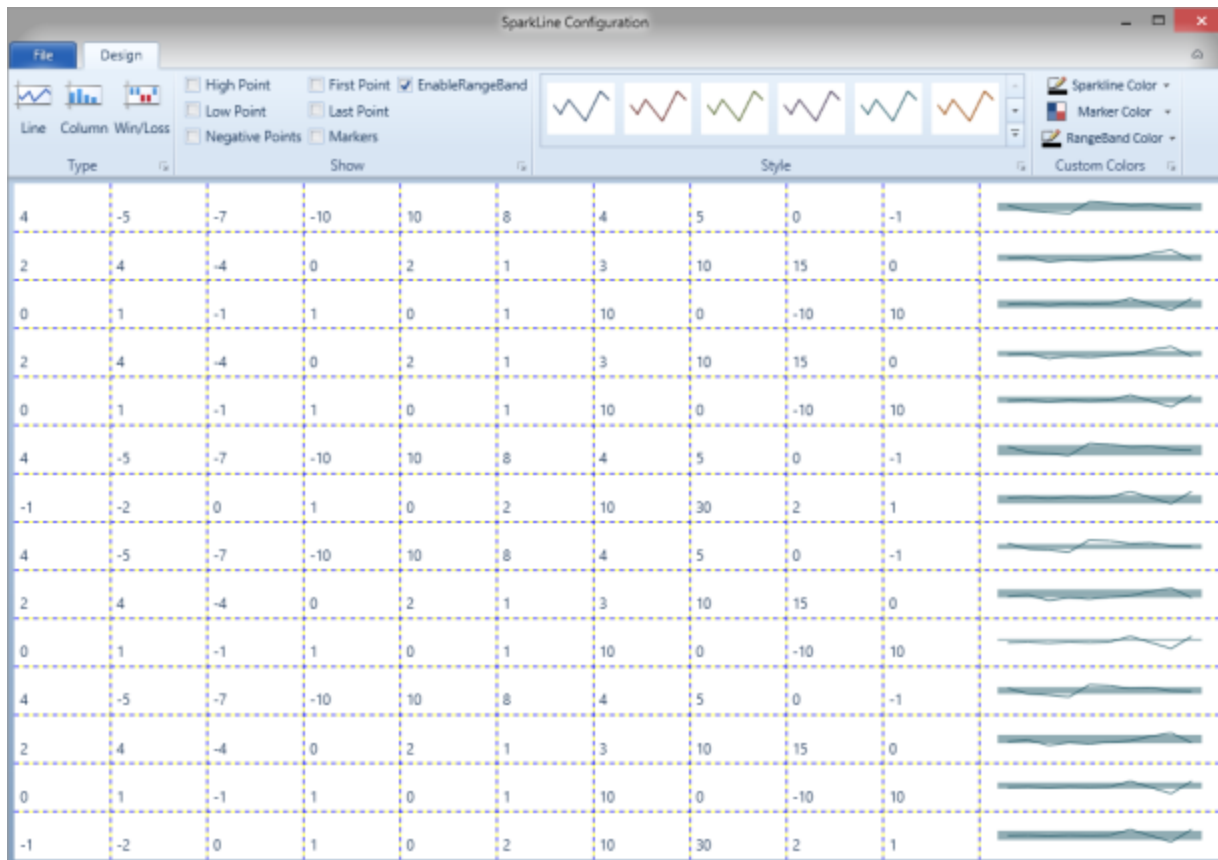
You can customize a particular range by using the `BandRange`, `EnableRangeBand` and `RangeBandInterior` properties.

Property	Description	Type	Data Type	Reference Links
<code>BandRange</code>	Gets or sets the start and end point of the Range Band.	Dependency Property	Double Range	NA
<code>IsEnableRangeBand</code>	Specifies whether the Range Band is visible.	Dependency Property	Bool	NA
<code>RangeBandInterior</code>	Sets the interior color of the Range Band.	Dependency Property	Brush	NA

The following code example shows how to configure the range band.

XML

```
<syncfusion:SparkLine Grid.Row="0" Grid.Column="10" ItemsSource="{Binding
Collections}" DataMemberPath="Y" SparkLineType="Line" BandRange="-5,8"
IsEnableRangeBand="True" RangeBandInterior="Red"/>
```



[Sample Link](#)

To access a Sparkline sample Demo:

1. Open the Syncfusion Dashboard.
2. Select User Interface.
3. Click the WPF drop-down list and select Explore Samples.
4. Browse to the path Chart.WPF\Samples\3.5\WindowsSamples\SparkLine\

Timeline in WPF Chart (Classic)

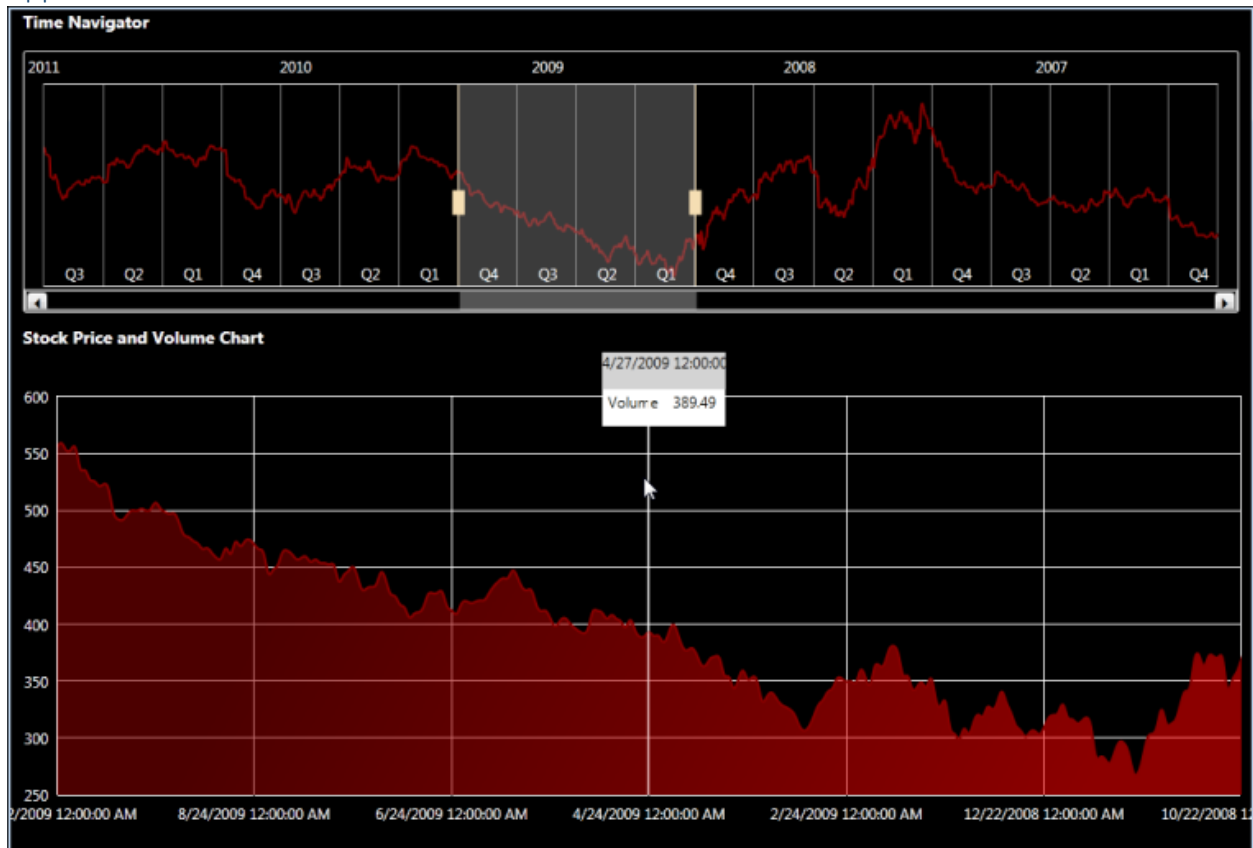
TimeLine control can be added to other controls like Chart, Grid, etc. It is mainly used to set the ViewPort for a limited time to view the selected data. Following are the list of characteristics of TimeLine Control:

- It gets the DataSource from the user and produces a limited set of data using the SelectedData property
- It includes the ViewPort to select the limited data
- The ViewPort can be dragged and dropped at any position inside the TimeLineControl
- It also includes the TimeLine indicator to change the size of the TimeLine Control
- The ViewPort and TimeLine Control can be customized

Use Case Scenario

TimeLine Control has been designed to visualize large number of data and to handle the selected data for instance, in StockMarket analysis; the stock price and volume of the selected data can be handled.

Appearance and Structure



Feature Summary

The features of TimeLine Control are:

- Panning of selected data can be done by using drag and drop in view port
- View port can also be resized to modify the selected data
- Double-clicking the view port displays the minimum time line interval
- When there is an unselected area in TimeLine control, the view port is automatically dragged to the selected area
- When mouse over is done on the view port it displays the minimum TimeLine interval

Adding TimeLine Control

The following code illustrates how to add TimeLine control:

XML

```
<sync:TimeLineControl Height="100" x:Name="rangeIndicator" Grid.Row="1"
DataSource="{Binding}" BindingPathX="X" BindingPathsY="Y"
TimeLineInterior="{Binding ElementName=timelineInterior, Path=SelectedItem,
Mode=TwoWay}"
ViewPortInterior="{Binding ElementName=viewport, Path=SelectedItem,
Mode=TwoWay}"
ViewLineInterior="{Binding ElementName=viewline, Path=SelectedItem,
Mode=TwoWay}" />
```

Samples Link

To view sample,

1. Open the WPF sample browser from the dashboard.
2. Navigate to WPF Chart-> TimeLine Control>TimeLine Control Demo.

Properties

Property	Description	Type	Data Type	Reference links
SelectedData	It returns the selected data in the ViewPort.	Dependency Property	Object	NA
DataSource	To give the input data for the TimeLineControl.	Dependency Property	IEnumerable	NA
BindingPathX	To set the X binding to the data.	Dependency Property	String	NA
BindingPathsY	To set the Y value binding to the Data.	Dependency Property	IEnumerable<string>	NA
StartDate	Specifies the starting date of the TimeLine.	Attached Property	Double	NA
EndDate	Specifies the ending date of the TimeLine.	Attached Property	Double	NA
StartValue	Specifies the starting value of the TimeLine.	DependencyProperty	Double	NA
EndValue	Specifies the ending value of the TimeLine.	DependencyProperty	Double	NA
ViewPortInterior	Sets the interior for the View Port.	Dependency Property	Brush	NA
ViewLineInterior	Sets the interior for the TimeLine.	Dependency Property	Brush	NA
TimeLineInterior	Sets the interior for the Timeline. series	Dependency property	Brush	NA

MinimumTimeLineInterval	To set the minimum interval for the view port.	Dependency property	Double	NA
-------------------------	--	---------------------	--------	----

Setting the Starting and Ending Date

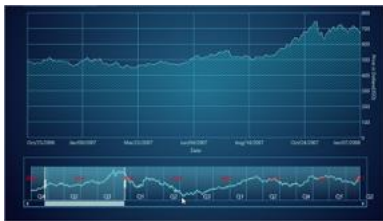
You can specify the starting date and ending date for the TimeLine control by using the StartDate and EndDate attached properties. This is applicable when the ValueType property of PrimayAxis is set to *DateTime*. The following code example illustrates this.

XML

```
<sync:TimeLineControl x:Name='timelineControl'
sync:TimeLineControl.StartDate='9/27/2006'
sync:TimeLineControl.EndDate='12/7/2007'>
<sync:TimeLineControl.PrimaryAxis>
<sync:ChartAxis ValueType='DateTime' />
</sync:TimeLineControl.PrimaryAxis>
</sync:TimeLineControl>
```

C#

```
TimeLineControl.SetStartDate(timelineControl, datalist[20].TimeStamp);
TimeLineControl.SetEndDate(timelineControl, datalist[40].TimeStamp);
```



Setting the Starting and Ending Value

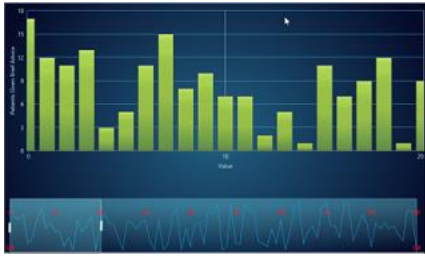
You can specify the starting value and ending value for the TimeLine control by using the StartValue and EndValue attached properties. This is applicable when the ValueType property of the PrimayAxis is set to *Double*. The following code example illustrates this.

XML

```
<sync:TimeLineControl x:Name='timelineControl'
sync:TimeLineControl.StartValue='0' sync:TimeLineControl.EndValue='20'>
<sync:TimeLineControl.PrimaryAxis>
<sync:ChartAxis ValueType='Double' />
</sync:TimeLineControl.PrimaryAxis>
</sync:TimeLineControl>
```

C#

```
TimeLineControl.SetStartValue(timelineControl, 0);
TimeLineControl.SetEndValue(timelineControl, 20);
```



How to

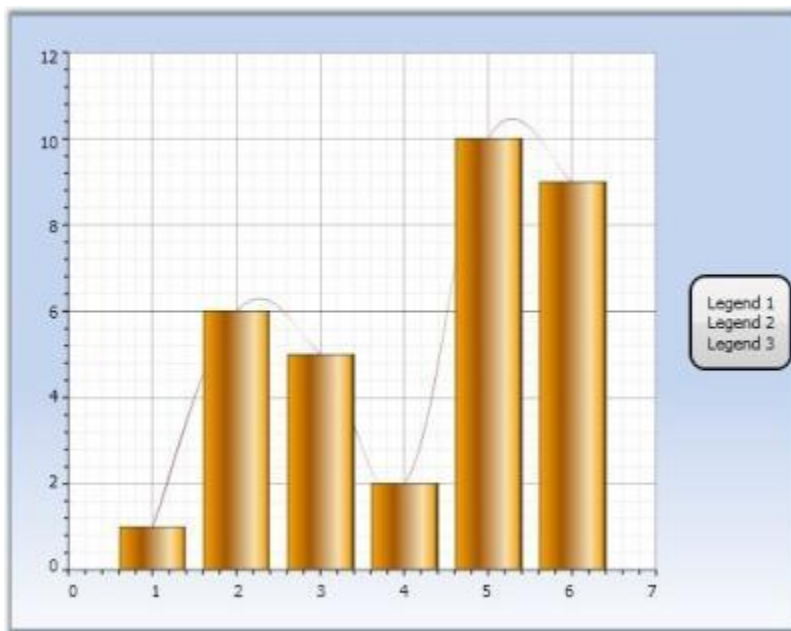
Add Custom Legends

It's easy to replace existing, default, legend items with custom items in Chart or Chart Area legends.

Remember to clear the existing default entries, before adding new custom items. Otherwise, this results in exceptions. The following lines of code can be used to add items to chart legend.

C#

```
ChartLegend legend = new ChartLegend();
legend.Items.Clear();
legend.Items.Add("Legend 1");
legend.Items.Add("Legend 2");
legend.Items.Add("Legend 3");
Chart1.Legends.Add(legend);
```



Auto scale Y axis when X axis is zoomed

The auto scale of y-axis when the x-axis is zoomed can be achieved by the following steps:

1. Get the start and end value of the visible range of X axis.
2. Filter out the point's collection using the RecordFilters and get the points within that range.
3. Get the maximum and minimum value of Y axis.
4. Change the range of the Y axis with the interval.

The following code example must be given under the VisibleRangeChanged event.

C#

```
void chartControl1_VisibleRangeChanged(object sender, EventArgs e)
{
    MinMaxInfo xInfo = chartControl1.PrimaryXAxis.VisibleRange;
    double max = double.MinValue;
    double min = double.MaxValue;
    bool toChange = false;
    double start = 0;
    double end = 0;
    // 1. Get the Start and End value of the visible range of X axis.
    start = xInfo.Min + xInfo.Interval;
    end = xInfo.Max + xInfo.Interval;
    // 2. Filter out the point's collection using the Record filter and get
    the points with in that range.
    GetVisiblePoints(start, end, ref max, ref min, ref toChange);
    // 4. Change the range of the Y axis with the interval.
    double steps = (max-min)/5;
    if (toChange)
    {
        m_Min = min - steps;
        m_Max = max + steps;
        chartControl1.PrimaryYAxis.Range = new MinMaxInfo(m_Min, m_Max, steps);
        chartControl1.ZoomFactorY = 1;
    }
}

private void GetVisiblePoints(double start, double end, ref double max, ref
double min, ref bool toChange)
{
    ChartPointIndexer points = this.chartControl1.Series[0].Points;
    Engine group = new Engine();
    group.SetSourceList(points);
    RecordFilterDescriptor rfd = new
RecordFilterDescriptor("X", FilterLogicalOperator.And, new
FilterCondition[] { new
FilterCondition(FilterCompareOperator.GreaterThanOrEqualTo, start), new
FilterCondition(FilterCompareOperator.LessThanOrEqualTo, end) });
    // add the record filter to group engine
    group.TableDescriptor.RecordFilters.Add(rfd);
    // 3. Get the maximum and minimum value of Y.
    foreach (Record rec in group.Table.FilteredRecords)
    {
        toChange = true;
        double[] yvalues = (double[])rec["YValues"];
        max = yvalues[0] > max ? yvalues[0] : max;
        min = yvalues[0] < min ? yvalues[0] : min;
    }
}
```

VB.NET

```
Private Sub chartControl1_VisibleRangeChanged(ByVal sender As Object, ByVal
e As EventArgs)
    Dim xInfo As MinMaxInfo = chartControl1.PrimaryXAxis.VisibleRange
    Dim max As Double = Double.MinValue
```

```

Dim min As Double = Double.MaxValue
Dim toChange As Boolean = False
Dim start As Double = 0
Dim [end] As Double = 0
' 1. Get the Start and End value of the visible range of X axis.
start = xInfo.Min + xInfo.Interval
[end] = xInfo.Max + xInfo.Interval
'2. Filter out the point's collection using the Record filter and get the
points within that range.
GetVisiblePoints(start, [end], max, min, toChange)
' 4. Change the range of the Y axis with the interval.
Dim steps As Double = (max-min)/5
If toChange Then
m_Min = min - steps
m_Max = max + steps
chartControl1.PrimaryYAxis.Range = New MinMaxInfo(m_Min, m_Max, steps)
chartControl1.ZoomFactorY = 1
End If
End Sub
Private Sub GetVisiblePoints(ByVal start As Double, ByVal [end] As Double,
ByRef max As Double, ByRef min As Double, ByRef toChange As Boolean)
Dim points As ChartPointIndexer = Me.chartControl1.Series(0).Points
Dim group As Engine = New Engine()
group.SetSourceList(points)
Dim rfd As RecordFilterDescriptor = New
RecordFilterDescriptor("X", FilterLogicalOperator.And, New
FilterCondition() {New
FilterCondition(FilterCompareOperator.GreaterThanOrEqualTo, start), New
FilterCondition(FilterCompareOperator.LessThanOrEqualTo, [end]) })
' add the record filter to group engine
group.TableDescriptor.RecordFilters.Add(rfd)
' 3. Get the maximum and minimum value of Y.
For Each rec As Record In group.Table.FilteredRecords
toChange = True
Dim yvalues() As Double = CType(rec("YValues"), Double())
max = If(yvalues(0) > max, yvalues(0), max)
min = If(yvalues(0) < min, yvalues(0), min)
Next rec
End Sub

```

Bind Commands With Chart Or Chart Area

Custom RoutedUICommands can be bound to the Chart or ChartArea just like any other control and InputBindings such as KeyboardGestures and MouseGestures can be set for those commands. Note that for KeyboardGestures and MouseGestures to work, Chart or ChartArea should be in focus. In the following sample, a custom command is created to change the series type of all the ChartSeries in the ChartArea to ChartType.Bar.

XML

```

<Window.Resources>
<XmlDataProvider x:Key="myXmlData">
<x:XData>
<Products xmlns="">
<Product X="1" Series1Y="3" Series1Y1="6" Series2Y="4" Series2Y1="6"
Series3Y="2" Series3Y1="2"/>

```

```

<Product X="2" Series1Y="2" Series1Y1="4" Series2Y="3" Series2Y1="5"
Series3Y="2.5" Series3Y1="6"/>
<Product X="3" Series1Y="3" Series1Y1="6" Series2Y="3.5" Series2Y1="5"
Series3Y="1" Series3Y1="4"/>
<Product X="4" Series1Y="2" Series1Y1="7" Series2Y="1" Series2Y1="6"
Series3Y="4" Series3Y1="6"/>
<Product X="5" Series1Y="1" Series1Y1="3" Series2Y="3.5" Series2Y1="1"
Series3Y="3" Series3Y1="8"/>
</Products>
</x:XData>
</XmlDataProvider>
</Window.Resources>
<Grid>
<sfchart:Chart Name="Chart1">
<sfchart:ChartArea GridBackground="White">
<sfchart:ChartSeries Name="Series1" Type="Column"
DataSource="{Binding Source={StaticResource myXmlData},
XPath=Products/Product}"
BindingPathX="X" BindingPathsY="Series1Y, Series1Y1" >
</sfchart:ChartSeries>
</sfchart:ChartArea>
</sfchart:Chart>
</Grid>

```

C#

```

public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();
        CommandBinding commandbinding = new
        CommandBinding(CustomCommand.ChangeSeriesType, OnChangeSeriesTypeExecute,
        OnChangeSeriesTypeCanExecute);
        CommandManager.RegisterClassCommandBinding(typeof(ChartArea),
        commandbinding);
        Chart1.Areas[0].Focusable = true;
    }
    private static void OnChangeSeriesTypeExecute(object target,
        ExecutedRoutedEventArgs args)
    {
        ChartArea area = (ChartArea)target;
        foreach (ChartSeries series in area.Series)
        {
            series.Type = ChartTypes.Bar;
        }
    }
    private static void OnChangeSeriesTypeCanExecute(object target,
        CanExecuteRoutedEventArgs args)
    {
        ChartArea area = (ChartArea)target;
        if (area != null)
        {
            if (area.Series.Count > 0)
            {
                args.CanExecute = true;
            }
        }
    }
}

```

```

    }
    else
    {
        args.CanExecute = false;
    }
    }
    else
    {
        args.CanExecute = false;
    }
    }
    }
    public class CustomCommand
    {
        private readonly static RoutedUICommand ch_type = new
        RoutedUICommand("ChangeSeriesType", "ChangeSeriesType",
        typeof(CustomCommand));
        public static RoutedUICommand ChangeSeriesType
        {
            get
            {
                ch_type.InputGestures.Add(new KeyGesture(Key.T, ModifierKeys.Control));
                return ch_type;
            }
        }
    }
}

```

Bind Data Using Data Context

When using multiple series' in the Chart, it might be convenient to set the Data Contexts at the Chart or Chart Area levels and refer to that context from the Chart Series.

XML

```

<sfchart:Chart Name="chart1" DataContext="{Binding Source={StaticResource
myXmlData}, XPath=Products/Product}" >
<sfchart:ChartArea View3DMode="True" >
<sfchart:ChartSeries Label="Sales" DataSource="{Binding}"
BindingPathX="Month" BindingPathsY="Sales" Type="Column">
</sfchart:ChartSeries>
<sfchart:ChartSeries Label="Projected Sales" DataSource="{Binding}"
BindingPathX="Month" BindingPathsY="Projected" Type="Column">
</sfchart:ChartSeries>
</sfchart:ChartArea>
</sfchart:Chart>

```

Change the position of the Chart Legend

Chart Legend can be docked by using the Dock property of DockPanel class. The Legend can be docked to the top, left, bottom or right of either the Chart or Chart Area; also it can be placed anywhere inside or outside the Chart Area (floating).

Property	Description
ChartDockPanel.Dock	Docks the Chart Legend in the Chart or Chart Area. The values provided are as follows:Top: Docks the Chart Legend to the top.Bottom: Docks the Chart Legend

	to the bottom.Right: Docks the Chart Legend to the right.Left: Docks the Chart Legend to the left.Floating: Chart Legend can be dragged anywhere inside or outside the Chart Area.
--	--

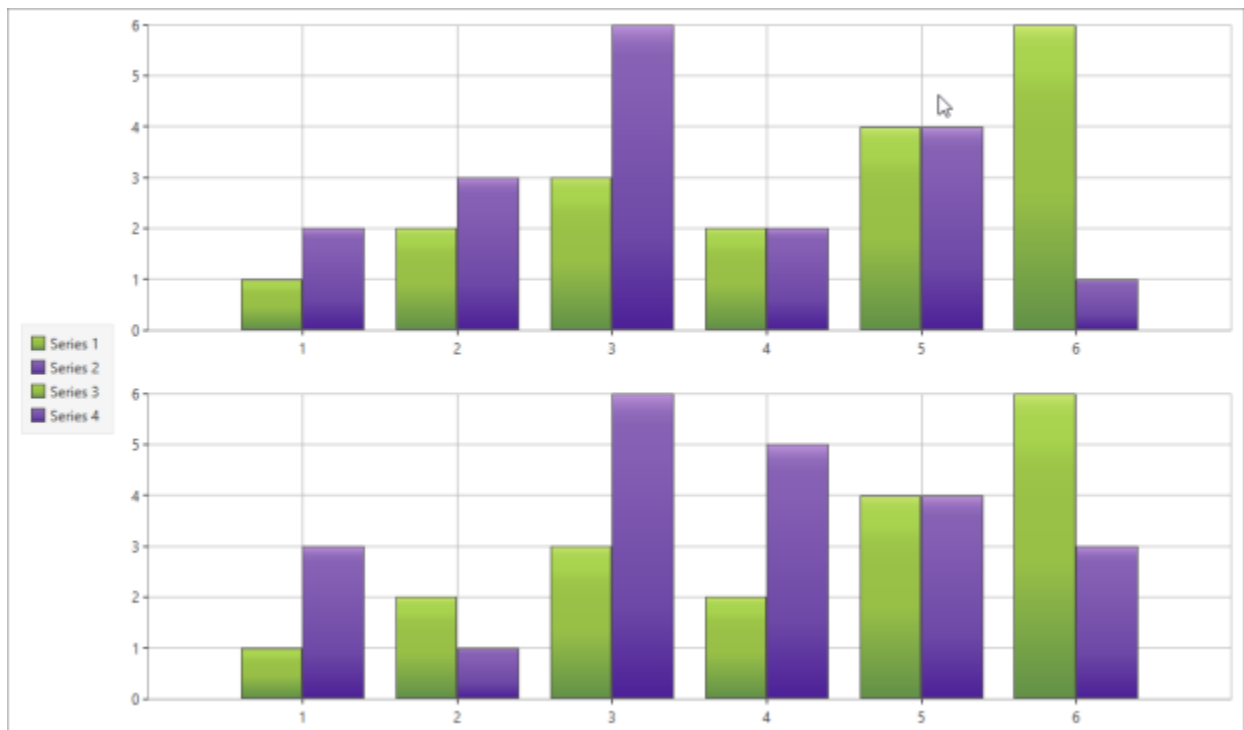
The following code example illustrates how to dock the Chart Legend in the Chart Area.

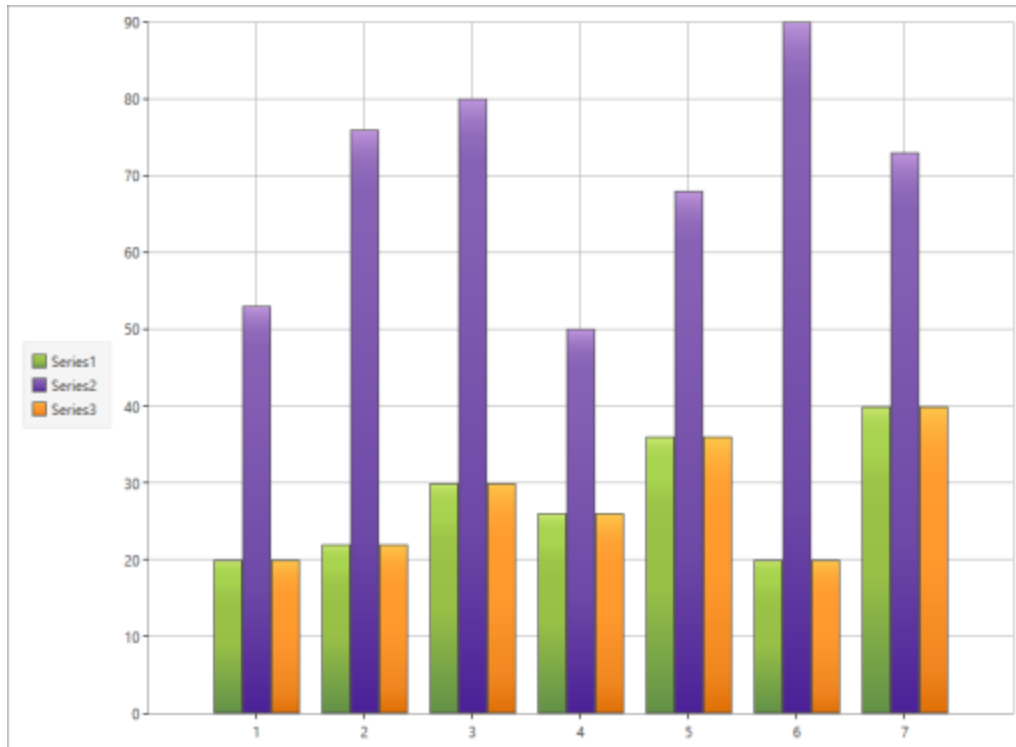
XML

```
<chart:Chart.Legends>
  <chart:ChartLegend chart:Chart.Dock="Left">
  </chart:ChartLegend>
</chart:Chart.Legends>
<syncfusion:ChartArea.Legend>
  <syncfusion:ChartLegend Name="Legend1"
  syncfusion:ChartDockPanel.Dock="Left"/>
</syncfusion:ChartArea.Legend>
```

C#

```
ChartDockPanel.SetDock(Legend1, ChartDock.Left);
```





Export Chart as Image

In order to export the Chart control as an image, the Save method needs to be called. Two overload methods are available to export the Chart:

- Save(Stream stream)
- Save(string fileName)
- Save(Stream stream, BitmapEncoder encoder)
- Save(Stream stream, Rect saveArea)
- Save(string fileName, BitmapEncoder encoder)
- Save(string fileName, Rect saveArea)
- Save(Stream stream, Rect saveArea, BitmapEncoder encoder)
- Save(string fileName, Rect saveArea, BitmapEncoder encoder)

The following code examples can be used to export the Chart as an image:

C#

```
SaveFileDialog saveFileDialog = new SaveFileDialog();
string C_imageFilesFilter =
"Bitmap(*.bmp)|*.bmp|JPEG(*.jpg;*.jpeg)|*.jpg;*.jpeg|Gif
(*.gif)|*.gif|TIFF(*.tiff)|*.tiff|PNG(*.png)|*.png|WDP(*.wdp)|*.wdp|Xps file
(*.xps)|*.xps|All files (*.*)|*.*";
saveFileDialog.Filter = C_imageFilesFilter;
if (saveFileDialog.ShowDialog() == true)
{
    Chart1.Save(saveFileDialog.FileName);
}
```


Use Custom Adorners In Chart Or Chart Area

Custom Adorners can be added to the Chart or ChartArea to add any custom UI elements over the Chart or ChartArea. The following code example illustrates adding and removing adorners to a ChartArea.

XML

```
<sfchart:Chart Name="Chart1">
  <sfchart:ChartArea GridBackground="White">
    <sfchart:ChartSeries Name="Series1" Type="Column"
      DataSource="{Binding Source={StaticResource myXmlData},
        XPath=Products/Product}"
      BindingPathX="X" BindingPathsY="Series1Y, Series1Y1" >
    </sfchart:ChartSeries>
  </sfchart:ChartArea>
</sfchart:Chart>
```

C#

```
public class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();
    }
    private void SetAdornmentClick(object sender, RoutedEventArgs e)
    {
        foreach (ChartArea area in Chart1.Areas)
        {
            AdornerLayer.GetAdornerLayer(area).Add(new CustomAdorner(area));
        }
    }
    private void RemoveAdornment(object sender, RoutedEventArgs e)
    {
        // Get adorner layer for Chart control.
        foreach (ChartArea area in Chart1.Areas)
        {
            AdornerLayer adornerLayer = AdornerLayer.GetAdornerLayer(area);
            // Iterate over each adorner.
            foreach (Adorner adorner in adornerLayer.GetAdorners(area))
            {
                if (adorner is CustomAdorner)
                {
                    adornerLayer.Remove(adorner);
                    break;
                }
            }
        }
    }
    internal class CustomAdorner : Adorner
    {
        public CustomAdorner(UIElement adornedElement)
        : base(adornedElement) { }
        protected override void OnRender(DrawingContext drawingContext)
        {
            Brush b = (Brush)new BrushConverter().ConvertFromString("#7A4047F7");
```

```
drawingContext.DrawRectangle(b, new Pen(Brushes.Red, 1), new Rect(new
Point(0, 0), DesiredSize));
FormattedText text = new FormattedText("This is a custom Syncfusion WPF
Chart adorer", Thread.CurrentThread.CurrentUICulture,
FlowDirection.LeftToRight, new Typeface("Arial"), 14, Brushes.White);
drawingContext.DrawText(text, new Point(10, 10));
base.OnRender(drawingContext);
}
}
```

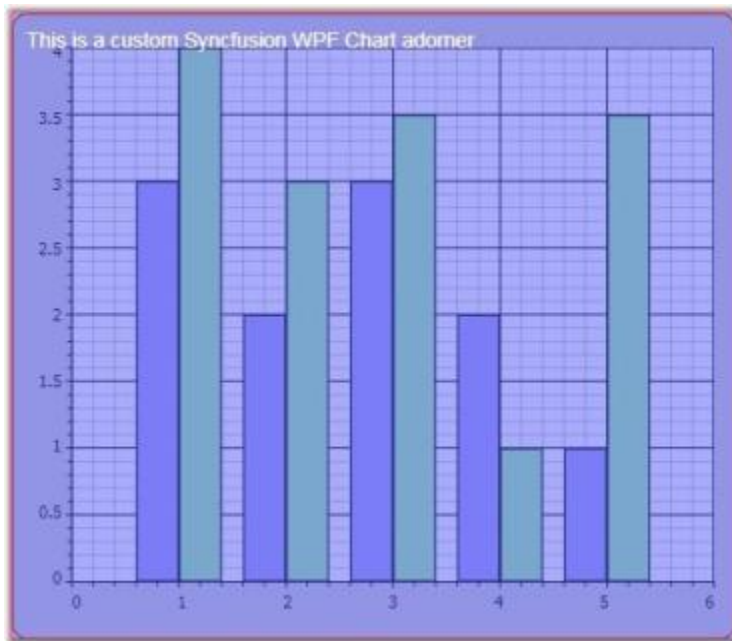


Diagram (classic)

WPF Diagram (classic) Overview

Essential Diagram WPF is an extensible and high-performance diagramming Framework for WPF applications. It can be used by the developers who want to develop Microsoft Visio-like interactive graphics and diagramming applications. It stores graphical objects in a node graph and renders those objects on the screen.

Note: A node graph is a structure consisting of nodes connected to each other by lines referred to as edges.

Essential Diagram supports both vector and raster graphics on the drawing surface.

Note: A raster (bitmap) image uses a grid of individual pixels where each pixel can be a different color or shade. Bitmaps are composed of pixels.

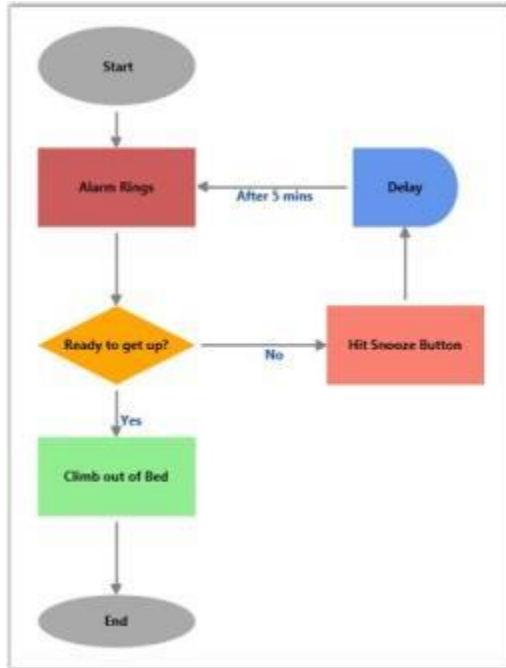
Vector graphics use mathematical relationships between points and the paths connecting them to describe an image. Vector graphics are composed of paths.

Essential Diagram WPF allows you to create interactive diagrams easily.

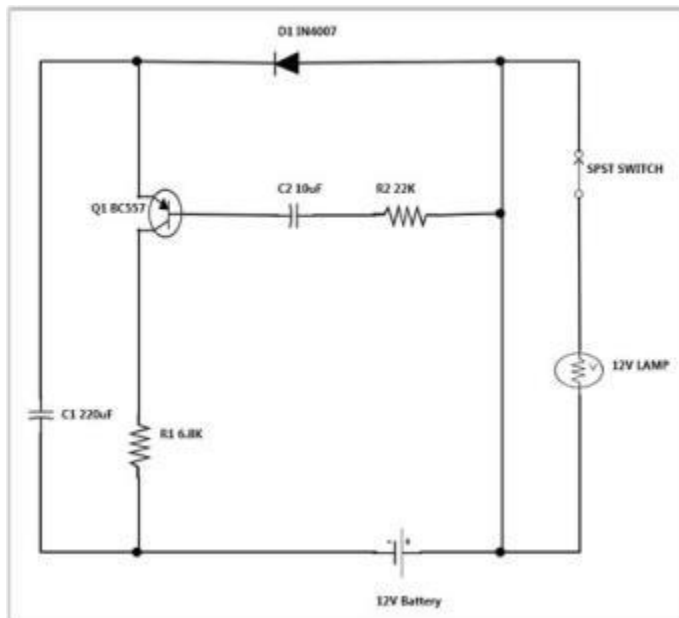
Real World Scenarios

Essential Diagram WPF finds its application in various fields; some of them are listed below.

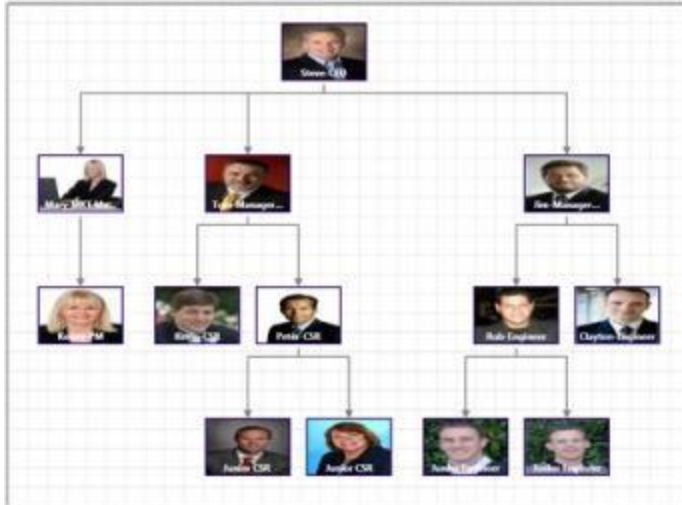
- Essential Diagram WPF can be effectively used to create process flow diagrams and flow charts. The connectors with a label depict the process flow from one level to another.



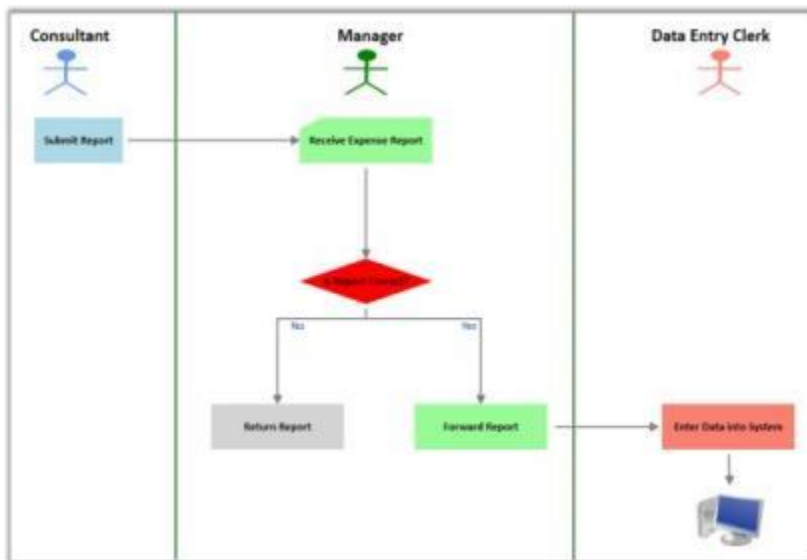
- Essential Diagram WPF can be used to create electrical diagrams. The electrical shapes can be loaded to the SymbolPalette and can be dragged to the drawing surface. Connections can then be made as desired and the circuit diagrams can be created.



- Essential Diagram WPF employs automatic layout algorithms to layout the nodes automatically in a tree structure. This kind of setting is typically useful in creating Organizational Layout and for data binding purposes.

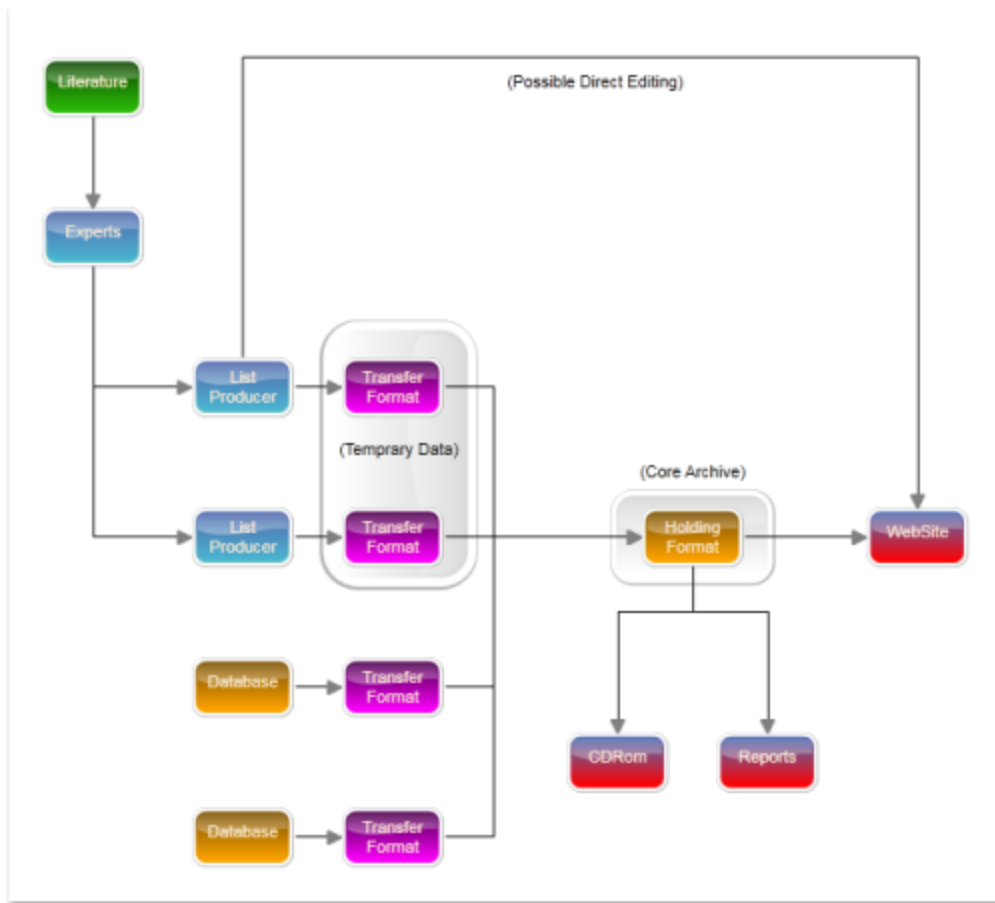


- Essential Diagram WPF allows you to create Swim lane diagrams, which groups a set of sub processes in a visual manner, by arranging them in lanes. Nodes can be manually placed to create swim lanes. The process in each lane may then be described using the nodes and the flow can be depicted using the connections as illustrated in the following diagram.

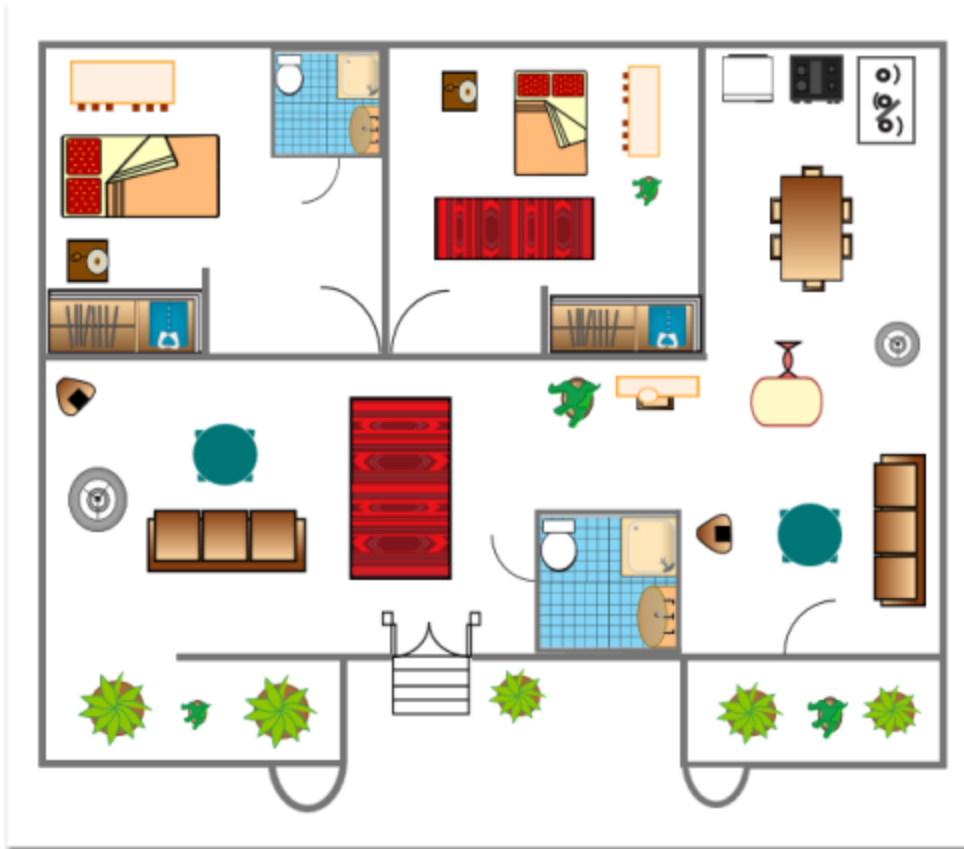


Swim Lane Diagram

- Essential Diagram WPF allows you to create a Data Flow Diagram, A data-flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. DFD's can also be used for the visualization of data processing.



- Essential Diagram WPF allows you to create a Floor Plan,



Key Features

The following are the key features of Essential Diagram WPF:

- **Nodes** - Nodes are graphical objects that can be drawn on the page by selecting them from the SymbolPalette and dropping them on the page.
- **Transformations** - The following transformations are provided:
 1. **Translate**: Ability to move the nodes.
 2. **Rotate**: Ability to rotate the nodes.
 3. **Scale**: Ability to resize the nodes.
- **Groups and Ungroup** - Essential Diagram WPF provides support to group and ungroup nodes. Grouping feature comes in handy when you want to apply the same edits to a number of objects and yet retain their individuality. All the operations performed on the group also affects the individual items in the group. However any item in the group can also be edited individually. On ungrouping, the items in the group again act as individual entities.
- **Layers** - Essential Diagram for WPF supports layer display. Numerous nodes and line connectors can be added to a layer and the visible property of its contents can be hidden by changing the visible property of the layer. A node or line connector can be added to any number of layers, and the node is visible only if the all layers to which this node or line connector belongs to are visible.
- **Connectors** - Connectors are the objects that are used to create a link between two nodes. Three types of connectors provided are as follows:
 1. **Orthogonal**
 2. **Bezier**

3. Straight

- **LineBridging** - Line Bridging creates a bridge for lines to smartly cross over other line at points of intersection.

When two line connectors meets each other, line with higher z-order will draw an arc over the line with lower z-order.

Only Straight and Orthogonal Connector type supports line bridging.

- **Custom Ports** - Essential Diagram WPF provides the ability to define custom ports for making connections. The `ConnectionPort` class can be used for defining custom ports on the nodes. Any number of ports can be defined on a node. The user can define a port on any part of the node and make connection to that port. The port's visibility can also be controlled. Several customizable properties have been provided for the port.
- **Decorator Shapes** - Decorator shapes can be added to the head and tail of the connectors. Three types of decorator shapes provided are as follows:
 1. Arrow
 2. Diamond
 3. Circle
- **Export** - Essential Diagram for WPF offers capabilities to export a diagram representation into various formats. The formats include JPEG, BMP, PNG, TIFF, GIF, WDP and XAML.
- **Command Architecture** - Essential Diagram for WPF provides several commands as follows.
 - Zoom Commands
 - Alignment Commands
 - Spacing and Sizing Commands
 - Group and Ungroup Commands
 - Undo and Redo Commands
 - Z-order Commands
 - Nudge Commands
 - Clipboard Commands
 - Delete Command
- **Zooming, Scrolling and Panning** - Zooming, scrolling, and panning are supported and can be achieved using sufficient interactive diagram tools.
- **Grid Lines** - The drawing area of the Diagram control can be rendered with horizontal and vertical Grid lines.
- **Rulers** - Horizontal and vertical rulers are provided to indicate the coordinates of the mouse position with respect to the view.
- **Measurement Units** - As different fields require different units of measure, several measurement units are provided such that the end-users can choose the unit that is most comfortable and suitable for their use.
- **SymbolPalette** - The `SymbolPalette` control displays the node shapes and allows a user to drag the symbols onto the diagram. It supports grouping and filtering of symbols, and it is implemented based on the Syncfusion's Gallery control. Also, custom shapes can be added to the `SymbolPalette`.
- **Label Editor** - A label editor is provided for each node and connector; it enables the user to edit labels at run time if `IsLabelEditable` property is set to true for the corresponding object.
- **Customizable** - The control is highly customizable and extensible. Customization is easy, and custom UI tools can be easily created and registered.

- Automatic Layout Management - Essential Diagram WPF provides the ability to set automatic layout for the nodes. Several layout types have been provided. They are:
- DirectedTree Layout - The DirectedTree layout arranges nodes in a tree-like structure. This layout can be applied to any diagram that is composed of a directed tree graph with unique root and child nodes.
- HierarchicalTree Layout - The HierarchicalTree layout also arranges nodes in a tree-like structure; however, unlike the directed tree layout, the nodes in hierarchical layout may have multiple parents hence avoiding the need to specify the root.
- RadialTree Layout - The Radial-TreeLayout is a specialization of the Directed Tree Layout Manager that employs a circular layout algorithm for locating the diagram nodes. The RadialTreeLayoutManager arranges nodes in a circular layout, positioning the root node at the center of the graph and the child nodes in a circular fashion around the root. Sub-trees formed by the branching of child nodes are located radially around the child nodes. This arrangement results in an ever-expanding concentric arrangement with radial proximity to the root node indicating the node level in the hierarchy.
- Table Layout - The Table layout arranges the nodes in a tabular structure based on specified intervals between them. The layout depends upon the number of nodes in each row and column specified. The nodes are assigned rows and columns based on the order in which they are added to the model and based on the maximum nodes allowed in that row and column.
- Serialization - The Diagram Page can be saved in XAML format for future use. The user can then load the saved page into the current view and start editing the page.
- Event Mechanisms - Several events have been provided for nodes and connections.
- Print and Print Preview - This feature enables the user to set a printer to be used, and it allows the user to define the pages and the number of copies that should be printed. It also provides an overview of the document, showing how the document will appear when printed.

User Guide Organization

The product comes with numerous samples as well as an extensive documentation to guide you. This User Guide provides detailed information on the features and functionalities of the Essential Diagram for WPF. It is organized into the following sections:

- Overview-This section gives a brief introduction to the product and its key features.
- Getting Started-This section guides you on getting started with WPF application, controls etc.
- Concepts and Features-The features of individual controls are illustrated with use case scenarios, code examples and screen shots under this section.

Document Conventions

The conventions below will help you to quickly identify the important sections of information, while using the content:

Convention	Description of the Icon
Note	Represents important information.
Example	Represents an example.
Tip	Represents useful hints, that will help you in using the controls and features.
Additional information	Represents additional information on the corresponding topic.

Getting Started with WPF Diagram (classic)

This section helps you to get started with Essential Diagram and has the following topics:

Diagram Architecture

The following is a general description about the important classes of Diagram WPF. These classes form the base of the control.

Diagram Control

The Diagram control is the base class, which contains the view and the model. It receives user input and translates it into actions and commands on the model and view. It also implements SymbolPalette and scrolling, and enables horizontal and vertical scrollbars when the size of the view exceeds the size of the window.

Diagram Model

A model represents data for an application and contains the logic for adding, accessing, and manipulating the data. Nodes and connectors are added to the Diagram Control using the Model property. A predefined layout can be applied using the LayoutType property of the DiagramModel, or the position of the nodes can be manually specified.

Diagram View

The view obtains data from the model and presents them to the user. It typically manages the overall layout of the data obtained from the model.

Apart from presenting the data, view also handles navigation between the items, and some aspects of item selection. The views also implements basic user interface features, such as rulers, and drag-and-drop. It handles the events, which occur on the objects, obtained from the model. Command mechanism is also implemented by the view.

A view can be constructed without a model, but a model must be provided before it can display useful information. Views can also render additional visual information that do not exist inside the model such as bounding boxes and grids. These additional view-specific objects are referred to as decorators, because they provide additional visual aids and window dressing to the view; but they are not actually a part of the model.

Diagram Page

The DiagramPage is just a container to hold the objects(nodes and connectors) added through the model. The DiagramView uses the page to display the diagram objects. As mentioned before, the view implements several basic user interface features like rulers, grids, events and commands. So therefore page is just a container to hold the graphical objects added through the model and the DiagramView uses it to display the objects.

SymbolPalette

The SymbolPalette control displays node shapes and allows a user to drag and drop symbols onto diagrams. It supports grouping and filtering symbols. It allows users to classify items as groups, so they can be navigated easily. Also, custom shapes can be added to the SymbolPalette.

SymbolPaletteGroup

A SymbolPalette group is a collection of SymbolPalette items. It is used to group the items in the SymbolPalette control based on classifications provided. The SymbolPalette group can be added to the SymbolPalette using the SymbolGroups property.

SymbolPaletteFilter

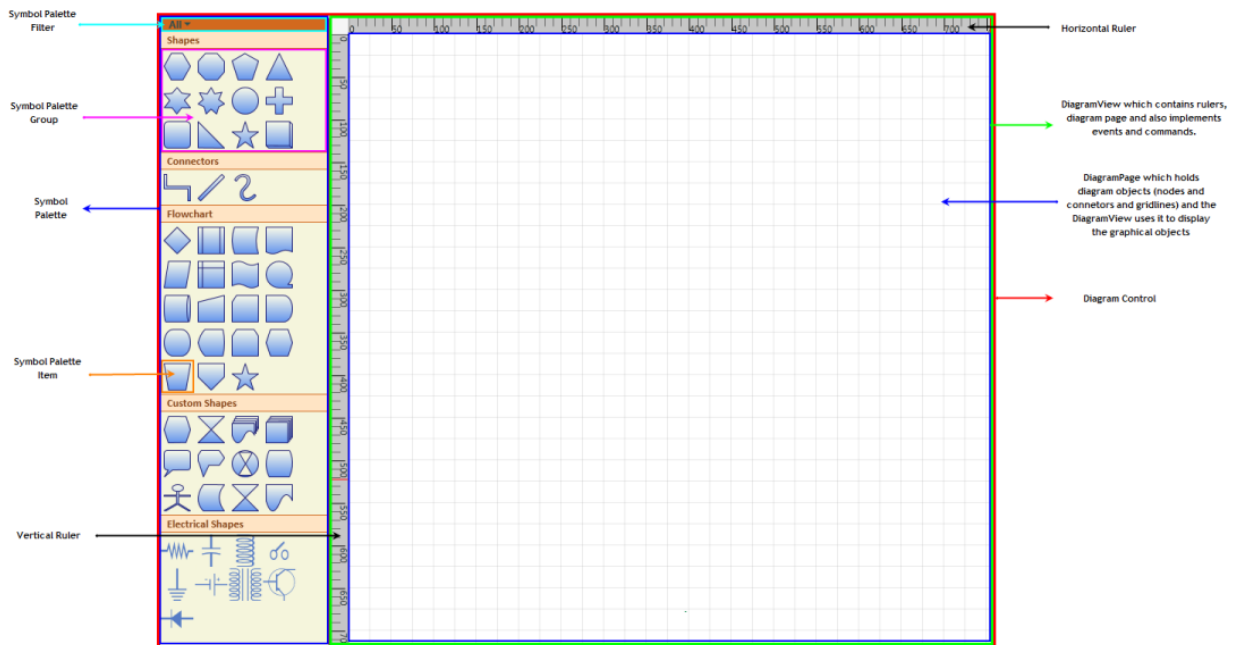
A SymbolPalette filter can be added to the SymbolPalette control, using the SymbolFilters property, so that only desired SymbolPalette groups get displayed.

SymbolPaletteItem

SymbolPalette items are contained in the SymbolPalette group. A SymbolPalette item does not restrict users to the type of content that can be added to it. A SymbolPalette item can be a text box, combo box, image, button, and so on.

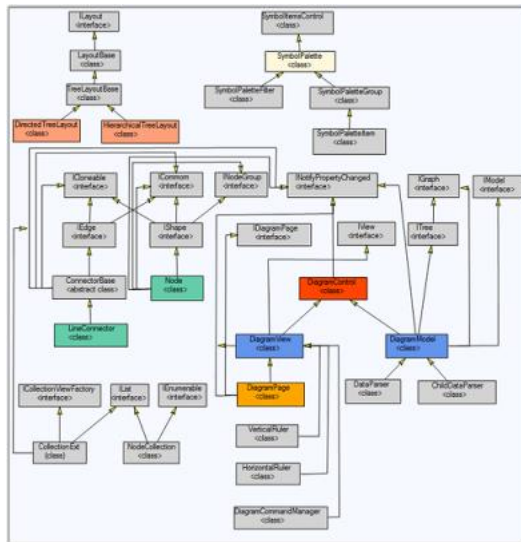
Horizontal / Vertical Ruler

Rulers display the coordinates of elements on the diagram page. Negative label values get displayed on the ruler in case the page is panned to the right side. On Zooming, the ruler values get adjusted accordingly, to match with the current Zoom level. At any point, the ruler value always indicates the exact coordinates of the page and its elements. So when the page is zoomed, the interval values get halved or doubled depending upon the zoom level.



Class Diagram

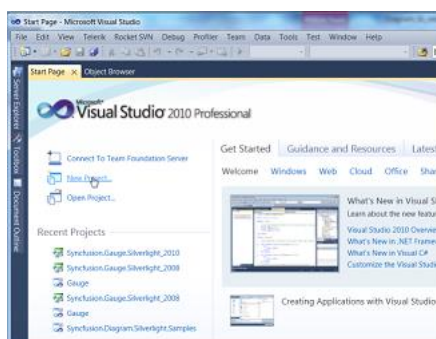
The class diagram for Essential Diagram WPF is as follows.



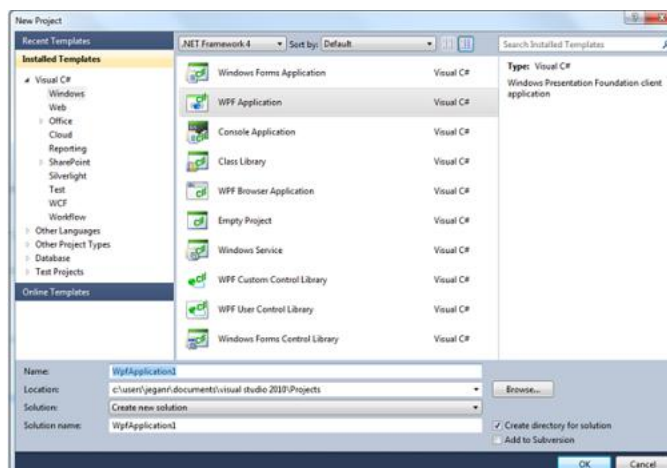
Creating a WPF application

This section illustrates the step-by-step procedure to create a WPF application.

1. Open Microsoft Visual Studio. Click New Project in the Start Page.



2. In the New Project dialog, select WPF Application template, name the project and click OK.



3. A new WPF application is created.

Creating a Diagram

Essential Diagram WPF can be used to create a rich Visio-like application. This Framework provides many utility controls to help you easily put an application together. End users can get started in minutes using this diagram control.

Following is a basic step to create DiagramControl and initialize the necessary properties. Details about individual parts are explained later in this documentation.

Create DiagramControl

The Diagram Control can be added to the application using the following code.

DiagramControl can be created in two ways,

- Through XAML
- Through Code Behind

HTML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:Diagram="clr-namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<Diagram:DiagramControl/>
</Grid>
</Window>
```

C#

```
DiagramControl dc = new DiagramControl();
```

VB.NET

```
Dim dc As New DiagramControl()
```

This shows a window with empty diagram control.

Enabling SymbolPalette

- Now you need to add the SymbolPalette to your newly created Diagram control. The SymbolPalette is displayed by setting the IsSymbolPaletteEnabled property to *True*. By default, it is set to *False*. The following code enables the SymbolPalette.

SymbolPalette can be enabled in two ways,

- Through XAML
- Through Code Behind

HTML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:Diagram="clr-
namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<Diagram:DiagramControl IsSymbolPaletteEnabled="True">
</Diagram:DiagramControl>
</Grid>
</Window>
```

C#

```
DiagramControl diagramcontrol = new DiagramControl();
diagramcontrol.IsSymbolPaletteEnabled = true;
```

VB.NET

```
Dim diagramcontrol As New DiagramControl()
diagramcontrol.IsSymbolPaletteEnabled = True
```

Create DiagramModel

- To add contents into the drawing area, use the Model property of the diagram control. The following code can be used to add the model.

DiagramModel can be created and assigned to DiagramControl's View Property using two ways,

- Through XAML
- Through Code Behind

HTML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:Diagram="clr-
namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<Diagram:DiagramControl IsSymbolPaletteEnabled="True">
<Diagram:DiagramControl.Model>
<Diagram:DiagramModel></Diagram:DiagramModel>
</Diagram:DiagramControl.Model>
</Diagram:DiagramControl>
</Grid>
</Window>
```

C#

```
DiagramControl dc = new DiagramControl();
dc.IsSymbolPaletteEnabled = true;
DiagramModel model = new DiagramModel();
dc.Model = model;
diagramgrid.Children.Add(dc);
```

VB.NET

```
Dim dc As New DiagramControl()
dc.IsSymbolPaletteEnabled = True
Dim model As New DiagramModel()
dc.Model = model
diagramgrid.Children.Add(dc)
```

Create DiagramView

- To display the drawing area, use the View property of the diagram control. The following code can be used to add the view.

DiagramView can be created and assigned to DiagramControl's View Property using two ways,

- Through XAML
- Through Code Behind

HTML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:Diagram="clr-namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<Diagram:DiagramControl IsSymbolPaletteEnabled="True">
<Diagram:DiagramControl.Model>
<Diagram:DiagramModel></Diagram:DiagramModel>
</Diagram:DiagramControl.Model>
<Diagram:DiagramControl.View>
<Diagram:DiagramView ></Diagram:DiagramView>
</Diagram:DiagramControl.View>
</Diagram:DiagramControl>
</Grid>
</Window>
```

C#

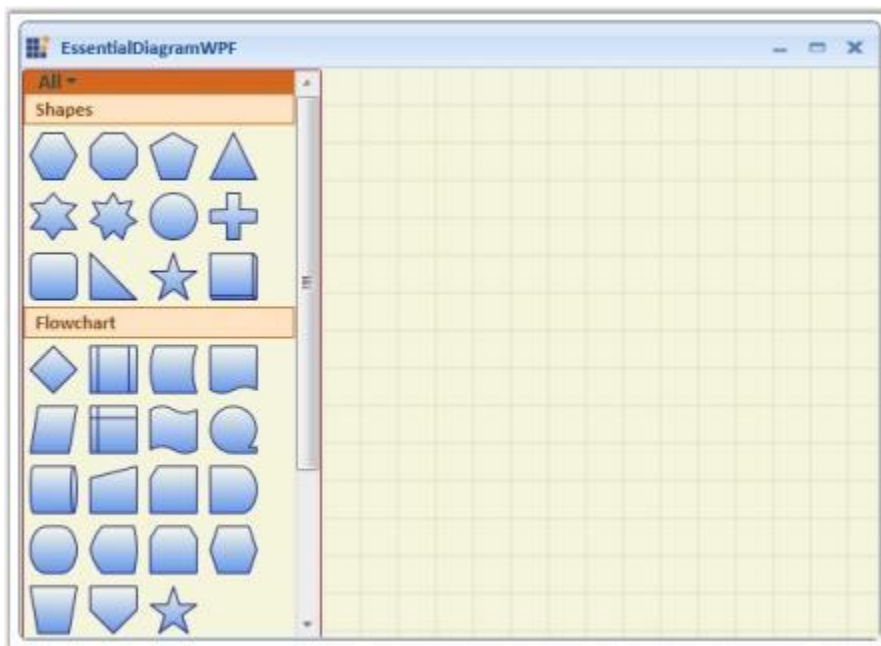
```
DiagramControl dc = new DiagramControl();
dc.IsSymbolPaletteEnabled = true;
DiagramView view = new DiagramView();
view.Bounds = new System.Drawing.Thickness(0, 0, 1000, 1000);
```

```
dc.View = view;  
diagramgrid.Children.Add(dc);
```

VB.NET

```
Dim dc As New DiagramControl()  
dc.IsSymbolPaletteEnabled = True  
Dim view As New DiagramView()  
view.Bounds = New System.Drawing.Thickness(0, 0, 1000, 1000)  
dc.View = view  
diagramgrid.Children.Add(dc)
```

- This creates a Diagram Control with the SymbolPalette and the drawing area as illustrated in the following image.



Note: For orthogonal and Bezier connectors, the connection always happens at the center of the node's edge.

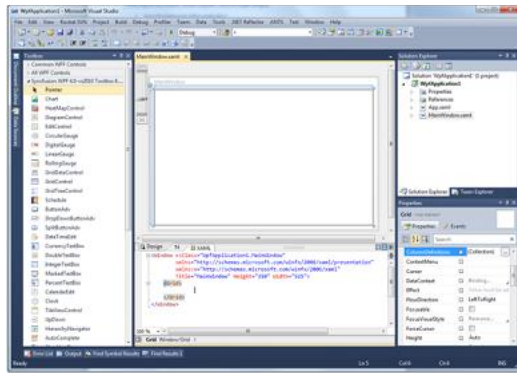
For straight line connectors, the connection happens at the intersection point of the edge and the line connector.

Creating Diagram control through Designer

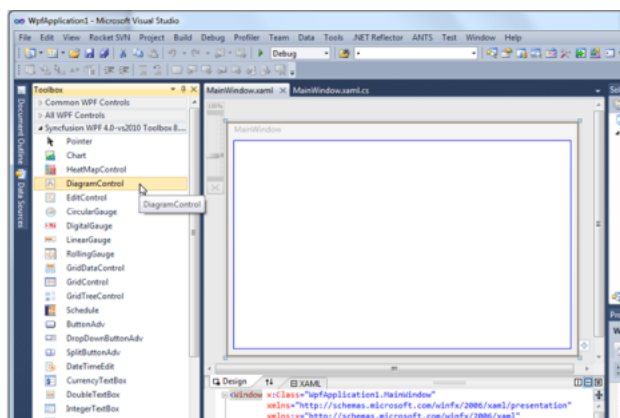
The Diagram Control can be added to the application using designer.

Following are the steps to create Diagram Control through Designer.

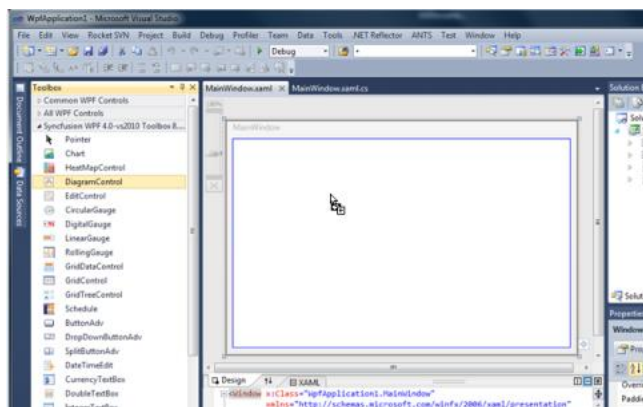
1. Open the XAML page of the application



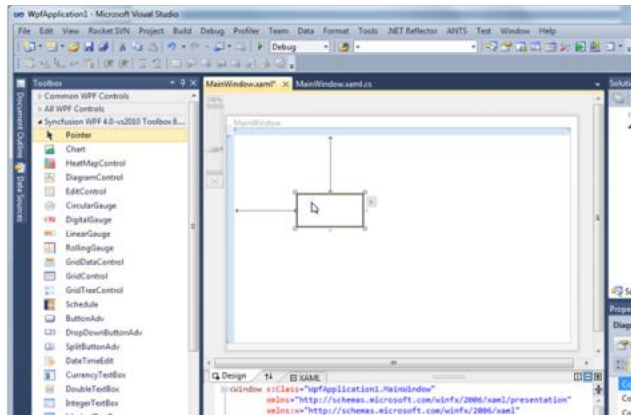
2. Select Diagram Control from ToolBox.



3. Drag the Diagram Control onto the Designer.



4. DiagramControl is added to the Page and also the assembly reference is added to the Project file.



Kindly refer to Add Diagram Model to the Diagram Control and Add Diagram View to the Diagram Control to add the model and view to the control.

Add Diagram Model to the Diagram Control

A model represents data for an application and contains the logic for adding, accessing, and manipulating the data.

Features

- Nodes and connectors are added to the Diagram Control using the Model property.
- A predefined layout is applied using the LayoutType property.
- A data template is applied to the layout using the Hierarchical DataTemplate property.

The following code shows how the Model property can be applied to the Diagram control.

HTML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="420" Width="600"
xmlns:Diagram="clr-namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<Diagram:DiagramControl IsSymbolPaletteEnabled="True">
<Diagram:DiagramControl.Model>
<Diagram:DiagramModel x:Name="diagramModel">
</Diagram:DiagramModel>
</Diagram:DiagramControl.Model>
<Diagram:DiagramControl.View>
<Diagram:DiagramView></Diagram:DiagramView>
</Diagram:DiagramControl.View>
</Diagram:DiagramControl>
</Grid>
</Window>
```

C#

```
DiagramControl dc = new DiagramControl();
dc.IsSymbolPaletteEnabled = true;
```

```
DiagramView view = new DiagramView();
view.Bounds = new System.Drawing.Thickness(0, 0, 1000, 1000);
dc.View = view;
DiagramModel diagramModel = new DiagramModel();
dc.Model = diagramModel;
diagramgrid.Children.Add(dc);
```

VB.NET

```
Dim dc As New DiagramControl()
dc.IsSymbolPaletteEnabled = True
Dim view As New DiagramView()
view.Bounds = New System.Drawing.Thickness(0, 0, 1000, 1000)
dc.View = view
Dim diagramModel As New DiagramModel()
dc.Model = diagramModel
diagramgrid.Children.Add(dc)
```

This adds a model to the Diagram Control, and defines Bounds Property for DiagramModel

Add Diagram View to the Diagram Control

The view obtains items of data from the model and presents them to the user. It typically manages the overall layout of the data obtained from model.

Apart from presenting the data, view also handles navigation between the items, and some aspects of item selection. The views also implement basic user interface features, such as rulers, and drag and drop. It handles the events, which occur on the objects, obtained from the model. Command mechanism is also implemented by the view.

A view can be constructed without a model, but a model must be provided before it can display useful information. Views can also render additional visual information that do not exist inside the model such as bounding boxes and grids. These additional view-specific objects are referred to as decorators, because they provide additional visual aids and window dressing to the view; but they are not actually a part of the model.

The following code illustrates adding a Diagram View to the Diagram control.

HTML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:Diagram="clr-namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<Diagram:DiagramControl IsSymbolPaletteEnabled="True">
<Diagram:DiagramControl.View>
<Diagram:DiagramView </Diagram:DiagramView>
</Diagram:DiagramControl.View>
</Diagram:DiagramControl>
</Grid>
</Window>
```

C#

```
DiagramControl dc = new DiagramControl();
dc.IsSymbolPaletteEnabled = true;
DiagramView view = new DiagramView();
view.Bounds = new System.Drawing.Thickness(0, 0, 1000, 1000);
dc.View = view;
diagramgrid.Children.Add(dc);
```

VB.NET

```
Dim dc As New DiagramControl()
dc.IsSymbolPaletteEnabled = True
Dim view As New DiagramView()
view.Bounds = New System.Drawing.Thickness(0, 0, 1000, 1000)
dc.View = view
diagramgrid.Children.Add(dc)
```

Manual Layout

The Essential Diagram WPF allows you to manually specify the layout of the page. The nodes can be positioned at any point on the diagram page. The OffsetX and OffsetY properties can be used to specify the position. Connections can then be made between the nodes using the various line connectors.

The nodes and the connectors need to be added to the Nodes and Connections collection of DiagramModel respectively. This gives a complete control over the placement of nodes on the page and enables the user to create diagrams as suited for their business needs.

The following code snippet shows how the manual layout may be specified.

C#

```
Node EssentialWPF = new Node(Guid.NewGuid(), "EssentialWPF");
EssentialWPF.Shape = Shapes.Ellipse;
EssentialWPF.Width = 100;
EssentialWPF.Height = 50;
EssentialWPF.OffsetX = 300;
EssentialWPF.OffsetY = 300;
EssentialWPF.Content = "Essential WPF";
Node EssentialTools = new Node(Guid.NewGuid(), "EssentialTools");
EssentialTools.Shape = Shapes.Ellipse;
EssentialTools.Width = 100;
EssentialTools.Height = 50;
EssentialTools.OffsetX = 300;
EssentialTools.OffsetY = 500;
EssentialTools.Content = "Essential Tools";
Node EssentialChart = new Node(Guid.NewGuid(), "EssentialChart");
EssentialChart.Shape = Shapes.Ellipse;
EssentialChart.Width = 100;
EssentialChart.Height = 50;
EssentialChart.OffsetX = 300;
EssentialChart.OffsetY = 100;
EssentialChart.Content = "Essential Chart";
Node EssentialDiagram = new Node(Guid.NewGuid(), "EssentialDiagram");
EssentialDiagram.Shape = Shapes.Ellipse;
EssentialDiagram.Width = 100;
EssentialDiagram.Height = 50;
```

```

EssentialDiagram.OffsetX = 100;
EssentialDiagram.OffsetY = 300;
EssentialDiagram.Content = "Essential Diagram";
Node EssentialEdit = new Node(Guid.NewGuid(), "EssentialEdit");
EssentialEdit.Shape = Shapes.Ellipse;
EssentialEdit.Width = 100;
EssentialEdit.Height = 50;
EssentialEdit.OffsetX = 500;
EssentialEdit.OffsetY = 300;
EssentialEdit.Content = "Essential Edit";
// Adding the nodes to the Diagram Model.
diagramModel.Nodes.Add(EssentialWPF);
diagramModel.Nodes.Add(EssentialTools);
diagramModel.Nodes.Add(EssentialChart);
diagramModel.Nodes.Add(EssentialDiagram);
diagramModel.Nodes.Add(EssentialEdit);
//Creating the Connections between the nodes.
Connect(EssentialWPF, EssentialTools);
Connect(EssentialChart, EssentialWPF);
Connect(EssentialDiagram, EssentialWPF);
Connect(EssentialEdit, EssentialWPF);
//Creating connection and adding to the model.
void Connect(Node HeadNode, Node TailNode)
{
    LineConnector connection = new LineConnector();
    connection.ConnectorType = ConnectorType.Orthogonal;
    // Specify the TailNode node.
    connection.TailNode = TailNode;
    //Specifying the Head Node.
    connection.HeadNode = HeadNode;
    connection.TailDecoratorShape = DecoratorShape.Arrow;
    //Adding to the Diagram Model.
    diagramModel.Connections.Add(connection);
}

```

VB.NET

```

Private EssentialWPF As New Node(Guid.NewGuid(), "EssentialWPF")
EssentialWPF.Shape = Shapes.Ellipse
EssentialWPF.Width = 100
EssentialWPF.Height = 50
EssentialWPF.OffsetX = 300
EssentialWPF.OffsetY = 300
EssentialWPF.Content = "Essential WPF"
Dim EssentialTools As New Node(Guid.NewGuid(), "EssentialTools")
EssentialTools.Shape = Shapes.Ellipse
EssentialTools.Width = 100
EssentialTools.Height = 50
EssentialTools.OffsetX = 300
EssentialTools.OffsetY = 500
EssentialTools.Content = "Essential Tools"
Dim EssentialChart As New Node(Guid.NewGuid(), "EssentialChart")
EssentialChart.Shape = Shapes.Ellipse
EssentialChart.Width = 100
EssentialChart.Height = 50
EssentialChart.OffsetX = 300

```

```
EssentialChart.OffsetY = 100
EssentialChart.Content = "Essential Chart"
Dim EssentialDiagram As New Node(Guid.NewGuid(), "EssentialDiagram")
EssentialDiagram.Shape = Shapes.Ellipse
EssentialDiagram.Width = 100
EssentialDiagram.Height = 50
EssentialDiagram.OffsetX = 100
EssentialDiagram.OffsetY = 300
EssentialDiagram.Content = "Essential Diagram"
Dim EssentialEdit As New Node(Guid.NewGuid(), "EssentialEdit")
EssentialEdit.Shape = Shapes.Ellipse
EssentialEdit.Width = 100
EssentialEdit.Height = 50
EssentialEdit.OffsetX = 500
EssentialEdit.OffsetY = 300
EssentialEdit.Content = "Essential Edit"
'Adding the nodes to the Diagram Model.
diagramModel.Nodes.Add(EssentialWPF)
diagramModel.Nodes.Add(EssentialTools)
diagramModel.Nodes.Add(EssentialChart)
diagramModel.Nodes.Add(EssentialDiagram)
diagramModel.Nodes.Add(EssentialEdit)
'Creating the Connections between the nodes.
Connect(EssentialWPF, EssentialTools)
Connect(EssentialChart, EssentialWPF)
Connect(EssentialDiagram, EssentialWPF)
Connect(EssentialEdit, EssentialWPF)
'Creating connection and adding to the model.
void Connect(Node HeadNode, Node TailNode)
Dim connection As New LineConnector()
connection.ConnectorType = ConnectorType.Orthogonal
'Specify the TailNode node.
connection.TailNode = TailNode
'Specifying the Head Node.
connection.HeadNode = HeadNode
connection.TailDecoratorShape = DecoratorShape.Arrow
'Adding to the Diagram Model.
diagramModel.Connections.Add(connection)
```



Automatic Layout

Essential Diagram WPF allows you to specify automatic layouts the nodes. Following layout types are available:

- Directed-Tree layout
- Hierarchical-Tree layout
- Radial-Tree layout
- Table layout

Property	Description	Type of the property	Value it accepts
VerticalSpacing	Gets or sets the Vertical spacing between nodes.	CLR property	Double
HorizontalSpacing	Gets or sets the Horizontal spacing between nodes.	CLR property	Double
SpaceBetweenSubTrees	Gets or sets the space between sub trees.	CLR property	Double

Orientation	Gets or sets the orientation.	CLR property	TreeOrientation.LeftRightTreeOrientation.RightLeftTreeOrientation.TopBottomTreeOrientation.BottomTop
EnableCycleDetection	Gets or sets a value indicating whether Cycle detection is enabled or not.	DependencyProperty	Boolean true/ false
TableExpandMode	Gets or sets the table expand mode.	DependencyProperty	ExpandMode.HorizontalExpandMode.Vertical
RowCount	Gets or sets the Row Count for the table layout.	DependencyProperty	int
ColumnCount	Gets or sets the Column Count for the table layout.	DependencyProperty	int
EnableLayoutWithVariedSizes	Gets or sets a value indicating whether to enable the varied size algorithm. In case the Model consists of the nodes of different sizes, this property can be set to true. This will align the differently sized nodes with respect to the center.	DependencyProperty	Boolean (true/ false)
Bounds	Gets or sets the bounds value which	CLR property	Thickness

	specifies the position of the root node in case of tree layout.		
BowtieSubTreePlacement	Gets or sets a value from the BowtieSubTreePlacement.	Attached Dependency property	BowtieSubTreePlacement

Directed-Tree Layout

The Directed-Tree Layout automatically arranges nodes in a tree-like structure. This enables you to position nodes in a tree-like fashion without specifying the coordinate location for each node. However, it is necessary to specify a layout root for the tree layout. The Directed-Tree layout will position the nodes based on the layout root.

Hierarchical-Tree Layout

The Hierarchical Tree Layout arranges nodes in a tree-like structure, where the nodes in hierarchical layout may have multiple parents. As a result, there is no need to specify the layout root.

Radial-Tree Layout

The Radial-Tree Layout Manager arranges nodes in a circular layout and positions the root-node at the center of the graph and child-nodes in a circular fashion around the root. Sub-trees formed by the branching of the child-node are located radially around the child-node.

Table Layout

Table layout is a layout manager that arranges the nodes in rows and column basis. The number of nodes in each row and column can be specified and the layout will take place accordingly.

Directed – Tree Layout

The Directed-Tree layout allows you to arrange the nodes in a tree-like structure. This layout can be applied to any diagram that comprises a directed tree graph with unique root and child nodes. This makes creating diagrams easier because the node position is determined automatically, based on the connections. However, it is necessary to specify a layout root for the tree layout. The Directed-Tree layout will position the nodes based on the layout root.

Orientation

The Layout Manager lets you orient the tree in many directions and create sophisticated arrangements. The Orientation property of the Diagram model can be used to specify the tree orientation.

- TopBottom - Places the root node at the top and the child nodes are arranged below the root node.
- BottomTop - Places the root node at the bottom and the child nodes are arranged above the root node.
- LeftRight - Places the root node at the left and the child nodes are arranged on the right side of the root node.
- RightLeft - Places the root node at the right and the child nodes are arranged on the left side of the root node.

The Bounds property of the DiagramView class can be used to specify the position of the root node based on which the entire tree gets generated.

Property	Description	Type of the property	Value it accepts
VerticalSpacing	Gets or sets the Vertical spacing between nodes.	CLR property	Double
HorizontalSpacing	Gets or sets the Horizontal spacing between nodes.	CLR property	Double
SpaceBetweenSubTrees	Gets or sets the space between sub trees.	CLR property	Double
Orientation	Gets or sets the orientation.	CLR property	TreeOrientation.LeftRightTreeOrientation.RightLeftTreeOrientation.TopBottomTreeOrientation.BottomTop
Bounds	Gets or sets the bounds value which specifies the position of the root node in case of	CLR property	Thickness

	tree layout.		
--	--------------	--	--

Methods

Name	Parameters	Return Type	Description	Reference Links
RefreshLayout()	Null	void	Refresh the layout	N/A

The following code shows how the automatic layout can be generated.

1. The LayoutType should be set to DirectedTreeLayout in DiagramModel class.

```
<Window x:Class="RadialTreeLayout_2008.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Radial Tree Layout Demo" WindowState="Normal"
WindowStartupLocation="CenterScreen" Name="MainWindow"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
FontWeight="Bold" xmlns:local="clr-namespace:RadialTreeLayout_2008" Height="1000" Width="900">
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
<!-- Model to add nodes and connections-->
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel LayoutType="DirectedTreeLayout" Orientation="TopBottom"
x:Name="diagramModel">
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<!--View to display nodes and connections added through model.-->
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
</Window>
```

2. Then, the nodes are defined and the connections are made.

```
Style s = (Style)this.Resources["{x:Type Diagram:Node}"];
Node CEO = CreateNode("CEO", "CEO of the company", "Steve-CEO", s);
Node SLS = CreateNode("ManagerSLS", "Tom-ManagerSLS", "Sales Manager of the company ", s);
Node Marketing = CreateNode("ManagerMarketing", "Mary-MKT Manager", "Marketing Manager of the company", s);
Node DEV = CreateNode("ManagerDEV", "Jim-Manager DEV", "Development Manager of the company", s);
Node CSR1 = Node DEV = CreateNode("CSR1", "Kevin-CSR", "CSR of the company", s);
Node CSR2 = Node DEV = CreateNode("CSR2", "Peter-CSR", "CSR of the company", s);
//The layout happens with respect to the layout root.
diagramModel.LayoutRoot = CEO;
//Add the nodes to the model.
diagramModel.Nodes.Add(CEO);
diagramModel.Nodes.Add(Marketing);
diagramModel.Nodes.Add(SLS);
diagramModel.Nodes.Add(DEV);
diagramModel.Nodes.Add(CSR1);
diagramModel.Nodes.Add(CSR2);
//Creating the Connections between the nodes.
Connect(CEO, Marketing);
Connect(CEO, SLS);
Connect(CEO, DEV);
Connect(SLS, CSR1);
Connect(SLS, CSR2);
Node CreateNode(string Name, string Label, string ToolTip, Style s)
{
    Node NewNode = new Node(Guid.NewGuid(), Name);
    NewNode.Label = Label;
    NewNode.ToolTip = ToolTip;
    NewNode.CustomPathStyle = s;
    return NewNode;
}
//Creating connection and adding to the model.
void Connect(Node HeadNode, Node TailNode)
```

```
{
LineConnector connection = new LineConnector();
connection.ConnectorType = ConnectorType.Orthogonal;
// Specify the TailNode node
connection.TailNode = TailNode;
//Specifying the Head Node
connection.HeadNode = HeadNode;
connection.TailDecoratorShape = DecoratorShape.Arrow;
//Adding to the Diagram Model
diagramModel.Connections.Add(connection);
}

Private s As Style = CType(Me.Resources("{x:Type Diagram:Node}"), Style)
Private CEO As Node = CreateNode("CEO", "CEO of the company", "Steve-CEO", s)
Private SLS As Node = CreateNode("ManagerSLS", "Tom-ManagerSLS", "Sales Manager of the company", s)
Private Marketing As Node = CreateNode("ManagerMarketing", "Mary-MKT Manager", "Marketing Manager of the company", s)
Private DEV As Node = CreateNode("ManagerDEV", "Jim-Manager DEV", "Development Manager of the company", s)
Private CSR1 As Node = Node DEV = CreateNode("CSR1", "Kevin-CSR", "CSR of the company", s)
Private CSR2 As Node = Node DEV = CreateNode("CSR2", "Peter-CSR", "CSR of the company", s)

'The layout happens with respect to the layout root.
diagramModel.LayoutRoot = CEO

'Add the nodes to the model.
diagramModel.Nodes.Add(CEO)
diagramModel.Nodes.Add(Marketing)
diagramModel.Nodes.Add(SLS)
diagramModel.Nodes.Add(DEV)
diagramModel.Nodes.Add(CSR1)
diagramModel.Nodes.Add(CSR2)

'Creating the Connections between the nodes.
Connect(CEO, Marketing)
Connect(CEO, SLS)
Connect(CEO, DEV)
```

```
Connect(SLS, CSR1)
```

```
Connect(SLS, CSR2)
```

```
Node CreateNode(String Name, String Label, String ToolTip, Style s)
```

```
Dim NewNode As New Node(Guid.NewGuid(), Name)
```

```
NewNode.Label = Label
```

```
NewNode.ToolTip = ToolTip
```

```
NewNode.CustomPathStyle = s
```

```
Return NewNode
```

```
'Creating connection and adding to the model.
```

```
void Connect(Node HeadNode, Node TailNode)
```

```
Dim connection As New LineConnector()
```

```
connection.ConnectorType = ConnectorType.Orthogonal
```

```
'Specify the TailNode node
```

```
connection.TailNode = TailNode
```

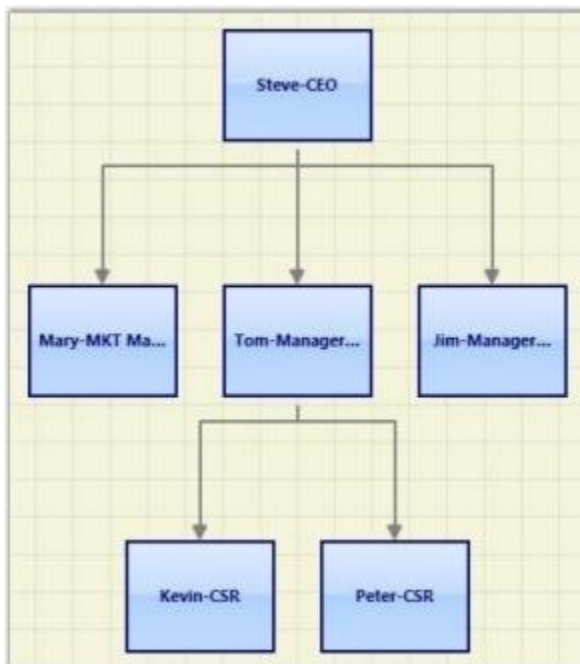
```
'Specifying the Head Node
```

```
connection.HeadNode = HeadNode
```

```
connection.TailDecoratorShape = DecoratorShape.Arrow
```

```
'Adding to the Diagram Model
```

```
diagramModel.Connections.Add(connection)
```



Layout Spacing

Spacing between the nodes with respect different levels and siblings are adjustable; this will be helpful to adjust the distance between nodes, so that it will meet many business needs. As this is a general topic between all layouts, detailed explanation about this can be found in Layout Spacing under DiagramModel.

Refresh Layout

When there are changes in content of the page link new nodes and connectors added, the layout has to be refreshed to get the page's content aligned again to give space for new contents. To refresh the layout please follow the following code snippet.

C#

```
DirectedTreeLayout tree = new DirectedTreeLayout(diagramModel, diagramView);
tree.RefreshLayout();
```

VB.NET

```
Dim tree As New DirectedTreeLayout(diagramModel, diagramView)
tree.RefreshLayout()
```

Here diagramModel and diagramView is an instance of DiagramModel and DiagramView respectively.

[Hierarchical – Tree Layout](#)

The Hierarchical Tree Layout arranges nodes in a tree-like structure, where the nodes in hierarchical layout may have multiple parents. As a result, there is no need to specify the layout root.

Orientation

The Layout Manager lets you orient the hierarchical tree in many directions. The Orientation property of Diagram Model can be used to specify the tree orientation.

- TopBottom - Places the root node at the top and the child nodes are arranged below the root node.
- BottomTop - Places the root node at the bottom and the child nodes are arranged above the root node.
- LeftRight - Places the root node at the left and the child nodes are arranged on the right side of the root node.
- RightLeft - Places the root node at the right and the child nodes are arranged on the left side of the root node.

Property	Description	Type of the property	Value it accepts
VerticalSpacing	Gets or sets the vertical spacing between	CLR property	Double

	en nodes.		
HorizontalSpacing	Gets or sets the Horizontal spacing between nodes.	CLR property	Double
SpaceBetweenSubTrees	Gets or sets the space between sub trees.	CLR property	Double
Orientation	Gets or sets the orientation.	CLR property	TreeOrientation.LeftRightTreeOrientation.RightLeftTreeOrientation.TopBottomTreeOrientation.BottomTop
EnableCycleDetection	Gets or sets a value indicating whether Cycle detection is enabled or not.	Dependency Property	Boolean (true/ false)
Bounds	Gets or sets the bounds value which specifi	CLR property	Thickness

	es the position of the root node in case of tree layout.		
--	--	--	--

Name	Parameters	Return Type	Description	Reference Links
RefreshLayout()	Null	void	Refresh the layout	N/A

The Bounds property of the DiagramView class can be used to specify the position of the root node, based on which the entire tree gets generated.

The following code example specifies how the Hierarchical-tree layout can be specified.

1. The LayoutType should be set to HierarchicalTreeLayout in DiagramModel class.

```
<Window x:Class="RadialTreeLayout_2008.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Radial Tree Layout Demo" WindowState="Normal"
WindowStartupLocation="CenterScreen" Name="MainWindow"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
FontWeight="Bold" xmlns:local="clr-namespace:RadialTreeLayout_2008" Height="1000" Width="900">
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
<!-- Model to add nodes and connections-->      <syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel LayoutType="HierarchicalTreeLayout" Orientation="TopBottom"
x:Name="diagramModel">      </syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<!--View to display nodes and connections added through model.-->
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Bounds="0,0,700,750"
Background="White" Name="diagramView" >
</syncfusion:DiagramView>
```



```
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
</Window>
```

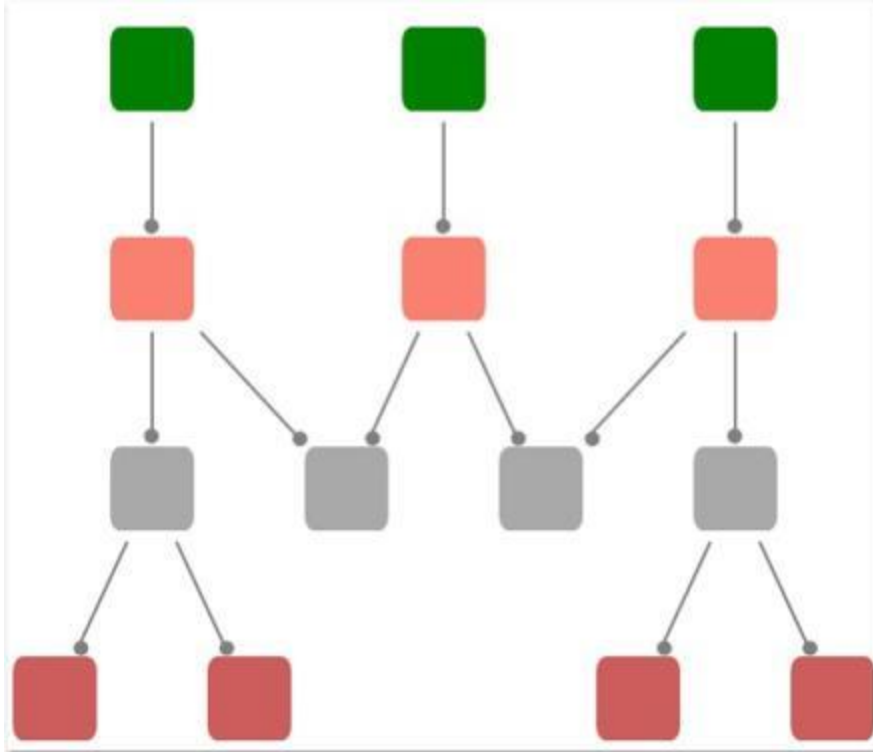
2. Then the nodes can be added and the connections can be specified as follows:

C#

```
Node n1 = new Node(Guid.NewGuid(), "n1");
n1.Level = 0;
Node n2 = new Node(Guid.NewGuid(), "n2");
n2.Level = 0;
Node n3 = new Node(Guid.NewGuid(), "n3");
n3.Level = 0;
Node n4 = new Node(Guid.NewGuid(), "n4");
n4.Level = 1;
Node n5 = new Node(Guid.NewGuid(), "n5");
n5.Level = 2;
Node n6 = new Node(Guid.NewGuid(), "n6");
n6.Level = 1;
Node n7 = new Node(Guid.NewGuid(), "n7");
n7.Level = 1;
Node n8 = new Node(Guid.NewGuid(), "n8");
n8.Level = 3;
Node n9 = new Node(Guid.NewGuid(), "n9");
n9.Level = 3;
Node n10 = new Node(Guid.NewGuid(), "n10");
n10.Level = 2;
Node n11 = new Node(Guid.NewGuid(), "n11");
n11.Level = 2;
Node n12 = new Node(Guid.NewGuid(), "n12");
n12.Level = 2;
Node n13 = new Node(Guid.NewGuid(), "n13");
n13.Level = 3;
Node n14 = new Node(Guid.NewGuid(), "n14");
n14.Level = 3;
//Adding nodes to the diagram Model.
diagramModel.Nodes.Add(n1);
diagramModel.Nodes.Add(n2);
diagramModel.Nodes.Add(n3);
diagramModel.Nodes.Add(n4);
diagramModel.Nodes.Add(n5);
diagramModel.Nodes.Add(n6);
diagramModel.Nodes.Add(n7);
diagramModel.Nodes.Add(n8);
diagramModel.Nodes.Add(n9);
diagramModel.Nodes.Add(n10);
diagramModel.Nodes.Add(n11);
diagramModel.Nodes.Add(n12);
diagramModel.Nodes.Add(n13);
diagramModel.Nodes.Add(n14);
//Creating connections between nodes.
Connect(n1, n4);
Connect(n6, n11);
Connect(n2, n6);
```

```
Connect(n6, n10);
Connect(n3, n7);
Connect(n5, n8);
Connect(n5, n9);
Connect(n4, n5);
Connect(n7, n10);
Connect(n4, n11);
Connect(n7, n12);
Connect(n12, n13);
Connect(n12, n14);
}
//Creating connection and adding to the model.
void Connect(Node HeadNode, Node TailNode)
{
    LineConnector connection = new LineConnector();
    connection.ConnectorType = ConnectorType.Straight;
    // Specify the TailNode node.
    connection.TailNode = TailNode;
    //Specifying the Head Node.
    connection.HeadNode = HeadNode;
    connection.TailDecoratorShape = DecoratorShape.Circle;
    //Adding to the Diagram Model.
    diagramModel.Connections.Add(connection);
}
Private n1 As New Node(Guid.NewGuid(), "n1")
n1.Level = 0
Dim n2 As New Node(Guid.NewGuid(), "n2")
n2.Level = 0
Dim n3 As New Node(Guid.NewGuid(), "n3")
n3.Level = 0
Dim n4 As New Node(Guid.NewGuid(), "n4")
n4.Level = 1
Dim n5 As New Node(Guid.NewGuid(), "n5")
n5.Level = 2
Dim n6 As New Node(Guid.NewGuid(), "n6")
n6.Level = 1
Dim n7 As New Node(Guid.NewGuid(), "n7")
n7.Level = 1
Dim n8 As New Node(Guid.NewGuid(), "n8")
n8.Level = 3
Dim n9 As New Node(Guid.NewGuid(), "n9")
n9.Level = 3
Dim n10 As New Node(Guid.NewGuid(), "n10")
n10.Level = 2
Dim n11 As New Node(Guid.NewGuid(), "n11")
n11.Level = 2
Dim n12 As New Node(Guid.NewGuid(), "n12")
n12.Level = 2
Dim n13 As New Node(Guid.NewGuid(), "n13")
n13.Level = 3
Dim n14 As New Node(Guid.NewGuid(), "n14")
n14.Level = 3
'Adding nodes to the diagram Model.
diagramModel.Nodes.Add(n1)
diagramModel.Nodes.Add(n2)
diagramModel.Nodes.Add(n3)
diagramModel.Nodes.Add(n4)
```

```
diagramModel.Nodes.Add(n5)
diagramModel.Nodes.Add(n6)
diagramModel.Nodes.Add(n7)
diagramModel.Nodes.Add(n8)
diagramModel.Nodes.Add(n9)
diagramModel.Nodes.Add(n10)
diagramModel.Nodes.Add(n11)
diagramModel.Nodes.Add(n12)
diagramModel.Nodes.Add(n13)
diagramModel.Nodes.Add(n14)
'Creating connections between nodes.
Connect(n1, n4)
Connect(n6, n11)
Connect(n2, n6)
Connect(n6, n10)
Connect(n3, n7)
Connect(n5, n8)
Connect(n5, n9)
Connect(n4, n5)
Connect(n7, n10)
Connect(n4, n11)
Connect(n7, n12)
Connect(n12, n13)
Connect(n12, n14)
'Creating connection and adding to the model.
void Connect(Node HeadNode, Node TailNode)
Dim connection As New LineConnector()
connection.ConnectorType = ConnectorType.Straight
'Specify the TailNode node.
connection.TailNode = TailNode
'Specifying the Head Node.
connection.HeadNode = HeadNode
connection.TailDecoratorShape = DecoratorShape.Circle
'Adding to the Diagram Model.
diagramModel.Connections.Add(connection)
```



Layout Spacing

Spacing between the nodes with respect different levels and siblings are adjustable; this will be helpful to adjust the distance between nodes, so that it will meet many business needs. As this is a general topic between all layouts, detailed explanation about this can be found in Layout Spacing under DiagramModel.

Cyclic path in Hierarchical-Tree Layout

Unlike Directed-Tree layout, hierarchical tree layout supports nodes with multiple parents, this will cause a cyclic path in the layout, a detailed explanation about this scenario can be found in Cyclic path in Hierarchical-Tree Layout under DiagramModel.

Refresh Layout

When there are changes in content of the page link new nodes and connectors added, the layout has to be refreshed to get the page's content aligned again to give space for new contents. To refresh the layout please follow the following code snippet.

C#

```
HierarchicalTreeLayout tree = new HierarchicalTreeLayout(diagramModel,
diagramView);
tree.RefreshLayout();
```

C#

```
Dim tree As New HierarchicalTreeLayout(diagramModel, diagramView)
tree.RefreshLayout()
```

Here diagramModel and diagramView is an instance of DiagramModel and DiagramView respectively.

Layout Spacing Refer Concepts and Features -> Diagram Model -> Layout Spacing

Tree Orientation Refer Concepts and Features -> Diagram Model -> Tree Orientation

Cyclic path in Hierarchical-Tree Layout: Refer Concepts and Features -> Diagram Model -> Cyclic path in Hierarchical – TreeLayout.

Radial – Tree Layout

The Radial-TreeLayout is a specialization of the Directed Tree Layout Manager that employs a circular layout algorithm for locating the diagram nodes. The Radial-Tree Layout arranges nodes in a circular layout, positioning the root node at the center of the graph and the child nodes in a circular fashion around the root. Sub-trees formed by the branching of child nodes are located radially around the child nodes. This arrangement results in an ever-expanding concentric arrangement with radial proximity to the root node indicating the node level in the hierarchy. However, it is necessary to specify a layout root for the tree layout. The Radial-Tree layout will position the nodes based on the layout root.

The Bounds property of the DiagramView class can be used to specify the position of the root node, based on which the entire tree gets generated.

Property	Description	Type of the property	Value it accepts
VerticalSpacing	Gets or sets the Vertical spacing between nodes.	CLR property	Double
HorizontalSpacing	Gets or sets the Horizontal spacing between nodes.	CLR property	Double
Bounds	Gets or sets the bounds value which specifies the position of the root node in case of tree layout.	CLR property	Thickness

Name	Parameters	Return Type	Description	Reference Links
RefreshLayout()	Null	void	Refresh the layout	N/A

1. The LayoutType should be set to RadialTreeLayout in DiagramModel class.

XML

```
<Window x:Class="RadialTreeLayout_2008.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Radial Tree Layout Demo" WindowState="Normal"
WindowStartupLocation="CenterScreen" Name="mainwindow"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
FontWeight="Bold" xmlns:local="clr-namespace:RadialTreeLayout_2008"
Height="1000" Width="900">
```

```

<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel x:Name="diagramModel"
      LayoutType="RadialTreeLayout">
    </syncfusion:DiagramModel>
  </syncfusion:DiagramControl.Model>
  <!--View to display nodes and connections added through model.-->
  <syncfusion:DiagramControl.View>
    <syncfusion:DiagramView Name="diagramView">
  </syncfusion:DiagramView>
  </syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
</Window>

```

2.Then the nodes can be added and the connections can be specified as follows:

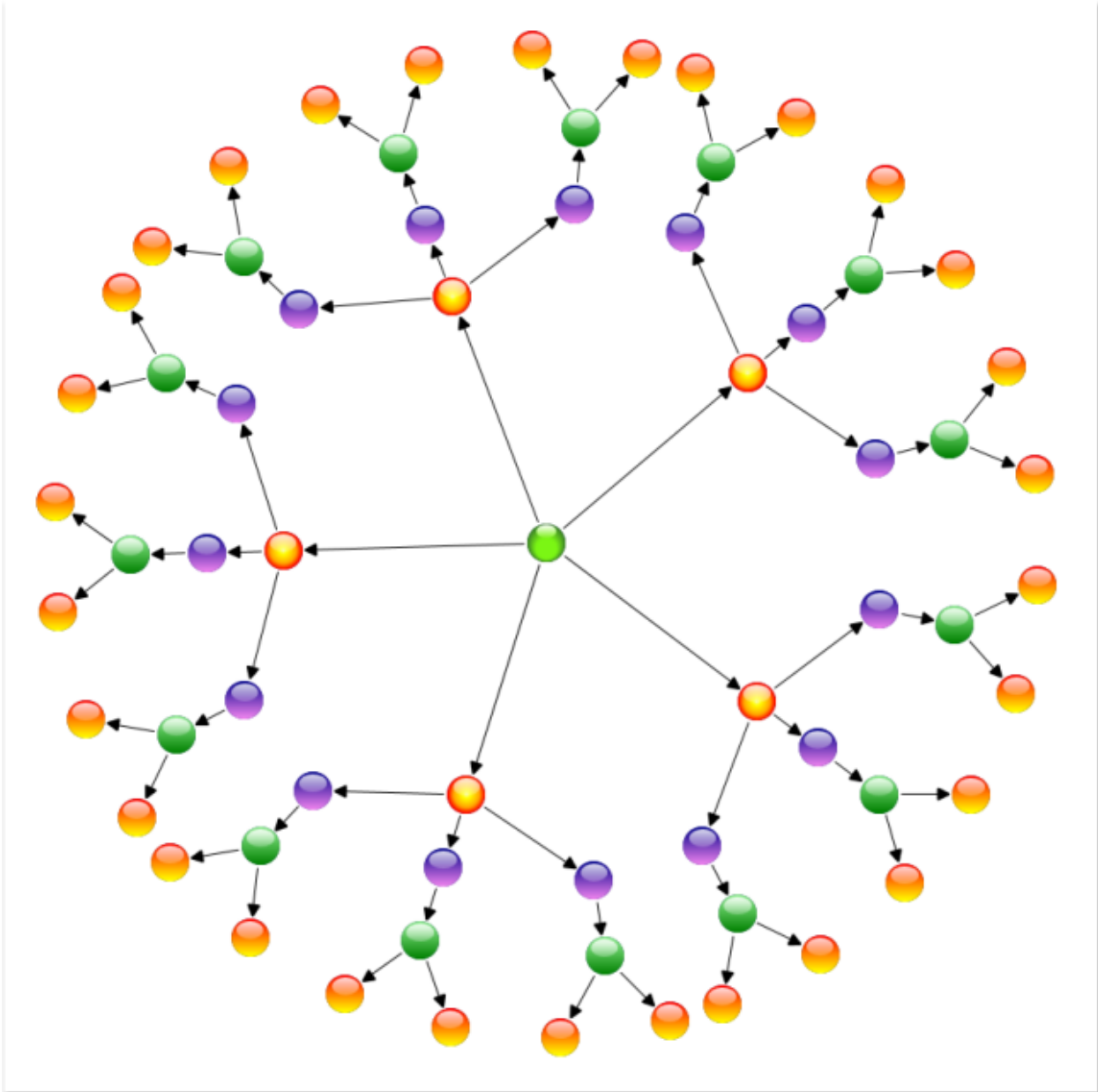
C#

```

//Define Spacings.
diagramModel.HorizontalSpacing = 10;
diagramModel.VerticalSpacing = 30;
Node n1 = new Node(Guid.NewGuid(), "n1");
n1.Level = 0;
Node n2 = new Node(Guid.NewGuid(), "n2");
n2.Level = 1;
Node n3 = new Node(Guid.NewGuid(), "n3");
n3.Level = 1;
Node n4 = new Node(Guid.NewGuid(), "n4");
n4.Level = 1;
Node n5 = new Node(Guid.NewGuid(), "n5");
n5.Level = 1;
Node n6 = new Node(Guid.NewGuid(), "n6");
n6.Level = 1;
//Adding nodes to the diagram Model.
diagramModel.Nodes.Add(n1);
diagramModel.Nodes.Add(n2);
diagramModel.Nodes.Add(n3);
diagramModel.Nodes.Add(n4);
diagramModel.Nodes.Add(n5);
diagramModel.Nodes.Add(n6);
//Creating connections between nodes.
Connect(n1, n2);
Connect(n1, n3);
Connect(n1, n4);
Connect(n1, n5);
Connect(n1, n6);
diagramModel.LayoutRoot = n1;
//Creating connection and adding to the model.
void Connect(Node HeadNode, Node TailNode)
{
  LineConnector connection = new LineConnector();
  connection.ConnectorType = ConnectorType.Straight;
  // Specify the TailNode node.
  connection.TailNode = TailNode;
}

```

```
//Specifying the Head Node.
connection.HeadNode = HeadNode;
connection.TailDecoratorShape = DecoratorShape.Circle;
//Adding to the Diagram Model.
diagramModel.Connections.Add(connection);
}
'Define Spacings.
diagramModel.HorizontalSpacing = 10
diagramModel.VerticalSpacing = 30
Dim n1 As New Node(Guid.NewGuid(), "n1")
n1.Level = 0
Dim n2 As New Node(Guid.NewGuid(), "n2")
n2.Level = 1
Dim n3 As New Node(Guid.NewGuid(), "n3")
n3.Level = 1
Dim n4 As New Node(Guid.NewGuid(), "n4")
n4.Level = 1
Dim n5 As New Node(Guid.NewGuid(), "n5")
n5.Level = 1
Dim n6 As New Node(Guid.NewGuid(), "n6")
n6.Level = 1
'Adding nodes to the diagram Model.
diagramModel.Nodes.Add(n1)
diagramModel.Nodes.Add(n2)
diagramModel.Nodes.Add(n3)
diagramModel.Nodes.Add(n4)
diagramModel.Nodes.Add(n5)
diagramModel.Nodes.Add(n6)
'Creating connections between nodes.
Connect(n1, n2)
Connect(n1, n3)
Connect(n1, n4)
Connect(n1, n5)
Connect(n1, n6)
diagramModel.LayoutRoot = n1
'Creating connection and adding to the model.
void Connect(Node HeadNode, Node TailNode)
Dim connection As New LineConnector()
connection.ConnectorType = ConnectorType.Straight
'Specify the TailNode node.
connection.TailNode = TailNode
'Specifying the Head Node.
connection.HeadNode = HeadNode
connection.TailDecoratorShape = DecoratorShape.Circle
'Adding to the Diagram Model.
diagramModel.Connections.Add(connection)
```



Layout Spacing

Spacing between the nodes with respect different levels and siblings are adjustable; this will be helpful to adjust the distance between nodes, so that it will meet many business needs. As this is a general topic between all layouts, detailed explanation about this can be found in Layout Spacing under DiagramModel.

Refresh Layout

When there are changes in content of the page link new nodes and connectors added, the layout has to be refreshed to get the page's content aligned again to give space for new contents. To refresh the layout please follow the following code snippet.

C#


```
RadialTreeLayout tree = new RadialTreeLayout(diagramModel, diagramView);
tree.RefreshLayout();
```

VB.NET

```
Dim tree As New RadialTreeLayout(diagramModel, diagramView)
tree.RefreshLayout()
```

Here diagramModel and diagramView is an instance of DiagramModel and DiagramView respectively.

Table Layout

Table layout arranges the nodes in a tabular structure based on specified intervals between them. The number of nodes in each row and column can be specified and the layout will take place accordingly. The nodes are assigned rows and columns based on the order in which they are added to the model and based on the maximum nodes allowed in that row and column. This layout enables to layout nodes automatically without the need to specify offset positions for each node.

Property	Description	Type of the property	Value it accepts
VerticalSpacing	Gets or sets the Vertical spacing between nodes.	CLR property	Double
HorizontalSpacing	Gets or sets the Horizontal spacing between nodes.	CLR property	Double
Orientation	Gets or sets the orientation.	CLR property	TreeOrientation.LeftRightTreeOrientation.RightLeftTreeOrientation.TopBottomTreeOrientation.BottomTop
TableExpandMode	Gets or sets	Dependency Property	ExpandMode.HorizontalExpandMode.Vertical

	the table expanded mode.		
RowCount	Gets or sets the Row Count for the table layout.	Dependency Property	int
ColumnCount	Gets or sets the Column Count for the table layout.	Dependency Property	int
EnableLayoutWithVariedSizes	Gets or sets a value indicating whether to enable the varied size algorithm. In case the Model consists of the nodes of different sizes,	Dependency Property	Boolean (true/ false)

	this property can be set to true. This will align the differently sized nodes with respect to the center.		
Bounds	Gets or sets the bounds value which specifies the position of the root node in case of tree layout.	CLR property	Thickness

Name	Parameters	Return Type	Description	Reference Links
RefreshLayout()	Null	void	Refresh the layout	N/A

The Layout Manager lets you orient the table in two directions, Horizontal and Vertical. The TableExpandMode property of Diagram Model is used to specify the orientation.

Horizontal:

When set to Horizontal, the RowCount is automatically calculated based on the number of nodes. The ColumnCount must be specified and the nodes will be arranged in the specified number of columns. When the maximum column count is reached, it starts placing the nodes in a new row.

Vertical:

When set to Vertical, the ColumnCount is automatically calculated based on the number of nodes. The RowCount must be specified and the nodes will be arranged in the specified number of rows. When the maximum row count is reached, it starts placing the nodes in a new column.

The Bounds property of the DiagramView class can be used to specify the position of the first node.

XML

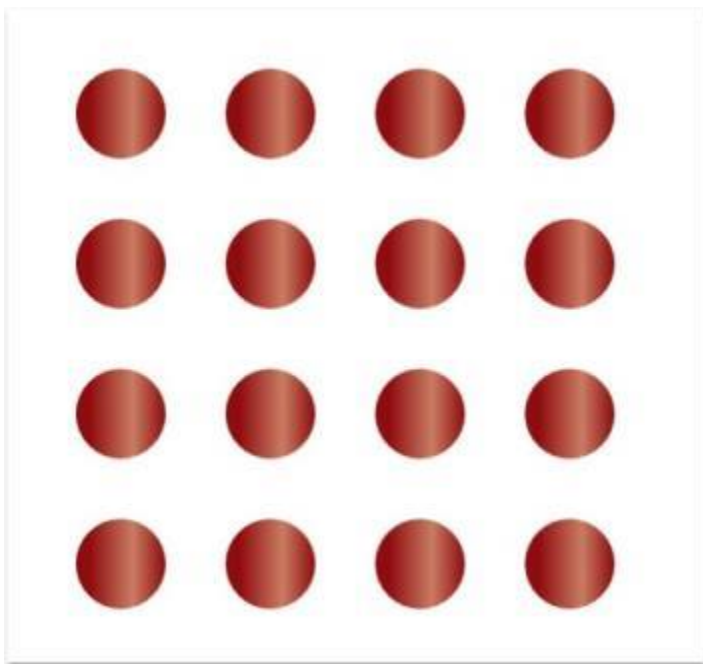
```
<Window x:Class="RadialTreeLayout_2008.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Radial Tree Layout Demo" WindowState="Normal"
WindowStartupLocation="CenterScreen" Name="mainwindow"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
FontWeight="Bold" xmlns:local="clr-namespace:RadialTreeLayout_2008"
Height="1000" Width="900">
<syncfusion:DiagramControl IsSymbolPaletteEnabled="False"
Name="diagramControl">
<!-- Model to add nodes and connections-->
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel
LayoutType="TableLayout"
EnableLayoutWithVariedSizes="False"
TableExpandMode="Horizontal"
HorizontalSpacing="50"
VerticalSpacing="50"
RowCount="4"
ColumnCount="4"
x:Name="diagramModel">
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<!--View to display nodes and connections added through model.-->
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Bounds="0,0,200,200" Name="diagramView"/>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl></Window>
```

C#

```
DiagramControl dc = new DiagramControl();
DiagramModel diagramModel = new DiagramModel();
dc.Model = diagramModel;
diagramModel.LayoutType = LayoutType.TableLayout;
diagramModel.TableExpandMode = ExpandMode.Horizontal;
diagramModel.EnableLayoutWithVariedSizes = false;
diagramModel.HorizontalSpacing = 50;
diagramModel.VerticalSpacing = 50;
diagramModel.RowCount = 10;
diagramModel.ColumnCount = 3;
```

VB.NET

```
Dim dc As New DiagramControl()  
Dim diagramModel As New DiagramModel()  
dc.Model = diagramModel  
diagramModel.LayoutType = LayoutType.TableLayout  
diagramModel.TableExpandMode = ExpandMode.Horizontal  
diagramModel.EnableLayoutWithVariedSizes = False  
diagramModel.HorizontalSpacing = 50  
diagramModel.VerticalSpacing = 50  
diagramModel.RowCount = 10  
diagramModel.ColumnCount = 3
```

**Layout Spacing**

Spacing between the nodes with respect different levels and siblings are adjustable; this will be helpful to adjust the distance between nodes, so that it will meet many business needs. As this is a general topic between all layouts, detailed explanation about this can be found in Layout Spacing under DiagramModel.

Refresh Layout

When there are changes in content of the page like new nodes or connectors added, the layout has to be refreshed to get the page's content aligned again to give space for new contents. To refresh the layout please follow the following code snippet.

C#

```
TableLayout tree = new TableLayout(diagramModel, diagramView);  
tree.RefreshLayout();
```

VB.NET

```
Dim tree As New TableLayout(diagramModel, diagramView)
```

```
tree.RefreshLayout()
```

Here diagramModel and diagramView is an instance of DiagramModel and DiagramView respectively.

There are some more customization supported for Table Layout, please refer the links following 'see also' topic.

See Also

Layout Spacing Refer Concepts and Features -> Diagram Model -> Layout Spacing

TableExpandMode Refer Concepts and Features -> Diagram Model -> TableExpandMode

RowCount and ColumnCount Refer Concepts and Features -> Diagram Model -> RowCount and ColumnCount

Enable TableLayout with varied Node sizes Refer Concepts and Features -> Diagram Model -> Enable TableLayout with varied Node size

BowTie Layout

The BowTie Diagram is a graphical representation of the risk assessment process. This can be used for assessing all type of risks.

Properties

Property	Description	Type	Data Type	Reference links
VerticalSpacing	Gets or sets the Vertical spacing between nodes.	CLR Property	Double	NA
HorizontalSpacing	Gets or sets the Horizontal spacing between nodes.	CLR Property	Double	NA
SpaceBetweenSubTrees	Gets or sets the space between sub trees.	CLR Property	Double	NA
Bounds	Gets or sets the bounds value which specifies the position of the root node in case of tree layout.	CLR Property	Thickness	NA
BowtieSubTreePlacement	Gets or sets a value from the BowtieSubTreePlacement.	Attached Dependency property	BowtieSubTreePlacement	NA

The following code illustrates how to generate the BowTie layout:

1. The LayoutType should be set to BowtieLayout in DiagramModel class.

HTML

```
<!--Diagram Control-->
```

```

<syncfusion:DiagramControl Name="diagramControl" Grid.Row="1">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel" LayoutType="BowtieLayout"/>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView" Bounds="0, 0, 1400, 700"/>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>

```

2. Then, the nodes are defined and the connections are made.

C#

```

//Tree spacing properties.
diagramModel.VerticalSpacing = 35;
diagramModel.HorizontalSpacing = 30;
diagramModel.SpaceBetweenSubTrees = 150;
//Defines the nodes and adds it to the model.
Node Root = AddNode("R", "Systems\nSecurity", Brushes.Red, Brushes.Black, 4,
Shapes.Ellipse);
this.diagramModel.LayoutRoot = Root;
diagramModel.LayoutType = LayoutType.BowtieLayout;
//creating the Left Tree.
createLeftNodes(Root, BowtieSubTreePlacement.Left);
//creating the Right Tree.
createRightNodes(Root, BowtieSubTreePlacement.Right);
//setting the Root Node.
this.diagramModel.LayoutRoot = Root;
public void createLeftNodes(Node Root, BowtieSubTreePlacement place)
{
//Defines the nodes.
Node n1 = AddNode("n1", "Hacking", Brushes.Blue, Brushes.Black, 1,
Shapes.Ellipse);
Node n2 = AddNode("n2", "Firewall", Brushes.Yellow, Brushes.Black, 1,
Shapes.RoundedRectangle);
Node n3 = AddNode("n3", "Identification", Brushes.Yellow, Brushes.Black, 1,
Shapes.RoundedRectangle);
Node n4 = AddNode("n4", "Authorization", Brushes.Yellow, Brushes.Black, 2,
Shapes.RoundedRectangle);
Node n5 = AddNode("n5", "Theft\n of\nInformation", Brushes.Blue,
Brushes.Black, 3, Shapes.Ellipse);
Node n6 = AddNode("n6", "Firewall", Brushes.Yellow, Brushes.Black, 3,
Shapes.RoundedRectangle);
Node n7 = AddNode("n7", "Network Access", Brushes.Yellow, Brushes.Black, 2,
Shapes.RoundedRectangle);
Node n8 = AddNode("n8", "Data Access", Brushes.Yellow, Brushes.Black, 2,
Shapes.RoundedRectangle);
DiagramControl.SetBowtieSubTreePlacement(n4, place);
DiagramControl.SetBowtieSubTreePlacement(n8, place);
//Creating connections between the nodes
Connect(n1, n2);
Connect(n2, n3);
Connect(n3, n4);
Connect(n4, Root);
Connect(n5, n6);

```

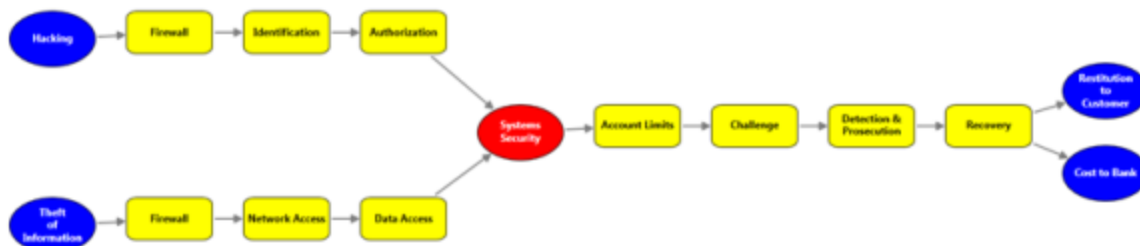
```

Connect(n6, n7);
Connect(n7, n8);
Connect(n8, Root);
}
//Defines the nodes.
public void createRightNodes(Node Root, BowtieSubTreePlacement place)
{
    //Defining the nodes.
    Node n1 = AddNode("n1", "Account Limits", Brushes.Yellow, Brushes.Black, 1,
        Shapes.RoundedRectangle);
    Node n2 = AddNode("n2", "Challenge", Brushes.Yellow, Brushes.Black, 1,
        Shapes.RoundedRectangle);
    Node n3 = AddNode("n3", "Detection & Prosecution", Brushes.Yellow,
        Brushes.Black, 2, Shapes.RoundedRectangle);
    Node n4 = AddNode("n4", "Recovery", Brushes.Yellow, Brushes.Black, 3,
        Shapes.RoundedRectangle);
    Node n5 = AddNode("n5", "Restitution\n to\nCustomer", Brushes.Blue,
        Brushes.Black, 3, Shapes.Ellipse);
    Node n6 = AddNode("n6", "Cost to Bank", Brushes.Blue, Brushes.Black, 3,
        Shapes.Ellipse);
    DiagramControl.SetBowtieSubTreePlacement(n1, place);
    //Creating connections between the nodes.
    Connect(Root, n1);
    Connect(n1, n2);
    Connect(n2, n3);
    Connect(n3, n4);
    Connect(n4, n5);
    Connect(n4, n6);
}
'Tree spacing properties.
diagramModel.VerticalSpacing = 35
diagramModel.HorizontalSpacing = 30
diagramModel.SpaceBetweenSubTrees = 150
'Defines the nodes and adds it to the model.
Dim Root As Node = AddNode("R", "Systems" & Constants.vbLf & "Security",
    Brushes.Red, Brushes.Black, 4, Shapes.Ellipse)
Me.diagramModel.LayoutRoot = Root
diagramModel.LayoutType = LayoutType.BowtieLayout
'creating the Left Side Tree
createLeftNodes(Root, BowtieSubTreePlacement.Left)
'creating the Right Tree
createRightNodes(Root, BowtieSubTreePlacement.Right)
'setting the Root Node
Me.diagramModel.LayoutRoot = Root
public void createLeftNodes(Node Root, BowtieSubTreePlacement place)
Defines nodes.
Dim n1 As Node = AddNode("n1", "Hacking", Brushes.Blue, Brushes.Black, 1,
    Shapes.Ellipse)
Dim n2 As Node = AddNode("n2", "Firewall", Brushes.Yellow, Brushes.Black, 1,
    Shapes.RoundedRectangle)
Dim n3 As Node = AddNode("n3", "Identification", Brushes.Yellow,
    Brushes.Black, 1, Shapes.RoundedRectangle)
Dim n4 As Node = AddNode("n4", "Authorization", Brushes.Yellow,
    Brushes.Black, 2, Shapes.RoundedRectangle)
Dim n5 As Node = AddNode("n5", "Theft" & Constants.vbLf & "of" &
    Constants.vbLf & "Information", Brushes.Blue, Brushes.Black, 3,
    Shapes.Ellipse)

```



```
Dim n6 As Node = AddNode("n6", "Firewall", Brushes.Yellow, Brushes.Black, 3,
Shapes.RoundedRectangle)
Dim n7 As Node = AddNode("n7", "Network Access", Brushes.Yellow,
Brushes.Black, 2, Shapes.RoundedRectangle)
Dim n8 As Node = AddNode("n8", "Data Access", Brushes.Yellow, Brushes.Black,
2, Shapes.RoundedRectangle)
DiagramControl.SetBowtieSubTreePlacement(n4, place)
DiagramControl.SetBowtieSubTreePlacement(n8, place)
'Creating connections between the nodes
Connect(n1, n2)
Connect(n2, n3)
Connect(n3, n4)
Connect(n4, Root)
Connect(n5, n6)
Connect(n6, n7)
Connect(n7, n8)
Connect(n8, Root)
'Defines the nodes.
public void createRightNodes(Node Root, BowtieSubTreePlacement place)
'Defining the nodes.
Dim n1 As Node = AddNode("n1", "Account Limits", Brushes.Yellow,
Brushes.Black, 1, Shapes.RoundedRectangle)
Dim n2 As Node = AddNode("n2", "Challenge", Brushes.Yellow, Brushes.Black,
1, Shapes.RoundedRectangle)
Dim n3 As Node = AddNode("n3", "Detection & Prosecution", Brushes.Yellow,
Brushes.Black, 2, Shapes.RoundedRectangle)
Dim n4 As Node = AddNode("n4", "Recovery", Brushes.Yellow, Brushes.Black, 3,
Shapes.RoundedRectangle)
Dim n5 As Node = AddNode("n5", "Restitution" & Constants.vbLf & "to" &
Constants.vbLf & "Customer", Brushes.Blue, Brushes.Black, 3, Shapes.Ellipse)
Dim n6 As Node = AddNode("n6", "Cost to Bank", Brushes.Blue, Brushes.Black,
3, Shapes.Ellipse)
DiagramControl.SetBowtieSubTreePlacement(n1, place)
'Creating connections between the nodes.
Connect(Root, n1)
Connect(n1, n2)
Connect(n2, n3)
Connect(n3, n4)
Connect(n4, n5)
Connect(n4, n6)
```



Layout Spacing

Spacing between the nodes with respect to different levels and siblings are adjustable. This will be helpful to adjust the distance between nodes, so that it will meet many business needs. As this is a general topic between all layouts, detailed explanation about this can be found in [Layout Spacing](#) under DiagramModel.

Refresh Layout

When there are changes in the content of the page-- such as adding new nodes and connectors to the layout, the page has to be refreshed to get the content aligned again. Only then it will give space for new contents. Refresh the layout as given in the following code example:

C#

```
BowtieLayout tree = new BowtieLayout (diagramModel, diagramView);
tree.RefreshLayout();
```

VB.NET

```
Dim tree As New BowtieLayout (diagramModel, diagramView)
tree.RefreshLayout()
```

Table Layout for Selected Nodes

This feature enables you to apply the table layout on selected nodes instead of applying it to the entire diagram. This arranges selected nodes or a given node collection in a tabular structure based on specified intervals between them. The number of nodes in each row and column can be specified and the layout will be applied accordingly.

Use Case Scenarios

- Users can easily make the layout with a specific collection of nodes called ordered nodes.
- Users can easily position the layout.
- Users can easily align the layout by using the layout alignment properties.
- Users can set a rectangle boundary around nodes by using the Layout Bounds property.

Properties

Property	Description	Type	Data Type
OrderedNodes	This property is used to get or set the Collection of Nodes for table layout.	Dependency property	List<IShape>

Sample Link

To view a sample of this feature:

1. Open Dashboard.
2. Click User Interface > WPF.
3. Click Run Samples.
4. Navigate to Diagram > Automatic Layout > Table Layout.

Adding Table Layout for selected Nodes

To apply a table layout to the selected nodes, assign the selected nodes to the OrderNodes property of the DiagramModel. You can also assign your own collection of IShape to the *OrderNodes* property. Then create an instance of the TableLayout and call the RefreshLayout method for this instance.

The following code illustrates this:

C#

```
// Assigning selected node to the OrderedNodes property.
diagramModel.OrderedNodes=
diagramView.SelectionList.OfType<IShape>().ToList();
// Create an instance of TableLayout and refresh it.
TableLayout table = new TableLayout(diagramModel, diagramView);
table.RefreshLayout();
```

VB.NET

```
'Assigning selected node to the OrderedNodes property.
diagramModel.OrderedNodes= diagramView.SelectionList.OfType(Of
IShape)().ToList()
'Create an instance of TableLayout and refresh it.
Dim table As New TableLayout(diagramModel, diagramView)
table.RefreshLayout()
```

When the code runs, the table layout will be applied to the specified node collection.

Note: If the OrderNodes property is set to null, then the table layout will be applied to the entire diagram.

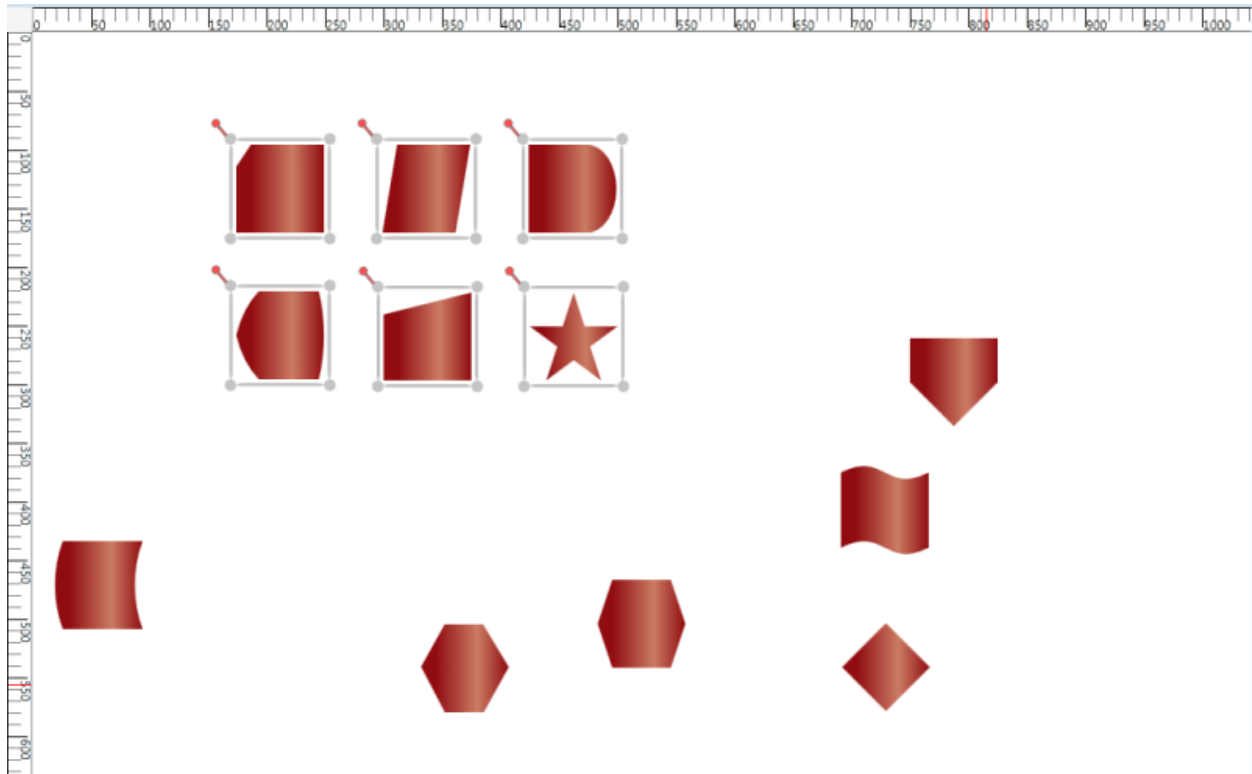


Table Layout Applied for Specified Nodes

[Aligning the Layout on a Specified Location](#)

To align the ordered nodes in a particular position, call the `TableLayout`'s `RefreshLayout` (Point PivotPoint) method and specify the particular point as a parameter. The layout will be positioned in the specified pivot point.

The following code illustrates this:

C#

```
// Assigning selected node to the OrderedNodes.
diagramModel.OrderedNodes=
diagramView.SelectionList.OfType<IShape>().ToList();
TableLayout table = new TableLayout(diagramModel, diagramView);
table.RefreshLayout(300,400);
```

VB.NET

```
' Assigning selected node to the OrderedNodes.
diagramModel.OrderedNodes= diagramView.SelectionList.OfType(Of
IShape)().ToList()
Dim table As New TableLayout(diagramModel, diagramView)
table.RefreshLayout(300,400)
```

[Removing Table Layout from the Specific Nodes](#)

You can remove the table layout applied to specific nodes. To achieve this set the `OrderedNodes` property of the `DiagramMode` to null, and call the `RefreshLayout` method of the `TableLayout`. The layout will be applied to the entire diagram. By default the `OrderedNodes` property is set to null.

The following code illustrates how to remove the layout from the specific nodes:

C#

```
// Set null value to the OrderedNodes property.
diagramModel.OrderedNodes = null;
TableLayout table = new TableLayout(diagramModel, diagramView);
table.RefreshLayout(300, 400);
```

Nodes in WPF Diagram (classic)

Nodes are graphical objects that can be placed on the page. It is usually used to represent visual data to be placed on the page.

Property	Description	Type of the property	Value it Accept	Any other dependencies/ sub properties associated
IsLabelEditable	Gets or sets a value indicating whether node's label can be edited or not. The default value is set to true.	Dependency property	Boolean (true/ false)	No
Label	Gets or sets the node label.	Dependency property	string	No
LabelVisibility	Gets or sets the label visibility.	Dependency property	Visibility.HiddenVisibility.CollapsedVisibility.Visible	No

LabelHorizontalAlignment	Specifies the horizontal alignment for the node label.	Dependency property	HorizontalAlignment.CenterHorizontalAlignment.LeftHorizontalAlignment.RightHorizontalAlignment.Stretch	No
LabelVerticalAlignment	Specifies the vertical alignment for the node label.	Dependency property	VerticalAlignment.BottomVerticalAlignment.CenterVerticalAlignment.StretchVerticalAlignment.Top	No
HorizontalContentAlignment	Specifies the horizontal alignment for the node content.	Dependency property	HorizontalAlignment.CenterHorizontalAlignment.LeftHorizontalAlignment.RightHorizontalAlignment.Stretch	No
VerticalContentAlignment	Specifies the vertical alignment for the node content.	Dependency property	VerticalAlignment.BottomVerticalAlignment.CenterVerticalAlignment.StretchVerticalAlignment.Top	No
LabelAngle	Gets or sets the angle of the node label.	Dependency property	double	No
Shape	Specifies the shape of the node.	Dependency property	Shapes	No
CustomPathStyle	Gets or sets the CustomPathStyle	Dependency property	Style	No

	for the node.			
Level	Gets or sets the node level.	Dependency property	int	No
OffsetX	Gets or sets the offset x value of the node.	CLR property	double	No
OffsetY	Gets or sets the offset y value of the node.	CLR property	double	No
Content	Gets or sets the node's content.	Dependency property	object	No
AllowMove	Gets or sets a value indicating whether node can be moved or not. The default value is set to true.	Dependency property	Boolean (true/ false)	No
AllowSelect	Gets or sets a value indicating whether node can be selected or not.	Dependency property	Boolean (true/ false)	No

	The default value is set to true.			
AllowRotate	Gets or sets a value indicating whether node can be rotated or not. The default value is set to true.	Dependency property	Boolean (true/ false)	No
AllowResize	Gets or sets a value indicating whether node can be resized or not. The default value is set to true.	Dependency property	Boolean (true/ false)	No
LabelTextTrimming	Gets or sets the text trimming style. Default value is CharacterEllipsis.	Dependency property	TextTrimming.CharacterEllipsisTextTrimming.NoneTextTrimming.WordEllipsis	No

LabelForeground	Gets or sets the foreground of the label. Default value is Black.	Dependency property	Brush	No
LabelBackground	Gets or sets the background of the label. Default value is White.	Dependency property	Brush	No
LabelFontStyle	Gets or sets the font style of the label. Default value is White.	Dependency property	FontStyle	No
LabelFontFamily	Gets or sets the font family of the label. Default value is Arial.	Dependency property	FontFamily	No
LabelTextAlignment	Gets or sets the text alignment of the label. Default value is Center.	Dependency property	TextAlignment.CenterTextAlignment.JustifyTextAlignment.LeftTextAlignment.Right	No
LabelFontSize	Gets or sets the font size	Dependency	Double	No

	of the label. Default value is 11.	property		
LabelFontWeight	Gets or sets the font weight of the label. Default value is SemiBold.	Dependency property	FontWeight	No
LabelTextWrapping	Gets or sets the text wrapping of the label. Default value is NoWrap.	Dependency property	TextWrapping.NoWrapTextWrapping.WrapTextWrapping.WrapWithOverflow	No
LabelWidth	Gets or sets the width of the label. Default value is node's width.	Dependency property	Double	No
IsLabelDraggable	Gets or sets the Label of Node is Dragging or Not.	Dependency property	bool(true/false)	False
LabelDisplacement	Gets or sets the different between the Original position	Dependency property	Point	Point(0, 0)

	and the Current position of the Node's Label.			
OutDegree	Gets the out-degree of the node, the number of edges for which this node is the source.	CLR property	int	No
OutEdges	Gets the collection of all outgoing edges for which this node is the source.	CLR property	CollectionExt	No
OutNeighbors	Gets the collection of adjacent nodes connected to this node by an outgoing edge (i.e., all nodes "pointed" to by this one)	CLR property	CollectionExt	No
InDegree	Gets the in-degree,	CLR property	int	No

	or the number of edges for which this node is the target.			
InEdges	Gets the collection of all incoming edges for which this node is the target.	CLR property	CollectionExt	No
InNeighbors	Gets the collection of all adjacent nodes connected to this node by an incoming edge (i.e., all nodes that "point" to this one).	CLR property	CollectionExt	No
Edges	Gets the collection of all incident edges for which this node is either the source or the target.	CLR property	CollectionExt	No

Neighbors	Gets an iterator over all nodes connected to this node.	CLR property	CollectionExt	No
Degree	Gets the degree of the node, or the number of edges for which this node is either the source or the target.	CLR property	int	No
ResizeThisNode	Gets or sets a value indicating whether to resize this node. Used for serialization purposes.	CLR property	Boolean (true/false)	No
ContentHitTestVisible	Gets or sets a value indicating whether content is hit test visible. Used for serialization		Boolean (true/false)	

	purposes internally.			
IntersectionMode	Specifies the Intersection mode for a node. The default value is OnBorder.	Dependency property	IntersectionMode.OnBorderIntersectionMode.OnContent	No
IsDragConnectionOver	Gets or sets a value indicating whether the connection drag is over.	CLR property	Boolean (true/false)	No
AllowDelete	Gets or sets a value indicating whether a node can be deleted.	Dependency property	Boolean (true/false)	No

Name	Parameters	Return Type	Description	Reference Links
Ports.Add(ConnectionPort)	ConnectionPort	void	To add a port to the node	Create Connection Port

- Node Shapes Refer Concepts and Features > Nodes > Node Shapes
- Custom Shape Refer Concepts and Features > Nodes > Node Shapes > Custom Shape
- Node Position Refer Concepts and Features > Nodes > Node Position

- Node Content Refer Concepts and Features -> Nodes > Node Content
- Node Label Refer Concepts and Features > Nodes > Node Label
- Customize the label of Nodes and LineConnectors Refer Concepts and Features -> General -> Customize the label of Nodes and LineConnectors
- Customize the ContextMenu of Nodes and LineConnectors Refer Concepts and Features -> General -> Customize the contextMenu of Nodes and LineConnectors

Create Node

Nodes are graphical objects that can be drawn on the page by selecting them from the SymbolPalette and dropping on the page, or they can be added through code behind using a model.

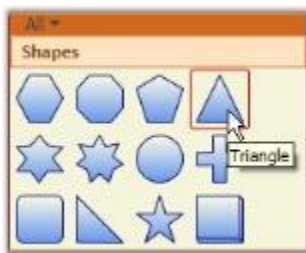
Node can be created and added into the DiagramModel in three ways,

- Through SymbolPalette
- Through XAML
- Through Code Behind

Adding Through SymbolPalette

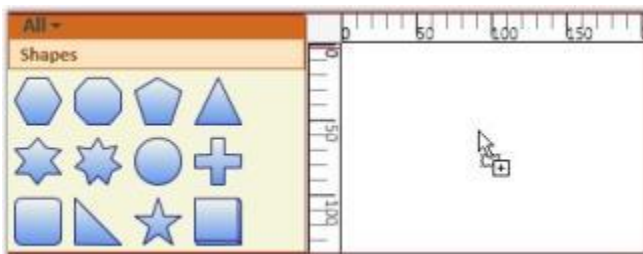
Steps for adding a node to the diagram using the SymbolPalette.

- Click the desired node on the SymbolPalette.



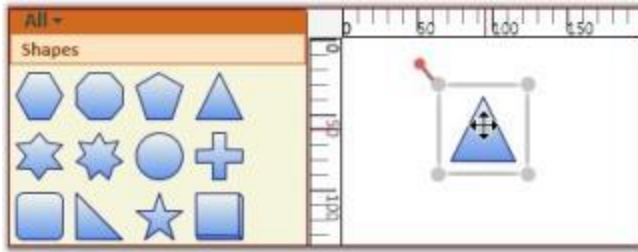
Item Selected

- While pressing the left mouse button, drag the pointer to the drawing area.



Item Dragged

- Now release the mouse button. The desired node is now on the drawing area at the point where the pointer was released.



Item Dropped

Adding Through XAML

Node can also be added into the model through XAML. The following code shows how it can be done.

HTML

```
<syncfusion:DiagramControl Name="diagramControl"
IsSymbolPaletteEnabled="True">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel">
<syncfusion:DiagramModel.Nodes>
<syncfusion:Node Width="200" Height="70" Shape="FlowChart_Card">
</syncfusion:Node>
</syncfusion:DiagramModel.Nodes>
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView IsZoomEnabled="True" Bounds="0,0,1200,1200"
Name="diagramView">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

Here, a node is created and added into the model.

Adding Through code behind

A Node can be created using one of the three different constructors

- Node ()
- Node (Guid)
- Node (Guid, sting)

Nodes can be added through the model. The following code shows how it can be done.

C#

```
Node n = new Node();
n.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim n As New Node()
n.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n)
```


- A node can be created with a new ID. The following code shows how it can be done.

C#

```
Node n = new Node(Guid.NewGuid());
n.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim n As New Node(Guid.NewGuid())
n.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n)
```

- A node can also be created with a new ID and a name. The following code shows how it can be done.

C#

```
Node n = new Node(Guid.NewGuid(), "First");
n.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim n As New Node(Guid.NewGuid(), "First")
n.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n)
```

NewGuid() creates a new instance of the Guid class. The string value sets the identifying name of the element. The name provides a reference so that the code-behind, such as event handler code, can refer to a markup element after it is constructed during processing by an XAML processor.

Note: If name is not specified for a node, a unique name will be automatically assigned to the node.

Node Shapes

Node's Shapes are collection of predefined geometry used to represent as Node's style. Node Shape visually lies between Node's Content and Node's Background.

Property	Description	Type of the property	Value it Accept	Any other dependencies/ sub properties associated
Shape	Specifies the shape of the node	Dependency property	Shapes	No
CustomPathStyle	Gets or sets the CustomPathStyle for the node	Dependency property	Style	No

Predefined Node Shapes

A node can be assigned with a shape using the Shape property. Several built-in shapes are provided. The user can select from any of the built-in shapes or specify their own custom shape using the CustomPathStyle property, which will be explained later in this user guide.

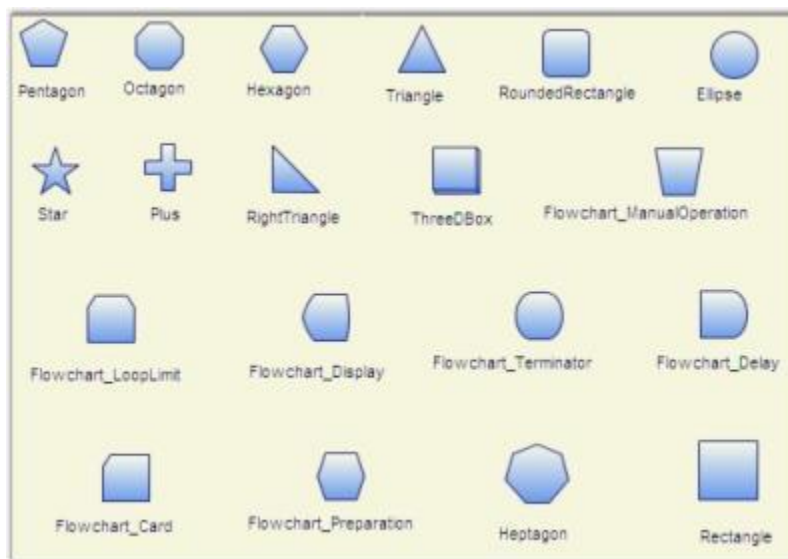
C#

```
Node n = new Node();
n.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim n As New Node()
n.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n)
```

The following is a list of built-in shapes.



Built-in Shapes



More Built-In Shapes

Customize the appearance of the pre-defined Shape

Appearance of the predefined shapes can be customized by applying Style for CustomPathStyle property.

Style can be applied for CustomPathStyle in two different ways,

- Through XAML
- Through Code Behind

HTML

```
<Style TargetType="{x:Type syncfusion:Node}" x:Key="{x:Type
syncfusion:Node}">
<Setter Property="CustomPathStyle">
<Setter.Value>
<Style TargetType="{x:Type Path}">
<Setter Property="Fill" Value="LightGray" />
</Style>
</Setter.Value>
</Setter>
</Style>
```

C#

```
Setter s = new Setter(System.Windows.Shapes.Path.FillProperty,
Brushes.LightGray);
Style style = new Style();
style.Setters.Add(s);
Node n = new Node();
n.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim s As New Setter(System.Windows.Shapes.Path.FillProperty,
Brushes.LightGray)
Dim style As New Style()
style.Setters.Add(s)
Dim n As New Node()
n.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n)
```

Custom Shape

You can specify your own custom shapes to be used for the node as follows. First create a style resource that contains your custom shape.

HTML

```
<Style TargetType="{x:Type Path}" x:Key="myNode">
<Setter Property="Data" Value="M200,239L200,200 240,239 280,202 320,238
281,279 240,244 198,279z"></Setter>
<Setter Property="Fill" Value="MidnightBlue" />
```

```
</Style>
```

Now use it for the node; the following code can be used as an example.

C#

```
Style s = (Style) this.Resources["myNode"];
Node n = new Node();
n.Shape = Shapes.CustomPath;
n.CustomPathStyle = s;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim s As Style = CType(Me.Resources("myNode"), Style)
Dim n As New Node()
n.Shape = Shapes.CustomPath
n.CustomPathStyle = s
diagramModel.Nodes.Add(n)
```



CustomNode

Node Content

Node is used to visually represent any UIElements using Content property. You can host any content inside the node using the Content property. Node supports control template too, by defined template for the nodes, business object can be assigned as Node's Content and the template will look after how to present the business object.

Note: A Node can have both Content and Shape at the same time, doing so Content will be placed over the Shape.

Property	Descri ption	Type of the proper ty	Value it Accept	Any other depend encies/ sub properti es associat ed
Content	Gets or sets the node's conte nt.	Depen dency proper ty	object	No

HorizontalContentAlignment	Specifies the horizontal alignment for the node content.	Dependency property	HorizontalAlignment.CenterHorizontalAlignment.LeftHorizontalAlignment.RightHorizontalAlignment.Stretch	No
VerticalContentAlignment	Specifies the vertical alignment for the node content.	Dependency property	VerticalAlignment.BottomVerticalAlignment.CenterVerticalAlignment.StretchVerticalAlignment.Top	No

UIElement Content

Node's Content can be specified in two ways,

- Through XAML
- Through Code Behind

HTML

```
<syncfusion:DiagramControl Name="diagramControl"
IsSymbolPaletteEnabled="True">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel">
<syncfusion:DiagramModel.Nodes>
<syncfusion:Node Width="200" Height="70" Shape="FlowChart_Card"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center">
<Button Content="Click Me!"></Button>
</syncfusion:Node>
</syncfusion:DiagramModel.Nodes>
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView IsZoomEnabled="True" Bounds="0,0,1200,1200"
Name="diagramView">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
Node n = new Node();
n.Shape = Shapes.FlowChart_Card;
Button b = new Button();
b.Content = "Click ME!";
n.Content = b;
(n.Content as Button).IsHitTestVisible = true;
```

VB.NET

```
Dim n As New Node()
n.Shape = Shapes.FlowChart_Card
Dim b As New Button()
b.Content = "Click ME!"
n.Content = b
TryCast(n.Content, Button).IsHitTestVisible = True
```

**NodeContent***Business-Object Content with ContentTemplate*

Node can even have a business object and a content template that can be used to define the how the business object should look line.

Assigning business object as Node's Content:

C#

```
//Business Object.
public class BusinessObject
{
    //Business Property.
    public string BusinessProperty
    {
        get;
        set;
    }
}
Node n = new Node();
//Assign an instance of Business Object to Node's Content.
n.Content = new BusinessObject() { BusinessProperty = "BusinessProperty" };
diagramModel.Nodes.Add(n);
```

VB.NET

```
'Business Object.
Public Class BusinessObject
```

```

'Business Property.
Private privateBusinessProperty As String
Public Property BusinessProperty() As String
Get
Return privateBusinessProperty
End Get
Set(ByVal value As String)
privateBusinessProperty = value
End Set
End Property
End Class
Private n As New Node()
'Assign an instance of Business Object to Node's Content.
n.Content = New BusinessObject() With {.BusinessProperty =
"BusinessProperty"}
diagramModel.Nodes.Add(n)

```

Specify ContentTemplate for node.

C#

```

<!-- Template for Node -->
<!-- Define the appearance of the Business Object -->
<DataTemplate x:Key="NodeTemplate">
<Border BorderThickness="2" BorderBrush="Blue" CornerRadius="5">
<TextBlock Text="{Binding Path=BusinessProperty}"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Border>
</DataTemplate>
<!-- Style for Node -->
<Style TargetType="{x:Type syncfusion:Node}" x:Key="{x:Type
syncfusion:Node}">
<Setter Property="Width" Value="175" />
<Setter Property="Height" Value="100" />
<Setter Property="VerticalContentAlignment" Value="Stretch"></Setter>
<Setter Property="HorizontalContentAlignment" Value="Stretch"></Setter>
<Setter Property="ContentTemplate" Value="{StaticResource NodeTemplate}" />
</Style>

```



Node's Content having Business Object with ContentTemplate

Node Position

The node's location on the drawing area can be manually specified using the node's OffsetX and OffsetY properties.

Property	Description	Type of the property	Value it Accept	Any other dependencies/ sub properties associated
OffsetX	Gets or sets the offset x value of the node.	CLR property	double	No
OffsetY	Gets or sets the offset y value of the node.	CLR property	double	No
AllowMove	Gets or sets a value indicating whether node can be moved or not. The default value is set to true.	Dependency property	Boolean (true/ false)	No

Note: There are two more properties called LogicalOffsetX and LogicalOffsetY which is used only for internal calculations and they do not support negative values,.So use only OffsetX and OffsetY property.

Specify node's location

Node's location can be changed in following two ways:

- At Runtime
- Through Code Behind

At Runtime

Node's location can be changed at run time by clicking and dragging the Node. To change the node's location, follow the steps below.

- Select the node that is to be dragged to change its location.
- Move the pointer and place it on the Node.
- The cursor will change to a four sided Arrow cursor.
- Now click and drag the node to the change the node's location. The node's OffsetX and OffsetY values will correspondingly change.

Through Code Behind

Node's location can be changed using the following code snippet.

C#

```
Node n = new Node();
n.OffsetX = 50;
n.OffsetY = 50;
n.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim n As New Node()
```



```
n.OffsetX = 50
n.OffsetY = 50
n.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n)
```

The node will be placed at the point: 50, 50.

Note: To dynamically change the position of the node, specify the offset values and call the `InvalidateMeasure()` of the `DiagramPage` as follows:

C#

```
node.OffsetX = 100;
node.OffsetY = 100;
DiagramPage page = new DiagramPage();
page = diagramView.Page as DiagramPage;
page.InvalidateMeasure();
```

VB.NET

```
node.OffsetX = 100
node.OffsetY = 100
Dim page As New DiagramPage()
page = TryCast(diagramView.Page, DiagramPage)
page.InvalidateMeasure()
```

In the above code, `node` refers to the instance of the node whose position is to be changed.

AllowMove

The `AllowMove` property can be used to enable/disable the node drag option.

When this property is set to `True`, it is possible to move the node. Otherwise the node cannot be moved.

The default value is `True`.

The `AllowMove` property can be set in the following way:

C#

```
Node nodeobject = new Node();
nodeobject.AllowMove = false;
```

VB.NET

```
Dim nodeobject As New Node()
nodeobject.AllowMove = False
```

Node Rotate

A node can be rotated to any angle by dragging the rotate thumb. A node always rotates on its center.

Property	Description	Type of the property	Value it Accept	Any other dependencies/ sub properties associated
----------	-------------	----------------------	-----------------	---

AllowRotate	Gets or sets a value indicating whether node can be rotated or not. The default value is set to True.	Dependency property	Boolean (true/ false)	No
-------------	---	---------------------	-----------------------	----

Rotate Node

The steps to rotate a node are as follows.

- Select the node to be rotated. The rotate thumb will be displayed in the top left corner.
- Click the rotate thumb and drag it clockwise or counterclockwise; the node will rotate correspondingly.



Node Rotation

AllowRotate

The AllowRotate property can be used to enable/disable rotation of the node.

Node rotation is enabled when this property is set to 'True'. Otherwise the rotation is disabled. The default value is 'True'.

The AllowRotate property can be set in the following way.

C#

```
Node nodeobject = new Node();
nodeobject.AllowRotate = false;
```

VB.NET

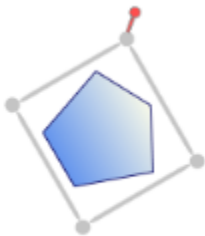
```
Dim nodeobject As New Node()
nodeobject.AllowRotate = False
```

Rotation through Code behind

Rotate angle property enables the rotation of the selected object with a specified angle. It enables the support to rotate all the selected Nodes.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
RotateAngle	Gets or sets the angle for the Rotation.	Dependency property	Double	No

After Rotating the Node:



RotateAngle with 60 degree

After rotating the Group



RotateAngle with 180 degree

Node Resize

Nodes can be resized to desired Width and Height by clicking and dragging the Resizer, based on the content alignment setting. The node's content will also be resized accordingly. Resizing is supported in eight directions.

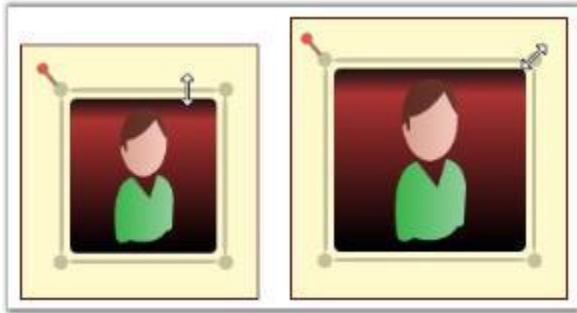
Property	Description	Type of the property	Value it Accept	Any other dependencies/ sub properties associated
AllowResize	Gets or sets a value indicating whether node can be resized or not. The default value is set to true.	Dependency property	Boolean (true/ false)	No

Resizing through the Resize handle

Resizing a node affects the node's width and height. To resize a node, follow the steps below.

- Select the node that is to be resized.
- Move the pointer to the edge that you want to resize.

- The cursor will change to one with two arrows.
- Now drag the edge to the size you want. The node's height and width will correspondingly change.



Node Resizing Illustrated

To resize both the width and height by the same factor, click and drag the corners of the resize adorer.

AllowResize

The AllowResize property can be used to enable/disable the node resizing.

When this property is set to 'True', it is possible to resize the node. Otherwise the node cannot be resized.

The default value is 'True'.

The AllowResize property can be set in the following way.

C#

```
Node nodeobject = new Node();  
nodeobject.AllowResize = false;
```

VB.NET

```
Dim nodeobject As New Node()  
nodeobject.AllowResize = False
```

Resize Single Node on Multiple Selection

When multiple nodes are selected, then by default on resizing any node in the selection list, all the other nodes also get resized. However the user can set this default behavior to false by specifying the EnableResizingCurrentNodeOnMultipleSelection property. Once this property is set to 'True', only the current node will get resized.

This property is in DiagramPage and can be set in the following ways.

C#

```
DiagramPage diagramPage = new DiagramPage();  
diagramPage.EnableResizingCurrentNodeOnMultipleSelection = true;
```

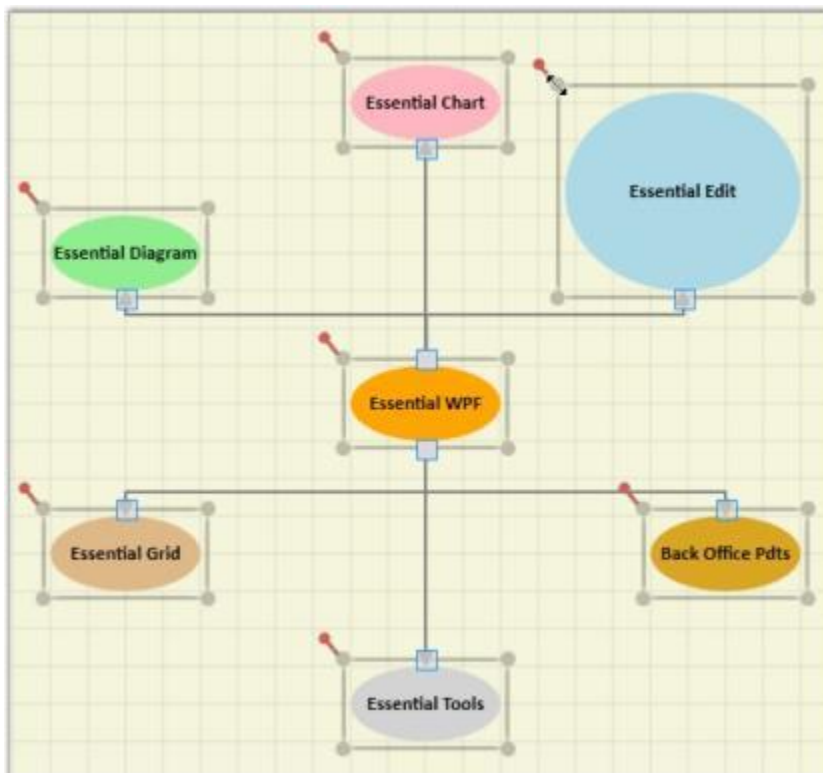
VB.NET

```
Dim diagramPage As New DiagramPage()
```

```
diagramPage.EnableResizingCurrentNodeOnMultipleSelection = True
```

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel x:Name="diagramModel">
    </syncfusion:DiagramModel>
  </syncfusion:DiagramControl.Model>
  <!--View to display nodes and connections added through model.-->
  <syncfusion:DiagramControl.View>
    <syncfusion:DiagramView Name="diagramView">
      <syncfusion:DiagramView.Page>
        <syncfusion:DiagramPage EnableResizingCurrentNodeOnMultipleSelection="True"
          x:Name="diagramPage"/>
      </syncfusion:DiagramView.Page>
    </syncfusion:DiagramView>
  </syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```



Resizing Single Node

Node Label

Label is a single line or multiline text that is displayed over the Node. This Label is used to textually represent a Node with a string that can be edited in run time, there are many properties used to change

the alignment and appearance settings. Label can be represented as multiline text using the TextWrapping property.

Property	Description	Type of the property	Value it Accept	Any other dependencies/ sub properties associated
IsLabelEditable	Gets or sets a value indicating whether node's label can be edited or not. The default value is set to True.	Dependency property	Boolean (true/ false)	No
Label	Gets or sets the node label.	Dependency property	string	No
LabelVisibility	Gets or sets the label visibility.	Dependency property	Visibility.HiddenVisibility.CollapsedVisibility.Visible	No
LabelHorizontalAlignment	Specifies the horizontal alignment for the node label.	Dependency property	HorizontalAlignment.CenterHorizontalAlignment.LeftHorizontalAlignment.RightHorizontalAlignment.Stretch	No
LabelVerticalAlignment	Specifies the vertical	Dependency	VerticalAlignment.BottomVerticalAlignment.CenterVerticalAlignment.StretchVerticalAlignment.Top	No

	alignmen t for the node label.	proper ty		
LabelAngle	Gets or sets the angle of the node label.	Depen dency proper ty	double	No
LabelTextTrim ming	Gets or sets the text trimming style. Default value is Character Ellipsis.	Depen dency proper ty	TextTrimming.CharacterEllipsisTextTrimming.NoneTextTrimmi ng.WordEllipsis	No
LabelForegrou nd	Gets or sets the foregrou nd of the label. Default value is Black.	Depen dency proper ty	Brush	No
LabelBackgrou nd	Gets or sets the backgrou nd of the label. Default value is White.	Depen dency proper ty	Brush	No
LabelFontStyle	Gets or sets the backgrou nd of the label. Default value is White.	Depen dency proper ty	FontStyle	No

LabelFontFamily	Gets or sets the font family of the label. Default value is Arial.	Dependency property	FontFamily	No
LabelTextAlignment	Gets or sets the text alignment of the label. Default value is Center.	Dependency property	TextAlignment.CenterTextAlignment.JustifyTextAlignment.LeftTextAlignment.Right	No
LabelFontSize	Gets or sets the font size of the label. Default value is 11.	Dependency property	Double	No
LabelFontWeight	Gets or sets the font weight of the label. Default value is SemiBold.	Dependency property	FontWeight	No
LabelTextWrapping	Gets or sets the text wrapping of the label. Default value is NoWrap.	Dependency property	TextWrapping.NoWrapTextWrapping.WrapTextWrapping.WrapWithOverflow	No

LabelWidth	Gets or sets the width of the label. Default value is node's width.	Dependency property	Double	No
EnableMultilineLabel	Gets or sets a value indicating whether the Node's label can be multiline or not. Default value is False.	Dependency Property	Boolean (True / False)	No
LabelTextDecorations	Gets or sets the text alignment of the label. Default value is Center.	Dependency property	TextDecorations.UnderlineTextDecorations.BaselineTextDecorations.OverLineTextDecorations.Strikethrough	No

Following is list of topics explained subsequently,

- Set Label for Node
- Label Editing
- Multiline label
- Label Visibility
- Label Angle

Set a label for the node using the Label property. The default value is an empty string. By default, the label is displayed at the top-center position.

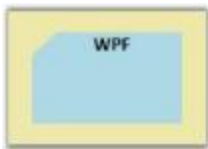
C#

```
Node n = new Node();
n.OffsetX = 50;
```

```
n.OffsetY = 50;  
n.Shape = Shapes.FlowChart_Card;  
n.Label = "WPF";  
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim n As New Node()  
n.OffsetX = 50  
n.OffsetY = 50  
n.Shape = Shapes.FlowChart_Card  
n.Label = "WPF"  
diagramModel.Nodes.Add(n)
```



Label

Label Editing

A node's label can be edited at run time by setting `IsLabelEditable` property to `True`.

C#

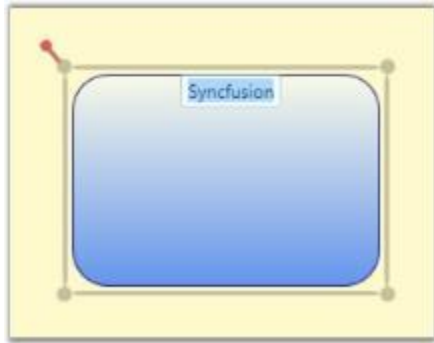
```
Node n = new Node();  
n.Shape = Shapes.RoundedRectangle;  
n.IsLabelEditable = true;
```

VB.NET

```
Dim n As New Node()  
n.Shape = Shapes.RoundedRectangle  
n.IsLabelEditable = True
```

A user can specify a label at run time by following the below mentioned steps.

- Double click the left mouse button on the node. A text box will appear with the cursor at the beginning.
- Now type the label name and press ENTER. The label will be displayed on the node. Press ESC key if you do not want to apply the new label value.



LabelEditor

Customize the Label of Nodes and Line Connectors Refer Concepts and Features -> General -> Customize the Label of Node and LineConnectors

Multiline label

Label text can be displayed in multiple lines using LabelTextWrapping property set to Wrap. If there is no enough space for the text to get displayed over the node in a single line then the text will get wrapped within the node boundaries and will start to display the label in multiple lines.

C#

```
Node n = new Node();
n.Shape = Shapes.RoundedRectangle;
n.Label = "This is Multiline Label";
n.Width = 75;
n.Height = 100;
n.LabelTextWrapping = TextWrapping.Wrap;
n.IsLabelEditable = true;
```

VB.NET

```
Dim n As New Node()
n.Shape = Shapes.RoundedRectangle
n.Label = "This is Multiline Label"
n.Width = 75
n.Height = 100
n.LabelTextWrapping = TextWrapping.Wrap
n.IsLabelEditable = True
```



Multi line label

Label Visibility

A label's visibility can be changed using the `LabelVisibility` property. The default value is visible.

C#

```
Node n = new Node();  
n.Shape = Shapes.FlowChart_Card;  
n.Label = "Syncfusion";  
n.LabelVisibility = Visibility.Hidden;
```

VB.NET

```
Dim n As New Node()  
n.Shape = Shapes.FlowChart_Card  
n.Label = "Syncfusion"  
n.LabelVisibility = Visibility.Hidden
```

Label Vertical Alignment & Label Horizontal Alignment

Vertical and horizontal alignments of a label is specified using `LabelVerticalAlignment` and `LabelHorizontalAlignment` properties. By default, `LabelVerticalAlignment` is set to Top and `LabelHorizontalAlignment` is set to Center.

C#

```
Node n = new Node();  
n.Shape = Shapes.FlowChart_Card;  
n.Label = "Diagram";  
n.LabelHorizontalAlignment = HorizontalAlignment.Center;  
n.LabelVerticalAlignment = VerticalAlignment.Center;
```

VB.NET

```
Dim n As New Node()  
n.Shape = Shapes.FlowChart_Card  
n.Label = "Diagram"  
n.LabelHorizontalAlignment = HorizontalAlignment.Center  
n.LabelVerticalAlignment = VerticalAlignment.Center
```

This will place the label at the center of the node.



LabelAlignment

LabelAngle

The user can rotate the label by a specified angle and display it using the below code snippet.

C#

```
Node n = new Node();
n.Shape = Shapes.FlowChart_Card;
n.Label = "Diagram";
n.LabelHorizontalAlignment = HorizontalAlignment.Right;
n.LabelVerticalAlignment = VerticalAlignment.Top;
n.Label = 45;
```

VB.NET

```
Dim n As New Node()
n.Shape = Shapes.FlowChart_Card
n.Label = "Diagram"
n.LabelHorizontalAlignment = HorizontalAlignment.Right
n.LabelVerticalAlignment = VerticalAlignment.Top
n.Label = 45
```



LabelAngle

Multiline Label Support for LabelEditor:

Node's Label can be set as Multiline Label by setting the EnableMultiline property as True. The default Value is False.

HTML

```
<syncfusion:Node Shape="FlowChart_Card" Height="100" Width="100"
OffsetX="500" OffsetY="500" Name="Node1" EnableMultilineLabel="true"/>
```

C#

```
Node node1 = new Node();
node1.Shape = Shapes.RoundedSquare;
node1.OffsetX = 500;
node1.OffsetY = 500;
node1.Width = 100;
node1.Height = 100;
node1.EnableMultilineLabel = true;
diagramModel.Nodes.Add(node1);
```

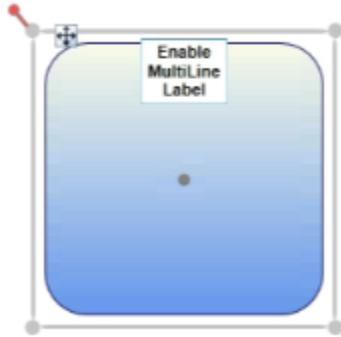
VB.NET

```
Dim node1 As New Node()
node1.Shape = Shapes.RoundedSquare
node1.OffsetX = 500
```

```

node1.OffsetY = 500
node1.Width = 100
node1.Height = 100
node1.EnableMultilineLabel = True
diagramModel.Nodes.Add(node1)

```



MultilineLabel for Node

Custom Label Support for Node

This feature enables you to customize the Label Position of the Nodes. The `IsLabelDraggable` and `LabelDisplacement` property is used to customize the Label position of the node. `LabelDisplacement` property depends on `LabelHorizontalAlignment` and `LabelVerticalAlignment`.

Property	Description	Type	Value It Accepts	Default Values
<code>IsLabelDraggable</code>	Gets or sets the Label of the Node to be Dragged or Not.	Dependency property	<code>bool(true/false)</code>	False
<code>LabelDisplacement</code>	Gets or sets the different between the Original position and the Current position of the Node's Label.	Dependency property	Point	<code>Point(0,0)</code>

Adding Custom Label Enhancements for Node to an Application

Set the LabelDisplacement for Node

Label is aligned within the bounds of Node using `LabelHorizontalAlignment` and `LabelVerticalAlignment` property, the `LabelDisplacement` can be used as to displace the Label from this original position. This value can be positive or negative.

C#

```

(node as Node).LabelDisplacement = new Point(100,100);

```

Label Dragging support for Node

The Label can be dragged from the Node at runtime, if this property is set to true. When a label is dragged, it will automatically update the LabelDisplacement value.

C#

```
(node as Node).IsLabelDraggable = true;
```

Gripper

Gripper is a small rectangle which appears near the top-left corner of the node. It facilitates the node drag operation. You can drag a node by clicking and dragging the gripper. This gripper will be useful when a content of a node is specified. If this content's IsHitTestVisible property is True clicking and dragging the node will be difficult or even impossible because all mouse operations will be handled by the Node's Content. In this scenario Gripper will be very helpful to drag the node in the page.

Property	Description	Type of the property	Value it accepts	Any other dependencies / sub properties associated
GripperVisibility	Gets or sets the gripper visibility.	Dependency property	Visibility.HiddenVisibility.CollapsedVisibility.Visible	No
GripperStyle	Gets or sets the gripper style.	Dependency property	Style	No

Gripper Visibility

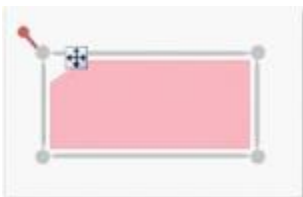
The gripper can be made visible by setting the GripperVisibility property to 'True'.

C#

```
Node n = new Node();
n.GripperVisibility = Visibility.Visible;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim n As New Node()
n.GripperVisibility = Visibility.Visible
diagramModel.Nodes.Add(n)
```



Gripper for Nodes

Gripper Style

Gripper can be customized using the GripperStyle property of the Node. However, on overriding the style, it is necessary to specify the Width, Height, HorizontalAlignment, VerticalAlignment and the Margin property to position the Gripper properly.

HTML

```
<Style x:Key="GripperStyle" TargetType="{x:Type syncfusion:Gripper}">
  <Setter Property="Width" Value="30"/>
  <Setter Property="Height" Value="30"/>
  <Setter Property="HorizontalAlignment" Value="Left"/>
  <Setter Property="VerticalAlignment" Value="Top"/>
  <Setter Property="Margin" Value="10,-15,0,0"/>
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type syncfusion:Gripper}">
        <Border Background="Salmon" CornerRadius="10" BorderThickness="2"
          BorderBrush="Gray">
          <Label HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
            Content="G" FontWeight="Bold"/>
        </Border>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

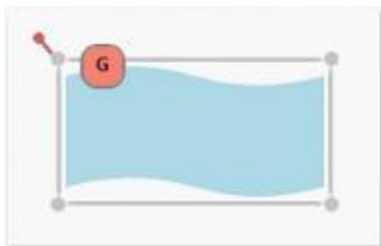
This can be applied to the node in the following way:

C#

```
Node node = new Node();
node.GripperStyle = this.Resources["GripperStyle"] as Style;
diagramModel.Nodes.Add(node);
```

VB.NET

```
Dim node As New Node()
node.GripperStyle = TryCast(Me.Resources("GripperStyle"), Style)
diagramModel.Nodes.Add(node)
```



Customized Gripper

Node Selection

A selected node is indicated using a rectangular resizer over the node's border. Many interactions using keyboard, mouse commands will affect the elements that is currently selected.

Property	Description	Type of the property	Value it Accept	Any other dependencies/ sub properties associated
AllowSelect	Gets or sets a value indicating whether node can be selected or not. The default value is set to True.	Dependency property	Boolean (true/ false)	No
IsSelected	Gets or sets a value indicating whether this instance is selected.	Dependency property	Boolean (true/ false)	No

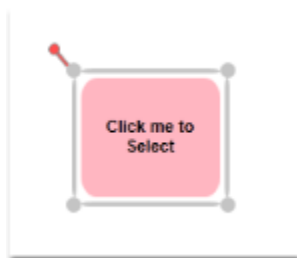
A node can be selected in two ways,

- At run time
- Through Code

Node can be selected at run time just by clicking on the node.



Node before selection



Node after selection

The above two images differentiates the appearance of the node before and after selection.

Node can also be selected using the IsSelected property of the Node.

C#

```
Node n = new Node();
```

```
n.Shape = Shapes.FlowChart_Card;  
n.IsSelected = true;  
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim n As New Node()  
n.Shape = Shapes.FlowChart_Card  
n.IsSelected = True  
diagramModel.Nodes.Add(n)
```

Node Multi-Selection

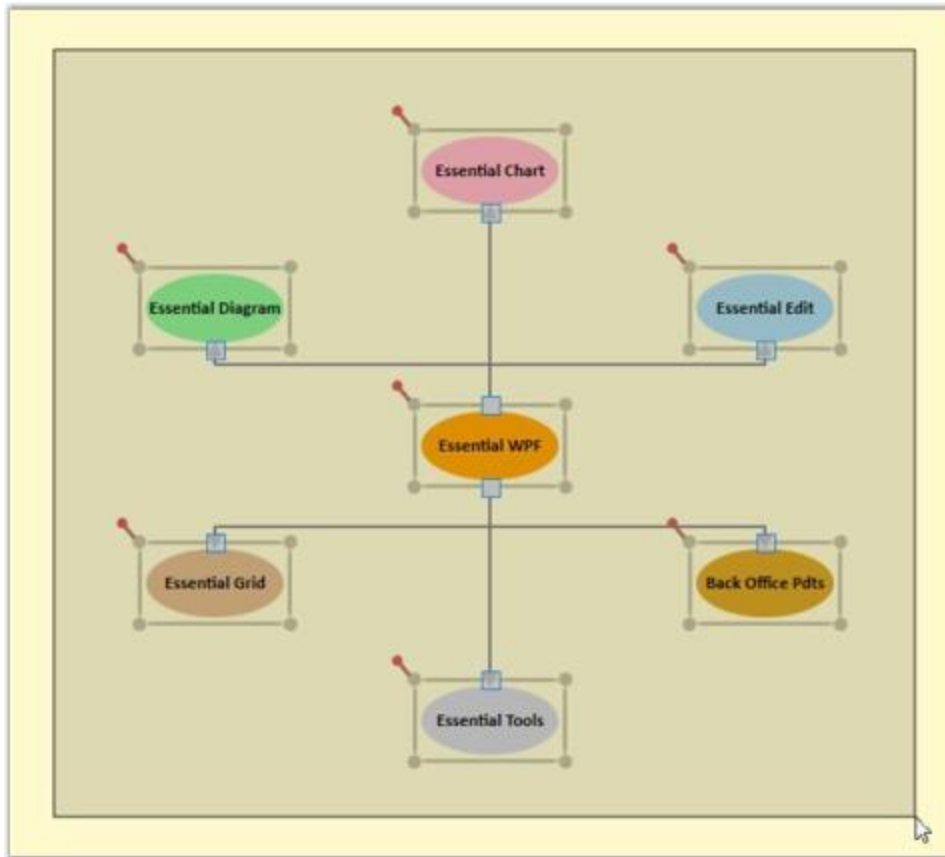
Multiple items on the drawing area can be selected.

Multiple selection can be performed by following the below mentioned steps.

- Items on the drawing area are selected only if they fall within the bounds of the drag adorer.
- The drag adorer is displayed when you click anywhere on the page and start dragging the pointer.
- The rectangle formed with the drag start-point as one of its points, and the point where the mouse button was released as its second point, defines the drag adorer's bounds.
- Nodes connected to one or more nodes are selected only if one of the connected nodes is also within the drag adorer bounds. The nodes and the connector connecting them act as a single selection.

Note: Resizing or moving any one item affects the other items by the same factor. However, rotating affects only the current node.

Items can be deselected by clicking on any part of the drawing area other than the selected items.



Multiple Selections

AllowSelect

The AllowSelect property can be used to enable/disable the node selection.

When this property is set to 'True', it is possible to select the node. Otherwise the node cannot be selected.

The default value is 'True'.

The AllowSelect property can be set in the following way:

C#

```
Node nodeobject = new Node();  
nodeobject.AllowSelect = false;
```

VB.NET

```
Dim nodeobject As New Node()  
nodeobject.AllowSelect = False
```

Select Nodes and Connectors Refer Concepts and Features -> General -> Select Nodes and Connectors

Select and Move Nodes

As this is a general topic to be share between Node and LineConnector, please refer general topic under Concepts and features using the following links.

- [Select Node and Connectors Refer Concepts and Features -> General -> Select Node and Connectors](#)
- [Move Node and Connectors Refer Concepts and Features -> General -> Move Node and Connectors](#)

Deleting Node Without its Edges

This feature enables the user to delete the node without deleting its dependent edges by using the DeletingMode property of the node.

Properties

Property	Description	Type	Data Type
DeletingMode	Decides whether the node alone has to be deleted or node along with its dependent edges has to be deleted.	Dependency Property	Enum

Enabling Deletion of Node Without its Edges

To enable the deletion of node, the DeletingMode property can be set to:

1. DeleteDependentEdges
2. None

By default, this property is set to DeleteDependentEdges.

Setting DeletingMode to None

If the DeletingMode property of the node is set to None while trying to delete the node, the node alone is deleted. The dependent edges are not deleted.

To set the DeletingMode property to None, use the following code.

C#

```
Node n1 = new Node();
n1.OffsetX = 200;
n1.OffsetY = 200;
n1.Shape = Shapes.Rectangle;
n1.DeletingMode = DeletingMode.None;
diagrammodel.Nodes.Add(n1);
```

VB.NET

```
Dim n1 As New Node()
n1.OffsetX = 200
n1.OffsetY = 200
n1.Shape = Shapes.Rectangle
n1.DeletingMode = DeletingMode.None
```

```
diagramModel.Nodes.Add(n1)
```

Setting DeletingMode to DeleteDependentEdges

If the DeletingMode property of the node is set to DeleteDependentEdges while trying to delete the node, the node along with the dependent edges is deleted.

To set the DeletingMode property to DeleteDependentEdges, use the following code example.

C#

```
Node n = new Node();
n.OffsetX = 100;
n.OffsetY = 100;
n.Shape = Shapes.Rectangle;
n.DeletingMode = DeletingMode.DeleteDependentEdges;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim n1 As New Node()
n1.OffsetX = 200
n1.OffsetY = 200
n1.Shape = Shapes.Rectangle
n1.DeletingMode = DeletingMode.DeleteDependentEdges
diagramModel.Nodes.Add(n1)
```

Customize the Label, Context Menu for Nodes

As this is a general topic to be share between Node and LineConnector, please refer general topic under Concepts and features using the following links.

- [Customize Label Refer Concepts and Features -> General -> Customize Label](#)
- [Customize ContextMenu Refer Concepts and Features -> General -> Customize ContextMenu](#)

Customization of Node Movement

While dragging the node in the diagram page, the user can customize the movement of the node by setting the following DiagramView properties. This customization is used to control the node dragging in a single direction.

Properties

Property	Description	Type	Data type
DirectionBehaviorEnabled	Sets or gets the DirectionBehaviorEnabled value.	Dependency property	Boolean
TranslateRailsMode	Specifies the direction in which the node has to be moved. The default value is None.	Dependency property	Enum
NodeMovementX	Sets or gets the key value for the node movement on X-axis.	Dependency property	Enum

NodeMovementY	Sets or gets the key value for the node movement on Y-axis.	Dependency property	Enum
---------------	---	---------------------	------

Adding Customization of Node in an Application

DirectionBehaviorEnabled

The DirectionBehaviorEnabled property restricts the node movement to only one direction at a time.

If the DirectionBehaviorEnabled property is set to 'True', the node is dragged only one direction; in which the node is initially dragged.

If the node is once dragged in X-direction, then the node can move only in X-direction for a while. Likewise, if the node is once dragged in Y-direction then the node can be moved only in Y-direction for a while.

Hence, the direction is decided at the beginning of each drag. By default, this property is set to *false*. Use the following code snippet to enable this property.

C#

```
DiagramView View1 = new DiagramView();
View1.DirectionBehaviorEnabled = true;
```

VB.NET

```
Dim view As New DiagramView ()
View1.DirectionBehaviorEnabled = True
```

This property can be disabled by setting it to 'false'.

C#

```
DiagramView View1 = new DiagramView();
View1.DirectionBehaviorEnabled = false;
```

VB.NET

```
Dim view As New DiagramView ()
View1.DirectionBehaviorEnabled = False
```

TranslateRailsMode

This property is used to define the node movement through code. This Enum property has three values.

1. TranslateRailsX
2. TranslateRailsY
3. None

By default, this property value is 'None'.

To enable the node movement only on X-axis, use the following code.

C#

```
DiagramView View1 = new DiagramView();  
View1.TranslateRailsMode = TranslateRailsMode.TranslateRailsX;
```

VB.NET

```
Dim view As New DiagramView ()  
View1. TranslateRailsMode = TranslateRailsMode.TranslateRailsX
```

To enable the node movement only in Y-axis, use the following code example.

C#

```
DiagramView View1 = new DiagramView();  
View1.TranslateRailsMode = TranslateRailsMode.TranslateRailsY;
```

VB.NET

```
Dim view As New DiagramView ()  
View1. TranslateRailsMode = TranslateRailsMode.TranslateRailsY
```

NodeMovementX/NodeMovementY

The node movement can be controlled by the keys.

NodeMovementX:

The keys that support NodeMovement on X-axis are: "Alt", "X" and "Z".

By default, the enum property value is *None*. Pressing and dragging, selected key controls the node movement only in X-axis.

C#

```
DiagramView View1 = new DiagramView();  
View1.NodeMovementX = NodeMovementX.X;  
View1.NodeMovementY = NodeMovementX.Y;
```

VB.NET

```
Dim view As New DiagramView ()  
View1.NodeMovementX = NodeMovementX.X  
View1.NodeMovementY = NodeMovementX.Y
```

NodeMovementY:

The key combinations that support the node movement on Y-axis are:

4. Alt+Y
5. Alt+Z
6. Y
7. Space

By default, the Enum property value is *None*. Pressing and dragging, selected Key controls the node movement only X-axis.

C#

```
DiagramView View1 = new DiagramView();
View1.NodeMovementX = NodeMovementX.X;
View1.NodeMovementY = NodeMovementX.Y;
```

VB.NET

```
Dim view As New DiagramView ()
View1.NodeMovementX = NodeMovementX.X
View1.NodeMovementY = NodeMovementX.Y
```

Resize Handler Customization

This feature provides different styles for Resize Handler. This enables you to customize the complete look and feel of the eight resize handler.

Use Case Scenarios

The appearance of the Resizer Handler can be effectively changed by applying different styles to the Thumb, using the Resize Handler Customization.

Creating Custom Style for Resize Handler

The Resize Handler consists of eight Resizer Thumbs. You can set different styles to a ResizerThumb by using the Resize Handler Properties.

Follow the below steps to create custom styles for Resize Handler.

Step1: Creating Style for ResizerThumb

Prepare styles with template for each ResizerThumb.

- Through XAML.

The following code illustrates how to create the style for ResizerThumb

HTML

```
<Style x:Key="TopLeftCornerResizerThump"
TargetType="syncfusion:ResizerThumb" >
<Setter Property="SnapsToDevicePixels" Value="True"/>
<Setter Property="OverridesDefaultStyle" Value="true"/>
<Setter Property="IsTabStop" Value="false"/>
<Setter Property="Focusable" Value="false"/>
<Setter Property="Height" Value="15"/>
<Setter Property="Width" Value="15"/>
<Setter Property="Cursor" Value="SizeNWSE"/>
<Setter Property="Margin" Value="-3 -3 0 0"/>
<Setter Property="VerticalAlignment" Value="Top"/>
<Setter Property="HorizontalAlignment" Value="Left"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:ResizerThumb">
<Grid>
<Rectangle HorizontalAlignment="Stretch" Margin="{TemplateBinding Margin}"
VerticalAlignment="{TemplateBinding VerticalAlignment}" StrokeDashCap="Flat"
```



```

Cursor="{TemplateBinding Cursor}" x:Name="PART_ReseizerThumb3" Stroke="Blue"
StrokeStartLineCap="Round" StrokeThickness="5"/>
<Rectangle HorizontalAlignment="{TemplateBinding HorizontalAlignment}"
VerticalAlignment="Stretch" Cursor="{TemplateBinding Cursor}" Stroke="Blue"
x:Name="PART_ReseizerThumb2" StrokeDashCap="Flat" StrokeThickness="5"
StrokeStartLineCap="Round" Margin="{TemplateBinding Margin}"/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

Step2: Assign the Style to Node

- Through XAML.

The following code illustrates how to assign the Resize Handler Style to Node

HTML

```

<Style TargetType="syncfusion:Node">
<Setter Property="TopResizer" Value="{StaticResource TopResizerThump}"/>
<Setter Property="LeftResizer" Value="{StaticResource LeftResizerThump}"/>
<Setter Property="RightResizer" Value="{StaticResource RightResizerThump}"/>
<Setter Property="BottomResizer" Value="{StaticResource
BottomResizerThump}"/>
<Setter Property="TopLeftCornerResizer" Value="{StaticResource
TopLeftCornerResizerThump}"/>
<Setter Property="TopRightCornerResizer" Value="{StaticResource
TopRightCornerResizerThump}"/>
<Setter Property="BottomLeftCornerResizer" Value="{StaticResource
BottomLeftCornerResizerThump}"/>
<Setter Property="BottomRightCornerResizer" Value="{StaticResource
BottomRightCornerResizerThump}"/>
</Style>

```

- Through Code behind

The following code illustrates how to assign the Resize Handler Style to Node.

C#

```

Node n = shape as Node;
n.TopResizer = this.Resources["TopResizerThump"] as Style;
n.LeftResizer = this.Resources["LeftResizerThump"] as Style;
n.RightResizer = this.Resources["RightResizerThump"] as Style;
n.BottomResizer = this.Resources["BottomResizerThump"] as Style;
n.TopLeftCornerResizer = this.Resources["TopLeftCornerResizerThump"] as
Style;
n.TopRightCornerResizer = this.Resources["TopRightCornerResizerThump"] as
Style;
n.BottomLeftCornerResizer = this.Resources["BottomLeftCornerResizerThump"] as
Style;

```

```
n.BottomRightCornerResizer =this.Resources["BottomRightCornerResizerThump"]
as Style;
```

VB.NET

```
Dim n As Node = TryCast(Shape, Node)
n.TopResizer = TryCast(Me.Resources("TopResizerThump"), Style)
n.LeftResizer =TryCast(Me.Resources("LeftResizerThump"), Style)
n.RightResizer =TryCast(Me.Resources("RightResizerThump"), Style)
n.BottomResizer =TryCast(Me.Resources("BottomResizerThump"), Style)
n.TopLeftCornerResizer =TryCast(Me.Resources("TopLeftCornerResizerThump"),
Style)
n.TopRightCornerResizer =TryCast(Me.Resources("TopRightCornerResizerThump"),
Style)
n.BottomLeftCornerResizer
=TryCast(Me.Resources("BottomLeftCornerResizerThump"), Style)
n.BottomRightCornerResizer
=TryCast(Me.Resources("BottomRightCornerResizerThump"), Style)
```

Setting ResizerThumb Template as null

ResizerThumb will not be visible When the ResizerThumb Template value as Null.

The following code illustrates how to set the ResizerThumb Template to Null.

HTML

```
<Style x:Key="TopResizerThump" TargetType="syncfusion:ResizerThumb">
<Setter Property="Template" Value="{x:Null}">
</Setter>
</Style>
```

Following is a sample screenshot of customized resizer that has only four corners.



Custom Style

Tables for Properties, Methods, and Events

Properties

Property	Description	Type	Data Type	Reference links
TopResizer	Gets or sets a value of TopResizer Style for Resize Handler	Dependency property	Style	No

BottomResizer	Gets or sets a value of BottomResizer Style for Resize Handler	Dependency property	Style	No
LeftResizer	Gets or sets a value of LeftResizer Style for Resize Handler	Dependency property	Style	No
RightResizer	Gets or sets a value of RightResizer Style for Resize Handle	Dependency property	Style	No
TopLeftResizer	Gets or sets a value of TopLeftResizer Style for Resize Handler	Dependency property	Style	No
TopRightResizer	Gets or sets a value of TopRightResizer Style for Resize Handler	Dependency property	Style	No
BottomLeftResizer	Gets or sets a value of BottomLeftResizer Style for Resize Handler	Dependency property	Style	No
BottomRightResizer	Gets or sets a value of BottomRightResizer Style for Resize Handler	Dependency property	Style	No

Sample Link

To view sample,

1. Open the WPF sample browser from the dashboard.
2. Navigate to WPF Diagram -> Editable Diagram->ResizerCustomization Demo

Edges, Degree and Neighbors

Edges Properties for Node:

The Edges properties for Node are used to retrieve the collection of incoming and outgoing connections for Node.

Property	Description	Type of Property	Any other dependencies/ sub-properties associated
Edges	Gets the collection of the incoming and outgoing connections of the node.	CLR Property â€œ read-only property	No
InEdges	Gets the collection of all incoming connections to the node.	CLR Property â€œ read-only property	No

OutEdges	Gets the collection of all outgoing connections from the node.	CLR Property â€œ read-only property	No
----------	--	-------------------------------------	----

The Edges, InEdges, and OutEdges properties can be retrieved as follows:

C#

```
//Getting the Edges of the Node
Node node = new Node();
CollectionExt ext = node.Edges;
//Getting the InEdges of the Node
Node node = new Node();
CollectionExt ext = node.InEdges;
//Getting the InEdges of the Node
Node node = new Node();
CollectionExt ext = node.OutEdges;
```

Degree Properties for Node:

The Degree properties for Node are used to retrieve the count of incoming and outgoing edges for Node.

Property	Description	Type of Property	Return Type	Any other dependencies/sub-properties associated
Degree	Gets the total number count of incoming and outgoing edges.	CLR Property â€œ read-only property	Integer	No
InDegree	Gets the number count of incoming edges.	CLR Property â€œ read-only property	Integer	No
OutDegree	Gets the number count of outgoing edges.	CLR Property â€œ read-only property	Integer	No

The Degree, InDegree, and OutDegree properties can be retrieved in the following way:

C#

```
//Getting the Degree of the Node
Node node = new Node();
int degree = node4.Degree;
//Getting the InDegree of the Node
Node node = new Node();
int indegree = node4.InDegree;
//Getting the OutDegree of the Node
Node node = new Node();
int outdegree = node4.OutDegree;
```

Neighbors Properties for Node:

The Neighbors properties for Node are used to retrieve the collection of all of the nodes connected to the specified Node.

Property	Description	Type of Property	Any other dependencies/sub-properties associated
Neighbors	Gets the collection of all of the nodes connected to the specified Node.	CLR Property â€œ read-only property	No
InNeighbors	Gets the collection of all adjacent nodes connected to the specified node. In other words, gets the collection of nodes that are connected to the node, or all nodes that "point" at this one.	CLR Property â€œ read-only property	No
OutNeighbors	Gets the collection of nodes that are connected from the node, or all nodes "pointed" to by this one.	CLR Property â€œ read-only property	No

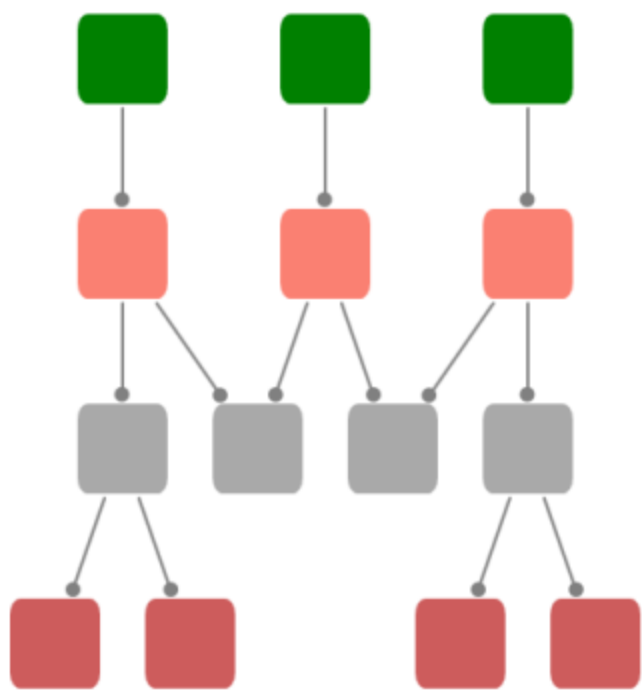
The Neighbors, InNeighbors, and OutNeighbors properties can be retrieved as illustrated in the following code:

C#

```
//Getting the Neighbors of the Node
Node node = new Node();
CollectionExt ext = node.Neighbors;
//Getting the InNeighbors of the Node
Node node = new Node();
CollectionExt ext = node.InNeighbors;
//Getting the OutNeighbors of the Node
Node node = new Node();
CollectionExt ext = node.OutNeighbors;
```

Node Layout

Essential Diagram for WPF provides layout representation for nodes. Numerous nodes and line connectors can be connected together to form various types of layouts, including hierarchical, tree, and table layouts.



Node Layout

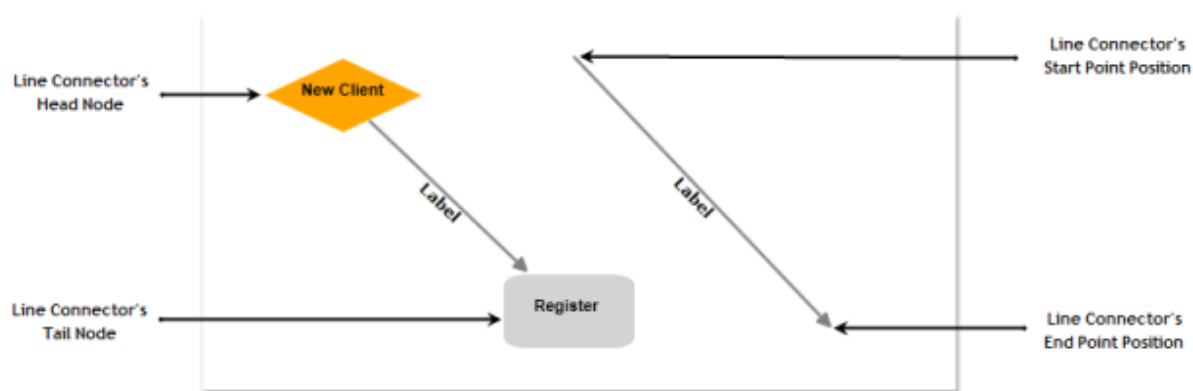
Properties:

Property	Description	Type of property	Value it Accepts	Any other dependencies/ sub properties associated
FirstChild	Gets node’s first tree child.	CLR property	Node	No
LastChild	Gets node’s last tree child.	CLR property	Node	No
ParentNode	Gets or sets the parent of the node.	CLR property	IShape	No
Row	Gets or sets the row.	CLR property	int	No
ParentEdge	Gets or sets the edge between this node and its parent node in tree structure.	CLR property	IEdge	No
NextSibling	Gets this node’s next tree sibling.	CLR property	IShape	No

IsExpanded	Gets or sets the values indicating whether this instance is expanded.	CLR property	Boolean (true/false)	No
ChildCount	Gets the number of tree children for this node.	CLR property	int	No
Column	Gets or sets the column.	CLR property	int	No

Line Connectors in WPF Diagram (classic)

Connectors are objects that are used to create a link between two nodes. Each connector has two ends whose position can be specified as point or directly connected to Node. One end of the connector can be defined either by using the 'Start Point Position' or 'Head Node', similarly other end can be defined using 'End Point Position' or 'Tail Node'.



Property	Description	Type of the property	Value it accepts	Any other dependencies / sub properties associated
EnableConnection	Gets or sets a value indicating whether connection is enabled or not.	Dependency property	Boolean (true/ false)	No
IsLabelEditable	Gets or sets a value indicating whether	Dependency property	Boolean (true/ false)	No

	line's label can be edited or not. Default value: True	property		
IntermediatePoints	Gets or sets the intermediate points.	Dependency property	List<Point>>	No
Label	Gets or sets the line's label. Default value: Empty String.	Dependency property	String	No
LabelTemplate	Gets or sets a template for the label. Default value: null.	Dependency property	DataTemplate	No
LabelVisibility	Gets or sets the label visibility. Default value: Visibility.Visible	Dependency property	Visibility.HiddenVisibility.CollapsedVisibility.Visible	No
LabelHorizontalAlignment	Gets or sets the node's label horizontal Alignment. Default value: HorizontalAlignment.Center	Dependency property	HorizontalAlignment.CenterHorizontalAlignment.LeftHorizontalAlignment.RightHorizontalAlignment.Stretch	No
ConnectionEndSpace	Gets or sets the distance between the connector end position and the node. Default Value: 6	CLR property	Double	No

ConnectorType	Gets or sets the connector type to be used. Three values namely Orthogonal, Straight and Bezier can be specified. Default Value: ConnectorType.Orthogonal	Dependency property	ConnectorType.OrthogonalConnectorType.BezierConnectorType.Straight	No
HeadNode	Gets or sets the head node of the connection. Default value: null.	Dependency property	IShape	No
TailNode	Gets or sets the tail node of the connection. Default value: null.	Dependency property	IShape	No
HeadDecoratorShape	Gets or sets the head decorator shape of the connection. Four values namely None, Arrow, Diamond and Circle can be specified. Default value: HeadDecoratorShape.None	CLR property	DecoratorShape.NoneDecoratorShape.ArrowDecoratorShape.DiamondDecoratorShape.Circle	No
TailDecoratorShape	Gets or sets the tail decorator shape of the connection. Four values	CLR property	DecoratorShape.NoneDecoratorShape.ArrowDecoratorShape.DiamondDecoratorShape.Circle	No

	namely None, Arrow , Diamond and Circle can be specified.Default value: TailDecoratorShape.Arrow			
HeadDecoratorStyle	Provides customization option for the head decorator shape.	CLR property	DecoratorStyle	No
TailDecoratorStyle	Provides customization option for the tail decorator shape.	CLR property	DecoratorStyle	No
LineStyle	Provides customization option for the line connector.	CLR property	LineStyle	No
LabelTextTrimming	Gets or sets the text trimming style. Default value is CharacterEllipsis.	Dependency property	TextTrimming.CharacterEllipsisTextTrimming.NoneTextTrimming.WordEllipsis	No
LabelForeground	Gets or sets the label foreground. Default value is Black.	Dependency property	Brush	No
LabelBackground	Gets or sets the label background. Default value is White.	Dependency property	Brush	No
LabelFontStyle	Gets or sets the label	Dependency	FontStyle	No

	background. Default value is White.	y prope rty		
LabelFontFa mily	Gets or sets the label font family. Default value is Arial.	Depe ndenc y prope rty	FontFamily	No
LabelTextAli gnment	Gets or sets the label text alignment. Default value is Center.	Depe ndenc y prope rty	TextAlignment.CenterTextAlignment.JustifyTextAlignm ent.LeftTextAlignment.Right	No
LabelFontSiz e	Gets or sets the label font size. Default value is 11.	Depe ndenc y prope rty	Double	No
LabelFontW eight	Gets or sets the label font weight. Default value is SemiBold.	Depe ndenc y prope rty	FontWeight	No
LabelTextWr apping	Gets or sets the label text wrapping. Default value is NoWrap.	Depe ndenc y prope rty	TextWrapping.NoWrapTextWrapping.WrapTextWrapp ing.WrapWithOverflow	No
LabelWidth	Gets or sets the label width. Default value is lineâ€™s width.	Depe ndenc y prope rty	Double	No
LineBridging Enabled	Gets or sets a value indicating whether line bridging is enabled.	Depe ndenc y prope rty	Boolean (true/ false)	No
FirstSegment Length	Gets or sets the FirstSegmentLy	Depe ndenc y	double	No

	length of the Orthogonal LineConnector	Property		
LastSegmentLength	Gets or sets FirstSegmentLength of the Orthogonal LineConnector	Dependency Property	double	No
AutoAdjustPoints	Gets or sets AutoAdjustPoints of Orthogonal LineConnector.	Dependency Property	Boolean(True/False)	No
LabelPosition	Gets or sets the Position of the LineConnector's Label from the DiagramPage.	Dependency property	Point	Point(0,0)
CustomLabelPosition	Gets or sets the Label of LineConnector is Dragging or Not.	Dependency property	Enum.CustomLabelPositions.AutoCustomLabelPositions.CustomCustomLabelPositions.Drag	CustomLabelPositions.Auto
LabelAngle	Gets or sets the angle of the Label of LineConnector.	Dependency property	double	0
Ports	Gets or sets connection ports for line connectors.	Dependency property	ObservableCollection<ConnectionPort>()	No
HeadDecoratorAngle	Gets or sets the angle at which the head decorator is	CLR property	Double	No

	to be positioned.			
HeadDecoratorPosition	Gets or sets the point where the head decorator is to be positioned.	CLR property	Point	No
TailDecoratorAngle	Gets or sets the angle at which the tail decorator is to be positioned.	CLR property	Double	No
TailDecoratorPosition	Gets or sets the point where the tail decorator is to be positioned.	CLR property	Point	No
IsDecoratorMovable	Gets or sets a value indicating whether the decorator can be moved.	Dependency property	Boolean (true/false)	No
IsDecoratorVisible	Gets or sets a value indicating whether the decorator can be moved.	Dependency property	Boolean (true/false)	No
LabelTextDecoration	Gets or sets a value for decorations for the label.	Dependency property	TextDecorationCollection	No
EnableCumulativeUpdate	Gets or sets a value indicating whether a line can be	Dependency property	Boolean (true/false)	No

	updated cumulatively.			
--	-----------------------	--	--	--

FirstSegmentLength:

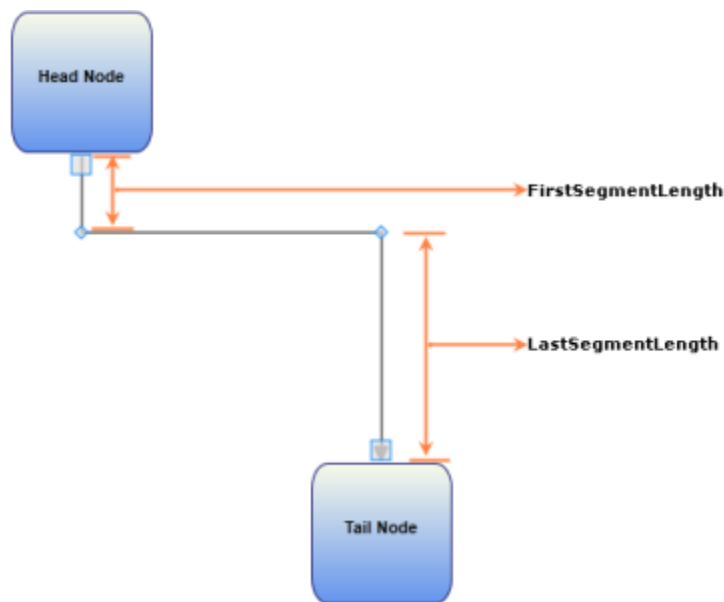
FirstSegmentLength defines the distance between StartPointPosition and the first IntermediatePoint. This property is applicable only for Orthogonal LineConnector whose end points are connected to ports.

LastSegmentLength:

LastSegmentLength defines the distance between the EndPointPosition and the last IntermediatePoint. This property is applicable only for Orthogonal LineConnector whose end points are connected to ports.

AutoAdjustPoints:

AutoAdjustPoints enables the Orthogonal LineConnector to adjust the intermediate points (add, remove, or modify intermediate points) depending upon the ports to which it is connected. This property is applicable only for Orthogonal LineConnector whose end points are connected to ports.



The following is a code snippet that connects two Nodes with a LineConnector with FirstSegmentLength and LastSegmentLength.

C#

```
Node headnode = new Node();
headnode.Shape = Shapes.RoundedRectangle;
headnode.Label = "Head Node";
headnode.Height = 100;
headnode.Width = 100;
headnode.OffsetX = 200;
headnode.OffsetY = 200;
diagramModel.Nodes.Add(headnode);
Node tailnode = new Node();
tailnode.Shape = Shapes.RoundedRectangle;
tailnode.Label = "Tail Node";
tailnode.Height = 100;
tailnode.Width = 100;
```

```

tailnode.OffsetX = 400;
tailnode.OffsetY = 500;
diagramModel.Nodes.Add(tailnode);
ConnectionPort port1 = new ConnectionPort(headnode, new Point(50, 100));
port1.Width = 10;
port1.Height = 10;
port1.PortShape = PortShapes.Circle;
headnode.Ports.Add(port1);
ConnectionPort port2 = new ConnectionPort(tailnode, new Point(50, 0));
port2.Width = 10;
port2.Height = 10;
port2.PortShape = PortShapes.Diamond;
tailnode.Ports.Add(port2);
LineConnector line = new LineConnector();
line.ConnectorType = ConnectorType.Orthogonal;
line.HeadNode = headnode;
line.TailNode = tailnode;
line.ConnectionHeadPort = port1;
line.ConnectionTailPort = port2;
line.FirstSegmentLength = 50;
line.LastSegmentLength = 100;
line.AutoAdjustPoints = true;
diagramModel.Connections.Add(line);

```

VB.NET

```

Dim headnode As New Node()
headnode.Shape = Shapes.RoundedRectangle
headnode.Label = "Head Node"
headnode.Height = 100
headnode.Width = 100
headnode.OffsetX = 200
headnode.OffsetY = 200
diagramModel.Nodes.Add(headnode)
Dim tailnode As New Node()
tailnode.Shape = Shapes.RoundedRectangle
tailnode.Label = "Tail Node"
tailnode.Height = 100
tailnode.Width = 100
tailnode.OffsetX = 400
tailnode.OffsetY = 50
diagramModel.Nodes.Add(tailnode)
Dim port1 As New ConnectionPort(headnode, New Point(50, 100))
port1.Width = 10
port1.Height = 10
port1.PortShape = PortShapes.Circle
headnode.Ports.Add(port1)
Dim port2 As New ConnectionPort(tailnode, New Point(50, 0))
port2.Width = 10
port2.Height = 10
port2.PortShape = PortShapes.Diamond
tailnode.Ports.Add(port2)
Dim line As New LineConnector()
line.ConnectorType = ConnectorType.Orthogonal
line.HeadNode = headnode
line.TailNode = tailnode

```

```

line.ConnectionHeadPort = port1
line.ConnectionTailPort = port2
line.FirstSegmentLength = 50
line.LastSegmentLength = 100
line.AutoAdjustPoints = True
diagramModel.Connections.Add(line)

```

- Connector Type
- Decorator Shapes
- Create Connection Port on Line Connector
- Customize Line Connectors
- Line Connector Label
- Customize the label of Nodes and LineConnectors
- Customize the ContextMenu of Nodes and LineConnectors
- LineBridging

Create Line Connector

Like nodes, connectors can also be added in two ways.

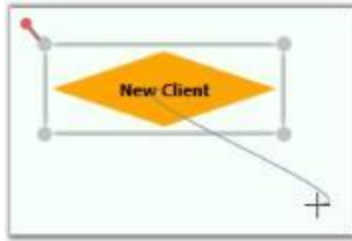
- At run time
- Through model

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
EnableConnection	Gets or sets a value indicating whether [enable connection].	Dependency property	Boolean (true/ false)	No

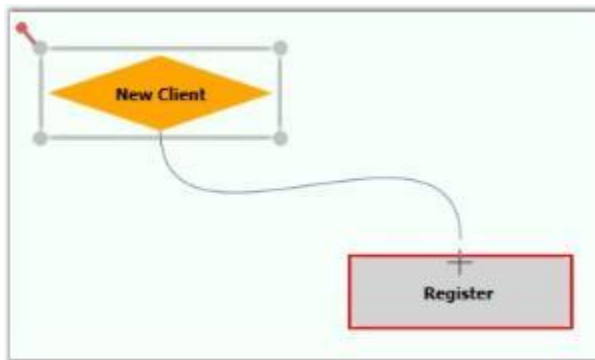
Steps for adding a connector to a diagram at run time

- First set the EnableConnection property of DiagramView to *True*. This has to be set to true every time the user makes a connection.
- Press the left mouse button while the pointer is over the node where the connection is to start. This acts as the head node for the connector.
- While the left button is pressed, drag the pointer to the node to where you want to create a link. The cursor will change to a cross while dragging.

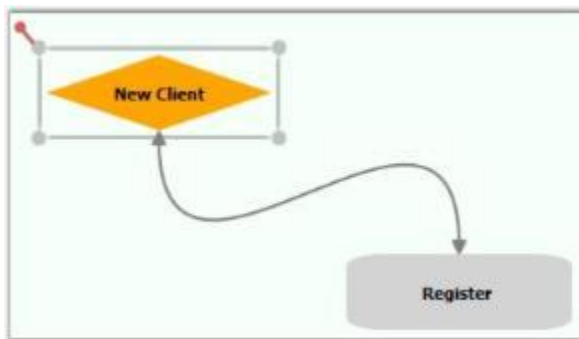
Note: If a node is not hit while making a connection, then no connector gets added.



- When you hit any node in the process, you can see an adorer showing what the link will look like if created.



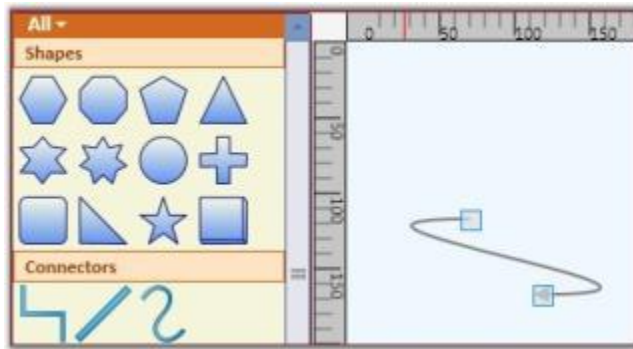
- Release the left button over the target node where you want to connect. This acts as the tail node for the connector and hence the link is created.



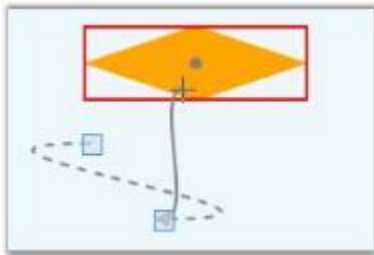
The connector's path geometry is dynamically created based on the start and end points and the connector type.

It is also possible to drag-and-drop line connectors from the SymbolPalette. Three shapes of the line connectors have been added in a group named "Connectors". The desired line can be dragged onto the page. Initially, the head node and the tail node will be null. The steps to be followed to add a line connector from SymbolPalette are as follows:

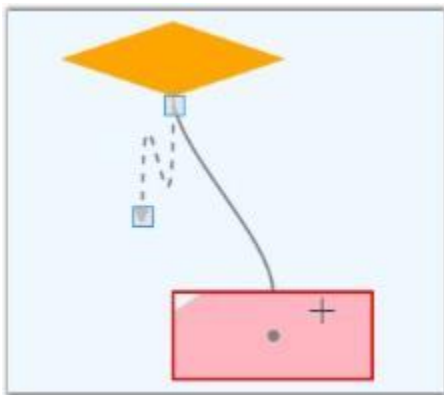
1. Drag the desired line connector onto the page.



2. Then drag the head thumb of the line connector to the desired node to make a connection.



3. Similarly drag the tail thumb of the line connector to the desired node. Now, this creates a link between the two nodes.



Add Connectors through a Model

You can create connections between nodes through a model. The Line Connector class is used to create the connection. We need to specify the head node and the tail node for the connection.

C#

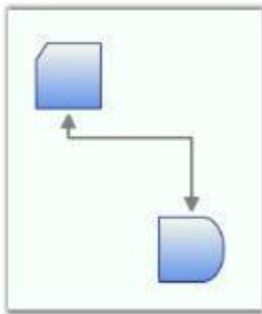
```
Node n1 = new Node();
n1.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n1);
n1.OffsetX = 50;
n1.OffsetY = 50;
Node n2 = new Node();
n2.Shape = Shapes.FlowChart_Delay;
diagramModel.Nodes.Add(n2);
```

```
n2.OffsetX = 150;
n2.OffsetY = 250;
LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
diagramModel.Connections.Add(l1);
```

VB.NET

```
Dim n1 As New Node()
n1.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n1)
n1.OffsetX = 50
n1.OffsetY = 50
Dim n2 As New Node()
n2.Shape = Shapes.FlowChart_Delay
diagramModel.Nodes.Add(n2)
n2.OffsetX = 150
n2.OffsetY = 250
Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
diagramModel.Connections.Add(l1)
```

This creates a connection between the two specified nodes.



Note: For orthogonal and Bezier connectors, the connection always happens at the center of the node's edge.

For straight line connectors, the connection happens at the intersection point of the edge and the line connector.

Setting Constraints for EnableConnection Property

This feature provides the ability to drag nodes and connection ports when the EnableConnection property is set to 'True'. The ConnectionMode enum is used to define the node or connection port for connecting.

Properties

Setting Constraints for EnableConnection Property

Property	Description	Type	Data Type
NodeModePortMode	Allows nodes to be dragged when	DependencyPropertyDependencyProperty	EnumEnum

	EnableConnection is enabled. Allows connection ports to be dragged when EnableConnection is enabled.		
--	--	--	--

Enabling Connection Mode Constraints for Nodes and Connection Ports

NodeMode

This property is used to define whether the node can be movable when the EnableConnection property of the DiagramView is set to true.

This property can be set to the following:

- Connect
- Move

By default, this property is set to Connect.

If the NodeMode property of DiagramView is set to Connect, the diagram nodes can be connected to other nodes.

To set this property to Connect, use the following code.

C#

```
DiagramView View1 = new DiagramView();
View1.NodeMode = ConnectionMode.Connect;
```

VB.NET

```
Dim view1 As New DiagramView()
View1.NodeMode = ConnectionMode.Connect
```

If the NodeMode property of DiagramView is set to Move, even though the EnableConnection is set to 'true' a node can be moved and will not be able to be connected to a line connector.

To set this property to Move, use the following code.

C#

```
DiagramView View1 = new DiagramView();
View1.NodeMode = ConnectionMode.Move;
```

VB.NET

```
Dim view As New DiagramView()
view.NodeMode = ConnectionMode.Move
```

PortMode

This property is used to define whether a connection port can be movable when the EnableConnection property of the DiagramView is set to true.

This property can be set to the following:

- Connect
- Move

By default, this property is set to Connect.

If the PortMode property of the DiagramView is set to Connect, we can create port-to-port connections.

To set the PortMode property to Connect, use the following code.

C#

```
DiagramView View1 = new DiagramView();
View1.PortMode = ConnectionMode.Connect;
```

VB.NET

```
Dim view As New DiagramView()
view.PortMode = ConnectionMode.Connect
```

If the PortMode property of the DiagramView is set to Move, even though EnableConnection is set to true, a port can be moved yet a port-to-port connection can't be created.

To set the PortMode property to Move, use the following code.

C#

```
DiagramView View1 = new DiagramView();
View1.PortMode = ConnectionMode.Move;
```

VB.NET

```
Dim view1 As New DiagramView()
View1.PortMode = ConnectionMode.Move
```

Connector Type

The ConnectorType property specifies the type of connector to be used for connection.

Property	Description	Type	Data Type
ConnectorType	Gets or sets the connector type to be used. There are four values namely Orthogo	Dependency property	ConnectorType.OrthogonalConnectorType.BezierConnectorType.StraightConnectorType.Arc

	nal, Straight, Bezier and Arc can be specified . Default Value is Orthogo nal.		
--	---	--	--

Following types of connectors are supported:

- Orthogonal—creates a line in which line segments (if any) are placed at right angles to each other.
- Bezier—Renders a Bezier curve with two points.
- Straight—renders a line with two points.
- Arc—creates a link between two nodes.

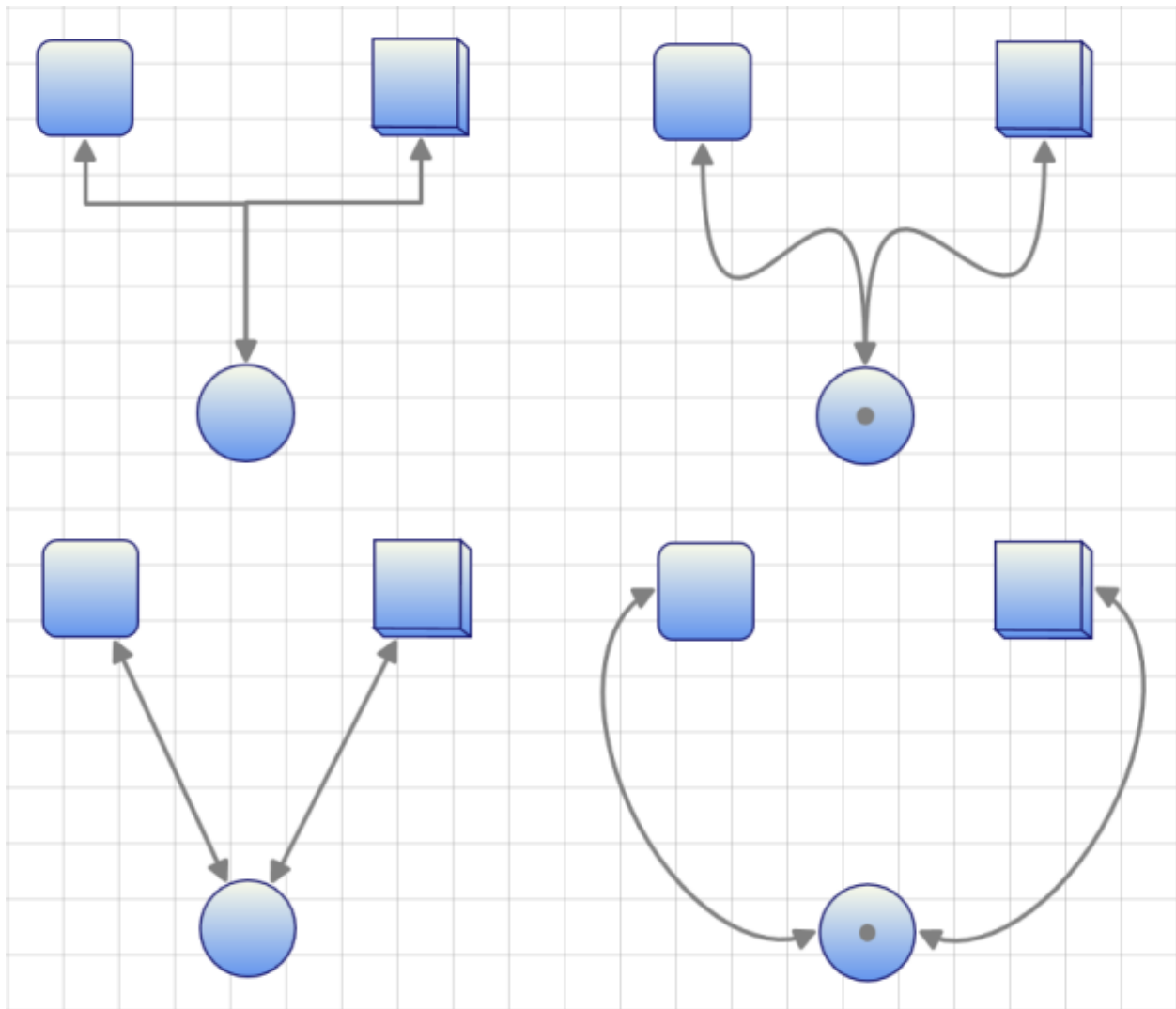
The following code illustrates how to set the connector type:

C#

```
LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
l1.ConnectorType = ConnectorType.Bezier;
diagramModel.Connections.Add(l1);
```

VB.NET

```
Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
l1.ConnectorType = ConnectorType.Bezier
diagramModel.Connections.Add(l1)
```



Arc Line Connector Type

Arc Line Connector creates link between two nodes. This can act as other line connectors like Bezier, Straight and Orthogonal. You can blend the Arc Line Connector and change its angel. Arc height and direction can be customized.

Properties

Property	Description	Type of the property	Data Type	Reference Links
ArcHeight	Gets or sets a value for the height of the Arc connector. The default value is 50.	Dependency property	double	NA
ArcDirection	Gets or sets a value for the direction of the Arc connector. The default value is Clockwise	Dependency property	SweepDirection	NA

Customizing Arc Line Connector type

Use the **ArcHeight** and the **ArcDirection** property of **ConnectorBase** to customize the height and direction of the Arc.

- **ArcHeight** – Gets or Sets the height of the Arc.
- **ArcDirection** – Gets or Sets the direction of the Arc

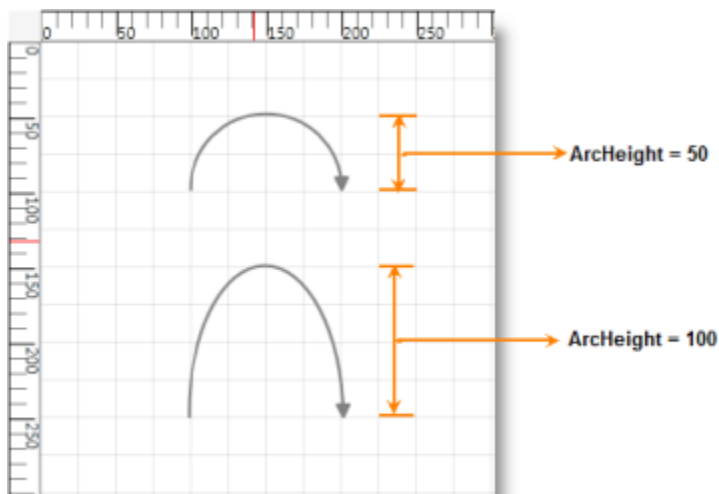
Following code illustrates how to customize the height and direction of the Arc:

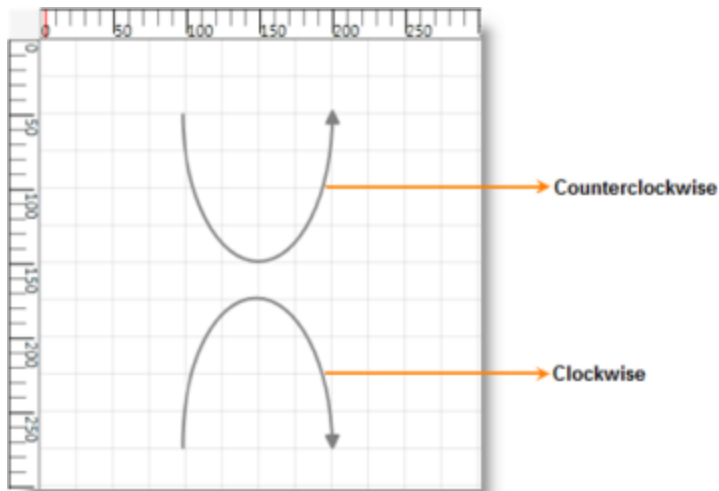
C#

```
LineConnector l = new LineConnector();  
l.ConnectorType = ConnectorType.Arc;  
l.ArcHeight = 100;  
//l.ArcDirection = SweepDirection.Clockwise; // Default  
l.ArcDirection = SweepDirection.Counterclockwise;  
l.StartPointPosition = new Point(50, 150);  
l.EndPointPosition = new Point(150, 150);  
diagramModel1.Connections.Add(l);
```

VB.NET

```
Dim l As New LineConnector()  
l.ConnectorType = ConnectorType.Arc  
l.ArcHeight = 100  
'l.ArcDirection = SweepDirection.Clockwise; // Default  
l.ArcDirection = SweepDirection.Counterclockwise  
l.StartPointPosition = New Point(50, 150)  
l.EndPointPosition = New Point(150, 150)  
diagramModel1.Connections.Add(l)
```





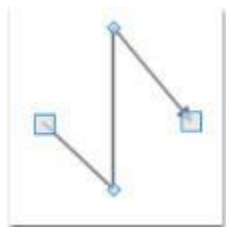
Polyline

Line connector can be used to draw polylines using `IntermediatePoints` property. Polylines are drawn using intermediate points for straight lines and orthogonal line connectors. For orthogonal lines, intermediate points are updated so that the adjacent line segments are always perpendicular to each other. These intermediate points are visually represented as vertex.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
<code>IntermediatePoints</code>	Gets or sets the intermediate points.	Dependency property	<code>List<Point></code>	No

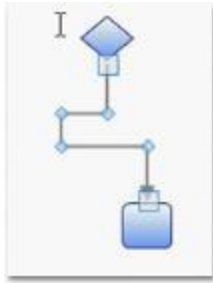
Polylines

Straight line connectors can be used as poly line by using `IntermediatePoints` property. This can be achieved at run time by holding `Ctrl + Shift` and Click on the line, or by simply changing the `IntermediatePoints` collection. This will reflect in the line connector.



Poly Orthogonal Lines

Orthogonal lines can have more than two intermediate points. All these Intermediate points are can be dragged. Unlike straight lines, orthogonal lines maintain their perpendicularity even after the intermediate points are dragged.



Intermediate Points

Adding Intermediate Points

Intermediate points can be added in two ways:

- Using Ctrl + Shift Key
- Through Code Behind

Intermediate points can be added at run time by holding Ctrl + Shift and clicking on the line.

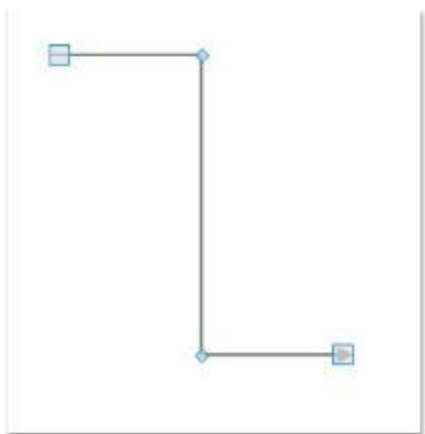
Intermediate points can be added programmatically. The following code snippet illustrates addition of intermediate lines.

C#

```
LineConnector lc = new LineConnector();  
lc.ConnectorType=ConnectorType.Straight;  
lc.StartPointPosition = new Point(100, 100);  
lc.EndPointPosition = new Point(300, 300);  
lc.IntermediatePoints.Add(new Point(200,100));  
lc.IntermediatePoints.Add(new Point(200,300));
```

VB.NET

```
Dim lc As New LineConnector()  
lc.ConnectorType=ConnectorType.Straight  
lc.StartPointPosition = New Point(100, 100)  
lc.EndPointPosition = New Point(300, 300)  
lc.IntermediatePoints.Add(New Point(200,100))  
lc.IntermediatePoints.Add(New Point(200,300))
```



Note: By default, an orthogonal line connector has two intermediate points. So, newly added intermediate points will need to be added beyond the default intermediate points. If users need to add new intermediate points to the line connector only, they must clear all of the default intermediate points and then add the intermediate points. Otherwise, the newly added intermediate points will be added beyond the default intermediate points.

Modifying Intermediate Points

Intermediate Points can be modified in two ways:

- Dragging the Vertex
- Through Code Behind

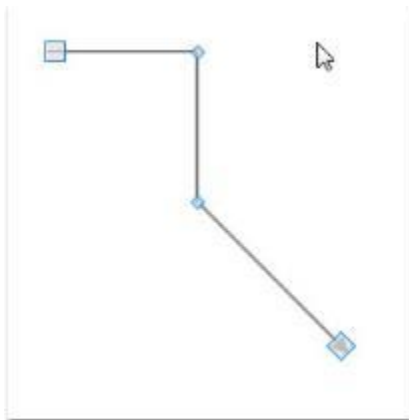
Intermediate points can be modified at run time by clicking and dragging the vertex of the line connector. Intermediate points can be modified programmatically also. The following code snippet illustrates modification of intermediate lines.

C#

```
LineConnector lc = new LineConnector();  
lc.ConnectorType=ConnectorType.Straight;  
lc.StartPointPosition = new Point(100, 100);  
lc.EndPointPosition = new Point(300, 300);  
lc.IntermediatePoints.Add(new Point(200,100));  
lc.IntermediatePoints.Add(new Point(200,300));  
lc.IntermediatePoints[1] = new Point(200,200);
```

VB.NET

```
Dim lc As New LineConnector()  
lc.ConnectorType=ConnectorType.Straight  
lc.StartPointPosition = New Point(100, 100)  
lc.EndPointPosition = New Point(300, 300)  
lc.IntermediatePoints.Add(New Point(200,100))  
lc.IntermediatePoints.Add(New Point(200,300))  
lc.IntermediatePoints(1) = New Point(200,200)
```



Delete Intermediate Points

Intermediate points can be deleted in two ways:

- Using Ctrl + Shift Key
- Through Code Behind

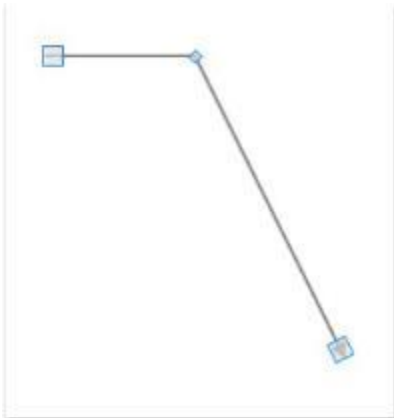
Intermediate points can be deleted by holding Ctrl + Shift and clicking on the vertex that represents intermediate point to be deleted. Intermediate points can be deleted programmatically also. The following code example illustrates deletion of intermediate lines.

C#

```
LineConnector lc = new LineConnector();
lc.ConnectorType=ConnectorType.Straight;
lc.StartPointPosition = new Point(100, 100);
lc.EndPointPosition = new Point(300, 300);
lc.IntermediatePoints.Add(new Point(200,100));
lc.IntermediatePoints.Add(new Point(200,300));
lc.IntermediatePoints.RemoveAt(1);
```

VB.NET

```
Dim lc As New LineConnector()
lc.ConnectorType=ConnectorType.Straight
lc.StartPointPosition = New Point(100, 100)
lc.EndPointPosition = New Point(300, 300)
lc.IntermediatePoints.Add(New Point(200,100))
lc.IntermediatePoints.Add(New Point(200,300))
lc.IntermediatePoints.RemoveAt(1)
```



Vertex Template for Intermediate Points

Vertex template for intermediate points can be set using ConnectorAdornerVertexStyle property of line connector. Custom styles can be set. The following code example illustrates the same.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
VertexStyle	Gets or sets the vertex style.	Dependency property	Style	No

HTML

```

<Window.Resources>
<ResourceDictionary>
<Style x:Key="vertexStyle" TargetType="{x:Type Thumb}">
<Setter Property="Width" Value="7"/>
<Setter Property="Height" Value="7"/>
<Setter Property="SnapsToDevicePixels" Value="true"/>
<Setter Property="RenderTransform">
<Setter.Value>
<TranslateTransform X="-3" Y="-3"/>
</Setter.Value>
</Setter>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Thumb}">
<Rectangle RenderTransformOrigin="0.5,0.5" Fill="Beige" Stroke="Black"
StrokeThickness="1" RadiusX="0" RadiusY="0">
<Rectangle.RenderTransform>
<RotateTransform Angle="45"/>
</Rectangle.RenderTransform>
</Rectangle>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>
</Window.Resources>

```

C#

```

LineConnector lc = new LineConnector();
lc.ConnectorType = ConnectorType.Straight;
lc.StartPointPosition = new Point(100, 100);
lc.EndPointPosition = new Point(300, 300);
lc.IntermediatePoints.Add(new Point(200, 100));
lc.IntermediatePoints.Add(new Point(200, 300));
lc.VertexStyle = this.Resources["vertexStyle"] as Style;

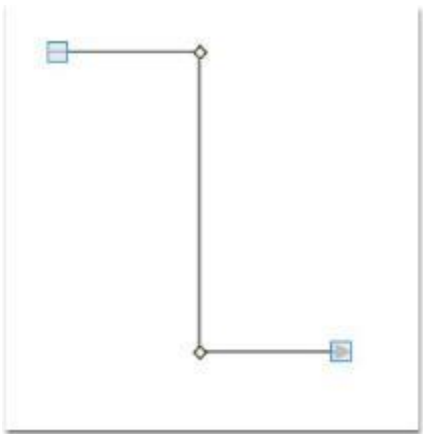
```

VB.NET

```

Dim lc As New LineConnector()
lc.ConnectorType = ConnectorType.Straight
lc.StartPointPosition = New Point(100, 100)
lc.EndPointPosition = New Point(300, 300)
lc.IntermediatePoints.Add(New Point(200, 100))
lc.IntermediatePoints.Add(New Point(200, 300))
lc.VertexStyle = TryCast(Me.Resources("vertexStyle"), Style)

```



Template for End Points

Vertex template for terminal points can be set using ConnectorAdornerThumbStyle property of line connector. Custom styles can be set. Following code example illustrates the same.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
DecoratorAdornerStyle	Gets or sets the decorator adorner style.	Dependency property	Style	No

HTML

```
<Window.Resources>
<ResourceDictionary>
<Style x:Key="decrator" TargetType="{x:Type Thumb}">
<Setter Property="Width" Value="14"/>
<Setter Property="Height" Value="14"/>
<Setter Property="SnapsToDevicePixels" Value="true"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Thumb}">
<Rectangle Fill="#66F5F5DC" Stroke="Black" StrokeThickness="1" RadiusX="0"
RadiusY="0"/>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>
</Window.Resources>
```

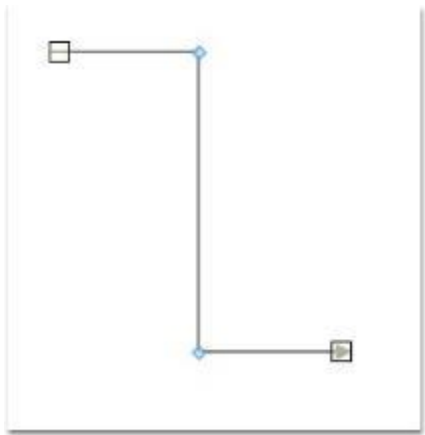
C#

```
LineConnector lc = new LineConnector();
lc.ConnectorType = ConnectorType.Straight;
lc.StartPointPosition = new Point(100, 100);
lc.EndPointPosition = new Point(300, 300);
```

```
lc.IntermediatePoints.Add(new Point(200, 100));
lc.IntermediatePoints.Add(new Point(200, 300));
lc.DecoratorAdornerStyle = this.Resources["decorator"] as Style;
```

VB.NET

```
Dim lc As New LineConnector()
lc.ConnectorType = ConnectorType.Straight
lc.StartPointPosition = New Point(100, 100)
lc.EndPointPosition = New Point(300, 300)
lc.IntermediatePoints.Add(New Point(200, 100))
lc.IntermediatePoints.Add(New Point(200, 300))
lc.DecoratorAdornerStyle = TryCast(Me.Resources("decorator"), Style)
```



Hide Vertex

The user can hide the vertex of a line connector by setting the `IsVertexVisible` property to `False`. The following code snippet illustrates the same.

Property Table

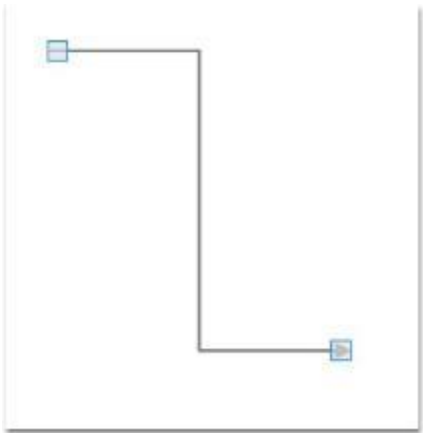
Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
<code>IsVertexVisible</code>	Gets or sets a value indicating whether this instance is vertex visible.	Dependency property	Style	No

C#

```
LineConnector lc = new LineConnector();
lc.ConnectorType = ConnectorType.Straight;
lc.StartPointPosition = new Point(100, 100);
lc.EndPointPosition = new Point(300, 300);
lc.IntermediatePoints.Add(new Point(200, 100));
lc.IntermediatePoints.Add(new Point(200, 300));
lc.IsVertexVisible = false;
```

VB.NET

```
Dim lc As New LineConnector()  
lc.ConnectorType = ConnectorType.Straight  
lc.StartPointPosition = New Point(100, 100)  
lc.EndPointPosition = New Point(300, 300)  
lc.IntermediatePoints.Add(New Point(200, 100))  
lc.IntermediatePoints.Add(New Point(200, 300))  
lc.IsVertexVisible = False
```



Arresting Vertex Drag

You can disable the drag operation on the vertex of a line connector by setting the `IsVertexMovable` property to `False`. The following code example illustrates the same.

Property Table

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
IsVertexMovable	Gets or sets a value indicating whether this instance is vertex movable.	Dependency property	Boolean (true/ false)	No

C#

```
LineConnector lc = new LineConnector();  
lc.ConnectorType = ConnectorType.Straight;  
lc.StartPointPosition = new Point(100, 100);  
lc.EndPointPosition = new Point(300, 300);  
lc.IntermediatePoints.Add(new Point(200, 100));  
lc.IntermediatePoints.Add(new Point(200, 300));  
lc.IsVertexMovable = false;
```


VB.NET

```

Dim lc As New LineConnector()
lc.ConnectorType = ConnectorType.Straight
lc.StartPointPosition = New Point(100, 100)
lc.EndPointPosition = New Point(300, 300)
lc.IntermediatePoints.Add(New Point(200, 100))
lc.IntermediatePoints.Add(New Point(200, 300))
lc.IsVertexMovable = False

```

The vertex drag of a line connector is arrested.

Decorator Shapes

Head and tail decorator shape properties provide an option to add arrows and to customize these arrows. End point decorators can be provided for all types of connectors. There are a number of shapes available for head and tail decorators.

Property Table

Property	Description	Type of the property	Value it accepts	Any other dependencies / sub properties associated
HeadDecoratorShape	Gets or sets the head decorator shape of the connection. Four values namely None, Arrow, Diamond and Circle can be specified. Default value: HeadDecoratorShape.None	CLR property	DecoratorShape.None DecoratorShape.Arrow DecoratorShape.Diamond DecoratorShape.Circle* DecoratorShape.Custom	No
TailDecoratorShape	Gets or sets the tail decorator shape of the connection. Four values namely None, Arrow, Diamond and Circle can be specified. Default value: TailDecoratorShape.Arrow	CLR property	DecoratorShape.None DecoratorShape.Arrow DecoratorShape.Diamond DecoratorShape.Circle* DecoratorShape.Custom	No
HeadDecoratorStyle	Provides customization option for the head decorator shape.	CLR property	DecoratorStyle	No
TailDecoratorStyle	Provides customization option for the tail decorator shape.	CLR property	DecoratorStyle	No

CustomHeadDecoratorStyle	Provides option for Custom Head Decorator Style for LineConnector	Style	Dependency Property	No
CustomTailDecoratorStyle	Provides option for Custom Tail Decorator Style for LineConnector	Style	Dependency Property	No

Arrow settings can be changed using HeadDecoratorShape and TailDecoratorShape properties. Both head and tail decorators consist of the same set of properties that allow one to customize the settings as required.

Types of decorator shapes

- Arrow
- Diamond
- Circle

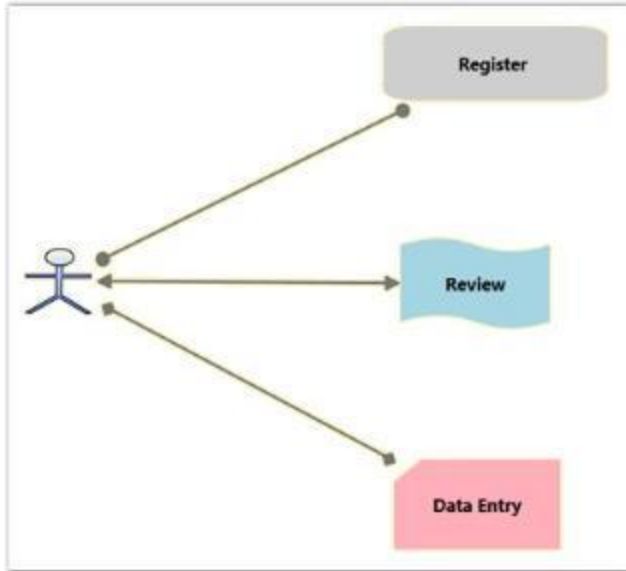
The following code shows how to set these properties.

C#

```
LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
l1.ConnectorType = ConnectorType.Bezier;
l1.HeadDecoratorShape = DecoratorShape.Diamond;
l1.TailDecoratorShape = DecoratorShape.Circle;
diagramModel.Connections.Add(l1);
```

VB.NET

```
Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
l1.ConnectorType = ConnectorType.Bezier
l1.HeadDecoratorShape = DecoratorShape.Diamond
l1.TailDecoratorShape = DecoratorShape.Circle
diagramModel.Connections.Add(l1)
```



CustomDecoratorShape

Decorator shapes can be customized in two ways.

- Using CustomHeadDecoratorStyle and CustomTailDecoratorStyle
- Using HeadDecoratorStyle and TailDecoratorStyle

CustomDecoratorShape using Style and Setters

HeadDecoratorShape and TailDecoratorShape can be customized by defining a required style for CustomHeadDecoratorStyle, and CustomTailDecoratorStyle respectively.

HTML

```

<Window.Resources>
<Style TargetType="{x:Type Path}" x:Key="Deco1">
<Setter Property="Data" Value="M 9,2 11,7 17,7 12,10 14,15 9,12 4,15 6,10
1,7 7,7 Z"></Setter>
<Setter Property="Width" Value="20"/>
<Setter Property="Fill" Value="MidnightBlue" />
<Setter Property="Height" Value="20"/>
</Style>
</Window.Resources>
  
```

HTML

```

<syncfusion:LineConnector ConnectorType="Bezier" Label="Line1"
LabelWidth="50" IsSelected="True" StartPointPosition="100,400"
EndPointPosition="100,500" HeadDecoratorShape="Custom"
CustomHeadDecoratorStyle="{StaticResource Deco1}"/>
  
```

C#

```

LineConnector l1 = new LineConnector();
l1.ConnectorType = ConnectorType.Bezier;
l1.HeadDecoratorShape = DecoratorShape.Custom;
  
```

```
l1.StartPointPosition = New Point(100, 100);
l1.EndPointPosition = New Point(200, 200);
l1.CustomHeadDecoratorStyle = this.Resources["deco1"] as Style;
diagramModel.Connections.Add(l1);
```

VB.NET

```
Dim l1 As New LineConnector()
l1.ConnectorType = ConnectorType.Bezier
l1.HeadDecoratorShape = DecoratorShape.Custom
l1.StartPointPosition = New Point(100, 100)
l1.EndPointPosition = New Point(200, 200)
l1.CustomHeadDecoratorStyle = TryCast(Me.Resources("Deco1"), Style)
diagramModel.Connections.Add(l1)
```

**HTML**

```
<Window.Resources>
<Style TargetType="{x:Type Path}" x:Key="Deco1">
<Setter Property="Data" Value="M 9,2 11,7 17,7 12,10 14,15 9,12 4,15 6,10
1,7 7,7 Z"></Setter>
<Setter Property="Width" Value="20"/>
<Setter Property="Fill" Value="MidnightBlue" />
<Setter Property="Height" Value="20"/>
</Style>
</Window.Resources>
```

HTML

```
<syncfusion:LineConnector ConnectorType="Orthogonal" Name="Line1"
LabelWidth="50" IsSelected="True" StartPointPosition="100,400"
EndPointPosition="100,500" TailDecoratorShape="Custom"
CustomTailDecoratorStyle="{StaticResource Deco1}"/>
```

C#

```
LineConnector l1 = new LineConnector();
l1.ConnectorType = ConnectorType.Orthogonal;
l1.HeadDecoratorShape = DecoratorShape.Custom;
l1.StartPointPosition = New Point(100, 100);
l1.EndPointPosition = New Point(200, 200);
l1.CustomTailDecoratorStyle = this.Resources["deco1"] as Style;
diagramModel.Connections.Add(l1);
```

VB.NET

```

Dim l1 As New LineConnector()
l1.ConnectorType = ConnectorType. Orthogonal
l1.TailDecoratorShape = DecoratorShape.Custom
l1.StartPointPosition = New Point(100, 100)
l1.EndPointPosition = New Point(200, 200)
l1.TailHeadDecoratorStyle = TryCast(Me.Resources("Deco1"), Style)
diagramModel.Connections.Add(l1)

```



CustomDecoratorShape using DecoratorStyle

HeadDecoratorShape and TailDecoratorShape can be customized by defining a required style for HeadDecoratorStyle and TailDecoratorStyle respectively.

C#

```

LineConnector line1 = new LineConnector();
line1.ConnectorType = ConnectorType.Straight;
line1.HeadDecoratorShape = DecoratorShape.Custom;
line1.HeadDecoratorStyle = new DecoratorStyle()
{
    Fill = Brushes.Red,
    Data = Geometry.Parse("M160.329212944922,-3.01862318403769L-2.5,130.5 - 5.00506481618589,311.336343115124 163.335290725089,448.012415349887 436.888368720338,448.012420654297 593,309 591.5,133.5 429.5,-1.5z"),
    Stroke = Brushes.Black,
    Height=20,
    Width=20
};
line1.StartPointPosition = new Point(100, 100);
line1.EndPointPosition = new Point(200, 200);
diagramModel.Connections.Add(line1);

```

VB.NET

```

Dim line1 As New LineConnector()
line1.ConnectorType = ConnectorType.Straight
line1.HeadDecoratorShape = DecoratorShape.[Custom]
line1.HeadDecoratorStyle = New DecoratorStyle() With { _
    .Fill = Brushes.Red, _
    .Data = Geometry.Parse("M160.329212944922,-3.01862318403769L-2.5,130.5 - 5.00506481618589,311.336343115124 163.335290725089,448.012415349887 436.888368720338,448.012420654297 593,309 591.5,133.5 429.5,-1.5z"), _
    .Stroke = Brushes.Black, _
    .Height = 20, _
    .Width = 20 _
}

```

```

}
line1.StartPointPosition = New Point(100, 100)
line1.EndPointPosition = New Point(200, 200)
diagramModel.Connections.Add(line1)

```



Customizing TailDecoratorShape with TailDecoratorStyle

C#

```

LineConnector line1 = new LineConnector();
line1.ConnectorType = ConnectorType.Bezier;
line1.TailDecoratorShape = DecoratorShape.Custom;
line1.TailDecoratorStyle = new DecoratorStyle()
{
    Fill = Brushes.Red,
    Data = Geometry.Parse("M160.329212944922,-3.01862318403769L-2.5,130.5 - 5.00506481618589,311.336343115124 163.335290725089,448.012415349887 436.888368720338,448.012420654297 593,309 591.5,133.5 429.5,-1.5z"),
    Stroke = Brushes.Black,
    Height=20,
    Width=20
};
line1.StartPointPosition = new Point(200, 200);
line1.EndPointPosition = new Point(100, 100);
diagramModel.Connections.Add(line1);

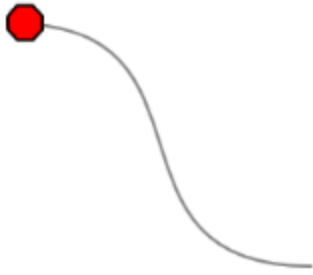
```

VB.NET

```

Dim line1 As New LineConnector()
line1.ConnectorType = ConnectorType.Bezier
line1.TailDecoratorShape = DecoratorShape.[Custom]
line1.TailDecoratorStyle = New DecoratorStyle() With
{
    .Fill = Brushes.Red, _
    .Data = Geometry.Parse("M160.329212944922,-3.01862318403769L-2.5,130.5 - 5.00506481618589,311.336343115124 163.335290725089,448.012415349887 436.888368720338,448.012420654297 593,309 591.5,133.5 429.5,-1.5z"), _
    .Stroke = Brushes.Black, _
    .Height = 20, _
    .Width = 20 _
}
line1.StartPointPosition = New Point(200, 200)
line1.EndPointPosition = New Point(100, 100)
diagramModel.Connections.Add(line1)

```



SegmentDecoratorShape

SegmentDecoratorShape provides the option to place decorator shapes for each segment of Straight and Orthogonal lines.

SegmentDecorator Shape

Property	Description	Type	Data Type	Reference links
Unit	Gets or sets the Unit for the SegmentDecoratorSettingsDefault Value is RelativeFraction	Dependency property	EnumLineUnit. AbsoluteFractionLineUnit. RelativeFractionLineUnit. AbsoluteValueLineUnit. RelativeValue	No
DecoratorOffset	Gets or sets DecoratorOffset for the SegmentDecorator	Property	Double	No
DecoratorShape	Gets or sets DecoratorShape for the SegmentDecorator	Property	DecoratorShape	No
CustomDecoratorStyle	Gets or sets CustomDecoratorStyle for the SegmentDecorator	Property	Style	No

To set unit property for Segment decorator setting

LineUnit property is used to access the following:

- AbsoluteFraction: the fraction values (double value between 0 to 1) entered are considered from the particular segment's StartPointPosition
- RelativeFraction: the fraction values (double value between 0 to 1) entered are considered from the previous DecoratorShape position
- AbsoluteValue: the pixel values (double) entered are considered from the particular segment's StartPointPosition
- RelativeValue: the pixel values (double) entered are considered from the previous DecoratorShape position

Through XAML

HTML

```
<syncfusion:LineConnector.SegmentDecoratorSettings>
<syncfusion:SegmentDecoratorSettings Unit="RelativeFraction">
<syncfusion:SegmentDecoratorSettings.SegmentDecorator>
<syncfusion:CollectionExt>
<syncfusion:SegmentDecorator DecoratorOffset="0.2" DecoratorShape="Arrow" />
<syncfusion:SegmentDecorator DecoratorOffset="0.5"
DecoratorShape="Diamond"/>
</syncfusion:CollectionExt>
</syncfusion:SegmentDecoratorSettings.SegmentDecorator>
</syncfusion:SegmentDecoratorSettings>
</syncfusion:LineConnector.SegmentDecoratorSettings>
```

C#

```
Through code behind
SegmentDecorator decorator1=new SegmentDecorator ();
decorator1.DecoratorOffset=0.2;
decorator1.DecoratorShape=Syncfusion.Windows.Diagram.DecoratorShape.Arrow;
SegmentDecorator decorator2=new SegmentDecorator ();
decorator2.DecoratorOffset=0.5;
decorator2.DecoratorShape=Syncfusion.Windows.Diagram.DecoratorShape.Diamond;
CollectionExt collection=new CollectionExt ();
collection.Add(decorator1);
collection.Add(decorator2);
SegmentDecoratorSettings decoratorsettings = new SegmentDecoratorSettings();
decoratorsettings.Unit = LineUnit.RelativeFraction;
decoratorsettings.SegmentDecorator = collection;
line.SegmentDecoratorSettings = decoratorsettings;
```

*DecoratorShape Customization***DecoratorStyle**

The decorator shapes used for the connector can be customized by specifying the property values under the CustomDecoratorStyle property. Same Style and shapes can be given to an entire line segment or different shapes and styles can be given to each part of the line segment, the various properties under the CustomDecoratorStyle property are as follows:

- Fill - Specifies the color to be used to fill the decorator
- StrokeThickness - Specifies the thickness value for the decorator's border
- Stroke - Specifies the color to be used for the border of the decorator
- StrokeStartLineCap - Specifies the shape used at the start of a line or segment
- StrokeEndLineCap - Specifies the shape at the end of a line or segment
- StrokeLineJoin - Specifies the shape that joins two lines or segments
- StrokeDashArray - Specifies a collection of double values that indicate the pattern of dashes and gaps used to outline shapes.
- Width- Specifies the Width of the shape
- Height- Specifies the Height of the shape

To set the same Style and Shape for the entire Line segment

HTML


```
<syncfusion:SegmentDecoratorSettings.CustomDecoratorStyle>
<Style TargetType="Path">
<Setter Property="Stretch" Value="Fill"/>
<Setter Property="Fill" Value="Red"/>
<Setter Property="Width" Value="15"/>
<Setter Property="Height" Value="15"/>
<Setter Property="Stroke" Value="Black"/>
<Setter Property="StrokeThickness" Value="2"/>
</Style>
</syncfusion:SegmentDecoratorSettings.CustomDecoratorStyle>
```

C#

```
Style decoratorstyle = new System.Windows.Style();
decoratorstyle.BasedOn = decoratorsettings.CustomDecoratorStyle;
decoratorstyle.TargetType = typeof(Path);
decoratorstyle.Setters.Add(new Setter(Path.StrokeProperty, new
SolidColorBrush(Colors.LightSteelBlue)));
decoratorstyle.Setters.Add(new Setter(Path.StrokeThicknessProperty, 2d));
decoratorstyle.Setters.Add(new Setter(Path.WidthProperty, 15d));
decoratorstyle.Setters.Add(new Setter(Path.HeightProperty, 15d));
decoratorstyle.Setters.Add(new Setter(Path.FillProperty, new
SolidColorBrush(Colors.YellowGreen)));
decoratorsettings.CustomDecoratorStyle = decoratorstyle;
line.SegmentDecoratorSettings = decoratorsettings;
```

To set different shapes and styles to the line segment

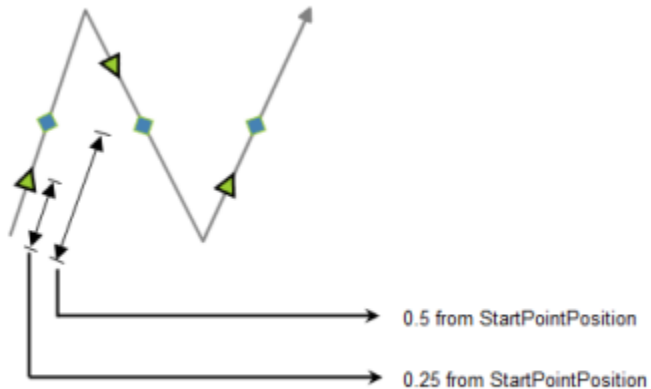
HTML

```
<syncfusion:SegmentDecoratorSettings.SegmentDecorator>
<syncfusion:CollectionExt>
<syncfusion:SegmentDecorator DecoratorOffset="0.2" DecoratorShape="Arrow"
CustomDecoratorStyle="{StaticResource de}" />
<syncfusion:SegmentDecorator DecoratorOffset="0.5"
DecoratorShape="Diamond"/>
</syncfusion:CollectionExt>
</syncfusion:SegmentDecoratorSettings.SegmentDecorator>
```

C#

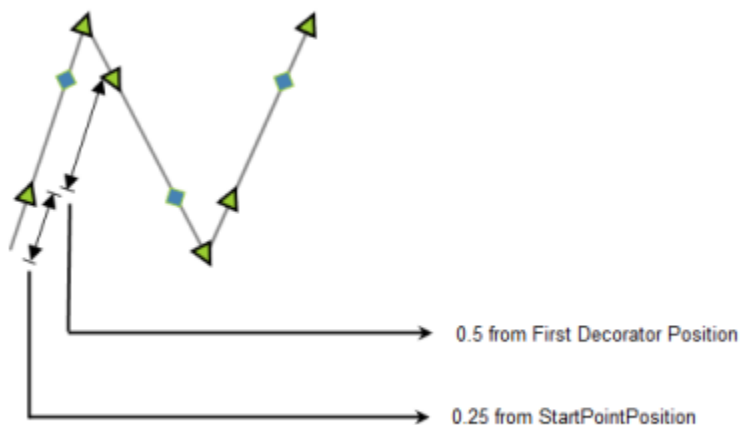
```
Style decoratorstyle = new System.Windows.Style();
decoratorstyle.BasedOn = decoratorsettings.CustomDecoratorStyle;
decoratorstyle.TargetType = typeof(Path);
decoratorstyle.Setters.Add(new Setter(Path.StrokeProperty, new
SolidColorBrush(Colors.LightSteelBlue)));
decoratorstyle.Setters.Add(new Setter(Path.StrokeThicknessProperty, 2d));
decoratorstyle.Setters.Add(new Setter(Path.WidthProperty, 15d));
decoratorstyle.Setters.Add(new Setter(Path.HeightProperty, 15d));
decoratorstyle.Setters.Add(new Setter(Path.FillProperty, new
SolidColorBrush(Colors.YellowGreen)));
decoratorsettings.CustomDecoratorStyle = decoratorstyle;
line.SegmentDecoratorSettings = decoratorsettings;
```

Appearance



HTML

```
<syncfusion:LineConnector.SegmentDecoratorSettings>
<syncfusion:SegmentDecoratorSettings Unit="AbsoluteFraction">
<syncfusion:SegmentDecoratorSettings.SegmentDecorator>
<syncfusion:CollectionExt>
<syncfusion:SegmentDecorator DecoratorOffset="0.25" DecoratorShape="Arrow"
/>
<syncfusion:SegmentDecorator DecoratorOffset="0.5"
DecoratorShape="Diamond"/>
</syncfusion:CollectionExt>
</syncfusion:SegmentDecoratorSettings.SegmentDecorator>
</syncfusion:SegmentDecoratorSettings>
</syncfusion:LineConnector.SegmentDecoratorSettings>
```



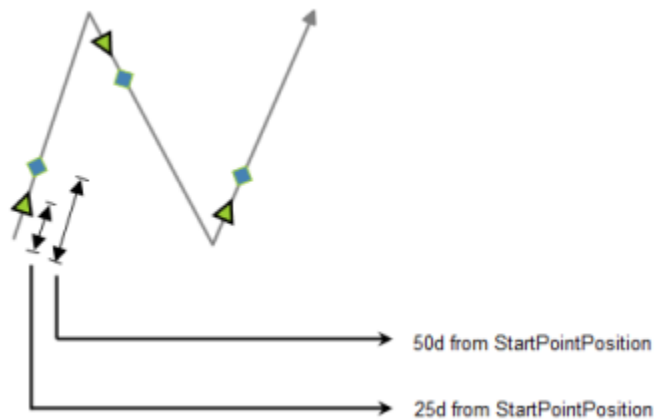
HTML

```
<syncfusion:LineConnector.SegmentDecoratorSettings>
```

```

<syncfusion:SegmentDecoratorSettings Unit="RelativeFraction">
  <syncfusion:SegmentDecoratorSettings.SegmentDecorator>
    <syncfusion:CollectionExt>
      <syncfusion:SegmentDecorator DecoratorOffset="0.25" DecoratorShape="Arrow"
      />
      <syncfusion:SegmentDecorator DecoratorOffset="0.5"
      DecoratorShape="Diamond"/>
    </syncfusion:CollectionExt>
  </syncfusion:SegmentDecoratorSettings.SegmentDecorator>
</syncfusion:SegmentDecoratorSettings>
</syncfusion:LineConnector.SegmentDecoratorSettings>

```

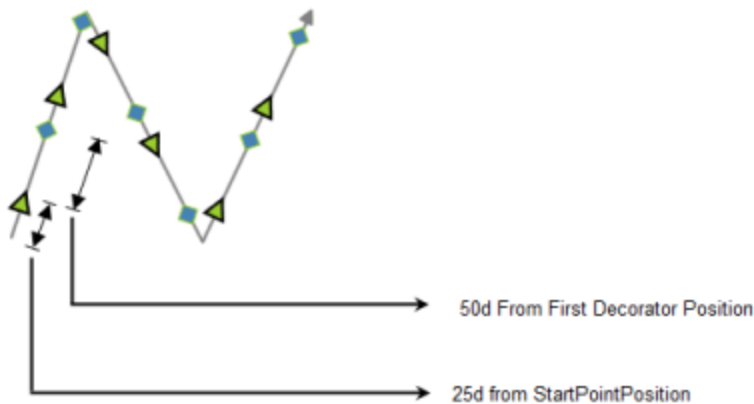


HTML

```

<syncfusion:LineConnector.SegmentDecoratorSettings>
  <syncfusion:SegmentDecoratorSettings Unit="AbsoluteValue">
    <syncfusion:SegmentDecoratorSettings.SegmentDecorator>
      <syncfusion:CollectionExt>
        <syncfusion:SegmentDecorator DecoratorOffset="25" DecoratorShape="Arrow" />
        <syncfusion:SegmentDecorator DecoratorOffset="50" DecoratorShape="Diamond"/>
      </syncfusion:CollectionExt>
    </syncfusion:SegmentDecoratorSettings.SegmentDecorator>
  </syncfusion:SegmentDecoratorSettings>
</syncfusion:LineConnector.SegmentDecoratorSettings>

```



HTML

```
<syncfusion:LineConnector.SegmentDecoratorSettings>
  <syncfusion:SegmentDecoratorSettings Unit="RelativeValue">
    <syncfusion:SegmentDecoratorSettings.SegmentDecorator>
      <syncfusion:CollectionExt>
        <syncfusion:SegmentDecorator DecoratorOffset="25" DecoratorShape="Arrow" />
        <syncfusion:SegmentDecorator DecoratorOffset="50" DecoratorShape="Diamond"/>
      </syncfusion:CollectionExt>
    </syncfusion:SegmentDecoratorSettings.SegmentDecorator>
  </syncfusion:SegmentDecoratorSettings>
</syncfusion:LineConnector.SegmentDecoratorSettings>
```

Customize Line Connectors

This topic describes two properties:

- LineStyle
- DecoratorStyle

Line Style

A connector can be customized by specifying the values under the LineStyle property. The various properties under LineStyle are,

- Fill - Specifies the color used to fill the connector.
- StrokeThickness - Specifies the thickness value for the connector's border.
- Stroke - Specifies the color used for the border of the connector.
- StrokeStartLineCap - Specifies the shape to be used at the start of a line or segment.
- StrokeEndLineCap - Specifies the shape at the end of a line or segment.
- StrokeLineJoin - Specifies the shape that joins two lines or segments.
- StrokeDashArray - Specifies a collection of double values that indicate the pattern of dashes and gaps used to outline shapes.

As an example, the Stroke property can be applied as follows.

C#

```

LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
l1.ConnectorType = ConnectorType.Bezier;
l1.LineStyle.Stroke = Brushes.Red;
diagramModel.Connections.Add(l1);

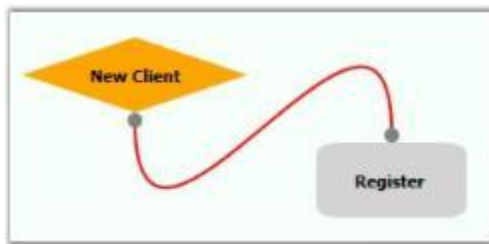
```

VB.NET

```

Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
l1.ConnectorType = ConnectorType.Bezier
l1.LineStyle.Stroke = Brushes.Red
diagramModel.Connections.Add(l1)

```



CustomPathStyle

A connector can be customized using CustomPathStyle. The CustomPathStyle property enables you to customize the appearance of LineConnector.

Properties

Property Table

Property	Description	Type	Data Type	Reference links
CustomPathStyle	Get or Set CustomPathStyle for LineConnector	Dependency Property	Style	NA

Applying Style for CustomPathStyle

Appearance of the LineConnector can be customized by applying style for the CustomPathStyle property. Style can be applied for CustomPathStyle as illustrated in the following code:

- Through XAML

HTML

```

<Window.Resources>
<Style TargetType="{x:Type Path}" x:Key="Deco1">
<Setter Property="Stroke" Value="Red" />
</Style>
</Window.Resources>
<Style TargetType="{x:Type syncfusion:LineConnector}" >
<Setter Property="HeadDecoratorShape" Value="Diamond" />

```

```
<Setter Property="TailDecoratorShape" Value="Diamond"/>
<!--set the stroke color-->
<Setter Property="CustomPathStyle"
Value="{StaticResource Decol}">
</Setter>
</Style>
```

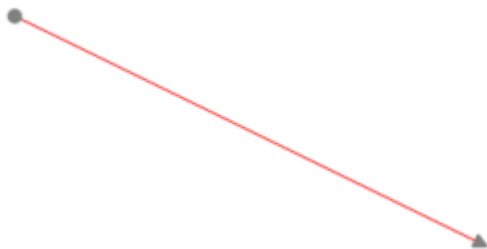
- Through Code-behind

C#

```
LineConnector l1 = new LineConnector();
l1.ConnectorType = ConnectorType.Bezier;
l1.HeadDecoratorShape = DecoratorShape.Custom;
l1.StartPointPosition = new Point(100, 100);
l1.EndPointPosition = new Point(200, 200);
l1.CustomPathStyle = this.Resources["Decol"] as Style;
diagramModel.Connections.Add(l1);
```

VB.NET

```
Dim l1 As New LineConnector()
l1.ConnectorType = ConnectorType.Bezier
l1.HeadDecoratorShape = DecoratorShape.Custom
l1.StartPointPosition = New Point(100, 100)
l1.EndPointPosition = New Point(200, 200)
l1.CustomPathStyle = TryCast(Me.Resources("Decol"), Style)
diagramModel.Connections.Add(l1)
```



DecoratorStyle

The decorator shapes used for the connector can be customized by specifying the property values under the `DecoratorStyle` property. To change the decorator style, the `HeadDecoratorStyle` and `TailDecoratorStyle` properties can be used.

The various properties under the `DecoratorStyle` property are as follows.

- `Fill` - Specifies the color used to fill the connector.
- `StrokeThickness` - Specifies the thickness value for the connector's border.
- `Stroke` - Specifies the color used for the border of the connector.
- `StrokeStartLineCap` - Specifies the shape to be used at the start of a line or segment.

- **StrokeEndLineCap** - Specifies the shape at the end of a line or segment.
- **StrokeLineJoin** - Specifies the shape that joins two lines or segments.
- **StrokeDashArray** - Specifies a collection of double values that indicate the pattern of dashes and gaps used to outline shapes.

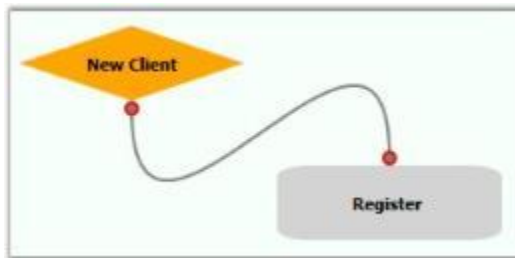
An example of the **Stroke** property can be applied to the head decorator as follows.

C#

```
LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
l1.ConnectorType = ConnectorType.Bezier;
l1.HeadDecoratorStyle.Stroke = Brushes.Red;
l1.TailDecoratorStyle.Stroke = Brushes.Red;
diagramModel.Connections.Add(l1);
```

VB.NET

```
Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
l1.ConnectorType = ConnectorType.Bezier
l1.HeadDecoratorStyle.Stroke = Brushes.Red
l1.TailDecoratorStyle.Stroke = Brushes.Red
diagramModel.Connections.Add(l1)
```



First Segment Orientation

This feature enables you to orient the **FirstSegment** of the **Orthogonal LineConnector** as needed.

This feature provides the following options to orient the first segment:

- **Auto** – The first segment of orthogonal **LineConnector** will always be perpendicular to the sides of the **HeadNode**, to which it is connected.
- **Horizontal** – The **FirstSegment** of the **Orthogonal LineConnector** will always be connected horizontally to the **HeadNode**.
- **Vertical** - The **FirstSegment** of the **Orthogonal LineConnector** will always be connected vertically to the **HeadNode**.

Use Case Scenarios

By default line connector will be drawn based on the space between the nodes. If you want to customize the default pattern, you can achieve this using this feature. This enables you to align the first segment of the connector and rest will be aligned based on this.

Tables for Properties, Methods, and Events

Properties

PropertyTable

Property	Description	Type	Data Type	Reference links
FirstSegmentOrientation	Gets or sets a value to orient the FirstSegment. Default Value is Auto	Dependency property	SegmentOrientation.Auto SegmentOrientation.Horizontal SegmentOrientation.Vertical	NA

Orienting the First Segment

You can orient the FirstSegment of the Orthogonal LineConnector using the FirstSegmentOrientation property.

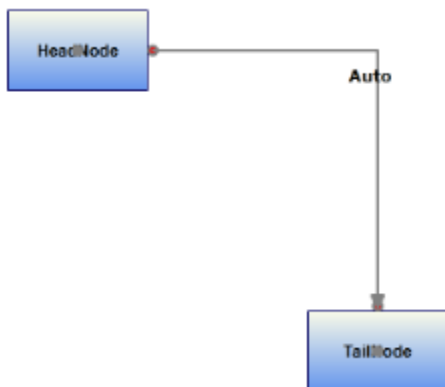
The following code illustrates how to set the FirstSegmentOrientation to Auto:

C#

```
LineConnector line = new LineConnector();
line.FirstSegmentOrientation = SegmentOrientation.Auto;
```

VB.NET

```
Dim line As New LineConnector()
line.FirstSegmentOrientation = SegmentOrientation.Auto
```



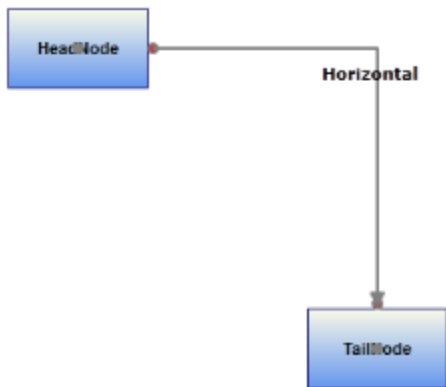
The following code illustrates how to set the FirstSegmentOrientation to Horizontal:

C#

```
LineConnector line = new LineConnector();
line.FirstSegmentOrientation = SegmentOrientation.Horizontal;
```


VB.NET

```
Dim line As New LineConnector()  
line.FirstSegmentOrientation = SegmentOrientation.Horizontal
```



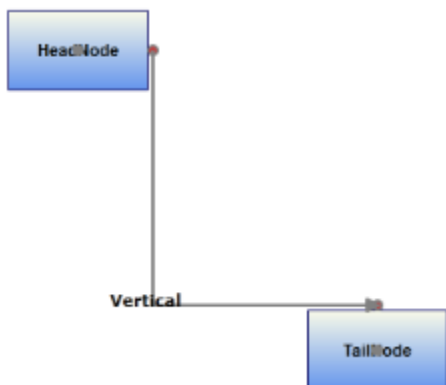
The following code illustrates how to set the FirstSegmentOrientation to Vertical.

C#

```
LineConnector line = new LineConnector();  
line.FirstSegmentOrientation = SegmentOrientation.Vertical;
```

VB.NET

```
Dim line As New LineConnector()  
line.FirstSegmentOrientation = SegmentOrientation.Vertical
```



Note: This FirstSegmentOrientation property is only works as expected when the LineConnector satisfies the following things.

- *LineConnector is connected between Nodes through ConnectionPort.*
- *When there is only one intermediate Point in Orthogonal LineConnector.*

Line Connector Label

Label is a single line or multiline text that is displayed over the Node. This Label is used to textually represent a LineConnector with a string that can be edited in run time, there are many properties that can be used to change the alignment and appearance settings. Label can be represented as multiline text using the TextWrapping property.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
IsLabelEditable	Gets or sets a value indicating whether line's label that can be edited or not. Default value is True.	Dependency property	Boolean (true/ false)	No
Label	Gets or sets the line's label. Default value is Empty String.	Dependency property	String	No
LabelTemplate	Gets or sets a template for the label. Default value is null.	Dependency property	DataTemplate	No
LabelVisibility	Gets or sets the label visibility. Default value is Visibility.Visible	Dependency property	Visibility.HiddenVisibility.CollapsedVisibility.Visible	No
LabelHorizontalAlignment	Gets or sets the node's label horizontal Alignment. Default value is HorizontalAlignment.Center	Dependency property	HorizontalAlignment.CenterHorizontalAlignment.LeftHorizontalAlignment.RightHorizontalAlignment.Stretch	No

LabelTextTrimming	Gets or sets the text trimming style . Default value is CharacterEllipsis.	Dependency property	TextTrimming.CharacterEllipsisTextTrimming.NoneTextTrimming.WordEllipsis	No
LabelForeground	Gets or sets the label foreground. Default value is Black.	Dependency property	Brush	No
LabelBackground	Gets or sets the label background. Default value is White.	Dependency property	Brush	No
LabelFontStyle	Gets or sets the label background. Default value is White.	Dependency property	FontStyle	No
LabelFontFamily	Gets or sets the label font family. Default value is Arial.	Dependency property	FontFamily	No
LabelTextAlignment	Gets or sets the label text alignment. Default value is Center.	Dependency property	TextAlignment.CenterTextAlignment.JustifyTextAlignment.LeftTextAlignment.Right	No
LabelFontSize	Gets or sets the label font size. Default value is 11.	Dependency property	Double	No
LabelFontWeight	Gets or sets the label font weight. Default value is SemiBold.	Dependency property	FontWeight	No
LabelTextWrapping	Gets or sets the label text wrapping.	Dependency	TextWrapping.NoWrapTextWrapping.WrapTextWrapping.WrapWithOverflow	No

	Default value is NoWrap.	property		
LabelWidth	Gets or sets the label width. Default value is lineâ€™s width.	Dependency property	Double	No
EnableMultilineLabel	Gets or sets a value indicating whether the label line can be multiline or not. Default value is False.	Dependency Property	Boolean (True / False)	No

A connector can be specified with a label, similar to node, using the Label property. The default value is an empty string. By default, the label starts at the center point of the connector.

C#

```
LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
l1.ConnectorType = ConnectorType.Bezier;
l1.Label = "Connect";
diagramModel.Connections.Add(l1);
```

VB.NET

```
Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
l1.ConnectorType = ConnectorType.Bezier
l1.Label = "Connect"
diagramModel.Connections.Add(l1)
```



Label Template

You can set a custom template for labels. The following code illustrates on how to set a label template. Create a DataTemplate and add the resource "text.png" to your application.

HTML

```
<DataTemplate x:Key="LabelCustomTemplate">
<StackPanel Orientation="Horizontal">
<Image Source="text.png" Width="20" Height="20"/>
<Border Background="AliceBlue">
<TextBlock Text="Hello"/>
</Border>
</StackPanel>
</DataTemplate>
```

Now, you can apply the template to the connector as follows.

C#

```
LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
l1.ConnectorType = ConnectorType.Straight;
l1.HeadDecoratorShape= DecoratorShape.Arrow;
l1.TailDecoratorShape= DecoratorShape.Arrow;
l1.HeadDecoratorStyle.Fill = new SolidColorBrush(Colors.LightGray);
l1.TailDecoratorStyle.Fill = new SolidColorBrush(Colors.LightGray);
l1.LabelTemplate = this.Resources["LabelCustomTemplate"] as DataTemplate ;
diagramModel.Connections.Add(l1);
```

VB.NET

```
Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
l1.ConnectorType = ConnectorType.Straight
l1.HeadDecoratorShape= DecoratorShape.Arrow
l1.TailDecoratorShape= DecoratorShape.Arrow
l1.HeadDecoratorStyle.Fill = New SolidColorBrush(Colors.LightGray)
l1.TailDecoratorStyle.Fill = New SolidColorBrush(Colors.LightGray)
l1.LabelTemplate = CType(Me.Resources("LabelCustomTemplate"), DataTemplate)
diagramModel.Connections.Add(l1)
```

The following screenshot illustrates "Hello" text on an Alice Blue background with an image on the left.



Multi line label

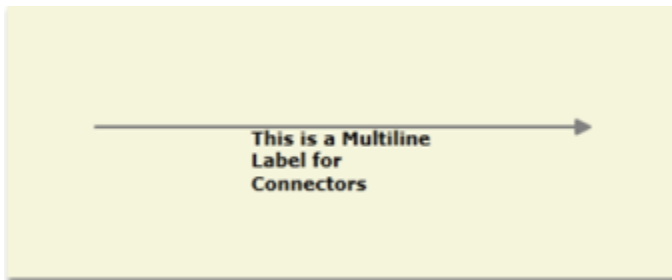
Label text can be displayed in multiple lines using LabelTextWrapping property set to wrap. If there is no enough space for the text to get displayed within connector in a single line then text will get wrapped within connector boundaries or LabelWidth and starts to display the label in multiple lines.

C#

```
LineConnector l = new LineConnector();  
l.Label = "This is a Multiline Label for Connectors";  
l.LabelHeight = 110;  
l.LabelTextWrapping = TextWrapping.Wrap;  
l.IsLabelEditable = true;
```

VB.NET

```
Dim l As New LineConnector()  
l.Label = "This is a Multiline Label for Connectors"  
l.LabelHeight = 110  
l.LabelTextWrapping = TextWrapping.Wrap  
l.IsLabelEditable = True
```

**Label Editing**

A connector's label can be edited at run time by setting `IsLabelEditable` to 'True'. The following code shows how it can be done.

C#

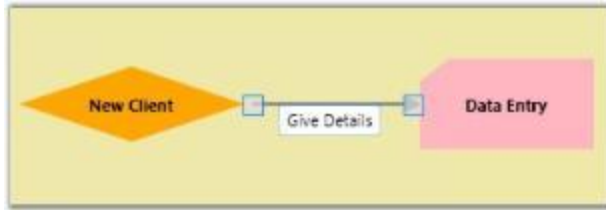
```
LineConnector l1 = new LineConnector();  
l1.Shape = Shapes.RoundedRectangle;  
l1.IsLabelEditable = true;
```

VB.NET

```
Dim l1 As New LineConnector()  
l1.Shape = Shapes.RoundedRectangle  
l1.IsLabelEditable = True
```

You can specify a label at run time by following the below mentioned steps.

- Double click the left mouse button on any part of the connector. A text box will appear with the cursor at the beginning.
- Now type the label name and press ENTER. The label will be displayed on the connector. Press ESC key if you do not want to apply the new label value.



Label Visibility

A label's visibility can be changed using the `LabelVisibility` property. The default value is visible.

C#

```
LineConnector l1 = new LineConnector();
l1.LabelVisibility = Visibility.Hidden;
```

VB.NET

```
Dim l1 As New LineConnector()
l1.LabelVisibility = Visibility.Hidden
```

The label will not get displayed.

Multiline Label Support for LabelEditor:

`LineConnector`'s Label can be set as Multiline Label by setting the `EnableMultiline` property as 'True'. The default Value is 'False'.

HTML

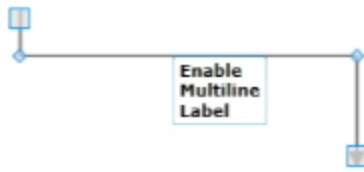
```
<syncfusion:LineConnector ConnectorType="Straight" Label="Line1"
IsSelected="True" StartPointPosition="100,400"
EndPointPosition="100,500" EnableMultilineLabel="True"
/>
```

C#

```
LineConnector line1 = new LineConnector();
line1.ConnectorType = ConnectorType.Orthogonal;
line1.StartPointPosition = new Point(100, 100);
line1.EndPointPosition = new Point(200, 200);
line1.EnableMultilineLabel = true;
diagramModel.Connections.Add(line1);
```

VB.NET

```
Dim line1 As New LineConnector()
line1.ConnectorType = ConnectorType.Orthogonal
line1.StartPointPosition = New Point(100, 100)
line1.EndPointPosition = New Point(200, 200)
line1.EnableMultilineLabel = True
diagramModel.Connections.Add(line1)
```



Custom Label Support for LineConnector

This feature enables you to customize the Label Position of the LineConnector. The CustomLabelPosition property for LineConnector are Auto, Drag and Custom.

- Drag: The label can be dragged
- Auto: Label position and angle will be updated internally based on the position of the LineConnector. This is a default value.
- Custom: You can customize the Label position

We can also set the Label Position using LabelPosition property of the LineConnector.

Property	Description	Type	Value It Accepts	Default Values	Any other dependencies/ sub properties associated
LabelPosition	Gets or sets the Position of the LineConnector's Label from the DiagramPage.	Dependency property	Point	Point(0,0)	No
CustomLabelPosition	Gets or sets the Label of the LineConnector to be Dragged or Not.	Dependency property	Enum.CustomLabelPositions.AutoCustomLabelPositions.CustomCustomLabelPositions.Drag	CustomLabelPositions.Auto	No

LabelAngle	Gets or sets the angle of the Label of LineConnector.	Dependency property	double	0	No
------------	---	---------------------	--------	---	----

Adding Custom Label Enhancements for LineConnector to an Application

Label Dragging support for LineConnector

The Label can be dragged from the Line Connector.

C#

```
(line as LineConnector).CustomLabelPosition = CustomLabelPositions.Drag;
```

Set the LabelPosition for LineConnector

When the values are given the position of the label will be exactly at the point of the specified values.

C#

```
(line as LineConnector).LabelPosition = new Point(100,100);
```

Set the LabelAngle for LineConnector

The labels rotate when values are given for the label angle.

C#

```
(line as LineConnector).LabelAngle = 45;
```

Label Orientation

Essential Diagram for WPF provides support to orient the LineConnector label as needed.

Use Case Scenarios

When the label overlaps with the nodes or connectors, it will not be legible. In such case you can use this feature to align the label to make it legible.

Property	Description	Type	Data Type	Reference links
LabelOrientation	Gets or sets a value to orient the labelDefault	Dependency property	LabelOrientation.AutoLabelOrientation.HorizontalLabelOrientation.Vertical	NA

	ult Value is Auto.			
--	-----------------------	--	--	--

Orienting the Label

You can orient the label using the **LabelOrientation** property. You can set this to Horizontal, Vertical or Auto. By default this is set to auto.

The following code illustrates how to set the LabelOrientation to Auto:

C#

```
LineConnector line = new LineConnector();
line.LabelOrientation = Syncfusion.Windows.Diagram.LabelOrientation.Auto;
```

VB.NET

```
Dim line As New LineConnector()
line.LabelOrientation = Syncfusion.Windows.Diagram.LabelOrientation.Auto
```

Note: When this property is set to Auto, the label will be positioned along the angle of the line drawn.



The following code illustrates how to set the LabelOrientation to Horizontal:

C#

```
LineConnector line = new LineConnector();
line.LabelOrientation =
Syncfusion.Windows.Diagram.LabelOrientation.Horizontal;
```

VB.NET

```
LineConnector line = new LineConnector();
line.LabelOrientation =
Syncfusion.Windows.Diagram.LabelOrientation.Horizontal;
```



The following code illustrates how to set the LabelOrientation to Vertical:

C#

```
LineConnector line = new LineConnector();  
line.LabelOrientation =  
Syncfusion.Windows.Diagram.LabelOrientation.Vertical;
```

C#

```
Dim line As New LineConnector()  
line.LabelOrientation = Syncfusion.Windows.Diagram.LabelOrientation.Vertical
```



Label Template Alignment

You can set an Alignment for the label template. The following code shows how to set the label template’s alignment. Label template supports horizontal alignment.

Properties

Property	Descri ption	Type of prop erty	Value it Accepts	Any other depend encies/ sub
----------	-----------------	-------------------------	------------------	--

				properties associated
LabelTemplateHorizontalAlignment	Specifies the horizontal alignment for the label template. The default value is Center.	Dependency property	HorizontalAlignment.Center, HorizontalAlignment.Left, HorizontalAlignment.Right, HorizontalAlignment.Stretch	No

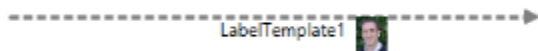
The following code illustrates how to set the LabelTemplateHorizontalAlignment to Auto:

C#

```
LineConnector Line = new DiagramView();
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.HorizontalAlignment.Auto
```

VB.NET

```
Dim Line As New LineConnector()
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.HorizontalAlignment.Auto
```



The following code illustrates how to set the LabelTemplateHorizontalAlignment to Left:

C#

```
LineConnector Line = new DiagramView();
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.HorizontalAlignment.Left;
```

VB.NET

```
Dim Line As New LineConnector()  
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.HorizontalA  
lignment.Left
```



The following code illustrates how to set the LabelTemplateHorizontalAlignment to Right:

C#

```
LineConnector Line = new DiagramView();  
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.HorizontalA  
lignment.Right;
```

VB.NET

```
Dim Line As New LineConnector()  
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.HorizontalA  
lignment.Right
```



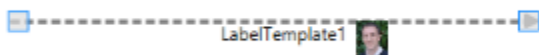
The following code illustrates how to set the LabelTemplateHorizontalAlignment to Stretch:

C#

```
LineConnector Line = new DiagramView();  
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.HorizontalA  
lignment.Stretch;
```

VB.NET

```
Dim Line As New LineConnector()  
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.HorizontalA  
lignment.Stretch
```

*Label Template Orientation*

You can set an orientation for the label template. The following property can be used to set the label template alignment.

Properties

Property	Description	Type of property	Value it Accepts	Any other dependencies/sub properties associated
LabelTemplateHorizontalAlignment	Specifies the horizontal alignment for label template. The default value is Center.	Dependency property	HorizontalAlignment.Center, HorizontalAlignment.Left, HorizontalAlignment.Right, HorizontalAlignment.Stretch	No

The following code illustrates how to set the LabelTemplateOrientation to Auto:

C#

```
LineConnector Line = new DiagramView();
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.LabelOrientation.Auto
```

VB.NET

```
Dim Line As New LineConnector()
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.LabelOrientation.Auto
```



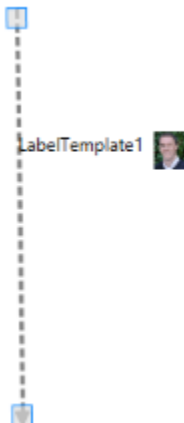
The following code illustrates how to set the `LabelTemplateOrientation` to Horizontal:

C#

```
LineConnector Line = new DiagramView();  
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.LabelOrientation.Horizontal
```

VB.NET

```
Dim Line As New LineConnector()  
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.LabelOrientation.Horizontal
```



The following code illustrates how to set the `LabelTemplateOrientation` to Vertical:

C#

```
LineConnector Line = new DiagramView();  
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.LabelOrientation.Vertical;
```

VB.NET

```
Dim Line As New LineConnector()
Line.LabelTemplateHorizontalAlignment=Syncfusion.Windows.Diagram.LabelOrientation.Vertical
```



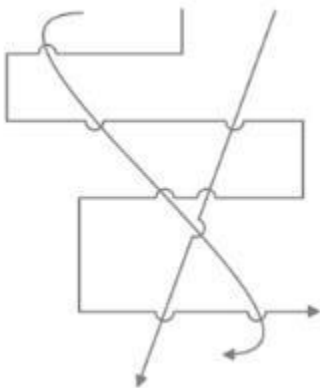
Line Bridging

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
LineBridgingEnabled	Gets or sets a value indicating whether line bridging is enabled.	Dependency property	Boolean (true/false)	No

Line Bridging creates a bridge for lines to smartly cross over other line at points of intersection.

When two line connectors meet each other, line with higher z-order will draw an arc over the line with lower z-order.

Only Straight and Orthogonal Connector type supports line bridging.



Enabling Line Bridging for LineConnector

LineBridging for a line connector can be enabled using the LineBridgingEnabled property.

By default this property will be set to 'False'.

C#

```
LineConnector lc = new LineConnector();  
lc.ConnectorType = ConnectorType.Straight;  
lc.LineBridgingEnabled = true;
```

VB.NET

```
Dim lc As New LineConnector()  
lc.ConnectorType = ConnectorType.Straight  
lc.LineBridgingEnabled = True
```

The line bridge is enabled.

Disable LineBridging from DiagramModel

When LineBridging for DiagramModel is disabled, LineBridging for all the lines will be disabled. You can change this binding by specifying a value for an individual LineConnector.

C#

```
DiagramModel model = new DiagramModel();  
model.LineBridgingEnabled = false;
```

VB.NET

```
Dim model As New DiagramModel()  
model.LineBridgingEnabled = False
```

The line bridge is disabled completely.

LineBridging Style

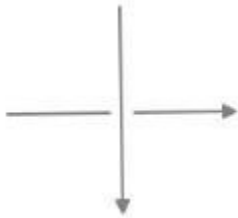
Default arc segment is customized by overriding CreateSegments method.

Example1: Draw an invisible segment, by setting IsStroked to false.

C#

```
protected override IEnumerable<PathSegment> CreateSegments(Point start,  
Point end, double angle)  
{  
    //InVisible ArcSegment  
    var collection = new List<PathSegment>()  
    {  
        new ArcSegment()  
        {  
            Point = end,  
            IsStroked = false,  
            SweepDirection = SweepDirection.Clockwise,  
            Size = new Size(1,1),  
            RotationAngle = 0,  
            IsLargeArc = false  
        }  
    };  
};
```

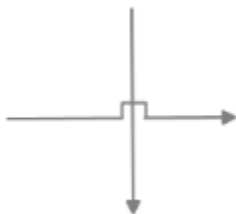
```
foreach (var line in collection)
{
    yield return line;
}
```



Example2: Draw a rectangular bridge instead of arc.

C#

```
protected override IEnumerable<PathSegment> CreateSegments(Point start,
    Point end, double angle)
{
    Matrix m = Matrix.Identity;
    m.RotateAt(angle, start.X, start.Y);
    MatrixTransform trans = new MatrixTransform(m);
    //List of LineSegements
    var coll = new List<PathSegment>()
    {
        new LineSegment() {Point = trans.Transform(new Point(start.X, start.Y - 10))},
        new LineSegment() {Point = trans.Transform(new Point(start.X + 16, start.Y - 10))},
        new LineSegment() {Point = trans.Transform(new Point(start.X + 16, start.Y))},
        new LineSegment() {Point = end}
    };
    foreach (var line in coll)
    {
        yield return line;
    }
}
```



Line Bridging Direction

Direction of the Line Bridge is customized using BridgeDirection property. This property decides which intersecting segment shows a bridge based path on the preferred direction. The Default value is BridgeDirection.Top.

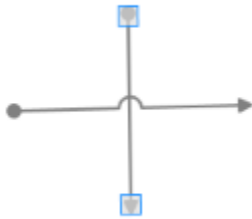
Properties	Description	Value
BridgeDirection	Gets or Sets the BridgeDirection for Horizontal and vertical lines.	EnumBridgeDirection.LeftBridgeDirection.RightBridgeDirection.TopBridgeDirection.Bottom

Example 1: Bridge for Horizontal Connector (with BridgeDirection.Top)

The following code example explains how to enable the Bridging and to set Bridge Direction.

C#

```
//Initializing Bridging and setting Bridge Direction
connector.LineBridgingEnabled=true;
diagramview.BridgeDirection=BridgeDirection.Top;
```



Example 2: Bridge for Vertical Connector (with BridgeDirection.Left)

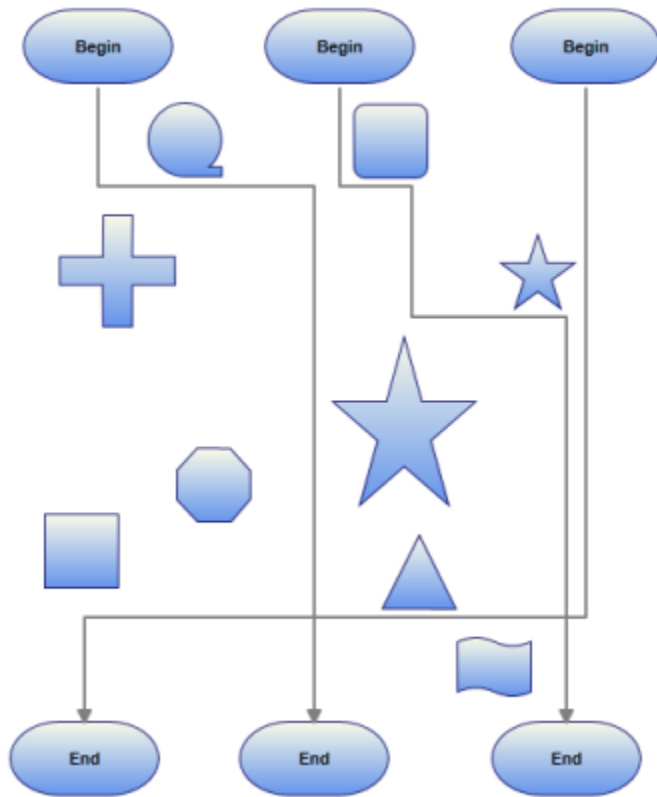
C#

```
// setting Bridge Direction
diagramview.BridgeDirection=BridgeDirection.Left
```



Line Routing

When a link is drawn between two nodes, by enabling the `LineRoutingEnabled` property of that link and the diagram view, and if any other node is found in between them, the line will be automatically re-routed around those nodes.



Property

Property	Description	Type	Data type	Reference links
<code>LineRoutingEnabled</code>	Specifies whether the links must be re-routed when nodes are found in the path. Default value is <code>True</code> .	Dependency	Boolean	NA

Disable Line Routing for LineConnector

Line Routing for a line connector can be disabled using the `LineRoutingEnabled` property.

By default this property will be set to 'True'.

C#

```
LineConnector lc = new LineConnector();
lc.ConnectorType = ConnectorType.Orthogonal;
lc.LineRoutingEnabled = false;
```

VB.NET

```
Dim lc As New LineConnector()
```

```
lc.ConnectorType = ConnectorType.Orthogonal
lc.LineBridgingEnabled = False
```

The Line Routing is disabled.

Enable LineRouting from DiagramView

When LineRouting for DiagramView is enabled, LineRouting for all the lines will be enabled. You can change this binding by specifying a value for an individual LineConnector.

C#

```
DiagramView view = new DiagramView ();
view.LineRoutingEnabled = true
```

VB.NET

```
Dim view As New DiagramView ()
view.LineRoutingEnabled = True
```

The Line Routing is enabled completely.

Note: Only Orthogonal Connector type supports Line Routing.

Node settings

By default, TreatAsObstacle property of the Node is set to true to avoid the lines overlapping them. If not set for a Node, then it will not be considered as an obstacle and the line might overlap on them.

Property

Property	Description	Type	Data type	Reference links
TreatAsObstacle	Gets or sets a value indicating whether the node treats as Obstacle.Default value is True.	Dependency	Boolean	NA

Customization of LineRouting

This feature defines when the connectors have to be routed, by setting the RoutingMode property of DiagramView. LineRouting can be executed after dragging the node by using this behavior.

Properties

Property	Description	Type	Data Type
RoutingMode	Decides when the connectors have to be routed.	Dependency Property	Enum

Adding Customization of LineRouting into Application

Enabling the LineRouting

To enable LineRouting, use the following code example:

C#

```
LineConnector line = new LineConnector();  
line.LineRoutingEnabled = true;
```

VB.NET

```
Dim line As New LineConnector()  
line.LineRoutingEnabled = True
```

RoutingMode

The RoutingMode property defines when the connectors have to be routed.

It can be set to:

1. DragEnd
2. Immediate

If this property is set to DragEnd, the connectors are routed only when the DiagramPage is stable. If the elements in the DiagramPage are moving or being dragged, the connectors are not routed. Only after the completion of the change, the connectors are routed.

To set the RoutingMode property to DragEnd, use the following code example.

C#

```
DiagramView View1 = new DiagramView();  
View1.RoutingMode = RoutingMode.DragEnd;
```

VB.NET

```
Dim view As New DiagramView()  
view.RoutingMode = RoutingMode.DragEnd
```

If this property is set to Immediate, the connectors are routed, while the element in the DiagramPage is dragged.

To set the RoutingMode property to Immediate, use the following the code example:

C#

```
DiagramView View1 = new DiagramView();  
View1.RoutingMode = RoutingMode.Immediate;
```

VB.NET

```
Dim view As New DiagramView()  
view.RoutingMode = RoutingMode.Immediate
```

Select, Move, Delete LineConnector

As this is a general topic to be share between Node and LineConnector, please refer general topic under Concepts and features using the following links.

- Select Node and Connectors
- Move Node and Connectors
- Delete Command

[Customize the Label, Context Menu for LineConnector](#)

As this is a general topic to be shared between Node and LineConnector, please refer general topic under Concepts and features using the following links.

- [Customize Label](#)
- [Customize ContextMenu](#)

[Connection Port in WPF Diagram \(classic\)](#)

Essential Diagram WPF provides the ability to define custom ports for making connections. The ConnectionPort class can be used for defining custom ports on the nodes. Any number of ports can be defined on a node. By default every node has a center port.

ConnectionPort has the following properties:

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
Left	Gets or sets the position of the port in the x coordinate. Default value: 0	Dependency property	Double	No
Top	Gets or sets the position of the port in the y coordinate. Default value: 0	Dependency property	Double	No
Node	The Node property specifies the container of the port.	CLR property	Node	No
PortShape	The PortShape property specifies the shape to be used for the port. Three types of shapes are provided:	CLR property	PortShapes.None PortShapes.Arrow PortShapes.Diamond PortShapes.Circle	No

	Arrow, Circle, Diamond.Default Value is PortShapes.Diamond			
PortStyle	The PortStyle property provides option for the customization of the ports.	CLR property	PortStyle	No

Node properties related to Connection Ports are:

Property	Description	Type of the property	Value it accepts	Any other dependencies / sub properties associated
PortVisibility	Gets or sets a value indicating whether all the ports of the node are visible or not.Default value is Visibility.Visible	Dependency property	Visibility.HiddenVisibility.CollapsedVisibility.Visible	No
AllowPortDragging	Gets or sets a value indicating whether the ports can be dragged or not.Default value is True.	Dependency property	Boolean (true/ false)	No

LineConnector properties related to Connection Port are:

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub
----------	-------------	----------------------	------------------	-----------------------------

				properties associated
ConnectionHeadPort	Gets or sets the head port of the connection. While specifying the ConnectionHeadPort, the node containing the port should be specified as the HeadNode of the connection. Default value is Null.	Dependency property	ConnectionPort	No
ConnectionTailPort	Gets or sets the tail port of the connection. While specifying the ConnectionTailPort, the node containing the port should be specified as the TailNode of the connection. Default value is Null.	Dependency property	ConnectionPort	No

- Create Connection Port Refer Concepts and Features -> Connection Port -> Create Connection Port
- PortShape Refer Concepts and Features -> Connection Port -> PortShape
- PortStyle Refer Concepts and Features -> Connection Port -> PortStyle
- PortVisibility Refer Concepts and Features -> Connection Port -> PortVisibility
- AllowPortDrag Refer Concepts and Features -> Connection Port -> AllowPortDrag
- Connections to Port Refer Concepts and Features -> Connection Port -> Connections to Port

Create Connection Port on Node

To add a port to the node, the port's position has to be specified using the Left and Top properties. The node which hosts the port should then be specified using the Node property. Finally the port should be added to the node's Ports collection.

The following code shows how to add a connection port to the node.

C#

```
Node node = new Node(Guid.NewGuid(), "Node1");
node.Shape = Shapes.RoundedSquare;
```

```

node.Width = 150;
node.Height = 50;
node.OffsetX = 250;
node.OffsetY = 100;
ConnectionPort port = new ConnectionPort();
port.Left = 50;
port.Top = 0;
port.Node = node;
node.Ports.Add(port);
diagramModel.Nodes.Add(node);

```

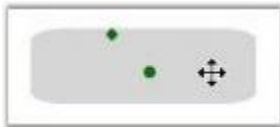
VB.NET

```

Dim node As New Node(Guid.NewGuid(), "Node1")
node.Shape = Shapes.RoundedSquare
node.Width = 150
node.Height = 50
node.OffsetX = 250
node.OffsetY = 100
Dim port As New ConnectionPort()
port.Left = 50
port.Top = 0
port.Node = node
node.Ports.Add(port)
diagramModel.Nodes.Add(node)

```

This adds a port to the node at the location (50,0) with respect to the node.



Connection Port

Note: The ports location should always be specified to be within the node's boundary. Therefore the values of the Left and Top property should always be less than the width and height of the node respectively.

Adding Connection Port at Runtime

This feature provides an option to add connection ports at runtime through mouse operations. Similarly, connection ports can be removed from a node through mouse operations.

Properties

Property	Description	Type	Data Type
AddConnectionPortEnabled	Property used to enable/disable the adding connection port dynamically.	SL	boolean

[Adding Connection Ports at Runtime to an Application](#)

[Adding a Connection Port at Runtime](#)

To allow connection ports to be added to a node dynamically, set the `AddConnectionPortEnabled` property of the node to `true`, then select the node. Hold `CTRL+SHIFT` and click on the node. The connection port will be added in the position where you clicked on the node.

C#

```
Node node = new Node();  
//Set True value to AddConnectionPortEnabled property.  
node.AddConnectionPortEnabled = true;
```

VB.NET

```
Dim node As New Node()  
'Set True value to AddConnectionPortEnabled property.  
node.AddConnectionPortEnabled = True
```

[Deleting a Connection Port at Runtime](#)

A connection port can be deleted from the node dynamically. With the `AllowDelete` property of a port set to `true`, hold `CTRL+SHIFT` and click on the connection port to delete it. The default value of the `AllowDelete` property is set to `true`. This can be disabled by setting the `AllowDelete` property to `false`.

C#

```
Node node = new Node();  
//Set false value to Connection Port disable Delete a port dynamically.  
(node.Ports[1] as ConnectionPort).AllowDelete = false;
```

VB.NET

```
Dim node As New Node()  
'Set false value to Connection Port disable Delete a port dynamically.  
TryCast(node.Ports(1), ConnectionPort).AllowDelete = False
```

Note: Removing a port directly from a node's `Ports` collection property will not be affected by setting the `AllowDelete` property of the connection port to `false`.

[Create Connection Port on Line Connector](#)

Connection ports for line connectors can be specified by using the `Edge` property. Multiple ports can be added to the line connectors by using the `Ports` property.

C#

```
LineConnector line = new LineConnector();  
ConnectionPort port = new ConnectionPort();  
port.PortStyle.Stroke = new SolidColorBrush(Colors.OrangeRed);  
port.PortStyle.Fill = new SolidColorBrush(Colors.OrangeRed);  
port.Width = 10;  
port.Height = 10;  
port.PortOffset = 0.5;  
port.PortVisibility = PortVisibility.AlwaysVisible;  
line.Ports.Add(port);  
port.Edge = line;
```

```
diagramModel.Connections.Add(line);
```

PortShape

Several predefined shapes have been provided for the ports. They are,

- Arrow
- Circle
- Diamond

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
PortShape	The PortShape property specifies the shape to be used for the port. Three types of shapes are provided: Arrow, Circle, Diamond. Default Value: PortShapes.Diamond	CLR property	PortShapes.NonePortShapes.ArrowPortShapes.DiamondPortShapes.Circle	No

The following code shows how a port shape can be selected for the port.

C#

```
Node node = new Node(Guid.NewGuid(), "Node1");
node.Shape = Shapes.RoundedSquare;
node.Width = 150;
node.Height = 50;
node.OffsetX = 250;
node.OffsetY = 100;
ConnectionPort port = new ConnectionPort();
port.Left = 50;
port.Top = 0;
port.Node = node;
port.PortShape = PortShapes.Arrow;
node.Ports.Add(port);
diagramModel.Nodes.Add(node);
```

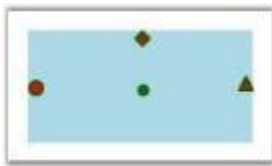
VB.NET

```
Dim node As New Node(Guid.NewGuid(), "Node1")
```

```

node.Shape = Shapes.RoundedSquare
node.Width = 150
node.Height = 50
node.OffsetX = 250
node.OffsetY = 100
Dim port As New ConnectionPort()
port.Left = 50
port.Top = 0
port.Node = node
port.PortShape = PortShapes.Arrow
node.Ports.Add(port)
diagramModel.Nodes.Add(node)

```



Port Shapes

Port Visibility

The PortVisibility property is used to set the visibility of connection ports on a node in one of three ways. This PortVisibility Enum class provides 3 types of enum.

1. OnMouseOver
2. AlwaysHidden
3. AlwaysVisible

Properties

Property	Description	Type	Data Type
PortVisibility	Gets and sets on which action the port will be visible or not.	Dependency	Enum

Adding PortVisibility to an Application

MouseOverNode

When the PortVisibility property of the node is set to MouseOverNode, the connection port will be visible only on mouse-over of the node. This is the default value of this property.

AlwaysHidden

The connection port will always be hidden, when PortVisibility property of the Node is set to AlwaysHidden.

AlwaysVisible

ConnectionPort will be visible always, when PortVisibility property of the Node is set to AlwaysVisible.

Information: Changes: For versions 10.2 and later, the type of PortVisibility property of nodes has been changed from 'Visibility' to 'PortVisibility' Enum.

Customize PortStyle

The port shapes can be customized by specifying the property values under the PortStyle property.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
PortStyle	The PortStyle property provides option for the customization of ports.	CLR property	PortStyle	No

The various properties under the PortStyle property are,

- Fill—specifies the color to be used to fill the port.
- StrokeThickness—specifies the thickness value of the port's border.
- Stroke—specifies the color to be used for the border of the port.
- StrokeStartLineCap—Specifies the shape used at the start of a line or segment.
- StrokeEndLineCap—specifies the shape at the end of a line or segment.
- StrokeLineJoin—specifies the shape that joins two lines or segments.

The following code shows how to set some of these properties.

C#

```
Node node = new Node(Guid.NewGuid(), "Node1");
node.Shape = Shapes.RoundedSquare;
node.Width = 150;
node.Height = 50;
node.OffsetX = 250;
node.OffsetY = 100;
ConnectionPort port = new ConnectionPort();
port.Left = 50;
port.Top = 0;
port.Node = node;
port.PortShape = PortShapes.Diamond;
port.PortStyle.Fill = Brushes.Orange;
port.PortStyle.Stroke = Brushes.Red;
node.Ports.Add(port);
diagramModel.Nodes.Add(node);
```

C#

```
Dim node As New Node(Guid.NewGuid(), "Node1")
node.Shape = Shapes.RoundedSquare
node.Width = 150
node.Height = 50
node.OffsetX = 250
node.OffsetY = 100
Dim port As New ConnectionPort()
port.Left = 50
port.Top = 0
port.Node = node
port.PortShape = PortShapes.Diamond
port.PortStyle.Fill = Brushes.Orange
port.PortStyle.Stroke = Brushes.Red
node.Ports.Add(port)
diagramModel.Nodes.Add(node)
```



Port Style

CustomPathStyle

The CustomPathStyle property enables you to customize the appearance of ConnectionPort.

Properties

Property	Description	Type	Data Type	Reference links
CustomPathStyle	Get or Set CustomPathStyle for ConnectionPort	Dependency Property	Style	NA

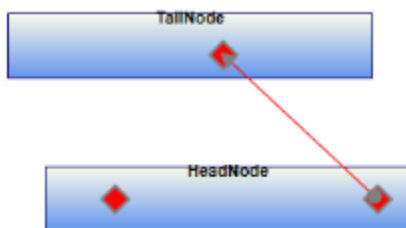
Adding CustomPathStyle for ConnectionPort to an Application

Appearance of the ConnectionPort can be customized by applying style for the CustomPathStyle property. Style can be applied for CustomPathStyle as illustrated in the following code:

Through XAML

HTML

```
<Style TargetType="{x:Type syncfusion:ConnectionPort}">
  <Setter Property="PortShape" Value="Diamond"/>
  <Setter Property="CustomPathStyle" >
    <Setter.Value>
      <Style TargetType="{x:Type Path}">
        <Setter Property="Stroke" Value="Red" />
        <Setter Property="StrokeThickness" Value="3"/>
      </Style>
    </Setter.Value>
  </Setter>
</Style>
```



Custom ConnectionPort using CustomPathStyle

AllowPortDrag

It is possible to move the ports on the node to a different location on the node. All that is needed to do is to drag the respective port to the desired location. The connections (if any) which are already connected to the nodes will also get updated accordingly. The node drag can be enabled/disabled using the AllowPortDrag property. By default it is set to 'False'.

The following code shows how to set the AllowPortDrag property.

C#

```
Node nodeObject = new Node(Guid.NewGuid(), "Node1");
diagramModel.Nodes.Add(nodeObject);
nodeObject.AllowPortDrag = true;
```

VB.NET

```
Dim nodeObject As New Node(Guid.NewGuid(), "Node1")
diagramModel.Nodes.Add(nodeObject)
nodeObject.AllowPortDrag = True
```

Connections to Ports

Connection to Ports on Node

Head and tail ends of a line connector can be connected to a connection port that exists in a node.

Make connection to the port through code behind

The ConnectionHeadPort and ConnectionTailPort properties can be used to specify the ports to be used for connecting them to the nodes. The HeadNode and TailNode should still be specified.

The following code shows how to connect to the ports.

C#

```
Node nodeObject = new Node(Guid.NewGuid(), "Node1");
diagramModel.Nodes.Add(nodeObject);
nodeObject.AllowPortDrag = true; Node node = new Node(Guid.NewGuid(),
"Node1");
node.Shape = Shapes.RoundedSquare;
node.Width = 150;
node.Height = 50;
node.OffsetX = 250;
node.OffsetY = 100;
ConnectionPort port = new ConnectionPort();
port.Left = 50;
port.Top = 20;
port.Node = node;
node.Ports.Add(port);
diagramModel.Nodes.Add(node);
Node node1 = new Node(Guid.NewGuid(), "Node1");
node1.Shape = Shapes.RoundedSquare;
node1.Width = 150;
node1.Height = 50;
node1.OffsetX = 250;
node1.OffsetY = 200;
ConnectionPort port1 = new ConnectionPort();
port1.Left = 20;
port1.Top = 20;
port1.Node = node1;
node1.Ports.Add(port1);
diagramModel.Nodes.Add(node1);
diagramModel.Nodes.Add(node1);
LineConnector o = new LineConnector();
```



```

o.ConnectorType = ConnectorType.Straight;
o.TailNode = node;
o.HeadNode = node1;
o.HeadDecoratorShape = DecoratorShape.None;
o.TailDecoratorShape = DecoratorShape.None;
o.ConnectionTailPort = port;
o.ConnectionHeadPort = port1;
LineConnector ol = new LineConnector();
ol.ConnectorType = ConnectorType.Straight;
ol.TailNode = node;
ol.HeadNode = node1;
ol.HeadDecoratorShape = DecoratorShape.None;
ol.TailDecoratorShape = DecoratorShape.None;
diagramModel.Connections.Add(o);
diagramModel.Connections.Add(ol);

```

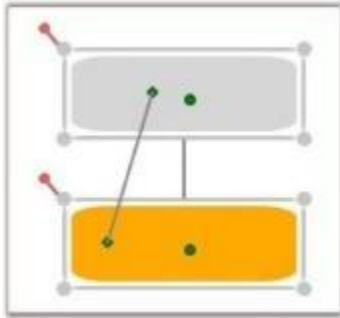
VB.NET

```

Dim nodeObject As New Node(Guid.NewGuid(), "Node1")
diagramModel.Nodes.Add(nodeObject)
nodeObject.AllowPortDrag = True
Dim node As New Node(Guid.NewGuid(), "Node1")
node.Shape = Shapes.RoundedSquare
node.Width = 150
node.Height = 50
node.OffsetX = 250
node.OffsetY = 100
Dim port As New ConnectionPort()
port.Left = 50
port.Top = 20
port.Node = node
node.Ports.Add(port)
diagramModel.Nodes.Add(node)
Dim node1 As New Node(Guid.NewGuid(), "Node1")
node1.Shape = Shapes.RoundedSquare
node1.Width = 150
node1.Height = 50
node1.OffsetX = 250
node1.OffsetY = 200
Dim port1 As New ConnectionPort()
port1.Left = 20
port1.Top = 20
port1.Node = node1
node1.Ports.Add(port1)
diagramModel.Nodes.Add(node1)
diagramModel.Nodes.Add(node1)
Dim o As New LineConnector()
o.ConnectorType = ConnectorType.Straight
o.TailNode = node
o.HeadNode = node1
o.HeadDecoratorShape = DecoratorShape.None
o.TailDecoratorShape = DecoratorShape.None
o.ConnectionTailPort = port
o.ConnectionHeadPort = port1
Dim ol As New LineConnector()
ol.ConnectorType = ConnectorType.Straight

```

```
o1.TailNode = node
o1.HeadNode = node1
o1.HeadDecoratorShape = DecoratorShape.None
o1.TailDecoratorShape = DecoratorShape.None
diagramModel.Connections.Add(o)
diagramModel.Connections.Add(o1)
```



Connecting to Port

Make connection to the port at run time

The following steps illustrate how to create a connection to the port at run time.

- To connect to a port on the node at run time, click the desired line icon and start dragging the mouse from the desired port on the node to the target node or port.
- As the mouse pointer moves over the ports, a red border will appear on the respective ports indicating that the mouse is over the port.
- To make a connection, drop the other end of the line on the desired port or node.
- Connecting to the center port will make the connection to the boundary of the node.
- If the node does not contain any port other than the default center port and if the end of line connector is dropped on the node, then the connection will take place on the node's boundary.
- In case the node contains a port, then the line connector should be dropped on the center port to connect to the node's boundary. A red border will appear around the node indicating that the connection will be made to the node's boundary.

Connection to Ports on Line Connector

Connections can be created between line connectors through connection ports. The connection ports can be added to the line connectors and dragged on the path lines.

Creating Connections between Line Connectors Programmatically

To make a connection between two lines, use the `ConnectionTailPort` or `ConnectionHeadPort` property of `LineConnector` and set a particular port.

C#

```
LineConnector line = new LineConnector();
ConnectionPort port = new ConnectionPort();
port.PortStyle.Stroke = new SolidColorBrush(Colors.OrangeRed);
port.PortStyle.Fill = new SolidColorBrush(Colors.OrangeRed);
port.Width = 10;
port.Height = 10;
port.PortOffset = 0.5;
```

```
port.PortVisibility = PortVisibility.AlwaysVisible;
line.Ports.Add(port);
port.Edge = line;
diagramModel.Connections.Add(line);
LineConnector line1 = new LineConnector();
line1.LineStyle.Stroke = new SolidColorBrush(Colors.Brown);
line1.TailDecoratorStyle.Stroke = new SolidColorBrush(Colors.Brown);
line1.TailDecoratorStyle.Fill = new SolidColorBrush(Colors.Brown);
line1.TailDecoratorShape = Syncfusion.Windows.Diagram.DecoratorShape.None;
line1.ConnectionTailPort = port;
diagramModel.Connections.Add(line1);
```

VB.NET

```
Dim line As New LineConnector()
Dim port As New ConnectionPort()
port.PortStyle.Stroke = New SolidColorBrush(Colors.OrangeRed)
port.PortStyle.Fill = New SolidColorBrush(Colors.OrangeRed)
port.Width = 10
port.Height = 10
port.PortOffset = 0.5
port.PortVisibility = PortVisibility.AlwaysVisible
line.Ports.Add(port)
port.Edge = line
diagramModel.Connections.Add(line)
Dim line1 As New LineConnector()
line1.LineStyle.Stroke = New SolidColorBrush(Colors.Brown)
line1.TailDecoratorStyle.Stroke = New SolidColorBrush(Colors.Brown)
line1.TailDecoratorStyle.Fill = New SolidColorBrush(Colors.Brown)
line1.TailDecoratorShape = Syncfusion.Windows.Diagram.DecoratorShape.None
line1.ConnectionTailPort = port
diagramModel.Connections.Add(line1)
```

Creating Connections between Line Connectors at Run Time

The connections can be created at run time using mouse operations. Drag one of the line connectors to a target line connector. It will automatically create a connection port on a dropped position to make a connection between the two line connectors. The automatically added connection port will be removed when disconnecting the line connectors from it.

Groups in WPF Diagram (classic)

Essential Diagram WPF provides support to group and ungroup nodes. Grouping feature comes in handy when you want to apply the same edits to a number of objects and yet retain their individuality. All the operations performed on the group also affect the individual items in the group. However any item in the group can also be edited individually. On ungrouping, the items in the group again act as individual entities.

A Group is essentially just another node added which acts as a container for other objects. Therefore a group node is referred to as the parent node, and the grouped objects are referred to as the children of the group.

The Group class is inherited from the Node class. Therefore all the node properties apply to a group too.

The following table lists the methods that are used for grouping.

Name	Parameter	Return Type	Description	Reference Type
Group.AddChild(INodeGroup)	INodeGroup	void	Adds the specified INodeGroup object to the group. INodeGroup provides an interface to the nodes and the connectors.	N/A
Group.RemoveChild(INodeGroup)	INodeGroup	void	Removes the specified INodeGroup object from the group. INodeGroup provides an interface to the nodes and the connectors.	N/A
AllowSingleChild	Gets or sets a value indicating whether grouping can be allowed for a single child.	CLR property	Boolean (true/false)	No
AlwaysSelected	Gets or sets a value indicating whether a group can be always selected.	CLR property	Boolean (true/false)	
NodeChildren	Gets the group children.	CLR property	CollectionExt	
ItemTemplate	Gets or sets the item template used to display the group items.	CLR property	HierarchicalDataTemplate	
GroupChildrenRef	Gets or sets the group children reference number. Used for serialization purposes.	CLR property	CollectionExt	
GroupNodes	Gets or sets the areas.	CLR property	IEnumerable	

Create Group

The following are the three ways to create a group in Essential Diagram WPF.

- By using Code Behind

- By using the Group Command
- By using the Context Menu

Grouping By Using Code Behind

The Group class enables to group nodes in Essential Diagram WPF. The AddChild method is used to add the elements to the group.

The following code example illustrates how to create a group by using code behind.

C#

```
public DiagramControl Control;
public DiagramModel Model;
public DiagramView View;
public Window1 ()
{
    Control = new DiagramControl ();
    Model = new DiagramModel ();
    View = new DiagramView ();
    Control.View = View;
    Control.Model = Model;
    View.Bounds = new Thickness(0, 0, 1000, 1000);
    Node n = new Node(Guid.NewGuid(), "Start");
    n.Shape = Shapes.FlowChart_Card;
    n.Level = 1;
    n.OffsetX = 150;
    n.OffsetY = 25;
    n.Width = 150;
    n.Height = 75;
    Node n1 = new Node(Guid.NewGuid(), "End");
    n1.Shape = Shapes.RoundedRectangle;
    n1.Level = 1;
    n1.OffsetX = 350;
    n1.OffsetY = 325;
    n1.Width = 100;
    n1.Height = 75;
    Model.Nodes.Add(n);
    Model.Nodes.Add(n1);
    Group g = new Group(Guid.NewGuid(), "group1");
    g.AddChild(n);
    g.AddChild(n1);
    Model.Nodes.Add(g);
}
```

VB.NET

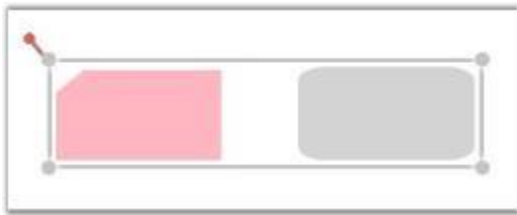
```
Public Control As DiagramControl
Public Model As DiagramModel
Public View As DiagramView
'INSTANT VB WARNING: The following constructor is declared outside of its
associated class:
'ORIGINAL LINE: public Window1 ()
Public Sub New()
    Control = New DiagramControl()
    Model = New DiagramModel()
    View = New DiagramView()
```

```

Control.View = View
Control.Model = Model
View.Bounds = New Thickness(0, 0, 1000, 1000)
Dim n As New Node(Guid.NewGuid(), "Start")
n.Shape = Shapes.FlowChart_Card
n.Level = 1
n.OffsetX = 150
n.OffsetY = 25
n.Width = 150
n.Height = 75
Dim n1 As New Node(Guid.NewGuid(), "End")
n1.Shape = Shapes.RoundedRectangle
n1.Level = 1
n1.OffsetX = 350
n1.OffsetY = 325
n1.Width = 100
n1.Height = 75
Model.Nodes.Add(n)
Model.Nodes.Add(n1)
Dim g As New Group(Guid.NewGuid(), "group1")
g.AddChild(n)
g.AddChild(n1)
Model.Nodes.Add(g)
End Sub

```

The following screenshot illustrates a group of two nodes created by using code behind.



Group of Two Nodes

Grouping By Using the Group Command

The Group command is used to group two or more objects.

The following code example illustrates how to group objects by using the Group command.

C#

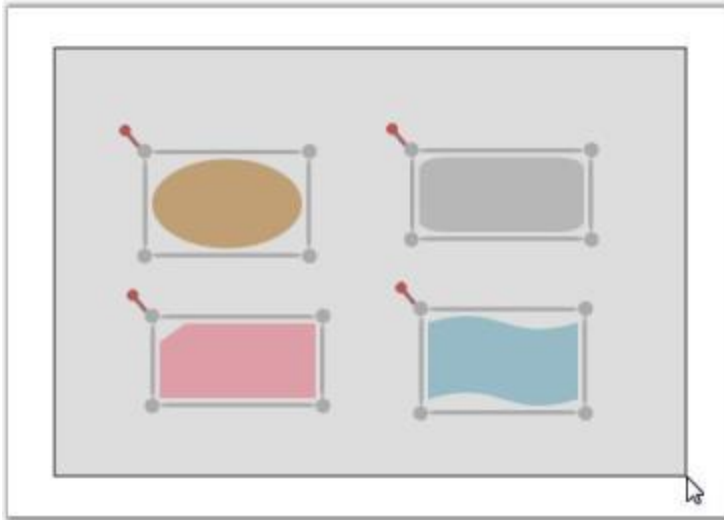
```
DiagramCommandManager.Group.Execute(diagramView.Page, diagramView);
```

VB.NET

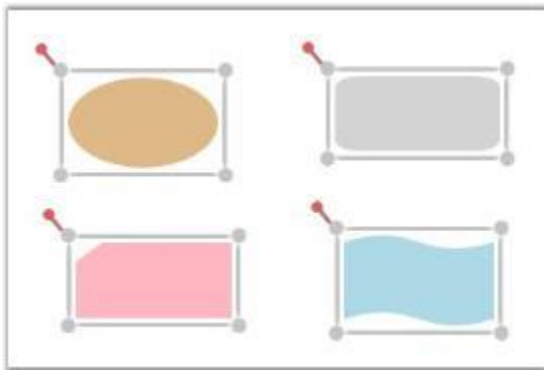
```
DiagramCommandManager.Group.Execute(diagramView.Page, diagramView)
```

The following steps illustrate how to create a group by using the Group command.

1. Select the objects to be grouped.



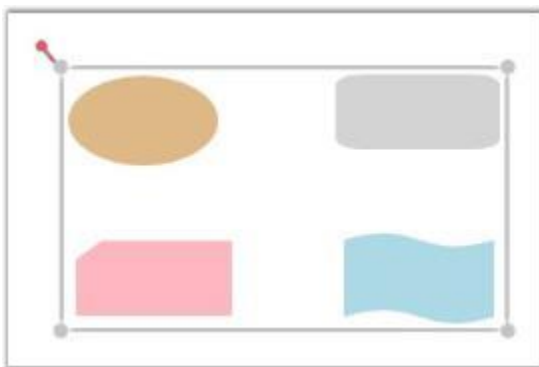
Selection of Objects to be Grouped



Selected Objects

2. Invoke the Group command. This creates a group.

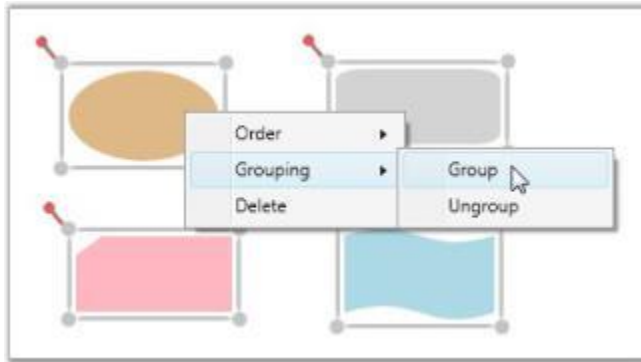
The new group is indicated by the selection rectangle which is displayed surrounding the objects in the group.



Grouped Objects inside the Selection Rectangle

Grouping By Using the Context Menu

You can also invoke the Group command by using the context menu which is displayed on right-clicking a particular Node or Line Connector.



Grouping By Using the Context Menu

Note: The Group command is enabled only when two or more objects are selected.

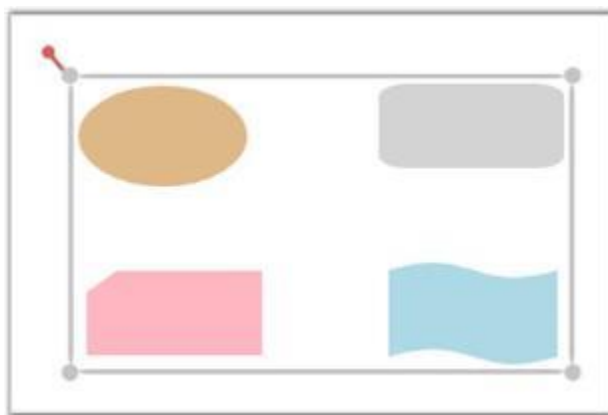
Multi-Select Nodes Refer Concepts and Features -> Nodes -> Node Resize -> Resize Single Node on Multiple Selection

Select Group

You can select a group by clicking on any one of its children. Consecutive clicks on a child object select the parent groups in the order of their creation. In a similar way, consecutive clicks on a child object leads to the selection of inner groups, and eventually the object itself, and the cycle continues. An object can belong to multiple groups, and groups may in turn have multiple subgroups.

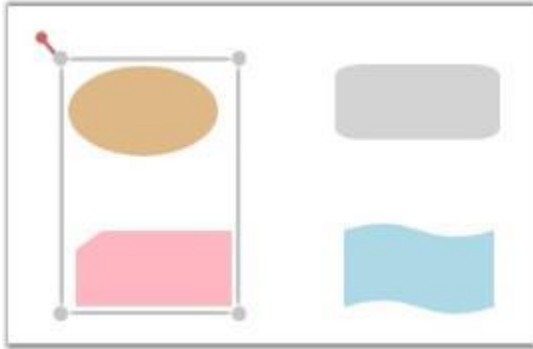
The following steps illustrate how to select an object which has two groups.

1. Click on the brown color node to select the outer group.



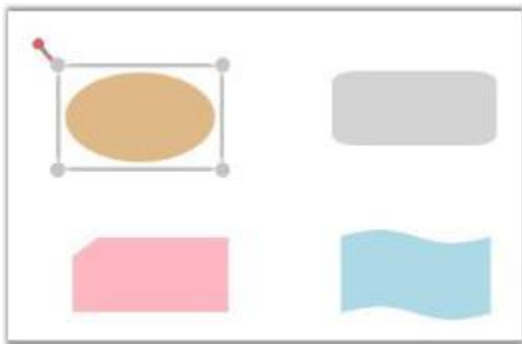
Outer Group Selected

2. Click again to select the inner group of which it is a part of.



Inner Group Selected

3. Finally, click again to select the child itself after all its groups have been traversed.

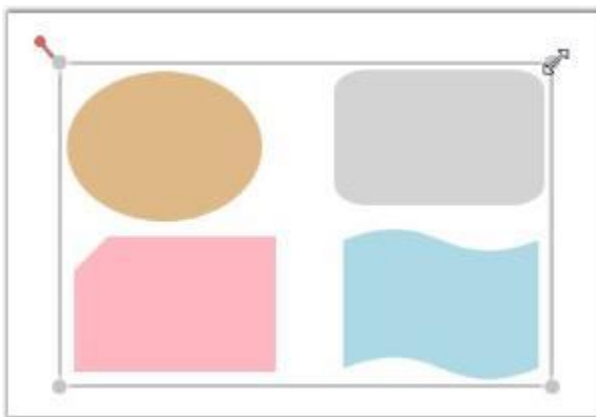


Selecting the Child Node Again

Edit Group

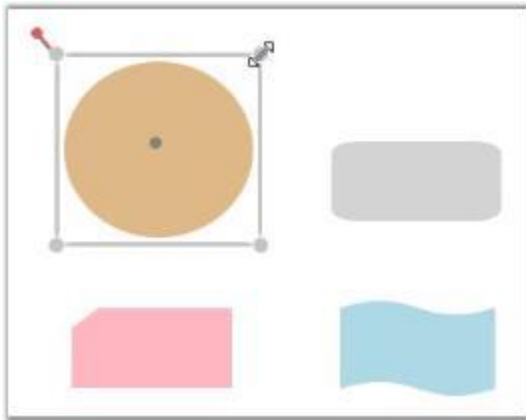
To edit a group, you have to make sure that the corresponding group is selected. The following features apply to the edits performed on an object.

- If the edit operation is performed on an object which is a group, then all its children are also affected. For example, resizing a group, automatically resizes its child objects to fit the selection area.



Resizing a Group

- If an individual object is selected, then the edit operation will be performed on that particular object only.



Resizing a Child Object

Once a group is edited, the group's selection rectangle updates its area to fit the child objects.



Updated Selection Rectangle

- Node Resize Refer Concepts and Features -> Nodes -> Node Resize
- Node Rotate Refer Concepts and Features -> Nodes -> Node Rotate

Connecting Groups

Grouping enables to connect to groups as well as to individual objects in a group. This section illustrates how to create connections to a group and its child objects.

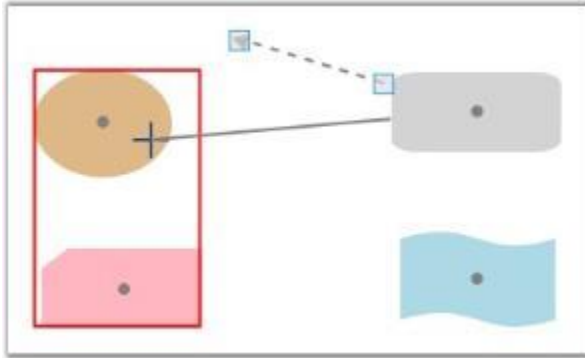
Creating Connections to a Group

The following steps illustrate how to create connections to a group.

1. Press the left mouse button while the mouse pointer is over any of the child objects of the group.

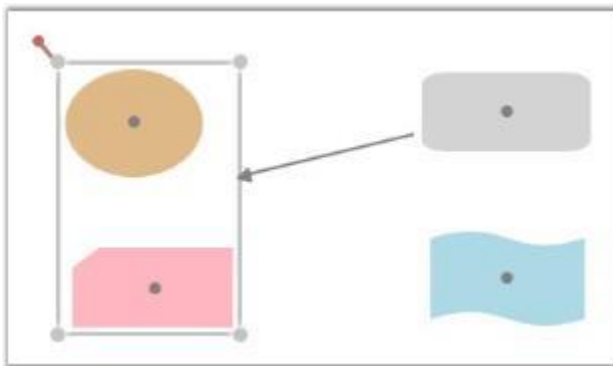
2. While the left mouse button is pressed, drag one end of the line connector to any of the group's children.

This will display a red adorer along the group's boundary, indicating that the connection is to be made to that particular group.



Creating Connection to a Group

3. Release the mouse button at that point to connect to the group's boundary.



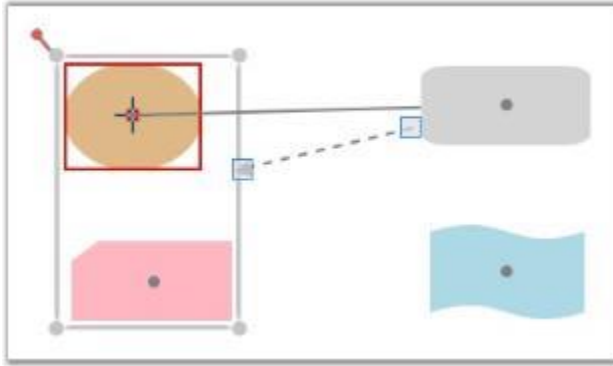
Connection Created to the Group

[Creating Connections to a Objects in a Group](#)

The following steps illustrate how to create connections to individual objects in a group.

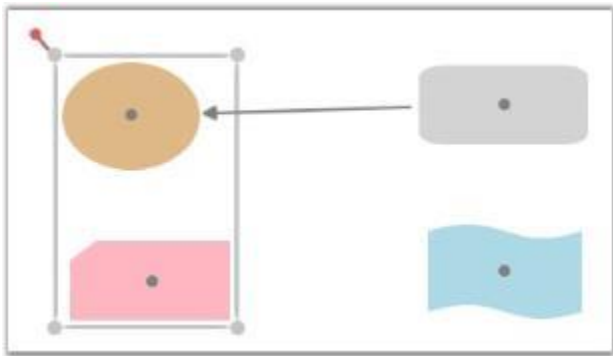
1. Press the left mouse button while the mouse pointer is over the node to which the connection is to be made.
2. While the left mouse button is pressed, drag the line connector to the center port of that particular node.

When the mouse pointer is over the center port, a red adorer will be displayed along the node's boundary, indicating that the connection is to be made to that particular node.



Creating Connection to a Child Object

3. Release the mouse button at that point to connect to the node's boundary.



Connection Created to the Child Object

Note: Connections cannot be created between a group and its own children. However child objects belonging to the same group can be connected to each other.

Create Line Connectors Refer Concepts and Features -> Line Connector -> Create Line Connector

Ungroup

Ungrouping a group deletes the group and removes all the child elements from the group. Once a group is ungrouped, the child elements behave as individual entities.

The following are the three ways to ungroup a group in Essential Diagram WPF.

- By using Code Behind
- By using the Ungroup Command
- By using the Context Menu

By Using Code Behind

The RemoveChild method is used to remove elements from a group.

For example, the following code illustrates how to remove node n from a group by using code behind.

C#

```

public DiagramControl Control;
public DiagramModel Model;
public DiagramView View;
public Window1 ()
{
    Control = new DiagramControl();
    Model = new DiagramModel();
    View = new DiagramView();
    Control.View = View;
    Control.Model = Model;
    View.Bounds = new Thickness(0, 0, 1000, 1000);
    Node n = new Node(Guid.NewGuid(), "Start");
    n.Shape = Shapes.FlowChart_Start;
    n.Level = 1;
    n.OffsetX = 150;
    n.OffsetY = 25;
    n.Width = 150;
    n.Height = 75;
    Node n1 = new Node(Guid.NewGuid(), "End");
    n1.Shape = Shapes.FlowChart_Start;
    n1.Level = 1;
    n1.OffsetX = 350;
    n1.OffsetY = 325;
    n1.Width = 100;
    n1.Height = 75;
    Model.Nodes.Add(n);
    Model.Nodes.Add(n1);
    Group g = new Group(Guid.NewGuid(), "group1");
    g.AddChild(n);
    g.AddChild(n1);
    Model.Nodes.Add(g);
    g.RemoveChild(n);
}

```

VB.NET

```

Public Control As DiagramControl
Public Model As DiagramModel
Public View As DiagramView
'INSTANT VB WARNING: The following constructor is declared outside of its
associated class:
'ORIGINAL LINE: public Window1 ()
Public Sub New()
    Control = New DiagramControl()
    Model = New DiagramModel()
    View = New DiagramView()
    Control.View = View
    Control.Model = Model
    View.Bounds = New Thickness(0, 0, 1000, 1000)
    Dim n As New Node(Guid.NewGuid(), "Start")
    n.Shape = Shapes.FlowChart_Start
    n.Level = 1
    n.OffsetX = 150
    n.OffsetY = 25
    n.Width = 150

```

```
n.Height = 75
Dim n1 As New Node(Guid.NewGuid(), "End")
n1.Shape = Shapes.FlowChart_Start
n1.Level = 1
n1.OffsetX = 350
n1.OffsetY = 325
n1.Width = 100
n1.Height = 75
Model.Nodes.Add(n)
Model.Nodes.Add(n1)
Dim g As New Group(Guid.NewGuid(), "group1")
g.AddChild(n)
g.AddChild(n1)
Model.Nodes.Add(g)
g.RemoveChild(n)
End Sub
```

The following screenshot illustrates a group of two nodes created by using code behind.

By Using the Ungroup Command

The Ungroup command is used to ungroup two or more objects.

The following code example illustrates how to ungroup objects by using the Ungroup command.

C#

```
DiagramCommandManager.Ungroup.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.Ungroup.Execute(diagramView.Page, diagramView)
```

The following steps illustrate how to ungroup a group by using the Ungroup command.

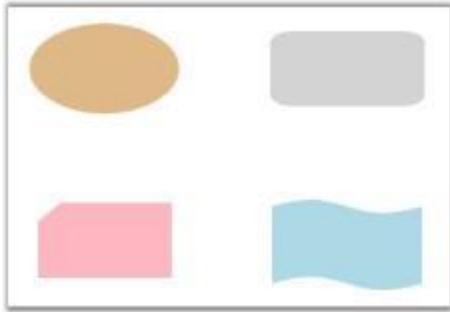
1. Select the group to be ungrouped.



Selecting a Group

2. Invoke the Ungroup command. This ungroup the nodes.

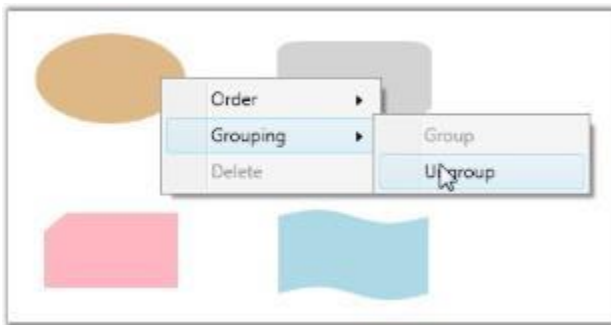
As soon as the group is ungrouped, the selection rectangle disappears indicating that the group has been ungrouped.



Objects Ungrouped

By Using the Context Menu

You can also invoke the Ungroup command by using the context menu which is displayed on right-clicking a particular Node or Line Connector.



Ungrouping By Using the Context Menu

Note: The Ungroup command is enabled only when two or more objects are selected.

Layers

Essential Diagram for WPF supports layer display. Numerous nodes and line connectors can be added to a layer and the visible property of its contents can be hidden by changing the visible property of the layer. A node or line connector can be added to any number of layers and the node is visible only if all layers to which this node or line connector belongs to are visible.

This feature will be useful when there are many nodes or connector in the page. If it is required to see only a particular part or particular set of nodes or connector create separate layers and categorize nodes and connectors in different layers. Then set the `IsVisible` of each layers accordingly.

The following topics are explained subsequently,

- Creating a Layer
- Adding the Layer to a Model
- Active Layer
- Hiding a Layer

Name	Parameters	Return Type	Description	Reference Links
------	------------	-------------	-------------	-----------------

Nodes.Add(Node)	Node	void	To add a node into the layer.	N/A
Lines.Add(Node)	LineConnector	Void	To add a line connector into the layer.	N/A

Creating a Layer

The following code example illustrates the creation of a layer with two Nodes and one LineConnector added.

C#

```
Node n1 = new Node();
n1.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n1);
n1.OffsetX = 50;
n1.OffsetY = 50;
Node n2 = new Node();
n2.Shape = Shapes.FlowChart_Delay;
diagramModel.Nodes.Add(n2);
n2.OffsetX = 150;
n2.OffsetY = 250;
LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
diagramModel.Connections.Add(l1);
Layer Lan1;
Lan1 = new Layer();
Lan1.Name = "Lan1";
Lan1.Nodes.Add(n1);
Lan1.Nodes.Add(n2);
Lan1.Lines.Add(l1);
Lan1.Background = Brushes.Transparent;
```

VB.NET

```
Dim n1 As New Node()
n1.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n1)
n1.OffsetX = 50
n1.OffsetY = 50
Dim n2 As New Node()
n2.Shape = Shapes.FlowChart_Delay
diagramModel.Nodes.Add(n2)
n2.OffsetX = 150
n2.OffsetY = 250
Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
diagramModel.Connections.Add(l1)
Dim Lan1 As Layer
Lan1 = New Layer()
Lan1.Name = "Lan1"
Lan1.Nodes.Add(n1)
Lan1.Nodes.Add(n2)
Lan1.Lines.Add(l1)
```



```
Lan1.Background = Brushes.Transparent  
### Adding the Layer to a Model  
The following code example illustrates the addition of a layer to a model.
```

C#

```
Node n1 = new Node();  
n1.Shape = Shapes.FlowChart_Card;  
diagramModel.Nodes.Add(n1);  
n1.OffsetX = 50;  
n1.OffsetY = 50;  
Node n2 = new Node();  
n2.Shape = Shapes.FlowChart_Delay;  
diagramModel.Nodes.Add(n2);  
n2.OffsetX = 150;  
n2.OffsetY = 250;  
LineConnector l1 = new LineConnector();  
l1.HeadNode = n1;  
l1.TailNode = n2;  
diagramModel.Connections.Add(l1);  
Layer Lan1;  
Lan1 = new Layer();  
Lan1.Name = "Lan1";  
Lan1.Nodes.Add(n1);  
Lan1.Nodes.Add(n2);  
Lan1.Lines.Add(l1);  
Lan1.Background = Brushes.Transparent;  
diagramModel.Layers.Add(Lan1)
```

VB.NET

```
Dim n1 As New Node()  
n1.Shape = Shapes.FlowChart_Card  
diagramModel.Nodes.Add(n1)  
n1.OffsetX = 50  
n1.OffsetY = 50  
Dim n2 As New Node()  
n2.Shape = Shapes.FlowChart_Delay  
diagramModel.Nodes.Add(n2)  
n2.OffsetX = 150  
n2.OffsetY = 250  
Dim l1 As New LineConnector()  
l1.HeadNode = n1  
l1.TailNode = n2  
diagramModel.Connections.Add(l1)  
Dim Lan1 As Layer  
Lan1 = New Layer()  
Lan1.Name = "Lan1"  
Lan1.Nodes.Add(n1)  
Lan1.Nodes.Add(n2)  
Lan1.Lines.Add(l1)  
Lan1.Background = Brushes.Transparent  
diagramModel.Layers.Add(Lan1)
```

Hidden or Active Layer

Active Layer

When a new Node or LineConnector is dropped from SymbolPalette into the DiagramPage, it will be added into all the active layers automatically. A layer can be activated or deactivated as shown in following code snippet.

C#

```
Node n1 = new Node();
n1.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n1);
n1.OffsetX = 50;
n1.OffsetY = 50;
Node n2 = new Node();
n2.Shape = Shapes.FlowChart_Delay;
diagramModel.Nodes.Add(n2);
n2.OffsetX = 150;
n2.OffsetY = 250;
LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
diagramModel.Connections.Add(l1);
Layer Lan1;
Lan1 = new Layer();
Lan1.Name = "Lan1";
Lan1.Nodes.Add(n1);
Lan1.Nodes.Add(n2);
Lan1.Lines.Add(l1);
Lan1.Background = Brushes.Transparent;
diagramModel.Layers.Add(Lan1);
Lan1.Active = false;
```

VB.NET

```
Dim n1 As New Node()
n1.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n1)
n1.OffsetX = 50
n1.OffsetY = 50
Dim n2 As New Node()
n2.Shape = Shapes.FlowChart_Delay
diagramModel.Nodes.Add(n2)
n2.OffsetX = 150
n2.OffsetY = 250
Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
diagramModel.Connections.Add(l1)
Dim Lan1 As Layer
Lan1 = New Layer()
Lan1.Name = "Lan1"
Lan1.Nodes.Add(n1)
Lan1.Nodes.Add(n2)
Lan1.Lines.Add(l1)
Lan1.Background = Brushes.Transparent
diagramModel.Layers.Add(Lan1)
```

```
Lan1.Active = False
```

Hiding a Layer

The following code example illustrates hiding a layer. When a layer is hidden all the nodes and connectors belonging to this layer will be hidden.

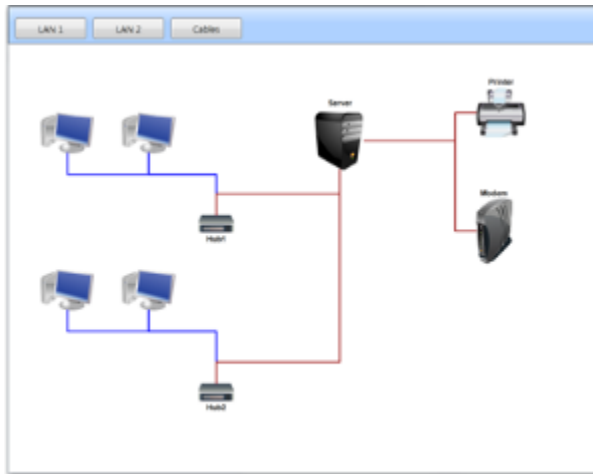
C#

```
Node n1 = new Node();
n1.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(n1);
n1.OffsetX = 50;
n1.OffsetY = 50;
Node n2 = new Node();
n2.Shape = Shapes.FlowChart_Delay;
diagramModel.Nodes.Add(n2);
n2.OffsetX = 150;
n2.OffsetY = 250;
LineConnector l1 = new LineConnector();
l1.HeadNode = n1;
l1.TailNode = n2;
diagramModel.Connections.Add(l1);
Layer Lan1;
Lan1 = new Layer();
Lan1.Name = "Lan1";
Lan1.Nodes.Add(n1);
Lan1.Nodes.Add(n2);
Lan1.Lines.Add(l1);
Lan1.Background = Brushes.Transparent;
diagramModel.Layers.Add(Lan1);
Lan1.Visible = false;
```

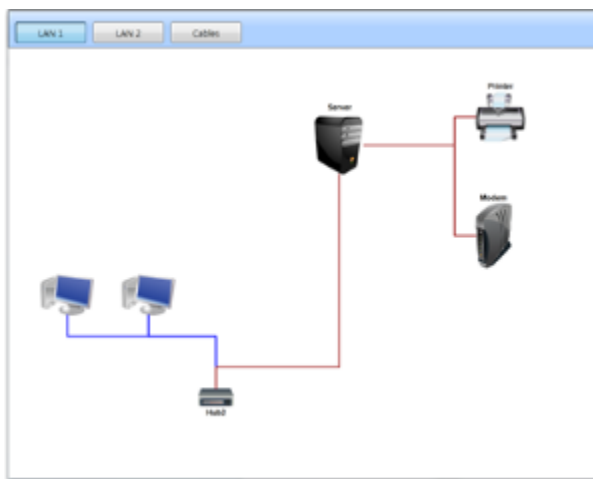
VB.NET

```
Dim n1 As New Node()
n1.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(n1)
n1.OffsetX = 50
n1.OffsetY = 50
Dim n2 As New Node()
n2.Shape = Shapes.FlowChart_Delay
diagramModel.Nodes.Add(n2)
n2.OffsetX = 150
n2.OffsetY = 250
Dim l1 As New LineConnector()
l1.HeadNode = n1
l1.TailNode = n2
diagramModel.Connections.Add(l1)
Dim Lan1 As Layer
Lan1 = New Layer()
Lan1.Name = "Lan1"
Lan1.Nodes.Add(n1)
Lan1.Nodes.Add(n2)
Lan1.Lines.Add(l1)
Lan1.Background = Brushes.Transparent
```

```
diagramModel.Layers.Add(Lan1)
Lan1.Visible = False
```



Before hiding the layer



After hiding the layer

Diagram Model in WPF Diagram (classic)

A model represents data for an application and contains the logic for adding, accessing, and manipulating the data. Nodes and connectors are added to the Diagram control using the Model property. A predefined layout can be applied using the LayoutType property of the DiagramModel. The position of the nodes can be manually specified.

- Bind data to Diagram Control Refer Concepts and Features -> Diagram Model -> Bind data to Diagram Control
- Tree Spacing Refer Concepts and Features -> Diagram Model -> Tree Spacing
- Tree Orientation Refer Concepts and Features -> Diagram Model -> Tree Orientation
- Table Expand Mode Refer Concepts and Features -> Diagram Model -> Table Expand Mode

Name	Parameters	Return Type	Description	Reference Links
Nodes.Add(object)	object,object should be of type Node	Void	To add a node into the Model.	Add a Node
Connections.Add(object)	object,object should be of type LineConnector	Void	To add a line connector into the Model.	Add a LineConnector
Layers.Add(Layer)	Layer	Void	To add a layer into the model.	Add a Layer

Layout

The following are general spacing properties used in many automatic layouts. Spacing refers to spaces between the nodes that lies in different levels of the tree layout and space between each node with their sibling.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
VerticalSpacing	Gets or sets the Vertical spacing between nodes.	CLR property	Double	No
HorizontalSpacing	Gets or sets the Horizontal spacing between	CLR property	Double	No

	en nodes.			
SpaceBetweenSubTrees	Gets or sets the space between sub trees.	CLR property	Double	No
LayoutVerticalAlignment	Specifies the vertical alignment for layout. The default value is Center.	Dependency property	VerticalAlignment.CenterVerticalAlignment.TopVerticalAlignment.BottomVerticalAlignment.Stretch	No
LayoutHorizontalAlignment	Specifies horizontal alignment for the layout.	Dependency property	HorizontalAlignment.CenterHorizontalAlignment.LeftHorizontalAlignment.RightHorizontalAlignment.Stretch	No
LayoutRoot	Gets or sets the layout root.	CLR property	IShape	No

You can set the horizontal and the vertical distance between the nodes in a tree layout using the HorizontalSpacing and VerticalSpacing properties. The spaces between sub-trees are specified using the SpaceBetweenSubTrees property.

Note: In case of Table layout, only the HorizontalSpacing and VerticalSpacing properties should be specified.

The following code illustrates these settings.

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel LayoutType="DirectedTreeLayout"
    HorizontalSpacing="50" VerticalSpacing="50" SpaceBetweenSubTrees="100"
    x:Name="diagramModel">
  </syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
  <!--View to display nodes and connections added through model.-->
  <syncfusion:DiagramControl.View>
    <syncfusion:DiagramView Name="diagramView">
  </syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DiagramModel diagramModel = new DiagramModel();diagramModel.VerticalSpacing
= 50;diagramModel.HorizontalSpacing = 50;diagramModel.SpaceBetweenSubTrees =
100;
```

VB.NET

```
Dim diagramModel As New DiagramModel()diagramModel.VerticalSpacing =
50diagramModel.HorizontalSpacing = 50diagramModel.SpaceBetweenSubTrees = 100
```

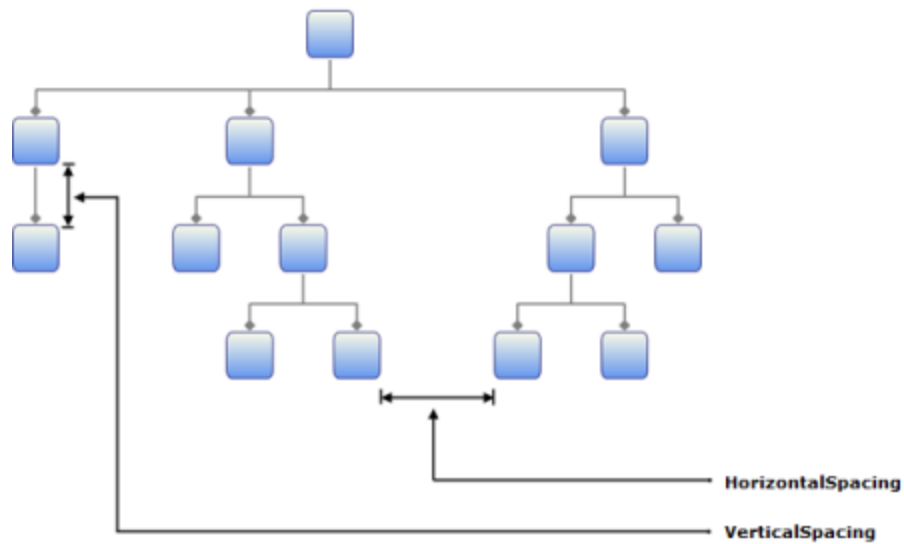
Pictorial Representation of Spacing Properties

The following are general spacing properties used in Directed and Hierarchical tree layouts:

- VerticalSpacing
- HorizontalSpacing
- SpaceBetweenSubTrees

Representation for Horizontal and Vertical Spacing Properties

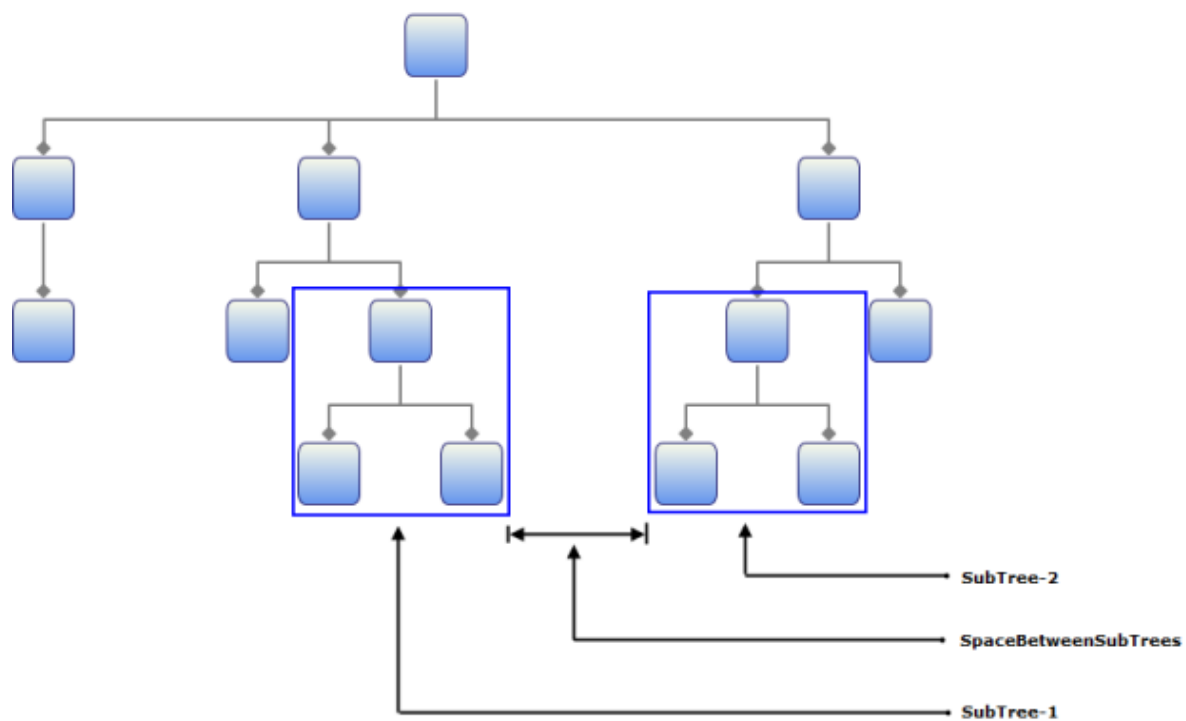
Horizontal Spacing refers to the spaces between the edges of the adjacent nodes (Siblings) and the Vertical Spacing refers to spaces between the nodes that lie at the next levels of the tree layout.



Horizontal and Vertical Spacing Properties

Representation for SpaceBetweenSubTrees properties

SpaceBetweenSubTrees refers to the spaces between adjacent Subtrees.



SpaceBetweenSubTrees property

Tree Orientation

The Layout Manager lets you orient the tree in many directions and can be used for the creation of many sophisticated arrangements. The Orientation property of Diagram Model can be used to specify the tree orientation.

Property	Description	Type of the property	Value it accepts	Any other dependencies/sub properties associated
Orientation	Gets or sets the orientation.	CLR property	TreeOrientation.LeftRightTreeOrientation.RightLeftTreeOrientation.TopBottomTreeOrientation.BottomTop	No

Following are the four orientations supported.

- TopBottom—Places the root node at the top and the child nodes are arranged below the root node.
- BottomTop—Places the root node at the Bottom and the child nodes are arranged above the root node.
- LeftRight—Places the root node at the Left and the child nodes are arranged on the right side of the root node.
- RightLeft—Places the root node at the Right and the child nodes are arranged on the left side of the root node.

The Bounds property of the DiagramView class can be used to specify the position of the root node based on which the entire tree gets generated.

The tree orientation can be set using the following code.

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel LayoutType="DirectedTreeLayout"
    Orientation="BottomTop" x:Name="diagramModel">
    </syncfusion:DiagramModel>
  </syncfusion:DiagramControl.Model>
  <!--View to display nodes and connections added through model.-->
  <syncfusion:DiagramControl.View>
    <syncfusion:DiagramView Name="diagramView">
    </syncfusion:DiagramView>
  </syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DiagramModel diagramModel = new DiagramModel();  
diagramModel.Orientation = TreeOrientation.BottomTop;
```

VB.NET

```
Dim diagramModel As New DiagramModel()  
diagramModel.Orientation = TreeOrientation.BottomTop
```

The tree orientation can be changed dynamically at run time using the following code for corresponding orientation types.

The following code may be specified in a Combobox SelectionChanged event.

C#

```
DirectedTreeLayout tree = new DirectedTreeLayout(diagramModel, diagramView);  
diagramModel.Orientation = TreeOrientation.RightLeft;  
tree.PrepareActivity(tree);  
tree.StartNodeArrangement();  
(diagramView.Page as DiagramPage).InvalidateMeasure();  
(diagramView.Page as DiagramPage).InvalidateArrange();
```

VB.NET

```
Dim tree As New DirectedTreeLayout(diagramModel, DiagramView)  
diagramModel.Orientation = TreeOrientation.RightLeft  
tree.PrepareActivity(tree)  
tree.StartNodeArrangement()  
TryCast(diagramView.Page, DiagramPage).InvalidateMeasure()  
TryCast(diagramView.Page, DiagramPage).InvalidateArrange()
```

The orientations are illustrated below.



BottomTop Orientation



TopBottom Orientation



LeftRight Orientation



RightLeft Orientation

Clear Nodes and Connections

Essential Diagram WPF allows you to clear the nodes and connections added to the diagram. It can be done by clearing the collections of nodes and connections from DiagramModel.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
Connections	Gets the connections.	CLR property	CollectionExt	No
Nodes	Gets the shapes.	CLR property	CollectionExt	No

The following lines of code can be used to clear nodes and connections.

C#

```
DiagramModel diagramModel = new DiagramModel();
diagramModel.Nodes.Clear();
diagramModel.Connections.Clear();
```

VB.NET

```
Dim diagramModel As New DiagramModel()
diagramModel.Nodes.Clear()
diagramModel.Connections.Clear()
```

Bind Data to Diagram Control

DiagramModel supports binding with business objects. Nodes will be created and added into the model depending upon the business object and template provided.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
ItemsSource	Gets or sets the source for the list of the items, the containers about to represent.	DependencyProperty	IEnumerable	No
HierarchicalDataTemplate	Gets or sets the HierarchicalDataTemplate for items.	DependencyProperty	HierarchicalDataTemplate	No
ItemTemplate	Gets or sets the ItemTemplate for items.	DependencyProperty	DataTemplate	No

HierarchicalDataTemplate and ItemsSource Properties

The ItemsSource property gets the source for the list of nodes to be added to the tree. The ItemTemplate property uses the items specified through the ItemsSource property specified in the data type.

The following code illustrates this.

XML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="420" Width="600"
```

```

xmlns:sfdiagram="clr-
namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Window.Resources>
<ResourceDictionary>
<local:CountrySalesList x:Key="myList"/>
<HierarchicalDataTemplate x:Key="myHierTemp" ItemsSource="{Binding
Path=RegionalSales}"
DataType="{x:Type local:CountrySale}">
</HierarchicalDataTemplate>
</ResourceDictionary>
</Window.Resources>
<Grid Name="diagramgrid">
<sfdiagram:DiagramControl IsSymbolPaletteEnabled="True">
<sfdiagram:DiagramModel x:Name="diagramModel"
LayoutType="DirectedTreeLayout"
HierarchicalDataTemplate="{StaticResource
myHierTemp}"></sfdiagram:DiagramModel>
</sfdiagram:DiagramControl.Model>
<sfdiagram:DiagramControl.View >
<sfdiagram:DiagramView Bounds="0,0,500,500"></sfdiagram:DiagramView>
</sfdiagram:DiagramControl.View>
</sfdiagram:DiagramControl>
</Grid>
</Window>

```

C#

```

this.Add(new CountrySale() { Name = "US", Sales = 28092 });
this[0].RegionalSales.Add(new RegionSale() { Name = "New York", Revenue =
2353 });
this[0].RegionalSales.Add(new RegionSale() { Name = "Los Angeles", Revenue =
3453 });
this[0].RegionalSales.Add(new RegionSale() { Name = "San Fransico", Revenue
= 8456 });

```

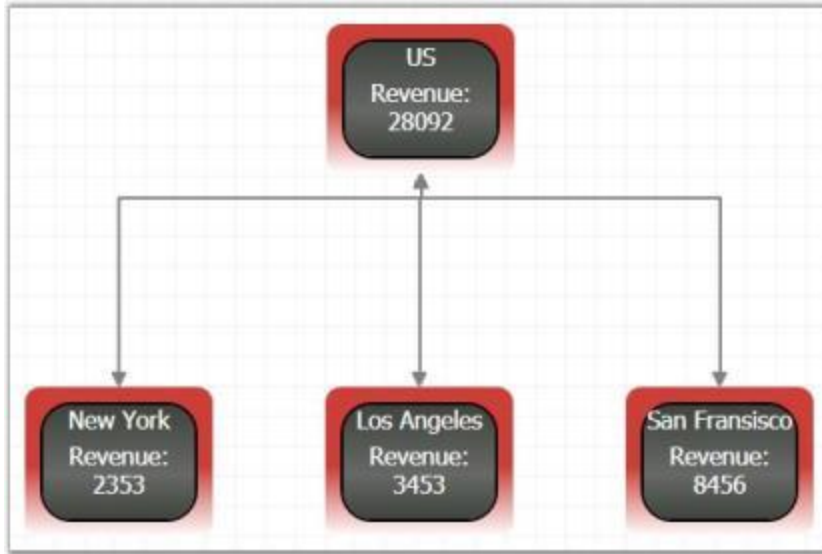
VB.NET

```

Me.Add(New CountrySale() With {.Name = "US", .Sales = 28092})
Me(0).RegionalSales.Add(New RegionSale() With {.Name = "New York", .Revenue
= 2353})
Me(0).RegionalSales.Add(New RegionSale() With {.Name = "Los Angeles",
.Revenue = 3453})
Me(0).RegionalSales.Add(New RegionSale() With {.Name = "San Fransico",
.Revenue = 8456})

```

This creates a tree view with CountrySale as the root node and region sales as the child nodes.



Data bound to the Diagram Control

Cyclic path in Hierarchical-Tree Layout

The Hierarchical-Tree layout provides support for creating cyclic paths. A cycle is said to exist if nodes are connected in a chain such that the last node in the chain is connected back to the first node. For example, if there are four nodes namely n1, n2, n3 and n4, such that n1 is connected to n2, n2 is connected to n3, n3 is connected to n4, and n4 is again connected to n1 (n1-->n2-->n3-->n4), then these nodes are said to form a cycle.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
EnableCycleDetection	Gets or sets a value indicating whether Cycle detection is enabled or not.	DependencyProperty	Boolean (true/false)	No

To specify a cyclic path is as giving input to the Hierarchical-Tree layout. EnableCycleDetection property must be set to 'True'. Enabling this property checks for cycles and makes connections accordingly.

Note: The EnableCycleDetection property takes effect only for the Hierarchical-Tree layout type of the Diagram Model.

The following code example illustrates how to set the EnableCycleDetection property.

HTML

```

<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel x:Name="diagramModel"
      LayoutType="HierarchicalTreeLayout" EnableCycleDetection="True"
  
```

```

Orientation="TopBottom" >
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<!--View to display nodes and connections added through model.-->
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>

```

C#

```

DiagramModel diagramModel = new DiagramModel();
diagramModel.Orientation = TreeOrientation.TopBottom;
diagramModel.LayoutType = LayoutType.HierarchicalTreeLayout;
diagramModel.EnableCycleDetection = true;
diagramControl.Model = diagramModel;

```

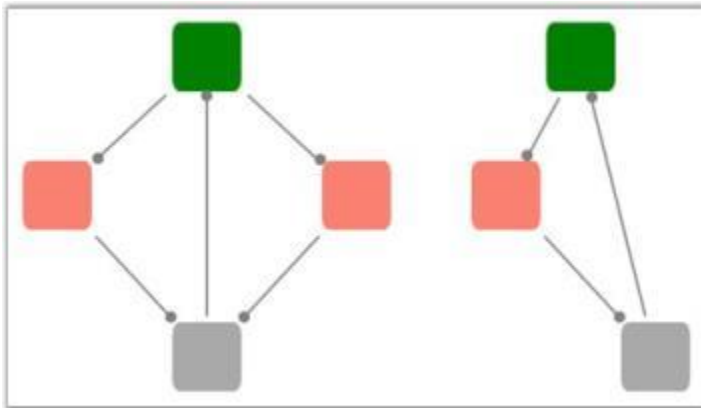
VB.NET

```

Dim diagramModel As New DiagramModel()
diagramModel.Orientation = TreeOrientation.TopBottom
diagramModel.LayoutType = LayoutType.HierarchicalTreeLayout
diagramModel.EnableCycleDetection = True
diagramControl.Model = diagramModel

```

The following screenshot illustrates Cyclic Paths in the Hierarchical-Tree layout.



Cyclic Paths In Hierarchical-Tree Layout

Note: If a cyclic path is specified as input when the EnableCycleDetection property is set to False, then a stack overflow exception is thrown as the loop goes on forever.

Advantages

Cyclic paths are very useful to demonstrate work flows which involve repeated processes.

Automatic Layout Refer Getting Started -> Automatic Layout

Table Expand Mode

The `TableExpandMode` property is an enumeration which takes two values, `Horizontal` and `Vertical`. Default value is `Horizontal`. It specifies how the table gets expanded when more items are added to the model.

Property

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
<code>TableExpandMode</code>	Gets or sets the table expand mode.	<code>DependencyProperty</code>	<code>ExpandMode.Horizontal</code> <code>ExpandMode.Vertical</code>	No

- When `TableExpandMode` is set to `Horizontal`, the row count is automatically calculated based on the number of nodes. The `ColumnCount` must be specified and the nodes will be arranged in the specified number of columns. When the maximum column count is reached, it starts placing the nodes in a new row.
- When `TableExpandMode` is set to `Vertical`, the column count is automatically calculated based on the number of nodes. The row count must be specified and the nodes will be arranged in the specified number of rows. When the maximum row count is reached, the nodes are placed in a new column.

The `TableExpandMode` can be set in the following way,

C#

```
DiagramControl dc = new DiagramControl();
DiagramModel diagramModel = new DiagramModel();
dc.Model = diagramModel;
diagramModel.TableExpandMode=TableExpandMode.Horizontal;
```

C#

```
Dim dc As New DiagramControl()
Dim diagramModel As New DiagramModel()
dc.Model = diagramModel
diagramModel.TableExpandMode=TableExpandMode.Horizontal
```

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramModel
    LayoutType="TableLayout"
```

```
TableExpandMode="Horizontal"  
HorizontalSpacing="50"  
VerticalSpacing="50"  
RowCount="4"  
ColumnCount="4"  
x:Name="diagramModel">  
</syncfusion:DiagramModel>  
<!--View to display nodes and connections added through model.-->  
<syncfusion:DiagramControl.View>  
<syncfusion:DiagramView Name="diagramView">  
</syncfusion:DiagramView>  
</syncfusion:DiagramControl.View>  
</syncfusion:DiagramControl>
```

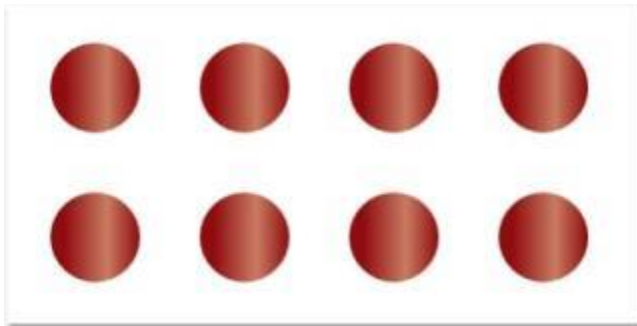


Table layout with TableExpandMode as Horizontal and ColumnCount

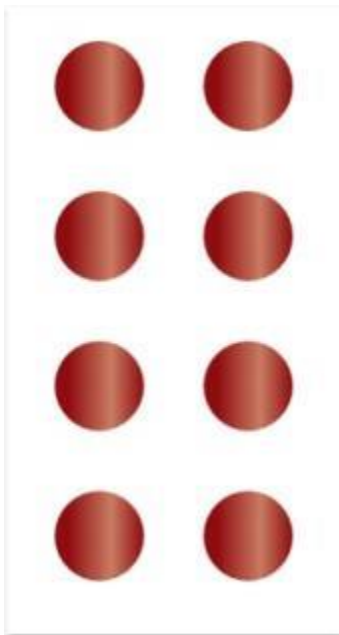


Table layout with TableExpandMode as Vertical and RowCount

Row Count and Column Count

RowCount and ColumnCount properties are used to specify the maximum number of rows and columns allowed in the table. Refer TableExpandMode property for more details.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
RowCount	Gets or sets the Row Count for the table layout.	DependencyProperty	int	No
ColumnCount	Gets or sets the Column Count for the table layout.	DependencyProperty	int	No

The RowCount and ColumnCount can be set in the following way:

C#

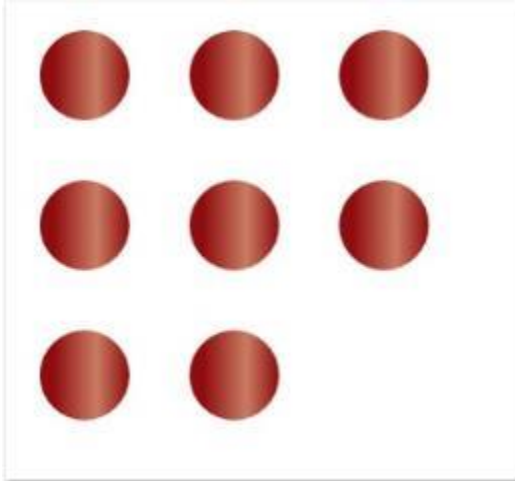
```
DiagramControl dc = new DiagramControl();
DiagramModel diagramModel = new DiagramModel();
dc.Model = diagramModel;
diagramModel.RowCount=6;
diagramModel.ColumnCount=5;
```

VB.NET

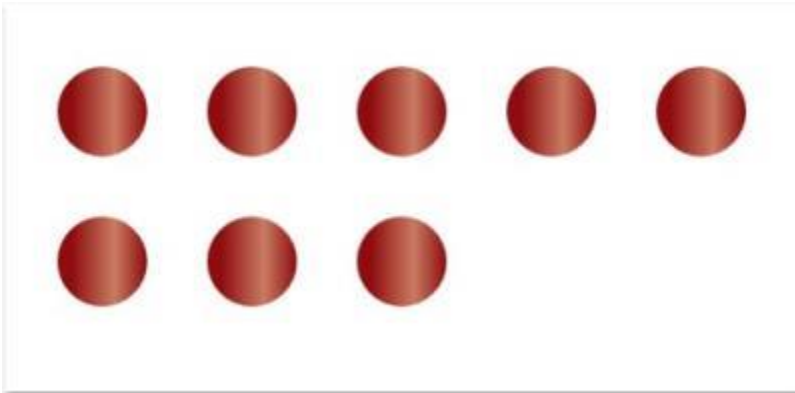
```
Dim dc As New DiagramControl()
Dim diagramModel As New DiagramModel()
dc.Model = diagramModel
diagramModel.RowCount=6
diagramModel.ColumnCount=5
```

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramModel
    LayoutType="TableLayout"
    TableExpandMode="Horizontal"
    HorizontalSpacing="50"
    VerticalSpacing="50"
    RowCount="3"
    ColumnCount="5"
    x:Name="diagramModel">
  </syncfusion:DiagramModel>
  <!--View to display nodes and connections added through model.-->
  <syncfusion:DiagramControl.View>
  <syncfusion:DiagramView Name="diagramView">
  </syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```



RowCount specified as 3



ColumnCount specified as 5

Enable Table Layout with Varied Node Sizes

When the `EnableLayoutWithVariedSizes` property is set to `True`, center aligns the content of each cell so that all the nodes in that row and column in the table get aligned with respect to the larger cell size. This property can be set to `true`, if the nodes are of different sizes (width and height).

Property	Description	Type of the property	Value it accepts	Any other dependencies/sub properties associated
<code>EnableLayoutWithVariedSizes</code>	Gets or sets a value indicating whether to enable the varied size algorithm. In case the Model consists of the nodes of different sizes,	<code>DependencyProperty</code>	Boolean (true/false)	No

	this property can be set to true. This will align the differently sized nodes with respect to the center.			
--	---	--	--	--

The EnableLayoutWithVariedSizes can be set in the following way:

C#

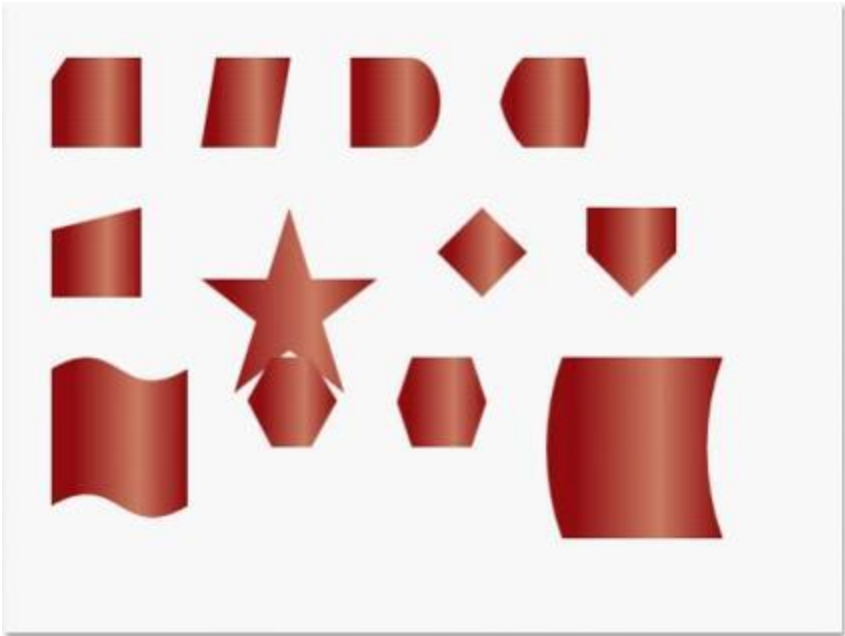
```
DiagramControl dc = new DiagramControl();
DiagramModel diagramModel = new DiagramModel();
dc.Model = diagramModel;
diagramModel.EnableLayoutWithVariedSizes = true;
```

VB.NET

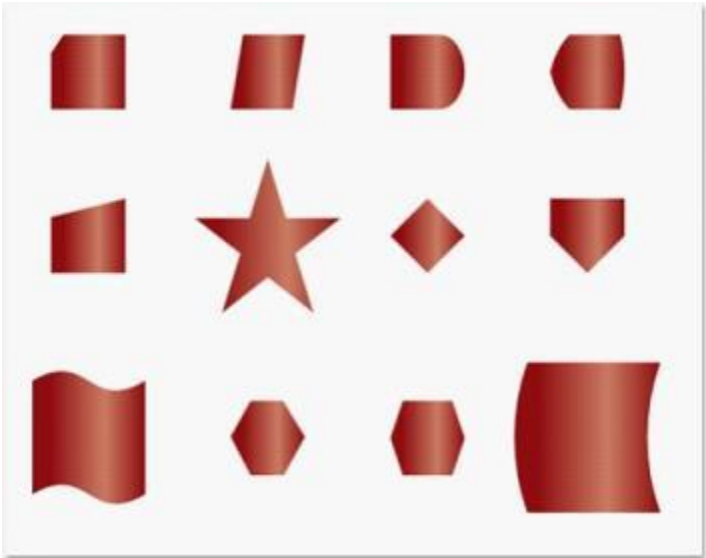
```
Dim dc As New DiagramControl()
Dim diagramModel As New DiagramModel()
dc.Model = diagramModel
diagramModel.EnableLayoutWithVariedSizes = True
```

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramModel
    LayoutType="TableLayout"
    TableExpandMode="Horizontal"
    HorizontalSpacing="50"
    VerticalSpacing="50"
    RowCount="3"
    ColumnCount="5"
    EnableLayoutWithVariedSizes="True"
    x:Name="diagramModel">
  </syncfusion:DiagramModel>
  <!--View to display nodes and connections added through model.-->
  <syncfusion:DiagramControl.View>
    <syncfusion:DiagramView Name="diagramView">
    </syncfusion:DiagramView>
  </syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```



EnableLayoutWithVariedSize set to false



EnableLayoutWithVariedSize set to true

Diagram View in WPF Diagram (classic)

The Diagram View is responsible for bringing the objects and data which are added into the view through the model. In other words, it deals with the visual representation of data. Zooming and panning are done with respect to the view.

Property	Description	Type of the property	Value it accepts	Any other dependencies/sub
----------	-------------	----------------------	------------------	----------------------------

				properties associated
IsZoomEnabled	Gets or sets a value indicating whether zoom is enabled or not. Default value is True.	Dependency property	Boolean(True/False)	No
IsPanEnabled	Gets or sets a value indicating whether pan is enabled or not. Default value is False.	Dependency property	Boolean(True/False)	No
ZoomFactor	Gets or sets a factor for the zoom. Default value is 0.2.	Dependency property	Double	No
Bounds	Gets or sets the bounds value which specifies the position of the root node in case of tree layout.	CLR property	Thickness	No

IsPageEditable	Gets or sets a value indicating whether page is enabled or not.Default value is True.	Dependency property	Boolean(True/False)	No
ShowVerticalRulers	Gets or sets a value indicating whether vertical rulers are displayed or not.Default value is True.	Dependency property	Boolean(True/False)	No
ShowHorizontalRulers	Gets or sets a value indicating whether horizontal rulers are displayed or not.Default value is True.	Dependency property	Boolean(True/False)	No
ShowVerticalGridLine	Gets or sets a value indicating whether vertical grid lines are displayed or not.Default	Dependency property	Boolean(True/False)	No

	value is True.			
ShowHorizontalGridLine	Gets or sets a value indicating whether horizontal grid lines are displayed or not. Default value is True.	Dependency property	Boolean(True/False)	No
PortVisibility	Gets or sets a value indicating whether all ports of all the nodes on the page are visible or not. If an individual node's PortVisibility is set, then the node's PortVisibility property will take precedence. Default value is Visibility.Visible	Dependency property	Visibility.Visible Visibility.Collapsed Visibility.Hidden	No
Page	Gets or sets the DiagramPage.	Dependency property	Panel	No

NodeContextMenu	Gets or sets context menu for the node.	Dependency property	ContextMenu	No
LineConnectorContextMenu	Gets or sets context menu for line connector.	Dependency property	ContextMenu	No
UndoRedoEnabled	Gets or sets a value indicating whether undo redo is enabled or not.	Dependency property	Boolean(True/False)	No
IsCutEnabled	Gets or sets a value indicating whether cut command is enabled. Default value is True.	Dependency property	Boolean(True/False)	No
IsCopyEnabled	Gets or sets a value copy command is enabled. Default value is True.	Dependency property	Boolean(True/False)	No
IsPasteEnabled	Gets or sets a value indicating whether paste	Dependency property	Boolean(True/False)	No

	command is enabled. Default value is True.			
IsPageSaved	Gets or sets a value indicating whether a page is saved.	Dependency property	Boolean (true/false)	No
IsConnectorDragCancel	Gets or sets whether the value indicating that connector drag can be canceled.	CLR property	Boolean (true/false)	No
IsNodeDragCancel	Gets or sets a value indicating that node drag can be canceled.	CLR property	Boolean (true/false)	No
HorizontalScrollBarVisibility	Specifies the visibility for the vertical scroll bar. The default value is Auto.	Dependency property	ScrollBarVisibility.AutoScrollBarVisibility.DisabledScrollBarVisibility.HiddenScrollBarVisibility.Visible	No
VerticalScrollBarVisibility	Specifies the visibility for the vertical scroll bar. The default value is Auto.	Dependency property	ScrollBarVisibility.AutoScrollBarVisibility.DisabledScrollBarVisibility.HiddenScrollBarVisibility.Visible	No

SelectionList	Gets the Selection List for the items.	CLR property	NodeCollection	No
---------------	--	--------------	----------------	----

XML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:sfdiagram="clr-namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<sfdiagram:DiagramControl IsSymbolPaletteEnabled="True">
<sfdiagram:DiagramControl.View>
<sfdiagram:DiagramView ></sfdiagram:DiagramView>
</sfdiagram:DiagramControl.View>
</sfdiagram:DiagramControl>
</Grid>
</Window>
```

C#

```
DiagramControl dc = new DiagramControl();
dc.IsSymbolPaletteEnabled = true;
DiagramView view = new DiagramView();
view.Bounds = new Thickness(0, 0, 1000, 1000);
dc.View = view;
diagramgrid.Children.Add(dc);
```

VB.NET

```
Dim dc As New DiagramControl()
dc.IsSymbolPaletteEnabled = True
Dim view As New DiagramView()
view.Bounds = New Thickness(0, 0, 1000, 1000)
dc.View = view
diagramgrid.Children.Add(dc)
```

The drawing area has many properties that can be used to customize a view.

- Create Rulers Refer Concepts and Features -> Diagram View -> Create Rulers
- Specify Bounds Refer Concepts and Features -> Diagram View -> Specify Bounds
- Panning Refer Concepts and Features -> Diagram View -> Panning
- Zoom Commands Refer Concepts and Features -> Diagram View -> Zoom Commands
- Create Page Refer Concepts and Features -> Diagram View -> Create Page
- Page Editing option Refer Concepts and Features -> Diagram View -> Page Editing option

- Grid Lines Refer Concepts and Features -> Diagram View -> Grid Lines
- Customize the ContextMenu of Nodes and LineConnectors Refer Concepts and Features -> General -> Customize the ContextMenu of Nodes and LineConnectors
- Undo and Redo Command. Refer Concepts and Features -> Diagram View -> Undo and Redo Command.

Create Rulers

Rulers display the coordinates of elements on the diagram page. Negative label values get displayed on the ruler in case the page is panned to the right side. On Zooming, the ruler values get adjusted accordingly, to match with the current Zoom level. At any point, the ruler value always indicates the exact coordinates of the page and its elements. So when the page is zoomed, the interval values get halved or doubled depending upon the zoom level. By default, labels of the major lines in rulers will represent the pixel values using a label when there is a change in Diagram Page's MeasurementUnit. The ruler and the Label will be updated so that the label indicates the respective unit values.

Panning, Zooming, MeasurementUnits are explained in later part of this documentation.

Horizontal and Vertical ruler can be initialized for DiagramView in two ways:

- Through XAML
- Through Code Behind

HTML

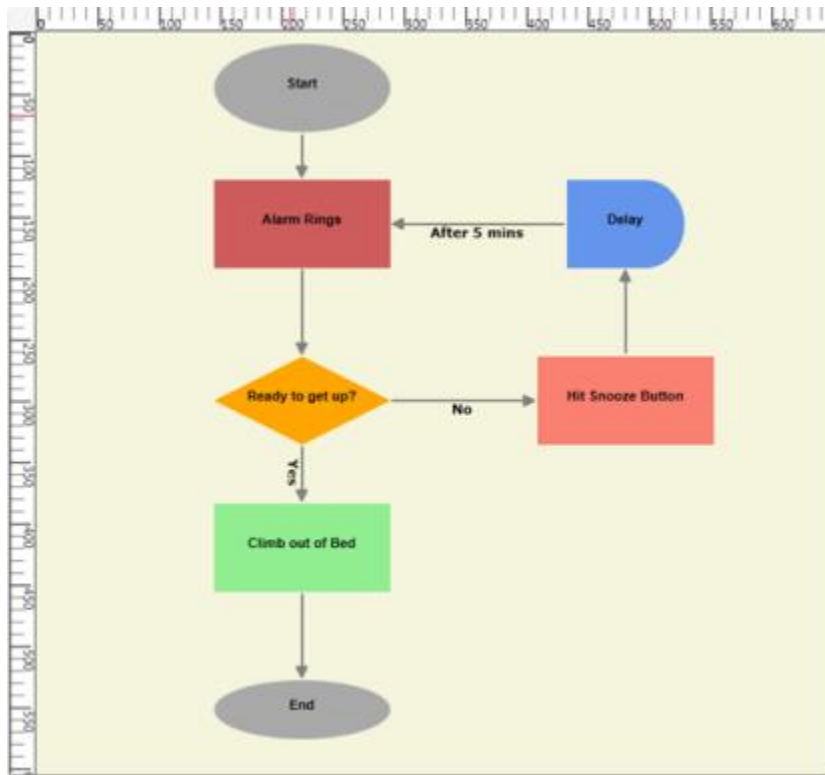
```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel x:Name="diagramModel">
    </syncfusion:DiagramModel>
  </syncfusion:DiagramControl.Model>
  <!--View to display nodes and connections added through model.-->
  <syncfusion:DiagramView IsPageEditable="True" Bounds="0,0,12,12"
  Name="diagramView">
    <syncfusion:DiagramView.HorizontalRuler>
    <syncfusion:HorizontalRuler Name="horizontalRuler" />
    </syncfusion:DiagramView.HorizontalRuler>
    <syncfusion:DiagramView.VerticalRuler>
    <syncfusion:VerticalRuler Name="verticalRuler" />
    </syncfusion:DiagramView.VerticalRuler >
  </syncfusion:DiagramView>
</syncfusion:DiagramControl>
```

C#

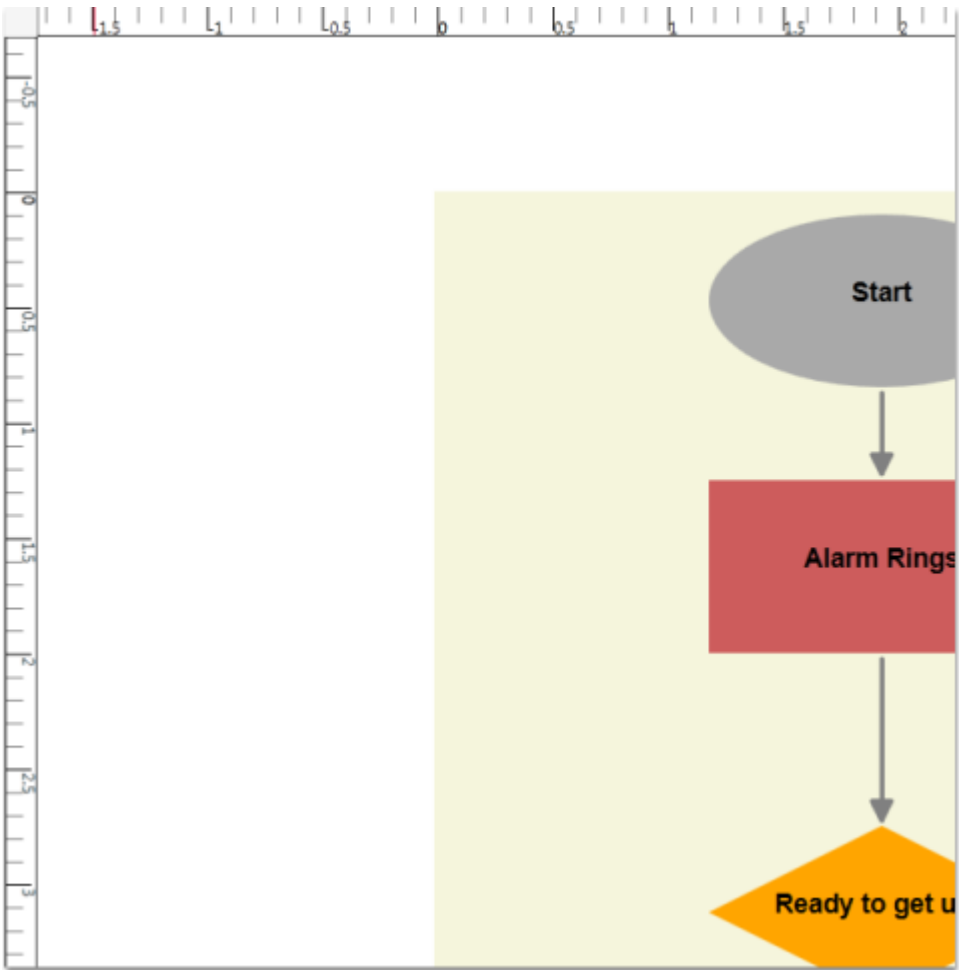
```
DiagramView diagramView = new DiagramView();
diagramView.HorizontalRuler = new HorizontalRuler();
diagramView.VerticalRuler as VerticalRuler();
```

VB.NET

```
Dim diagramView As New DiagramView()  
diagramView.HorizontalRuler = New HorizontalRuler()  
TryCast(diagramView.VerticalRuler, VerticalRuler())
```



Rulers

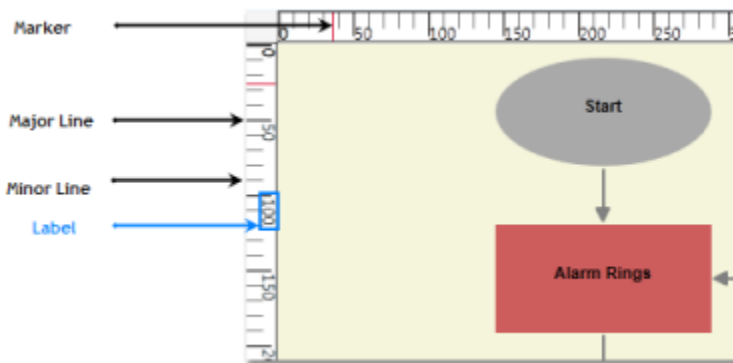


Ruler after Panning, zooming and Measurement unit as Inch

Several customizable options have been provided for the horizontal and vertical rulers. These are common for both the rulers.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
Background	Get or sets the ruler's background color.	Dependency property	Brush	No
MarkerBrush	Gets or sets the MarkerBrush.	Dependency property	Brush	No
LabelFontColor	Gets or sets the LabelFontColor.	Dependency property	Brush	No
MinorLinesStroke	Gets or sets the MinorLinesStroke.	Dependency property	Brush	No

MajorLinesStroke	Gets or sets the MajorLinesStroke.	Dependency property	Brush	No
MajorLinesThickness	Gets or sets the MajorLinesThickness.	Dependency property	Double	No
MinorLinesThickness	Gets or sets the MinorLinesThickness.	Dependency property	Double	No
MarkerThickness	Gets or sets the MarkerThickness.	Dependency property	Double	No



Ruler Terminology

The following code shows how the properties can be set.

HTML

```
<syncfusion:DiagramView IsPageEditable="True" Bounds="0,0,12,12"
Name="diagramView">
<syncfusion:DiagramView.HorizontalRuler>
<syncfusion:HorizontalRuler Name="horizontalRuler" Background="#FFC6C6C6"
LabelFontColor="Green"/>
</syncfusion:DiagramView.HorizontalRuler>
<syncfusion:DiagramView.VerticalRuler>
<syncfusion:VerticalRuler Name="verticalRuler" Background="#FFC6C6C6"
LabelFontColor="Green"/>
</syncfusion:DiagramView.VerticalRuler >
</syncfusion:DiagramView>
```

C#

```
DiagramView diagramView = new DiagramView();
(diagramView.HorizontalRuler as HorizontalRuler).LabelFontColor =
Brushes.Green;
(diagramView.VerticalRuler as VerticalRuler).LabelFontColor = Brushes.Green;
```

VB.NET

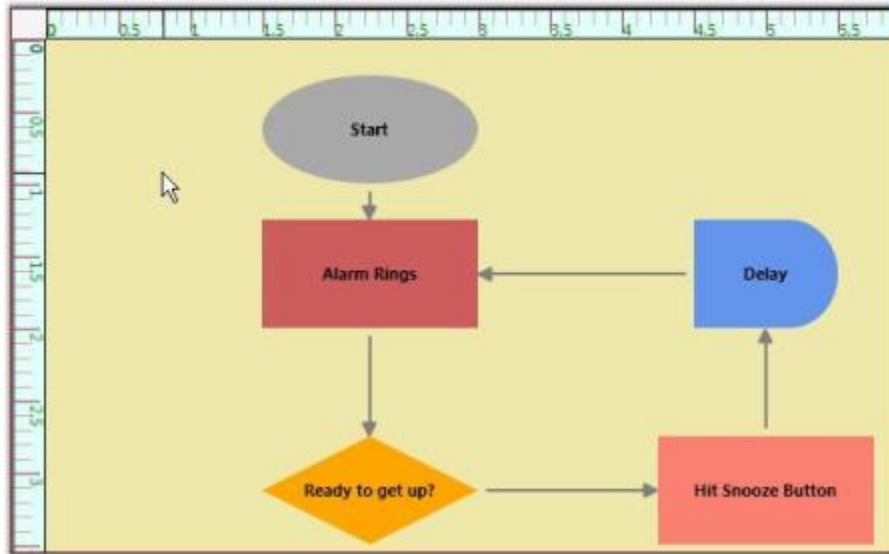
```
Dim diagramView As New DiagramView()
```



```

TryCast(diagramView.HorizontalRuler, HorizontalRuler).LabelFontColor =
Brushes.Green
TryCast(diagramView.VerticalRuler, VerticalRuler).LabelFontColor =
Brushes.Green

```



Custom Ruler

Specify Bounds

The Bounds property of the Diagram View class enables a user to specify the rectangular area where the tree layout is to be displayed. The root of the tree layout is placed at the center of the bounds value.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
Bounds	Gets or sets the bounds value which specifies the position of the root node in case of tree layout.	CLR property	Thickness	No

The following code can be used to set the Bounds property.

Bounds can be specified in two ways,

- Through XAML
- Through Code Behind

XAML

```

<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"

```

```

xmlns:sfdiagram="clr-
namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<sfdiagram:DiagramControl IsSymbolPaletteEnabled="True" >
<sfdiagram:DiagramControl.View >
<sfdiagram:DiagramView Bounds="0,0,500,500">
</sfdiagram:DiagramView>
</sfdiagram:DiagramControl.View>
</sfdiagram:DiagramControl>
</Grid>
</Window>

```

C#

```

DiagramControl dc = new DiagramControl();
dc.IsSymbolPaletteEnabled = true;
DiagramView view = new DiagramView();
view.Bounds = new System.Drawing.Thickness(0, 0, 500, 500);
dc.View = view;
diagramgrid.Children.Add(dc);

```

VB.NET

```

Dim dc As New DiagramControl()
dc.IsSymbolPaletteEnabled = True
Dim view As New DiagramView()
view.Bounds = New System.Drawing.Thickness(0, 0, 500, 500)
dc.View = view
diagramgrid.Children.Add(dc)

```

Panning

Essential Diagram WPF provides the ability to pan a page. Panning is used to move the contents of page both horizontally and vertically by holding down a mouse button and then moving the mouse.

Property	Description	Type Of the property	Value it accepts	Any other dependencies/ sub properties associated
IsPanEnabled	Gets or sets a value indicating whether pan is enabled or not.Default value is False.	Dependency property	Boolean(True/False)	No

Steps for panning a page

1. A page can be panned by setting the IsPanEnabled property to *True*.

C#

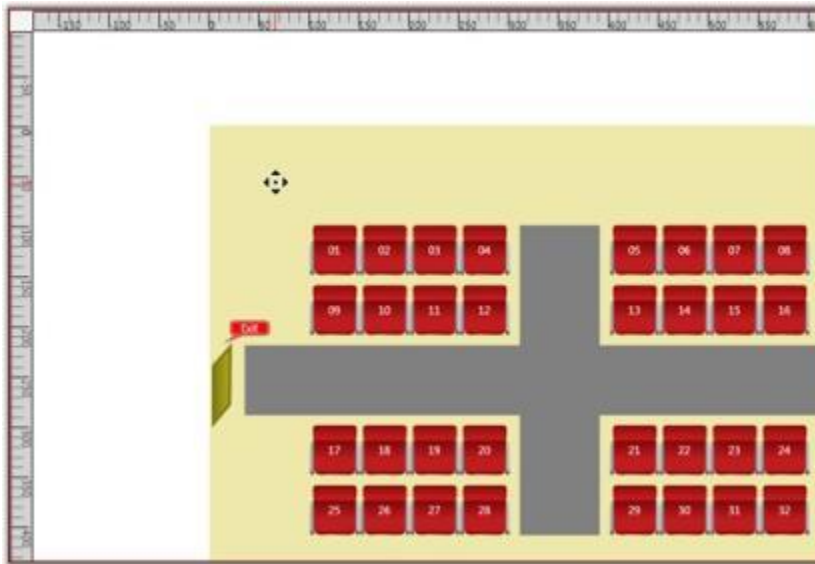
```
DiagramView diagramView = new DiagramView();
diagramView.IsPanningEnabled = true;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.IsPanningEnabled = True
```

2. Click and drag the diagram page to the desired position. Note that the negative rulers get displayed while panning to the right.

Note: No other operations can be performed on page elements while IsPanningEnabled is set to True.



Pan

Creating Page

The Diagram View has a Page property which refers to the DiagramPage class. The DiagramPage displays the nodes and connections which are added through the model.

Property	Description	Type of the property	Value it accepts	Any other dependencies/sub properties associated
Page	Gets or sets the DiagramPage.	Dependency property	DiagramPage	No

The DiagramPage can be created for DiagramView in following two ways,

- Through XAML
- Through Code Behind

XAML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:sfdiagram="clr-
namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1" >
<Grid Name="diagramgrid">
<sfdiagram:DiagramControl IsSymbolPaletteEnabled="True" >
<sfdiagram:DiagramControl.Model>
<sfdiagram:DiagramModel x:Name="diagramModel" >
</sfdiagram:DiagramModel>
</sfdiagram:DiagramControl.Model>
<sfdiagram:DiagramControl.View >
<sfdiagram:DiagramView ShowHorizontalGridLine="True"
ShowVerticalGridLine="True">
<syncfusion:DiagramView.Page>
<syncfusion:DiagramPage x:Name="diagramPage" GridHorizontalOffset="50"
GridVerticalOffset="50"/>
</syncfusion:DiagramView.Page>
</sfdiagram:DiagramView>
</sfdiagram:DiagramControl.View>
</sfdiagram:DiagramControl>
</Grid>
</Window>

```

C#

```

DiagramControl dc = new DiagramControl();
dc.IsSymbolPaletteEnabled = true;
DiagramView view = new DiagramView();
dc.View = view;
(diagramView.Page as DiagramPage).GridHorizontalOffset = 50;
(diagramView.Page as DiagramPage).GridVerticalOffset = 50;

```

VB.NET

```

Dim dc As New DiagramControl()
dc.IsSymbolPaletteEnabled = True
Dim view As New DiagramView()
dc.View = view
TryCast(diagramView.Page, DiagramPage).GridHorizontalOffset = 50
TryCast(diagramView.Page, DiagramPage).GridVerticalOffset = 50

```

Page editing option

You can disable editing the page by setting the `IsPageEditable` property to `False`. No operation can then be performed on the page or its contents. Default value is `True`.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
IsPageEditable	Gets or sets a value indicating whether page is enabled or	Dependency property	Boolean(True/False)	No

	not.Default value is True.			
--	----------------------------	--	--	--

This property is in DiagramView and can be set in the following ways.

- Through XAML
- Through Code Behind

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel x:Name="diagramModel">
    </syncfusion:DiagramModel>
  </syncfusion:DiagramControl.Model>
  <!--View to display nodes and connections added through model.-->
  <syncfusion:DiagramControl.View>
    <syncfusion:DiagramView IsPageEditable="True" Name="diagramView">
    </syncfusion:DiagramView>
  </syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DiagramView diagramView = new DiagramView();
diagramView.IsPageEditable = true;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.IsPageEditable = True
```

Clear the Selection List on Right-Click

By default, the selection list can be cleared by right-clicking on the diagram page. This clearing can be enabled or disabled using the `ClearSelectionOnRightClick` property.

Properties

Property	Description	Type	Data Type
ClearSelectionOnRightClick	This property is used to enable/disable the selection list clearing on right-click.	Dependency	Boolean

Adding ClearSelectionOnRightclick Property to an Application

Clearing the selection list on right-click can be enabled using the `ClearingSelectionOnRightClick` property of the `DiagramView`. By default this property is set to true. Clearing selection can be controlled if the `ClearingSelectionOnRightClick` property of the `DiagramView` is set to false.

C#

```
DiagramView diagramView = new DiagramView();  
//Disable Clearing Selection list on RightClick.  
diagramView.ClearSelectionOnRightClick = false;
```

VB.NET

```
Dim diagramView As New DiagramView()  
'Disable Clearing Selection list on RightClick.  
diagramView.ClearSelectionOnRightClick = False
```

Fit-to-Page Support

Fit-to-page will bring the whole diagram into the viewport using the zooming options. This helps you view the whole diagram page in the viewport without having to scroll.

Use Case Scenarios

In a large network diagram each object will be far from another. You need to scroll the diagram page to explore the objects that are outside the viewport. Using this feature you can bring the whole diagram into the viewport area.

Properties

Property	Description	Type	Data Type
EnableFitToPage	Gets or sets the FitToPage property of the diagram page.	Dependency	Boolean
FitToPageCommand	This ICommand is used to execute the fit-to-page on demand.	Dependency	ICommand

Sample Link

To view a sample of this feature:

1. Open Dashboard.
2. Click User Interface > WPF.
3. Click Run Samples.
4. Navigate to Diagram> Overview Demo.

Note: A demo of this feature is included in the Overview sample.

Enabling Fit-to-Page

You can enable the feature in two methods. They are:

- Using Property
- Using Command

Using Property

You can enable fit-to-page to bring the whole diagram within the viewport, either by zooming in or by zooming out.

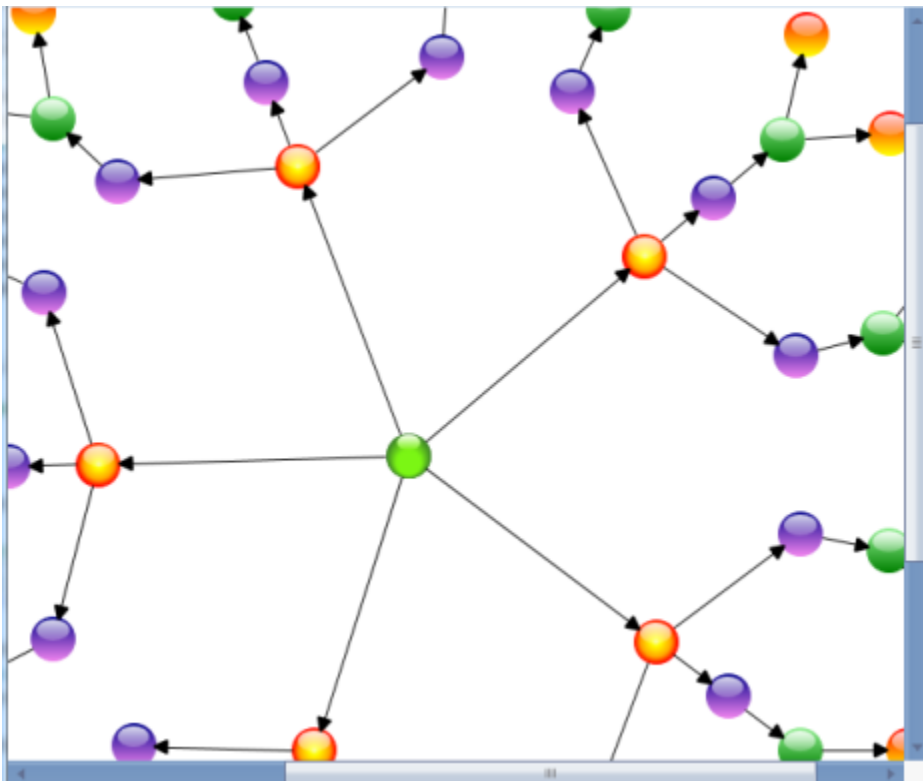
To enable this feature, set the `EnableFitToPage` property of the diagram view to `True`. The following code illustrates this:

HTML

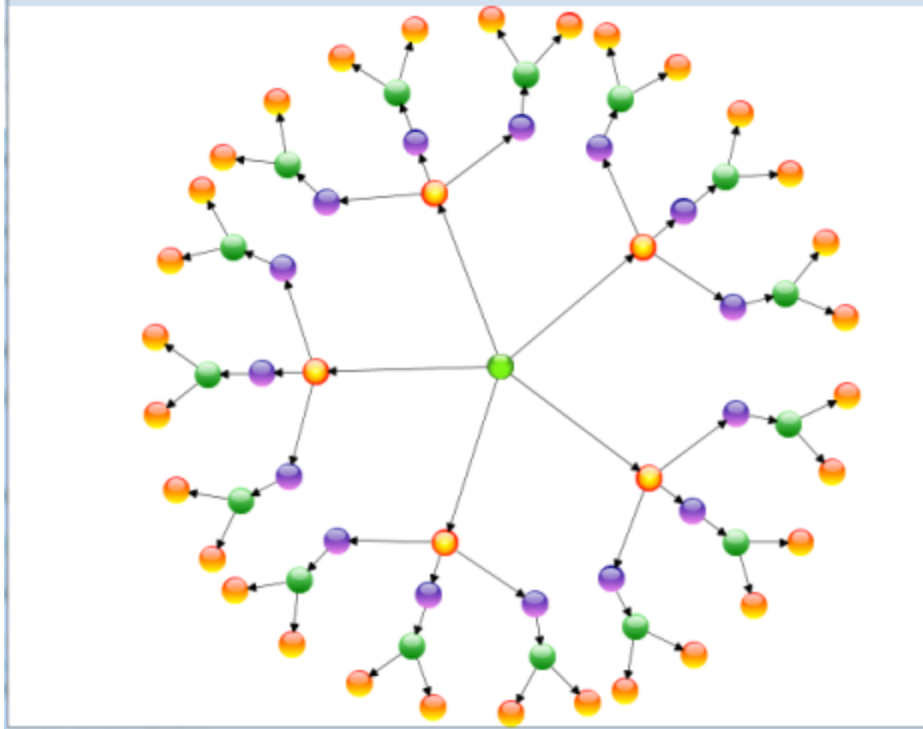
```
<Syncfusion:DiagramControl Grid.Row="1" Name="diagramControl" >  
  <Syncfusion:DiagramControl.Model>  
    <Syncfusion:DiagramModel x:Name="diagramModel">  
    </Syncfusion:DiagramModel>  
  </Syncfusion:DiagramControl.Model>  
  <Syncfusion:DiagramControl.View>  
    <Syncfusion:DiagramView EnableFitToPage="True" Name="diagramView">  
    </Syncfusion:DiagramView>  
  </Syncfusion:DiagramControl.View>  
</Syncfusion:DiagramControl>
```

C#

```
// Enable FitToPage of the DiagramView  
diagramView.EnableFitToPage = true;
```



Fit-to-Page Disabled



Fit-to-Page Enabled

[Using Command](#)

You can execute a fit-to-page command to bring the whole diagram within the viewport either by zooming in or zooming out.

Note: Using this command you can fit the content within the viewport even if the `EnableFitToPage` property of the diagram view is set to `False`.

The following code illustrates how to bring the whole diagram into the viewport using a command:

HTML

```
<Button Command="FitToPage" CommandTarget="{Binding ElementName=diagramView}"
  "CommandParameter="{Binding ElementName=diagramPage}">
  FitToPage
</Button>
```

C#

```
DiagramCommandManager.FitToPage.Execute.(diagramView.Page, diagramView);
```

[Table Layout for Selected Nodes](#)

From version 10.1.0.44, Essential Diagram for WPF enables you to apply the table layout on selected nodes instead of applying it to the entire diagram. This arranges selected nodes or a given node collection in a tabular structure based on specified intervals between them. The number of nodes in each row and column can be specified and the layout will be applied accordingly.

Use Case Scenarios

- Users can easily make the layout with a specific collection of nodes called ordered nodes.
- Users can easily position the layout.
- Users can easily align the layout by using the layout alignment properties.
- Users can set a rectangle boundary around nodes by using the Layout Bounds property.

Properties

Property	Description	Type	Data Type
OrderedNodes	This property is used to get or set the Collection of Nodes for table layout.	Dependency property	List>IShape<

Sample Link

To view a sample of this feature:

1. Open Dashboard.
2. Click User Interface > WPF.
3. Click Run Samples.
4. Navigate to Diagram > Automatic Layout > Table Layout.

Adding Table Layout for selected Nodes

To apply a table layout to the selected nodes, assign the selected nodes to the OrderNodes property of the DiagramModel. You can also assign your own collection of IShape to the OrderNodes property. Then create an instance of the TableLayout and call the RefreshLayout method for this instance.

The following code illustrates this:

C#

```
// Assigning selected node to the OrderedNodes property.
diagramModel.OrderedNodes=
diagramView.SelectionList.OfType<IShape>().ToList();
// Create an instance of TableLayout and refresh it.
TableLayout table = new TableLayout(diagramModel, diagramView);
table.RefreshLayout();
```

VB.NET

```
'Assigning selected node to the OrderedNodes property.
diagramModel.OrderedNodes= diagramView.SelectionList.OfType(Of
IShape)().ToList()
'Create an instance of TableLayout and refresh it.
Dim table As New TableLayout(diagramModel, diagramView)
table.RefreshLayout()
```

When the code runs, the table layout will be applied to the specified node collection.

Note: If the OrderNodes property is set to null, then the table layout will be applied to the entire diagram.

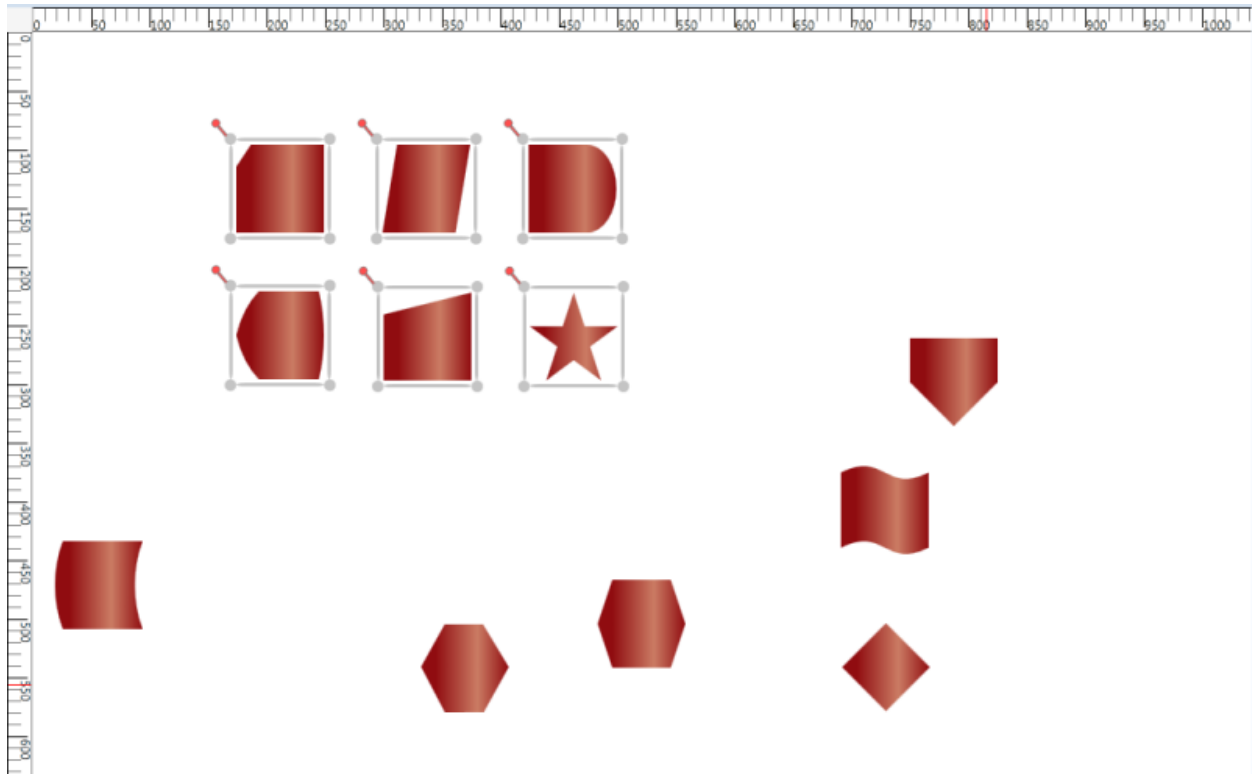


Table Layout Applied for Specified Nodes

Aligning the Layout on a Specified Location

To align the ordered nodes in a particular position, call the *TableLayout's RefreshLayout* (Point PivotPoint) method and specify the particular point as a parameter. The layout will be positioned in the specified pivot point.

The following code illustrates this:

C#

```
// Assigning selected node to the OrderedNodes.
diagramModel.OrderedNodes=
diagramView.SelectionList.OfType<IShape>().ToList();
TableLayout table = new TableLayout(diagramModel, diagramView);
table.RefreshLayout(300,400);
```

VB.NET

```
\Assigning selected node to the OrderedNodes.
diagramModel.OrderedNodes= diagramView.SelectionList.OfType(of
IShape)().ToList()
Dim table As New TableLayout(diagramModel, diagramView)
table.RefreshLayout(300,400)
```

Removing Table Layout from the Specific Nodes

You can remove the table layout applied to specific nodes. To achieve this set the *OrderedNodes* property of the *DiagramMode* to null, and call the *RefreshLayout* method of the *TableLayout*. The layout will be applied to the entire diagram. By default the *OrderedNodes* property is set to null.

The following code illustrates how to remove the layout from the specific nodes:

C#

```
// Set null value to the OrderedNodes property.
diagramModel.OrderedNodes = null;
TableLayout table = new TableLayout(diagramModel, diagramView);
table.RefreshLayout(300, 400);
```

VB.NET

```
' Set null value to the OrderedNodes property.
diagramModel.OrderedNodes = Nothing
Dim table As New TableLayout(diagramModel, diagramView)
table.RefreshLayout(300, 400)
```

PageMargin

The PageMargin property is used to maintain the distance between DiagramPage and DiagramView element.

Properties

Property	Description	Type	Data Type
PageMargin	Gets or the diagram Page Margin.	Dependency Property	Thickness

Adding PageMargin to an Application

PageMargin can be set to diagram page by using DiagramView's PageMargin property. The default value is set to 0. The property is in DiagramView and the value can be set in following ways:

- Through XAML
- Through Code behind

HTML

```
<syncfusion:DiagramControl Name="diagramControl"
IsSymbolPaletteEnabled="True">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel"/>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView" Margin="10,20,10,20"/>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

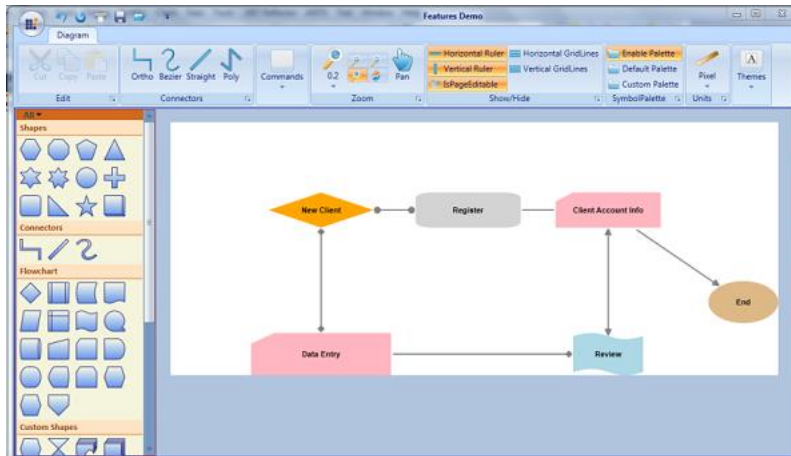
C#

```
DiagramView diagramView = new DiagramView();
diagramView.PageMargin = new Thickness(10, 20, 10, 20);
```

VB.NET

```
Dim diagramView As New DiagramView()
```

```
diagramView.PageMargin = New Thickness(10, 20, 10, 20)
```



Features Demo

Virtualization for DiagramControl

Virtualization

Virtualization is the process of loading the diagram page elements that are available in the visible area of the diagram control, i.e page elements that lie within the viewport of the ScrollViewer will be in loaded state and the rest will not be loaded until they come into view.

This feature gives optimizes performance while loading and dragging items to diagram control when many Nodes and LineConnectors are added in the diagram page.

Use Case Scenarios

The loading time and the UI response will be proportional to the number of elements used in a page. When you want to display a page with large number of Nodes and LineConnectors, such as floor plan application, the processing speed will be slow in user interaction. If virtualization is enabled, application will load only elements that lie in the visible area. This leads to fast loading and fast user interactivity.

Tables for Properties, Methods, and Events

Properties

Property	Description	Type	Data Type	Reference links
EnableVirtualization	Gets or sets a value indicating whether the diagram page can be virtualized. The default value is set to false.	Dependency Property	Boolean	No
Enable Caching	Gets or sets a value indicating whether the loaded object in diagram page can be virtualized. The default value is true.	Dependency Property	Boolean	

Adding Virtualization feature to an Application

EnableVirtualization Property:

To enable virtualization set the EnableVirtualization property to true. Page elements within the viewport alone will be loaded. The default value is false.

The property is in DiagramView and can be set in the following methods.

- Through XAML
- Through Code behind

The following code illustrates how to set EnableVirtualization property through XAML.

HTML

```
<!--DiagramControl-->
<syncfusion:DiagramControl Name="diagramControl" >
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel" >
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView EnableVirtualization="True" Name="diagramView"
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

The following code illustrates how to set EnableVirtualization property through Code behind.

C#

```
DiagramView diagramView = new DiagramView();
diagramView.EnableVirtualization = true;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.EnableVirtualization = True
### EnableCaching Property:
EnableCaching decides whether the element should be in loaded state or
unloaded state, when the element is outside the viewport area. To set the
element in unloaded state set the EnableCaching to false. To set it in
loaded state set the EnableCaching to true.
```

C#

```
DiagramView diagramView = new DiagramView();
diagramView.EnableCaching = true;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.EnableCaching = True
```

Node/LineConnector AllowVirtualization Property:

AllowVirtualization property is used to enable/disable the Node/LineConnector virtualization. When AllowVirtualization is set to false for an element that lies outside the viewport It will be in loaded state when Virtualization is enabled. The default value is true.

The AllowVirtualization property can be set as given in the following code.

C#

```
//Node Virtualization
Node NodeObject = new Node();
NodeObject.AllowVirtualization = true;
//LineConnector Virtualization
LineConnector LineConnectorObject = new LineConnector();
LineConnectorObject.AllowVirtualization = true;
```

VB.NET

```
'Node Virtualization
Dim NodeObject As New Node()
NodeObject.AllowVirtualization = True
'LineConnector Virtualization
Dim LineConnectorObject As New LineConnector()
LineConnectorObject.AllowVirtualization = True
```

Limitations:

Due to virtualization behavior there are some limitations in the diagram control. They are:

1. As Gridlines and Rulers are not visualized, when node or line connector is placed at a distance for example 2,000,000 pixels away, rendering will take place from zero to end. This leads to performance issues in rendering Gridlines or Rulers.
2. Save and load is not supported for Nodes and LineConnectors that are in unloaded state.
3. When diagram is virtualized, many nodes will be in unloaded state and their Width and Height will not be set. As the automatic layout depends on the size of the node, predefined width and height for the node is required for updating the layout.

Measurement Units

As different fields require different units of measure, several measurement units are provided such that you can choose the unit that is most comfortable and suitable to use. All basic properties can be defined in the specified measurement unit. It is also possible to dynamically change the units at run-time. The rulers are updated accordingly to represent the coordinates in the currently selected unit.

Property	Description	Type of the Property	Value it Accepts	Any other Dependencies

				/ Sub-Properties Associated
Measurement Units	Gets or sets the measurement unit property.	DependencyProperty	MeasureUnits.PixelMeasureUnits.PointMeasureUnits.DocumentMeasureUnits.DisplayMeasureUnits.SixteenthInchMeasureUnits.EighthInchMeasureUnits.QuarterInchMeasureUnits.HalfInchMeasureUnits.InchMeasureUnits.FootMeasureUnits.YardMeasureUnits.MileMeasureUnits.MillimeterMeasureUnits.CentimeterMeasureUnits.MeterMeasureUnits.Kilometer	No

The measurement units property can be specified in the following way.

HTML

```
<sfdiagram:DiagramControl IsSymbolPaletteEnabled="True" >
  <sfdiagram:DiagramControl.Model>
    <sfdiagram:DiagramModel x:Name="diagramModel" >
      </sfdiagram:DiagramModel>
    </sfdiagram:DiagramControl.Model>
    <sfdiagram:DiagramControl.View >
      <sfdiagram:DiagramView ShowHorizontalGridLine="True"
        ShowVerticalGridLine="True">
        <syncfusion:DiagramView.Page>
          <syncfusion:DiagramPage x:Name="diagramPage" MeasurementUnits="Inch"/>
        </syncfusion:DiagramView.Page>
      </sfdiagram:DiagramView>
    </sfdiagram:DiagramControl.View>
  </sfdiagram:DiagramControl>
```

C#

```
DiagramControl dc = new DiagramControl();  
dc.IsSymbolPaletteEnabled = true;  
DiagramView view = new DiagramView();  
(view.Page as DiagramPage).MeasurementUnits = MeasureUnits.Inch;
```

VB.NET

```
Dim dc As New DiagramControl()  
dc.IsSymbolPaletteEnabled = True  
Dim view As New DiagramView()  
TryCast(view.Page, DiagramPage).MeasurementUnits = MeasureUnits.Inch
```

Once the Measurement unit is specified, all the values must be specified with respect to that unit. For instance, If the unit is set to Inch then the node's properties can be set as follows.

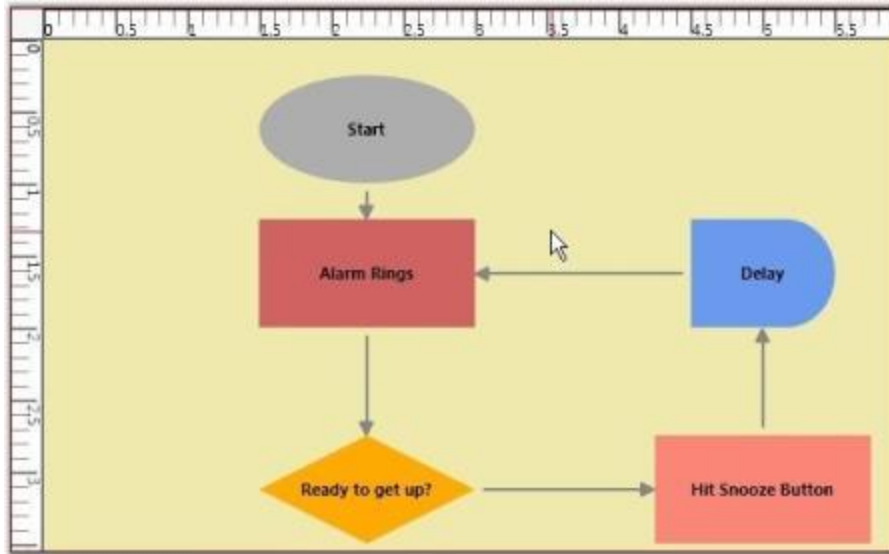
C#

```
Node n1 = new Node(Guid.NewGuid(), "Node1");  
n1.IsLabelEditable = true;  
n1.Label = "Alarm Rings";  
n1.OffsetX = 1.5;  
n1.OffsetY = 1.25;  
n1.Width = 1.5;  
n1.Height = 0.75;  
n1.LabelVerticalAlignment = VerticalAlignment.Center;  
diagramModel.Nodes.Add(n1);
```

VB.NET

```
Dim n1 As New Node(Guid.NewGuid(), "Node1")  
n1.IsLabelEditable = True  
n1.Label = "Alarm Rings"  
n1.OffsetX = 1.5  
n1.OffsetY = 1.25  
n1.Width = 1.5  
n1.Height = 0.75  
n1.LabelVerticalAlignment = VerticalAlignment.Center  
diagramModel.Nodes.Add(n1)
```

You can also dynamically change the units at runtime. The ruler values get changed according to the measurement unit selected. The rulers then indicate the position of the graphical objects with respect to the selected measurement unit.



Units changed to Inches

DateTime Unit

This feature enables the ruler to display the position of diagram contents (nodes, line connectors) in a date-time format. This feature also allows users to access nodes, line connectors, and double values as *DateTime* types.

Properties

Property	Description	Type	Data Type
DateTimeSettings	Used to get or set the value for <i>DateTime</i> unit values for the ruler.	Dependency	DateTimeSettings

Adding DateTime Unit Support for Ruler to an Application

DateTimeSettings

DateTime unit support enables the ruler to display the position of diagram contents (nodes, line connectors) in the *DateTime* format.

The *DateTime* settings have two important properties to display the date-time in the ruler.

1. *TimeSpan*—Gets or sets the *TimeSpan* for a unit. The default value is a new *TimeSpan* (1, 0, 0, 0).
2. *PixelUnit*—Gets or sets the number of pixels to be considered for *TimeSpan* specified. The default value is 100d.

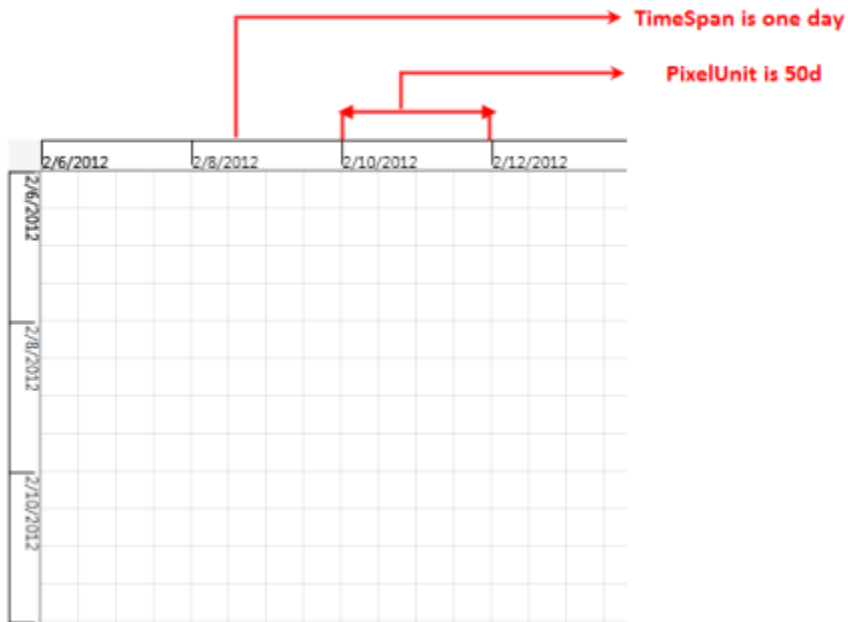
C#

```

DiagramView diagramView = new DiagramView();
//Setting DateTimeSettings property value.
diagramView.DateTimeSettings = new DateTimeSettings(new TimeSpan(1, 0, 0, 0, 0), 50);
  
```

VB.NET

```
Dim diagramView As New DiagramView()
'Setting DateTimeSettings property value.
diagramView.DateTimeSettings=New DateTimeSettings(New
TimeSpan(1, 0, 0, 0, 0), 50)
```



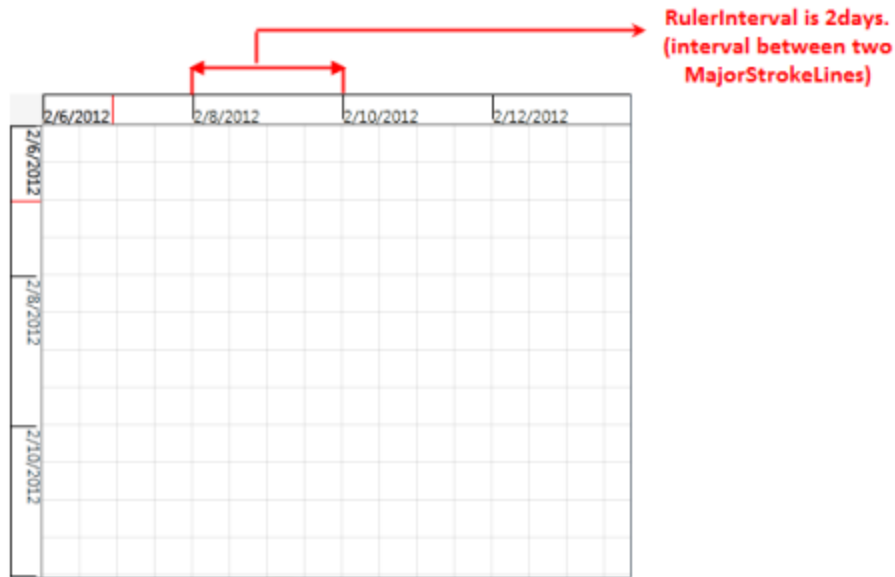
Ruler Representing 50 Pixels as One Day

Ruler Interval

RulerInterval represents the interval (distance) between two major stroke lines of the ruler.

C#

```
DiagramView diagramView = new DiagramView();
diagramView.DateTimeSettings.RulerInterval = new TimeSpan(2, 0, 0, 0, 0);
```



RulerInterval Set as Two Days

Enable DateTimeSettings

By default, the `DateTimeSettings` property is set to false. A node's `DateTime`-related property will work only after enabling this property. Also, `DateTime` values will be displayed only after enabling this feature.

C#

```
DiagramView diagramView = new DiagramView();  
diagramView.DateTimeSettings.IsEnabled = true;
```

DateTime Format

The `CustomFormatString` property is used to specify the format of the date-time to be displayed in the ruler. The default value is "{0:d}".

The available standard date and time format strings are explained in the following link:

<https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-date-and-time-format-strings>

C#

```
DiagramView diagramView = new DiagramView();  
diagramView.DateTimeSettings.CustomFormatString = "{0:HH:mm}";
```

OriginofRuler

The origin (0,0) of the diagram page can be represented as a `DateTime` value by using the `OriginDateX` and `OriginDateY` properties in the ruler.

- `OriginDateX` represents the origin of the ruler on the x-axis. The default value is a new `DateTime` (2000, 1, 1).
- `OriginDateY` represents the origin of the ruler in the y-axis. The default value is a new `DateTime` (2000, 1, 1).

C#

```
DiagramView diagramView = new DiagramView();  
diagramView.DateTimeSettings.CustomFormatString = "{0:HH:mm}";  
diagramView.DateTimeSettings.OriginDateX = DateTime.Now;  
diagramView.DateTimeSettings.OriginDateY = DateTime.Now;
```

ShowDateTime

The ShowDateTime property of the ruler is used to display the DateTime values (based on PixelUnit, TimeSpan, RulerInterval, etc.) in the ruler. The default value is false. Refer to the following code snippet:

C#

```
DiagramView diagramView = new DiagramView();  
diagramView.HorizontalRuler.ShowDateTime = true;  
diagramView.VerticalRuler.ShowDateTime = true;
```

Node

The following properties are used to specify the position and size of the node when the DateTime support in the ruler is enabled.

1. Position
 - StartDateX is similar to the OffsetX property of the node.
 - StartDateY is similar to the OffsetY property of the node.
2. Size
 - DurationX represents the width of the node.
 - DurationY represents the height of the node.

Position

Positioning of the node in the diagram page can be accessed using DateTime values instead of double values (OffsetX, OffsetY) though the StartDateX and StartDateY properties. It accepts the DateTime value.

C#

```
Node node = new Node();  
node.StartDateX = DateTime.Now;  
node.StartDateY = DateTime.Now + new TimeSpan(3, 0, 0, 0);
```

Size

The size of the node can be accessed using DateTime values instead of double values (Width, Height) though the DurationX and DurationY properties. It accepts the DateTime value.

C#

```
Node node = new Node();  
node.DurationX = new TimeSpan(4, 5, 45, 24, 4799);  
node.DurationY = new TimeSpan(3, 6, 43, 12, 0);
```

Note: Use these node properties after the nodes are loaded. Also, the unit conversion methods can be used (we have provided methods for conversion as explained below) for converting the double values into DateTime and vice versa.

Methods for Converting

We have provided some methods to convert the DateTime and TimeSpan as double and vice versa. Please refer to the table and code snippet.

Method	Description	Return type	Arguments
ToPixel	Convert DateTime to double value.	double	DateTime
ToPixel	Convert TimeSpan to double value.	double	TimeSpan
ToDateTime	Convert double to DateTime value.	DateTime	double
ToTimeSpan	Convert double to TimeSpan value.	TimeSpan	double

Node

This code example demonstrates how to create the position and size of the node using the methods listed above.

C#

```
Node n = new Node() {Shape = Shapes.Star };
n.OffsetX = diagramView.DateTimeSettings.ToPixel(new DateTime(2012, 2, 10));
n.OffsetY = diagramView.DateTimeSettings.ToPixel(new DateTime(2012, 2, 10));
n.Width = diagramView.DateTimeSettings.ToPixel(new DateTime(2012, 2, 11));
n.Height = diagramView.DateTimeSettings.ToPixel(new DateTime(2012, 2, 11));
diagramModel.Nodes.Add(n);
```

LineConnector

This code example demonstrates how to create the start point position and end point position of the line connector using the methods listed above.

C#

```
LineConnector line = new LineConnector();
//Converting the DateTime into double
double x=diagramView.DateTimeSettings.ToPixel(new DateTime(2012, 2, 10));
double y=diagramView.DateTimeSettings.ToPixel(new DateTime(2012, 2, 10));
line.StartPointPosition = new Point(x,y);
//Converting the DateTime into double
double x1 = diagramView.DateTimeSettings.ToPixel(new DateTime(2012, 2, 10));
double y1=diagramView.DateTimeSettings.ToPixel(new DateTime(2012, 2, 15));
line.EndPointPosition = new Point(x1,y1);
line.ConnectorType = ConnectorType.Straight;
diagramModel.Connections.Add(line);
```

Grid Lines

The drawing area of a DiagramControl can be rendered with horizontal and vertical grid lines to allow for proper positioning of the nodes.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
ShowVerticalGridLine	Gets or sets a value indicating whether vertical grid lines are displayed or not. Default value is True.	Dependency property	Boolean(True/False)	No
ShowHorizontalGridLine	Gets or sets a value indicating whether horizontal grid lines are displayed or not. Default value is True.	Dependency property	Boolean(True/False)	No

Show or Hide Grid Lines

The horizontal and vertical grid lines can be enabled or disabled using the ShowHorizontalGridLine and ShowVerticalGridLine properties.

The following code can be used to set these properties.

XML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:sfdiagram="clr-namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
  <Grid Name="diagramgrid">
    <sfdiagram:DiagramControl IsSymbolPaletteEnabled="True" >
      <sfdiagram:DiagramControl.Model>
        <sfdiagram:DiagramModel x:Name="diagramModel" >
        </sfdiagram:DiagramModel>
      </sfdiagram:DiagramControl.Model>
      <sfdiagram:DiagramControl.View >
        <sfdiagram:DiagramView ShowHorizontalGridLine="True"
ShowVerticalGridLine="True">
        </sfdiagram:DiagramView>
      </sfdiagram:DiagramControl.View>
    </sfdiagram:DiagramControl>
  </Grid>
</Window>
```

C#

```
DiagramControl dc = new DiagramControl();
dc.IsSymbolPaletteEnabled = true;
```

```

DiagramView view = new DiagramView();
view.ShowHorizontalGridLine = true;
view.ShowVerticalGridLine = true;
view.Bounds = new System.Drawing.Thickness(0, 0, 1000, 1000);
dc.View = view;
diagramgrid.Children.Add(dc);

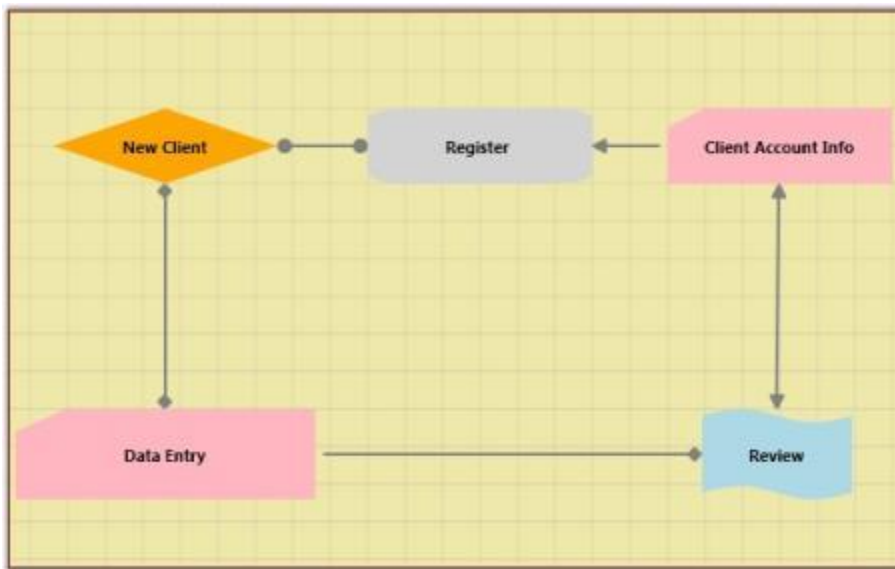
```

VB.NET

```

Dim dc As New DiagramControl()
dc.IsSymbolPaletteEnabled = True
Dim view As New DiagramView()
view.ShowHorizontalGridLine = True
view.ShowVerticalGridLine = True
view.Bounds = New System.Drawing.Thickness(0, 0, 1000, 1000)
dc.View = view
diagramgrid.Children.Add(dc)

```

**GridLines***GridLine Offset*

The vertical and horizontal spacing between grid lines can be specified using the `GridVerticalOffset` and `GridHorizontalOffset` properties respectively.

The default value is 25d for both properties.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
<code>GridHorizontalOffset</code>	Gets or sets the <code>HorizontalOffset</code> value of the grid.	CLR Property	double	No

GridVerticalOffset	Gets or sets the VerticalOffset value of the grid.	CLR Property	double	No
--------------------	--	--------------	--------	----

The following code can be used to set these properties.

XML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:sfdiagram="clr-
namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<sfdiagram:DiagramControl IsSymbolPaletteEnabled="True" >
<sfdiagram:DiagramControl.Model>
<sfdiagram:DiagramModel x:Name="diagramModel" >
</sfdiagram:DiagramModel>
</sfdiagram:DiagramControl.Model>
<sfdiagram:DiagramControl.View >
<sfdiagram:DiagramView ShowHorizontalGridLine="True"
ShowVerticalGridLine="True">
<syncfusion:DiagramView.Page>
<syncfusion:DiagramPage x:Name="diagramPage" MeasurementUnits="Pixels"
GridHorizontalOffset="100"
GridVerticalOffset="100"/>
</syncfusion:DiagramView.Page>
</sfdiagram:DiagramView>
</sfdiagram:DiagramControl.View>
</sfdiagram:DiagramControl>
</Grid>
</Window>
```

C#

```
DiagramControl dc = new DiagramControl();
dc.IsSymbolPaletteEnabled = true;
DiagramView view = new DiagramView();
view.ShowHorizontalGridLine = true;
view.ShowVerticalGridLine = true;
(view.Page as DiagramPage).GridHorizontalOffset = 100;
(view.Page as DiagramPage).GridVerticalOffset = 100;
dc.View = view;
diagramgrid.Children.Add(dc);
```

VB.NET

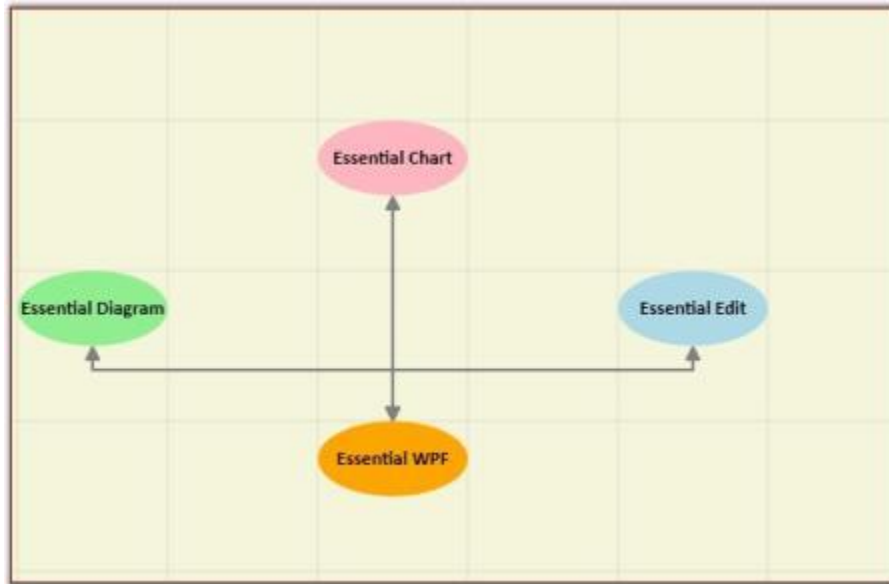
```
Dim dc As New DiagramControl()
dc.IsSymbolPaletteEnabled = True
Dim view As New DiagramView()
view.ShowHorizontalGridLine = True
view.ShowVerticalGridLine = True
```



```

TryCast(view.Page, DiagramPage).GridHorizontalOffset = 100
TryCast(view.Page, DiagramPage).GridVerticalOffset = 100
dc.View = view
diagramgrid.Children.Add(dc)

```



GridOffset

Customizing the GridLineStyle

The following code illustrates how to customize GridLineStyle:

C#

```

Pen pen = new Pen(Brushes.Gray, 1)
{
    DashCap = PenLineCap.Triangle,
    Thickness=3,
    DashStyle = new DashStyle(new double[] { 2, 8 }, 1),
    Brush=new SolidColorBrush(Colors.MidnightBlue)
};
Pen pen1 = new Pen(Brushes.Gray, 1)
{
    DashCap = PenLineCap.Round,
    Thickness=2,
    DashStyle = new DashStyle(new double[] { 2, 8 },1),
    Brush=new SolidColorBrush(Colors.Green)
};
diagramView.HorizontalGridLineStyle = pen1;
diagramView.VerticalGridLineStyle = pen;

```

VB.NET

```

Dim pen As New Pen(Brushes.Gray, 1) With {.DashCap = PenLineCap.Triangle,
.DashStyle = New DashStyle(New Double() { 2, 8 }, 1), .Brush = New
SolidColorBrush(Colors.MidnightBlue)}

```

```
Dim pen1 As New Pen(Brushes.Gray, 1) With {.DashCap = PenLineCap.Round,  
.DashStyle = New DashStyle(New Double() { 2, 8 }, 1), .Brush = New  
SolidColorBrush(Colors.Green) }  
diagramView.HorizontalGridLineStyle = pen1  
diagramView.VerticalGridLineStyle = pen
```

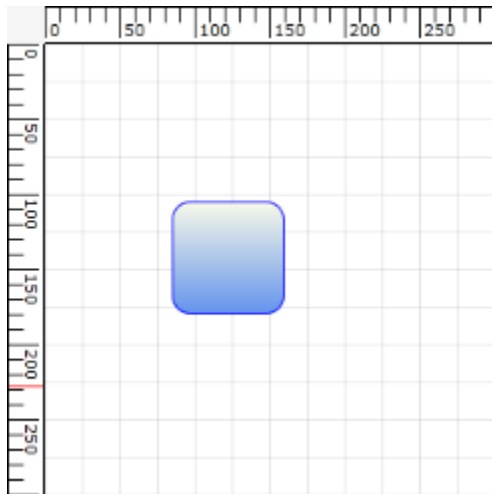
Snapping

Snap to Grid

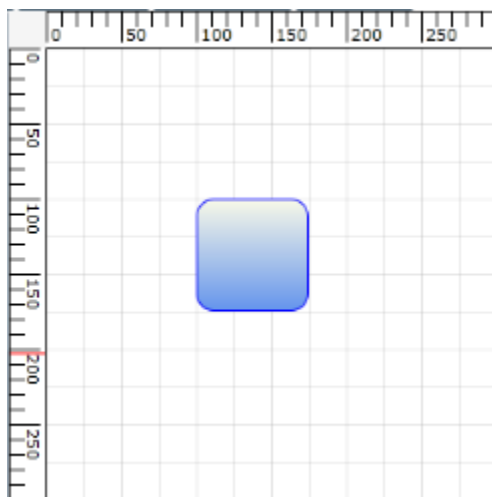
The Snap to Grid feature enables dragging nodes and connectors in multiples of offset values, which is specified by using DiagramView's `SnapOffsetX` and `SnapOffsetY` properties. For example, if a node is dragged when `SnapOffsetX` is set to 25, then the nodes `OffsetX` value will change in multiples of 25.

Use Case Scenarios

Users can snap objects with respect to grid lines in the Design environment by using Snap to Grid instead of smooth dragging.



Node Before Snapping



Node After Snapping

Enabling Snap to Grid

The Snap to Grid feature for nodes and connectors can be enabled by setting DiagramView's `SnapToHorizontalGrid` and `SnapToVerticalGrid` properties to "True", as shown in the following code snippets.

In the following code example, `diagramView` is an instance of `DiagramView`.

HTML

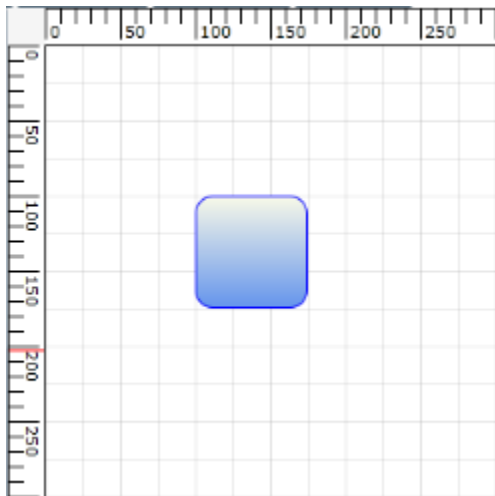
```
<syncfusion:DiagramView x:Name="diagramView" SnapToHorizontalGrid="True"
SnapToVerticalGrid="True" >
</syncfusion:DiagramView>
```

C#

```
// Enable snap to vertical grid.
diagramView.SnapToVerticalGrid = True;
// Enable snap to horizontal grid.
diagramView.SnapToHorizontalGrid = True;
```

VB.NET

```
'Enable snap to vertical grid.
diagramView.SnapToVerticalGrid = True
'Enable snap to horizontal grid.
diagramView.SnapToHorizontalGrid = True
```



Snap to Grid Enabled

Customizing Snap to Grid Offset Values

By default, the `SnapOffsetX` and `SnapOffsetY` values are set to 25 pixels. However, these values can be changed so that objects will snap to the horizontal grid by using `SnapOffsetX` and snap to the vertical grid by using `SnapOffsetY`, as shown in the following code example.

In the following code example, `diagramView` is an instance of `DiagramView`.

HTML

```
<syncfusion:DiagramView x:Name="diagramView" SnapOffsetX ="50" SnapOffsetY
="50">
</syncfusion:DiagramView>
```

C#

```
diagramView.SnapOffsetX = 50;
diagramView.SnapOffsetY = 50;
```

VB.NET

```
diagramView.SnapOffsetX = 50
diagramView.SnapOffsetY = 50
```

Note: SnapToGrid will snap objects based on the offset values specified in DiagramView's SnapOffsetX and SnapOffsetY values and it works independently from grid lines. However, to snap objects along with the grid lines, specify the same offset values for grid lines and snap offset.

Also, snapping of objects will occur only when the objects are dragged during runtime. Even after snapping is enabled, users can specify their own offset values in code behind.

The properties of the Snap to Grid feature are described in the following tabulation:

Property	Description	Type	Data Type	Reference links
SnapOffsetX	Snaps to the horizontal offset value.	Dependency property	double	Not applicable
SnapOffsetY	Snaps to the vertical offset value.	Dependency property	double	Not applicable
SnapToHorizontalGrid	Enables or disables snap to horizontal grid.	Dependency property	bool, true/false	Not applicable
SnapToVerticalGrid	Enables or disables snap to vertical grid.	Dependency property	bool, true/false	Not applicable

[Sample Link](#)

To view a sample:

Open the Diagram Sample Browser from the dashboard. (Refer to the Samples and Location chapter.)

Navigate to Editable Diagram > SnapToGrid Demo.

[Snap-To-Node](#)

This feature enables you to properly align a node with neighboring nodes. This is useful when multiples nodes need to be aligned horizontally or vertically to each other. While dragging the nodes, drawing guidelines will be shown to indicate how the nodes are aligned.

Properties

Property	Description	Type	Data Type
Left	This property is used to enable/disable the left alignment support.	Dependency	bool
Top	This property is used to enable/disable the top alignment support.	Dependency	bool
Right	This property is used to enable/disable the right (width oriented) alignment support.	Dependency	bool
Bottom	This property is used to enable/disable the bottom (height oriented) alignment support.	Dependency	bool
CenterX	This property is used to enable/disable the center alignment support based on X-coordinate alignment.	Dependency	bool
CenterY	This property is used to enable/disable the center alignment support based on Y-coordinate alignment.	Dependency	bool
SnapLinePen	This property is used to customize drawing guideline properties such as brush, thickness, etc.	Dependency	Pen
SnapAdjustmentDistance	This property is used to adjust the snapping distance.	Dependency	double

Note: By default, all the properties of the SnapSettings class are set to true.

Enabling Snap-To-Node Feature

To enable the snap-to-node feature, create an instance for the SnapSettings class of DiagramView and set the EnableSnapNode property of the SnapSettings class to true. By default, the value of the EnableSnapNode property is set to NULL.

C#

```
DiagramView diagramView = new DiagramView();
diagramView.SnapSettings = new SnapSettings();
diagramView.SnapSettings.EnableSnapNode = true;
```

SnapSettings

By default, two nodes can be snapped in six positions. To disable snapping on a particular direction, you need to set the corresponding property to false.

Customizing the Snapping Lines

Snapping to object can be done in six ways. They are:

1. Left
2. Top
3. Right

4. Bottom
5. CenterX
6. CenterY

Customizing the Drawing Guidelines

The appearance of the drawing guidelines can be customized using the `SnapLinePen` property of the `SnapSettings` class.

C#

```
DiagramView diagramView = new DiagramView();  
//Creating Instance for SnapSettings class.  
SnapSettings snapSettings = new SnapSettings();  
//Customize the SnapAutoAdjustment distance between the elements.  
snapSettings.SnapAdjustmentDistance = 25;  
//Customize the SnapLine Pen Color, Thickness.etc....  
snapSettings.SnapLinePen = new Pen() { Brush = Brushes.Blue, Thickness = 0.7  
};  
//Assinging the SnapSettings instance to SnapSettings property of the  
DiagramView.  
diagramView.SnapSettings = snapSettings;
```

Enabling/Disabling Drawing Guidelines

By default, all the properties of the `SnapSettings` class are set to true. To disable the drawing guidelines, set these properties to false as explained in the following code example.

C#

```
DiagramView diagramView = new DiagramView();  
//Creating Instance for SnapSettings class.  
SnapSettings snapSettings=new SnapSettings();  
//Customize the SnapAutoAdjustment distance between the elements.  
snapSettings.SnapAdjustmentDistance = 25;  
//Disable Left Alignment Support line.  
snapSettings.Left = false;  
//Disable Right Alignment Support line.  
snapSettings.Right = false;  
//Disable CenterX Alignment Support line.  
snapSettings.CenterX = false;  
//Assinging the SanpSettings instance to SnapSettings property of the  
DiagramView.  
diagramView.SnapSettings = snapSettings;
```

Snapping Port Support

This support is used to align a connection port with other ports horizontally or vertically.

Enabling the Snapping Port Support

To enable this feature, create an instance for the `SnapSettings` class of `DiagramView` and set the `EnableSnapPort` property of the `SnapSettings` class to true.

C#

```
DiagramView diagramView = new DiagramView();  
diagramView.SnapSettings = new SnapSettings();
```

```
diagramView.SnapSettings.EnableSnapPort = true;
```

Disabling the Snapping Port Feature

To disable this feature, set the EnableSnapPort property of the SnapSettings class to false. The following code example illustrates this.

C#

```
DiagramView diagramView = new DiagramView();
diagramView.SnapSettings.EnableSnapPort = false;
```

Zoom Commands

The diagram page can be zoomed in and out. Zooming can be achieved in the following two ways.

- Using the zoom commands.
- Using the mouse wheel.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
IsZoomEnabled	Gets or sets a value indicating whether zoom is enabled or not. Default value is True.	Dependency property	Boolean(True/False)	No
ZoomFactor	Gets or sets a factor for the zoom. Default value is 0.2.	Dependency property	Double	No

Steps to zooming using the zoom commands

Zooming in

The following code can be added to a Button's click event to facilitate zooming in.

C#

```
DiagramView diagramView = new DiagramView();
ZoomCommands.ZoomIn.Execute(diagramView.Page, diagramView);
```

VB.NET

```
Dim diagramView As New DiagramView()
ZoomCommands.ZoomIn.Execute(diagramView.Page, diagramView)
The diagram page elements will be zoomed in each time the button is clicked.
```

Zoom out

The following code can be added to a button's click event to facilitate zooming out.

C#

```
DiagramView diagramView = new DiagramView();  
ZoomCommands.ZoomOut.Execute(diagramView.Page, diagramView);
```

VB.NET

```
Dim diagramView As New DiagramView()  
ZoomCommands.ZoomOut.Execute(diagramView.Page, diagramView)
```

The diagram page elements will be zoomed out each time the button is clicked.

Steps to zooming using the mouse wheel

1. Ensure the IsZoomEnabled property is set to True. By default it is set to true.

C#

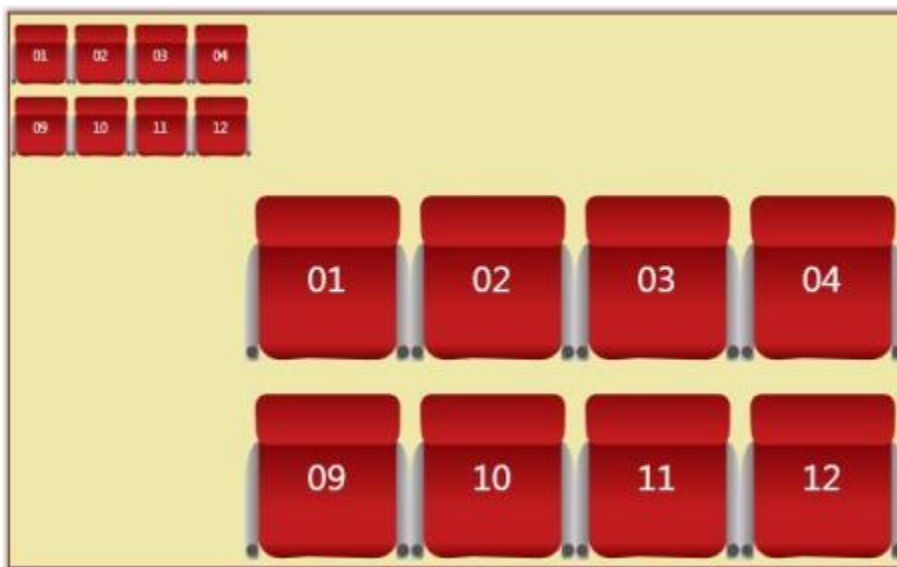
```
DiagramView diagramView = new DiagramView();  
diagramView.IsZoomEnabled = true;
```

VB.NET

```
Dim diagramView As New DiagramView()  
diagramView.IsZoomEnabled = True
```

2. Now while the CTRL key is pressed, roll the mouse wheel up to zoom in or down to zoom out.

Note: All other operations can be performed on page elements while IsZoomEnabled is set to True.



Zooming the Diagram Control

Zoom Factor

Essential Diagram WPF allows you to set the factor by which you can zoom in or out. This factor can be specified using the ZoomFactor property. The default value is 0.2.

The following code can be used to set the ZoomFactor property.

HTML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:sfdiagram="clr-
namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
<Grid Name="diagramgrid">
<sfdiagram:DiagramControl IsSymbolPaletteEnabled="True">
<sfdiagram:DiagramControl.View>
<sfdiagram:DiagramView Name="diagramView" ZoomFactor="0.5">
</sfdiagram:DiagramView>
</sfdiagram:DiagramControl.View>
</sfdiagram:DiagramControl>
</Grid>
</Window>
```

C#

```
DiagramView diagramView = new DiagramView();
diagramView.ZoomFactor = 0.5;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.ZoomFactor = 0.5
```

Nudge Commands

Nudge commands allows you to move the selected objects on the page by 1 pixel. This can be done in two ways:

Using Nudge Commands

NudgeUp

Moves the selected object to the top by 1 pixel.

C#

```
DiagramCommandManager.NudgeUp.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.NudgeUp.Execute(diagramView.Page, diagramView)
```

NudgeDown

Moves the selected object to the bottom by 1 pixel.

C#

```
DiagramCommandManager.NudgeDown.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.NudgeDown.Execute(diagramView.Page, diagramView)
```

NudgeLeft

Moves the selected object to the left by 1 pixel.

C#

```
DiagramCommandManager.NudgeLeft.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.NudgeLeft.Execute(diagramView.Page, diagramView)
```

NudgeRight

Moves the selected object to the right by 1 pixel.

C#

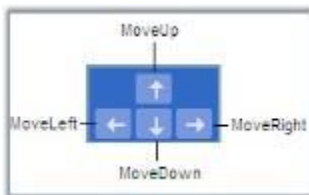
```
DiagramCommandManager.NudgeRight.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.NudgeRight.Execute(diagramView.Page, diagramView)
```

Nudge by using Arrow Keys

The corresponding arrow keys can be used to move the selected objects to top, bottom, left or right.

**Nudge by using Arrow Keys**

Nudge commands are particularly useful for accurate placement of objects on the page as it allows you to move by 1 pixel each time.

Clipboard Commands

Clipboard commands are used to perform cut copy and paste operations.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated

IsCutEnabled	Gets or sets a value indicating whether cut command is enabled. Default value is True.	Dependency property	Boolean(True/False)	No
IsCopyEnabled	Gets or sets a value indicating whether copy command is enabled. Default value is True.	Dependency property	Boolean(True/False)	No
IsPasteEnabled	Gets or sets a value indicating whether paste command is enabled. Default value is True.	Dependency property	Boolean(True/False)	No

Clipboard commands allow you to cut or copy the selected objects in the page to the clipboard and paste the valid clipboard content into the page. This can be done in two ways:

Using Clipboard Commands

Cut Command

Cut the selected objects from the page into the Clipboard.

HTML

```
<Button Command="Cut" CommandTarget="{Binding ElementName=diagramView}"
CommandParameter="{Binding ElementName=diagramPage}">
Cut
</Button>
```

C#

```
ApplicationCommands.Cut.Execute(diagramView.Page, diagramView);
```

VB.NET

```
ApplicationCommands.Cut.Execute(diagramView.Page, diagramView)</td></tr>
```

Copy Command

Copies the selected objects from the page into the Clipboard.

HTML

```
<Button Command="Copy" CommandTarget="{Binding ElementName=diagramView}"
CommandParameter="{Binding ElementName=diagramPage}">
Copy</Button>
```

C#

```
ApplicationCommands.Copy.Execute(diagramView.Page, diagramView);
```

VB.NET

```
ApplicationCommands.Copy.Execute(diagramView.Page, diagramView)
```

Paste Command

Paste the contents of the valid clipboard into the page.

HTML

```
<Button Command="Paste" CommandTarget="{Binding ElementName=diagramView}"
CommandParameter="{Binding ElementName=diagramPage}">
Paste
</Button>
```

HTML

```
ApplicationCommands.Paste.Execute(diagramView.Page, diagramView);
```

C#

```
ApplicationCommands.Paste.Execute(diagramView.Page, diagramView)
```

Using Shortcut Keys

- Cut – Ctrl + X
- Copy – Ctrl + C
- Paste – Ctrl + V

Disable Clipboard Commands

Each clipboard command can be disabled as shown in the following code example.

HTML

```
<Window x:Class="RadialTreeLayout_2008.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Disable Cut Copy Paste Demo" WindowState="Maximized"
WindowStartupLocation="CenterScreen" Name="mainwindow"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf" Icon="App.ico"
xmlns:local="clr-namespace:DisableCutCopyPaste_2008"
>
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel">
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<!-- Disable Cut, Copy and Past -->
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView IsCutEnabled="False" IsCopyEnabled="False"
IsPasteEnabled="False" Name="diagramView">
</syncfusion:DiagramView>
```

```
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
</Window>
```

C#

```
diagramView.IsCutEnabled = false;
diagramView.IsCopyEnabled = false;
diagramView.IsPasteEnabled = false;
```

VB.NET

```
diagramView.IsCutEnabled = False
diagramView.IsCopyEnabled = False
diagramView.IsPasteEnabled = False
```

ZOrder Commands

The ordering commands allows you to change the z-index value of the selected objects (nodes and connectors) on the page. The objects can be made to go back or front so that they get displayed over other objects in case two or more objects overlap.

The commands are listed as follows.

BringToFront

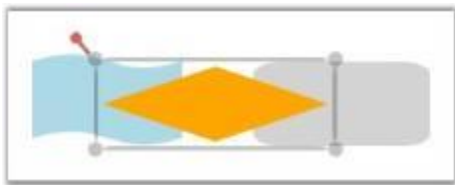
Moves the selected object over other objects by increasing the z-index to maximum value.

C#

```
DiagramCommandManager.BringToFront.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.BringToFront.Execute(diagramView.Page, diagramView)
```

**Bring To Front***SendToBack*

Moves the selected object behind all other objects by setting the z-index to 0.

C#

```
DiagramCommandManager.SendToBack.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.SendToBack.Execute(diagramView.Page, diagramView)
```



Send To Back

MoveForward

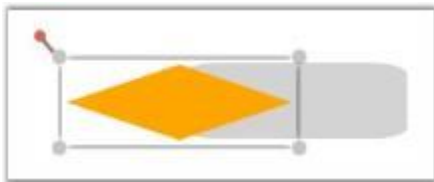
Increases the z-index value of the selected object by 1.

HTML

```
DiagramCommandManager.MoveForward.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.MoveForward.Execute(diagramView.Page, diagramView)
```



Move Forward

SendBackward

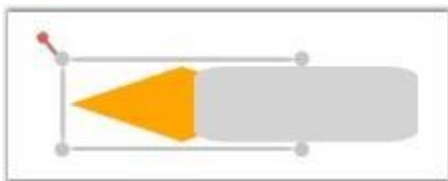
Decreases the z-index value of the selected object by 1.

C#

```
DiagramCommandManager.SendBackward.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.SendBackward.Execute(diagramView.Page, diagramView)
```



Send Backward

ZOrder Mode

Essential Diagram for WPF provides support for the ZOrder mode for diagram view elements. This allows the user to decide whether the ZOrdering of diagram view elements should be done by index or be visually based.

Properties

Property	Description	Type of property	Value it Accepts	Any other dependencies/ sub properties associated
ZOrderMode	Specifies the ZOrder mode for diagram view elements. The default value is Index.	Dependency property	ZOrderModes.VisualZOrderModes.Index	No

This property is in DiagramView and can be set in the following ways.

- Through XAML
- Through Code Behind

The following code illustrates how to set the ZorderMode as Index:

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections.-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel x:Name="diagramModel">
    </syncfusion:DiagramModel>
  </syncfusion:DiagramControl.Model>
  <!--View to display nodes and connections added through the model.-->
  <syncfusion:DiagramControl.View>
    <syncfusion:DiagramView Name="diagramView" ZOrderMode="Index">
    </syncfusion:DiagramView>
  </syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DiagramView diagramView = new DiagramView();
diagramview.ZOrderMode = ZOrderModes.Index;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramview.ZOrderMode = ZOrderModes.Index;
```

The following code illustrates how to set the ZOrderMode as Visual:

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections.-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel x:Name="diagramModel">
    </syncfusion:DiagramModel>
  </syncfusion:DiagramControl.Model>
  <!--View to display nodes and connections added through the model.-->
  <syncfusion:DiagramControl.View>
    <syncfusion:DiagramView Name="diagramView" ZOrderMode="Index">
    </syncfusion:DiagramView>
  </syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DiagramView diagramView = new DiagramView();
diagramview.ZOrderMode = ZOrderModes.Visual;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramview.ZOrderMode = ZOrderModes.Visual;
```

Alignment Commands

Alignment commands enable you to align selected objects (nodes and connectors) on a page with respect to a reference object. The first object in the selection is considered the reference object.

The following alignment commands are used to align objects.

Left Alignment

The AlignLeft command aligns all selected objects along the left corner of the reference object.

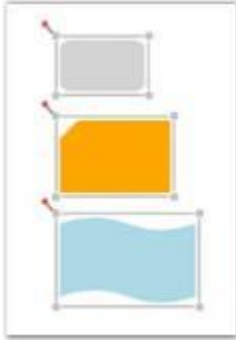
HTML

```
DiagramCommandManager.AlignLeft.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.AlignLeft.Execute(diagramView.Page, diagramView)
```

The following screenshot illustrates how the last two nodes are aligned to the left with respect to the first node.



AlignLeft command applied to Diagram Objects

Center Alignment (Horizontal Axis)

The AlignCenter command aligns all selected objects to the center. This command center-aligns selected objects with respect to the horizontal axis, i.e., by changing the x-coordinate of the object.

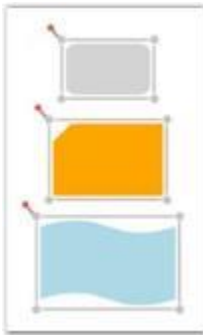
C#

```
DiagramCommandManager.AlignCenter.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.AlignCenter.Execute(diagramView.Page, diagramView)
```

The following screenshot illustrates how the last two nodes are aligned to the center with respect to the horizontal axis of the first node.



AlignCenter command applied to Diagram Objects

Right Alignment

The AlignRight command aligns all selected objects along the right corner of the reference object.

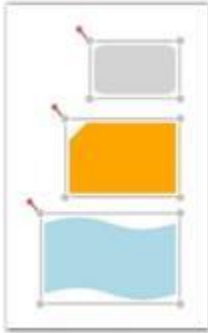
C#

```
DiagramCommandManager.AlignRight.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.AlignRight.Execute(diagramView.Page, diagramView)
```

The following screenshot illustrates how the last two nodes are aligned to the right with respect to the first node.



AlignRight command applied to Diagram Objects

Top Alignment

The AlignTop command aligns all selected objects along the top surface of the reference object.

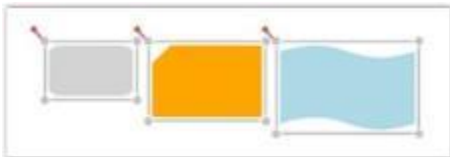
C#

```
DiagramCommandManager.AlignTop.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.AlignTop.Execute(diagramView.Page, diagramView)
```

The following screenshot illustrates how the last two nodes are aligned to the top with respect to the first node.



AlignTop command applied to Diagram Objects

Center Alignment (Vertical Axis)

The AlignMiddle command aligns all selected objects at the center. This command center-aligns selected objects with respect to the vertical axis, i.e., by changing the y-coordinate of the object.

C#

```
DiagramCommandManager.AlignMiddle.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.AlignMiddle.Execute(diagramView.Page, diagramView)
```

The following screenshot illustrates how the last two nodes are aligned to the center with respect to the vertical axis of the first node.



AlignMiddle command applied to Diagram Objects

Bottom Alignment

The AlignBottom command aligns all selected objects along the bottom surface of the reference object.

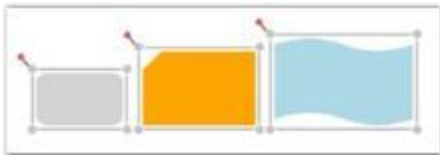
C#

```
DiagramCommandManager.AlignBottom.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.AlignBottom.Execute(diagramView.Page, diagramView)
```

The following screenshot illustrates how the last two nodes are aligned to the bottom with respect to the first node.



AlignBottom command applied to Diagram Objects

Note: The connector gets aligned only when the head node and the tail node of the connector is Null.

Alignment commands are useful for ordering the layout of the objects on a page and provide a professional appearance to the diagram.

Spacing Commands

Spacing commands enables you to place selected objects (nodes and connectors) on the page at equal intervals from each other. The objects are spaced within the bounds of the first and last objects in the selection object.

The following spacing commands are used to space objects.

Horizontal Spacing

The SpaceAcross command spaces selected objects with equal horizontal distance between them.

C#

```
DiagramCommandManager.SpaceAcross.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.SpaceAcross.Execute(diagramView.Page, diagramView)
```

The following screenshot illustrates horizontally spaced objects.



SpaceAcross command applied to Diagram Objects

Vertical Spacing

The SpaceDown command spaces selected objects with equal vertical distance between them.

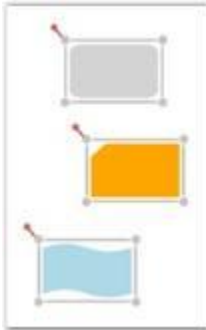
C#

```
DiagramCommandManager.SpaceDown.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.SpaceDown.Execute(diagramView.Page, diagramView)
```

The following screenshot illustrates vertically spaced objects.



SpaceDown command applied to Diagram Objects

Note: The connector gets spaced only when the head node and the tail node of the connector is Null.

Delete Command

Diagram WPF provides support to delete nodes and connectors by using the Delete command.

The following steps illustrate how to delete a node or connector.

1. Select the node or the connector to be deleted.
2. Press the DELETE key. The selected node or connector will be deleted.

Note: When a node is deleted, all the connectors connected to that node are also deleted. Deleting a connector leads to the deletion of that particular connector only.

The following code example illustrates how to invoke the Delete command.

C#

```
DiagramCommandManager.Delete.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.Delete.Execute(diagramView.Page, diagramView)
```

Sizing Commands

Sizing commands enable you to resize selected objects (nodes and connectors) on the page. The selected objects get resized with respect to the first object in the selection list.

The following sizing commands are used to resize objects.

Height Customization

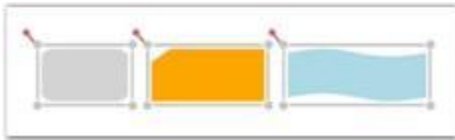
The SameHeight command resizes selected objects to the height of the first object in the selection list.

C#

```
DiagramCommandManager.SameHeight.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.SameHeight.Execute(diagramView.Page, diagramView)
```



SameHeight command applied to Diagram Objects

Note: The width of the selected object remains the same.

Width Customization

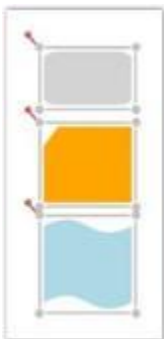
The SameWidth command resizes selected objects to the width of the first object in the selection list.

C#

```
DiagramCommandManager.SameWidth.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.SameWidth.Execute(diagramView.Page, diagramView)
```



SameWidth command applied to Diagram Objects

Note: The height of the selected object remains the same.

Height and Width Customization

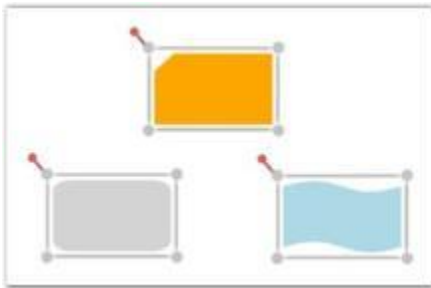
The SameSize command resizes selected objects to the height and width of the first object in the selection list.

C#

```
DiagramCommandManager.SameSize.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.SameSize.Execute(diagramView.Page, diagramView)
```



SameSize command applied to Diagram Objects

Note: The connector gets spaced only when the head node and the tail node of the connector is Null.

Undo and Redo Command

Undo command reverses the last action performed. For example: Some of the basic operations like translation, rotation, resizing, grouping, ungrouping, changing z-order, addition, deletion etc., which are performed on diagram objects (Nodes and Line Connectors) can be reversed. Redo command undoes the last Undo action. Alternatively these commands can be executed using the keyboard shortcuts; Ctrl +Z for Undo command and Ctrl+Y for Redo command.

Undo Command can be specified in the following way.

C#

```
DiagramCommandManager.Undo.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.Undo.Execute(diagramView.Page, diagramView)
```

Redo Command can be specified in the following way.

C#

```
DiagramCommandManager.Redo.Execute(diagramView.Page, diagramView);
```

VB.NET

```
DiagramCommandManager.Redo.Execute(diagramView.Page, diagramView)
```

Disable Undo and Redo

Disabling Undo and Redo is helpful when the Diagram control has large number of nodes and line connectors where insertion and deletion are very frequently used. This property can be disabled so that all the references are removed for the stack. This implies that deleted nodes will lose their references and Garbage collected.

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
UndoRedoEnabled	Gets or sets a value indicating whether undo redo is enabled or not.	Dependency property	Boolean(True/False)	No

The following code example shows how to disable Undo Redo operation.

HTML

```
<sfDiagram:DiagramControl IsSymbolPaletteEnabled="True" >
<sfDiagram:DiagramControl.Model>
<sfDiagram:DiagramModel x:Name="diagramModel" >
</sfDiagram:DiagramModel>
</sfDiagram:DiagramControl.Model>
<sfDiagram:DiagramControl.View >
<sfDiagram:DiagramView UndoRedoEnabled="False"
ShowHorizontalGridLine="True" ShowVerticalGridLine="True">
</sfDiagram:DiagramView>
</sfDiagram:DiagramControl.View>
</sfDiagram:DiagramControl>
```

C#

```
DiagramView diagramView = new DiagramView();
diagramView.UndoRedoEnabled = false;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.UndoRedoEnabled = False
```

Clearing Undo Redo Stack

The following code example illustrates how to clear Undo Redo stack.

C#

```
DiagramView diagramView = new DiagramView();
diagramView.ClearUndoRedoStack();
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.ClearUndoRedoStack()
```

Printing Enhancements for Diagram Page

This feature enables you to print a copy of the diagram page, with or without using Print Dialog Box. This feature comes with two functionalities:

1. Printing a diagram using Print Dialog and Print Preview, and
2. Printing a diagram without using Print Dialog.

Use Case Scenarios

To print the diagram page, you can use this feature as it enables printing with different functionalities.

Tables for Properties, Methods and Events

Properties

Property	Description	Type	Value It Accepts	Default Values	Any other dependencies/ sub properties associated
ShowDialog	Gets or sets the Print Dialog to show or not.	CLR property	Bool (true/false)	True	No
PrintStretch	Gets or sets the page stretch for printing the document.	CLR property	Stretch.Fill,Stretch.None,Stretch.Uniform,Stretch.UniformToFill	-	No

Methods

Method	Description	Parameters	Return Type	Reference links
Print	Prints the diagram page using Print Dialog Box and Print Preview	void	void	No
Print	Prints the diagram page without using Print Dialog Box and Print Preview	PrintParameters	void	No

[Sample Link](#)

To view the sample for this feature:

1. Open the WPF Sample Browser from the Dashboard.
2. Navigate to Diagram -> Static Diagram ->Export Demo.

Adding Printing Enhancements for Diagram Page to an Application

This feature enables you to print a copy of diagram though:

- PrintPreview,
- Without PrintDialog (though code behind)

[Print Preview](#)

Diagram can be printed though PrintPreview using following code snippet:

- Through Code behind.

C#

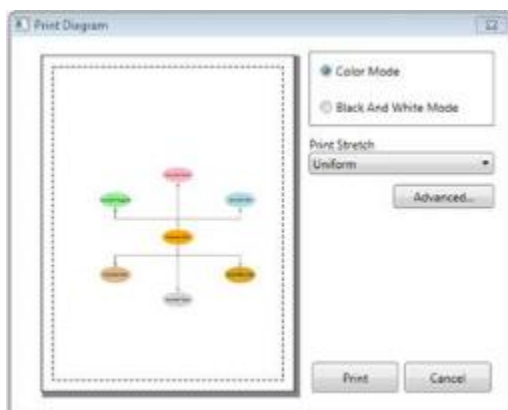
```
DiagramView diagramView = new DiagramView();  
diagramView.Print();
```

VB.NET

```
Dim diagramView As New DiagramView()  
diagramView.Print()
```

The following custom options can be customized using PrintPreview.

- Print Preview—To view the page before printing
- Different modes—To select printing such as Color, and Black and White
- Stretch—To adjust the fit of the image on the page



Print and PrintPreview Dialog Box

Printing a Diagram without PrintDialog Box

Diagram can be printed without using `PrintDialog` or `PrintPreview`, and by sending `PrintPreview` as an argument for printing as shown blow:

Print without dialog box

C#

```
DiagramView diagramView = new DiagramView();
PrintParameters p = new PrintParameters();
p.ShowDialog = false;
p.PrintStretch = Stretch.Fill;
diagramView.Print(p);
```

VB.NET

```
Dim diagramView As New DiagramView()
Dim p As New PrintParameters()
p.ShowDialog = False
p.PrintStretch = Stretch.Fill
diagramView.Print(p)
```

Support to Print the Diagram Shapes with Effects

Essential Diagram for WPF provides support to print the diagram shapes with the applied effects. When effects are applied to the nodes, they cannot be printed properly, due to the framework limitation. This feature enables you to overcome this limitation.

Use Case Scenarios

When you want to print a diagram page, in which you have applied effects for the nodes, you can use this feature to achieve this.

Properties

Property	Description	Type	Data Type	Reference links
CustomEffect	Gets or sets a value of the applied effect of the Node.	Dependency property	Effect	NA

Printing the Diagram Shapes with Effects

You can print the diagram shapes with the applied effects using the *CustomEffect* property. The following code illustrates this:

C#

```
DropShadowEffect effect = new DropShadowEffect();
effect.BlurRadius = 10;
Node node = new Node();
node.Shape = Shapes.Ellipse;
node.CustomEffect = effect;
diagramModel.Nodes.Add(node);
```

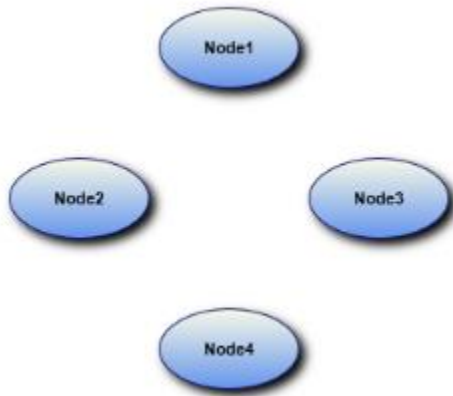
VB.NET

```
Dim effect As New DropShadowEffect()
```

```

effect.BlurRadius = 10
Dim node As New Node()
node.Shape = Shapes.Ellipse
node.CustomEffect = effect
diagramModel.Nodes.Add(node)

```



Printed Nodes with Effects

Drawing Tools

This feature enables you to draw different shapes and lines. The drawn shapes and lines will be converted into Node and LineConnector respectively.

The following shapes and lines are available in DrawingTools:

1. Ellipse
2. Rectangle
3. Rounded Rectangle
4. Polygon
5. Straight Line
6. Bezier Line
7. Orthogonal Line
8. Polyline

Use Case Scenarios

DrawingTools such as Microsoft Paint and Expression Blend support drawing a particular shape continually on a page. This feature too, enables you to draw a shape repeatedly without selecting it manually each time.

Properties

Property	Description	Type of the Property	Value It Accepts	Any Other Dependencies/Sub
----------	-------------	----------------------	------------------	----------------------------

				properties Associated
EnableDrawingTools	Gets or sets a value indicating whether EnableDrawingTools is enabled or not.	Dependency property	Boolean(True/False)	No
DrawingTools	Gets or sets the ShapeType to be used. Default value is DrawingTools.Ellipse	Dependency property	DrawingTools.EllipseDrawingTools.RectangleDrawingTool.RoundedRectangleDrawingTools.PolygonDrawingTools.StraightLineDrawingTools.BezierLineDrawingTools.OrthogonalLineDrawingTools.PolyLine	No

[Sample Link](#)

To view a sample:

1. Open the WPF sample browser from the dashboard.
2. Navigate to WPF Diagram > Product Showcase > Features demo

[Enable Drawing Tools](#)

To enable DrawingTools set EnableDrawingTools property to true. Shapes and line connectors are enabled, when the EnableDrawingTools property is enabled. By default, the value is set to false.

DrawingTools can be enabled using two methods.

- Through XAML.

The following code illustrates how to enable the DrawingTools.

HTML

```
<syncfusion:DiagramControl Name="diagramControl"
```

```
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel">
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView" EnableDrawingTools="True">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

- Through Code behind

The following code illustrates how to enable the DrawingTools.

C#

```
DiagramView diagramView = new DiagramView();
diagramView.EnableDrawingTools = true;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.EnableDrawingTools = True
```

Note: When the EnableDrawingTools is set to True, it has to be disabled manually, i.e. it cannot be disabled automatically. This will facilitate drawing shapes or lines continually until EnableDrawingTools is set to false manually. ObjectDrawn Event

The ObjectDrawn event will be raised when the node or line connector is drawn using the EnableConnection/Drawing Tools property of the diagram view. This event is used to identify the source of the drawing object.

Events

Event	Description	Arguments
ObjectDrawn	Raised when the node or line connector will be drawn using the Drawing Tools property. Event cannot be cancelled.	Drawing Object--Get the object that is drawn (node or line connector). Drawing Type--Get the type of object. For example, if the drawing object is a node, the drawing type is a Rectangle, RoundedRectangle, Ellipse, etc. If the drawing object is a line connector, the drawing type is an Arc, BezierLine, OrthogonalLine, etc.

The following code illustrates this feature.

In XAML

HTML

```
<syncfusion:DiagramControl.View>
```

```
<syncfusion:DiagramView Name="diagramView"
ObjectDrawn="diagramView_ObjectDrawn">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
```

In C#

C#

```
diagramView.ObjectDrawn += new
Syncfusion.Windows.Diagram.DrawingToolEventHandler(diagramView_ObjectDrawn);
//Event handler.
void diagramView_ObjectDrawn(object sender,
Syncfusion.Windows.Diagram.DrawingToolEventArgs evtArgs)
{
//User-specified code.
```

Drawing Mode

Essential Diagram for WPF provides support for Drawing mode in the diagram view drawing tool. It can be used to draw elements in diagram view by continuously or only once.

Properties

Property	Description	Type of property	Value it Accepts	Any other dependencies/sub properties associated
DrawingMode	Specifies the drawing mode for the drawing tool. The default value is Default.	Dependency property	DrawingMode.ContinuousDrawingMode.Default	No

This property is in DiagramView and can be set in the following ways:

- Through XAML
- Through code behind

The following code illustrates how to set the DrawingMode as Default:

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
<!-- Model to add nodes and connections.-->
<syncfusion:DiagramControl.Model>
```

```

<syncfusion:DiagramModel x:Name="diagramModel">
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<!--View to display nodes and connections added through the model.-->
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView" DrawingMode="Default">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>

```

C#

```

DiagramView diagramView = new DiagramView();
diagramview.ZOrderMode = DrawingMode.Default;

```

VB.NET

```

Dim diagramView As New DiagramView()
diagramview.ZOrderMode = DrawingMode.Default;

```



DrawingMode is Default

The following code illustrates how to set the DrawingMode as Default:

HTML

```

<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
<!-- Model to add nodes and connections.-->
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel">
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<!--View to display nodes and connections added through the model.-->
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView" DrawingMode="Continous">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>

```

C#

```

DiagramView diagramView = new DiagramView();
diagramview.ZOrderMode = DrawingMode.Continuous;

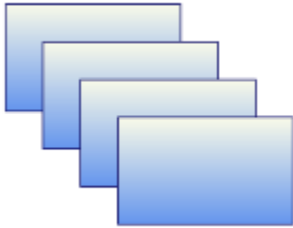
```

VB.NET

```

Dim diagramView As New DiagramView()
diagramview.ZOrderMode = DrawingMode.Continuous;

```



DrawingMode is Continuous

Select a Drawing Tool

DrawingTools consist of different shapes and line connectors. You can choose one of the DrawingTools at a time. By default, it is set to Ellipse.

The DrawingTool selection in Diagram View can be set in two methods:

- Through XAML.

The following code illustrates how to select a DrawingTool.

HTML

```
<syncfusion:DiagramControl Name="diagramControl">
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel x:Name="diagramModel">
    </syncfusion:DiagramModel>
  </syncfusion:DiagramControl.Model>
  <syncfusion:DiagramControl.View>
    <syncfusion:DiagramView Name="diagramView" DrawingTool="Polygon">
    <syncfusion:DiagramView.Page>
    </syncfusion:DiagramView>
  </syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

- Through Code behind

The following code illustrates how to select a DrawingTool.

C#

```
DiagramView diagramView = new DiagramView();
diagramView.DrawingTool = DrawingTools.Rectangle;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.DrawingTool = DrawingTools.Rectangle
```

Steps for Drawing

You can draw on a page by clicking and drag on the page.

Follow are the below steps to draw a shape or a line:

- Set the EnableDrawingTools property of DiagramView to true.
- Select the DrawingTool as required from DrawingTools option.
- Click and drag. Preview of the drawing will be displayed.
- Release the mouse. Shape or line will be drawn.

Note: These steps are common for all shapes and lines drawing, except Polygon and Polyline.

Shape Drawing

Preview Ellipse – while Drawing



Ellipse Preview

Ellipse – After Drawing.



Ellipse(Node)

Line Drawing

Bezier Line Preview – While Drawing



Bezier Line Preview

Bezier Line – After Drawing



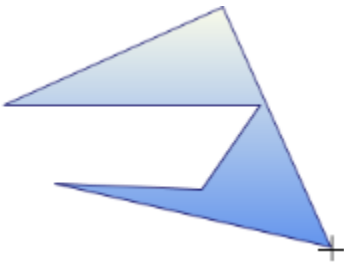
Bezier Line(Line Connector)

Note: *The drawn shape will be converted into a Node. The drawn line will be converted into a LineConnector. You can continually draw the selected shape. Lines cannot be drawn continually.*

Steps for drawing a Polygon and Polyline Drawing:

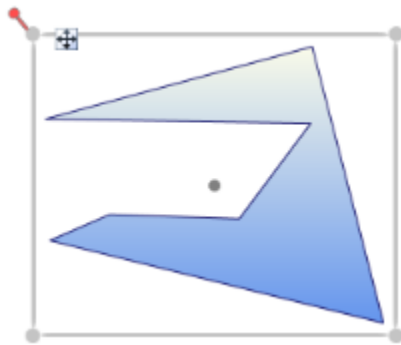
- Set the EnabledDrawingTools property of DiagramView to be true.
- Select the DrawingTool as required from DrawingTools option.
- Click, where you want the first point for polygon (or) polyline.
- Drag the mouse pointer. Preview of the drawing will be displayed.
- Click, where you want to place the Intermediate points of Polygon (or) Polyline
- Right-click to complete the drawing.

Preview Polygon – While Drawing



Polygon Preview

Polygon – After Drawing



Polygon (Node)

[Sample Link](#)

To view a sample:

1. Open the WPF sample browser from the dashboard.
2. Navigate to WPF Diagram > Product Showcase > Features demo.

[Preview Style for Line Connectors](#)

This feature allows you to modify the preview of a line connector by using the CustomPathStyle property of the DiagramView.

Properties

Property	Description	Type	Data Type
CustomPathStyle	Defines the path or style that has to be applied in the preview of a line connector.	Dependency Property	Style

Adding Options for Dashed Line Connector to an Application_

Defining the Preview Style

The preview style for line connectors can be defined either through XAML or code behind.

To define a preview style, see the following code. This code sample is for setting the preview of the line connector as dashed.

HTML

```
<Style TargetType="{x:Type Path}" x:Key="MyStyle">
  <Setter Property="StrokeDashArray" Value="2,2"/>
</Style>
```

Applying the Preview Style

To apply the preview style to a line connector, it has to be assigned to the CustomPathStyle property of DiagramView.

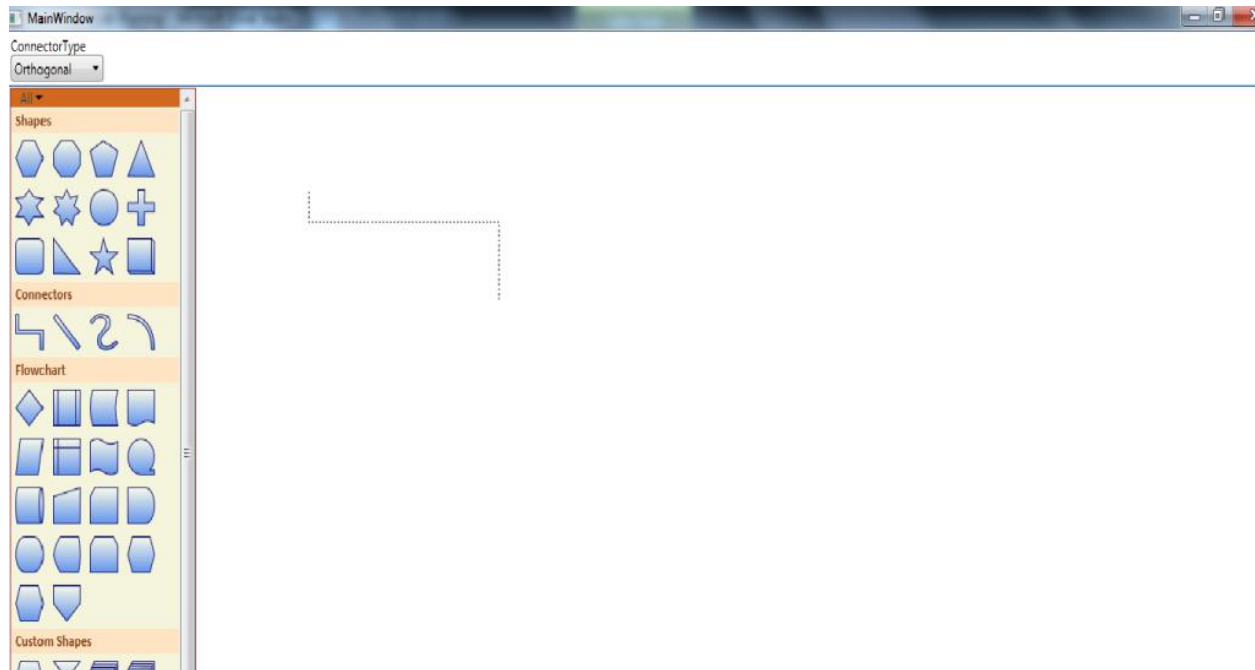
C#

```
DiagramView View1 = new DiagramView();
View1.CustomPathStyle = this.Resources["MyStyle"] as Style;
```

VB.NET

```
Dim view1 As New DiagramView()
View1.CustomPathStyle = this.Resources["MyStyle"] as Style
```

The CustomPathStyle property of DiagramView applies the style that is assigned to the preview of the drawing objects.



Preview of Dashed Orthogonal Line Connector

[Export Diagram](#)

Essential Diagram for WPF can be exported to the following File formats:

- Exporting into Image File Format
- Bitmap(.bmp)
- JPEG
- PNG
- GIF
- TIFF
- WDP
- Exporting into XPS Format
- Export to Clipboard

The Diagram can be exported using following overloaded methods:

- Stream
- Stream
- Stream with BitmapEncoder
- Stream with Rect
- Stream with Rect and BitmapEncoder
- FileName
- FileName
- FileName with BitmapEncoder
- FileName with Rect
- FileName with Rect and BitmapEncoder
- XPS
- Stream

- Stream with Rect
- FileName
- FileName with Rect

The following code illustrates how to export the Page as an image file.

Stream

Page can be exported into any streams such as FileStream and MemoryStream.

C#

```
System.IO.FileStream filestream = new
System.IO.FileStream("Diagram.jpeg", System.IO.FileMode.Create);
diagramview.Save(filestream);
```

Stream with Rect and Encoder

You can also save to a stream with type of encoder and rectangle portion to be clipped.

C#

```
System.IO.FileStream filestream = new
System.IO.FileStream("Diagram.jpeg", System.IO.FileMode.Create);
TiffBitmapEncoder encoder = new TiffBitmapEncoder();
Rect rect = new Rect(new Point(100, 100), new Point(300, 300));
diagramview.Save(filestream, rect, encoder);
```

Filename

Filename with Path

You can also export Page as an image file with directory path and file name.

C#

```
diagramview.Save(@"D:\ Diagram.jpeg");
```

Filename with Rect and Encoder

You can also specify the name of the file directly, specified portions of the DiagramPage, type of encoder such as TiffBitmapEncoder and GifBitmapEncoder.

C#

```
TiffBitmapEncoder encoder = new TiffBitmapEncoder();
Rect rect = new Rect(new Point(100, 100), new Point(300, 300));
string filename = " Diagram.jpeg";
diagramview.Save(filename, rect, encoder);
```

XPS

The DiagramPage can be exported in different ways:

- Stream
- Filename with Path
- Filename with Rect
- Stream with Rect

Stream

You can also save to a stream.

The following code illustrates how to save the Page.

C#

```
System.IO.FileStream filestream = new System.IO.FileStream("Diagram.xps",
System.IO.FileMode.Create);
diagramview.SaveToXps(filestream);
```

Stream with Rect

You can also save to a stream with specified size of the save area.

C#

```
System.IO.FileStream filestream = new
System.IO.FileStream(@"D:\Diagram.xps", System.IO.FileMode.Create);
Rect rect = new Rect(new Point(200, 200), new Point(500, 500));
diagramview.SaveToXps(filestream, rect);
```

Filename with Path

You can also specify the name of the file directly in the save method.

C#

```
diagramview.SaveToXps(@"D:\Diagram.xps", rect);
```

Filename with Rect

You can also specify the name of the file directly; Specified size of the save area.in the save method.

C#

```
Rect rect = new Rect(new Point(100, 100), new Point(500, 500));
diagramview.SaveToXps(@"D:\Diagram.xps", rect);
```

Shrink and Expand Options

While exporting a diagram as an image, the exported image size can be shrunk or expanded to a specific size. An aspect ratio of 1:1 will always be maintained while exporting. If the diagram size and the proposed size are in different ratios, the size of the image will be decided internally based on the ImageStretch property.

Parameters

Parameter	Description	Type of Property	Value It Accepts	Other Dependencies/ Subproperties Associated
-----------	-------------	------------------	------------------	--

filename	Target file name where the image will be saved (e.g., @"E:\Fig3.jpg")	CLR property	String	None
imagesize	Target image size. This property can be set to fit or expand the diagram to the specified size (e.g.: new Size(500, 500)).	CLR property	Size	None
ImageStretch	Decides whether the target image will be the same size as the diagram, expanded to a specified size, or shrunk to a specified size.	CLR property	EnumImageStretch.NoneImageStretch.BestFitImageStretch.ExpandImageStretch.Shrink	None

The following table lists the four options available for the ImageStretch property and their descriptions.

Option	Description
None	Target image will be the same size as the actual diagram.
Expand	Expands a small diagram to the specified size while exporting.

Shrink	Shrinks a huge diagram to the specified size while exporting.
BestFit	(Expand Shrink). Decides whether the diagram should be expanded or shrunk and will export it as an image.

The following code sample demonstrates how to use the `ImageSize` and `ImageStretch` options.

C#

```
DiagramView diagramView = new DiagramView();
diagramView.Save(@"E:\Fig3.jpg", new Size(500, 500), ImageStretch.Shrink);
```

Note: By default, when exporting a diagram that is huge in size (for example, 30,000 × 2,000), an 'Out of memory' exception is raised. By shrinking the diagram to a smaller size, the memory occupied will be less, so the exception will not be raised.

Export to Clipboard

Content of the Page can be copied onto the clipboard as an image, and can be pasted into applications such as Paint and Microsoft Word.

The following Code illustrates how to export the Page content into clipboard.

C#

```
diagramview.CopyToClipboard();
```

SizeToContent

`SizeToContent` support enables you to resize the diagram page depending upon the content size. This option can also be disabled. You can also define the size of diagram page and restrict the diagram element from moving out of a specified area.

Properties

Name	Description	Type	Data Type	Reference Links
SizeToContent	Specifies whether <code>SizeToContent</code> behavior is enabled or disabled.	Dependency Property	Bool	NA
BoundaryConstraintsArea	Defines the diagram page area.	Dependency Property	Rect	NA
BoundaryConstarintsEnabled	Restricts the diagram element moving out of a specified area.	Dependency Property	Bool	NA
PageBackgroundEffect	Sets the page background effect.	Dependency Property	Effect	NA
OffPageBackground	Sets the background brush for area outside Diagram Page.	Dependency Property	Brush	NA

Enabling SizeToContent

To enable SizeToContent support, set the SizeToContent property of DiagramView to true. To disable, set this to false. Default value is true.

Following code illustrates how to enable SizeToContent support:

HTML

```
<syncfusion:DiagramControl Name="diagramControl"
IsSymbolPaletteEnabled="True">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel"></syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView SizeToContent="True" Name="diagramView"/>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DiagramView diagramView = new DiagramView();
diagramView.SizeToContent = true;
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.SizeToContent = True
```

Defining Diagram Page

You can define the Diagram Page area. Use the BoundaryConstraintsArea property of DiagramView to define Diagram Page area.

Following code illustrates how to define Diagram Page area:

HTML

```
<syncfusion:DiagramControl Name="diagramControl"
IsSymbolPaletteEnabled="True">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel"></syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView BoundaryConstraintsArea="100,100,850,1195"
Name="diagramView"/>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DiagramView diagramView = new DiagramView();
diagramView.BoundaryConstraintsArea = new Rect(100, 100, 850, 1195);
```

VB.NET

```
Dim diagramView As New DiagramView()  
diagramView.BoundaryConstraintsArea = New Rect(100, 100, 850, 1195)
```

Restricting the Diagram Element

You can restrict the diagram element moving out of a specified area. Use the `BoundaryConstraintsEnabled` property of `DiagramView` for this purpose.

Following code illustrates how to restrict diagram element moving out of specific area:

HTML

```
<syncfusion:DiagramControl Name="diagramControl"  
IsSymbolPaletteEnabled="True">  
<syncfusion:DiagramControl.Model>  
<syncfusion:DiagramModel x:Name="diagramModel"></syncfusion:DiagramModel>  
</syncfusion:DiagramControl.Model>  
<syncfusion:DiagramControl.View>  
<syncfusion:DiagramView BoundaryConstraintsEnabled="True"  
Name="diagramView"/>  
</syncfusion:DiagramControl.View>  
</syncfusion:DiagramControl>
```

C#

```
DiagramView diagramView = new DiagramView();  
diagramView.BoundaryConstraintsEnabled = true;
```

VB.NET

```
Dim diagramView As New DiagramView()  
diagramView.BoundaryConstraintsEnabled = True
```

Customizing options

Following properties enables to customize the appearance of the diagram page:

- `PageBackground` specifies the background of the diagram page.
- `OffPageBackground` specifies the off page background.
- `PageEffect` specifies the background effect of the diagram page.

Customizing Background

To customize the background, use the `PageBackground` of `DiagramView`. Default value is White.

Following code illustrated how to customize the background:

HTML

```
<syncfusion:DiagramControl Name="diagramControl"  
IsSymbolPaletteEnabled="True">  
<syncfusion:DiagramControl.Model>  
<syncfusion:DiagramModel x:Name="diagramModel"></syncfusion:DiagramModel>  
</syncfusion:DiagramControl.Model>  
<syncfusion:DiagramControl.View>  
<syncfusion:DiagramView OffPageBackground="White" PageBackground="White"  
Name="diagramView"/>  
</syncfusion:DiagramControl.View>  
</syncfusion:DiagramControl>
```

```
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DiagramView diagramView = new DiagramView();
diagramView.PageBackground = new SolidColorBrush(Colors.Gray);
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.PageBackground = New SolidColorBrush(Colors.Gray)
```

Customizing the Off Page's Background

To customize the background of the Off Page, use the `OffPageBackground` property of `DiagramView`. Default value is White.

Following code illustrates how to customize the background of the Off Page:

HTML

```
<syncfusion:DiagramControl Name="diagramControl"
IsSymbolPaletteEnabled="True">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel"></syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView OffPageBackground="White" PageBackground="White"
Name="diagramView"/>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DiagramView diagramView = new DiagramView();
diagramView.OffPageBackground = new SolidColorBrush(Colors.White);
```

VB.NET

```
Dim diagramView As New DiagramView()
diagramView.OffPageBackground = New SolidColorBrush(Colors.White)
```

Customizing page effect

To customize the page effect, use the `PageEffect` property of `DiagramView`.

Following code illustrates how to customize the page effect:

HTML

```
<syncfusion:DiagramControl Name="diagramControl"
IsSymbolPaletteEnabled="True">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel"></syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
```

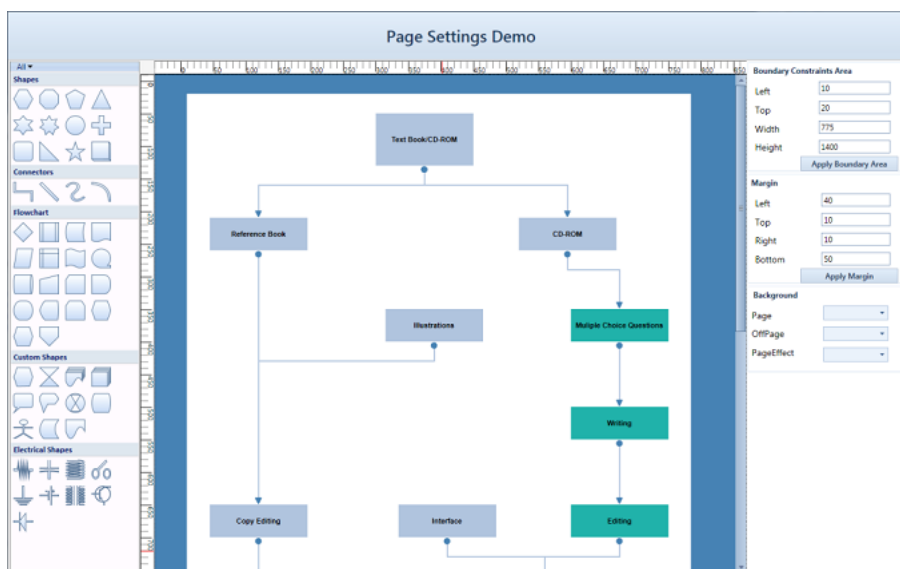
```
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView OffPageBackground="White" PageBackground="White"
Name="diagramView">
<syncfusion:DiagramView.BackgroundEffect>
<DropShadowEffect BlurRadius="12" Color="Black" Direction="-350"
ShadowDepth="30"/>
</syncfusion:DiagramView.BackgroundEffect>
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

C#

```
DropShadowEffect drop = new DropShadowEffect();
drop.BlurRadius = 25;
drop.Color = new Color { B = 205, G = 148, R = 79 };
drop.Direction = 450;
drop.ShadowDepth = -2;
diagramView.BackgroundEffect = drop;
```

VB.NET

```
Dim drop As New DropShadowEffect()
drop.BlurRadius = 25
drop.Color = New Color With {.B = 205, .G = 148, .R = 79}
drop.Direction = 450
drop.ShadowDepth = -2
diagramView.BackgroundEffect = drop
```



Customized Diagram Page

Touch Support

Touch support for diagram view has the following features:

- Drag and Drop from the SymbolPalette

- Dragging the Node and LineConnector on DiagramView
- Panning
- Multiple selection

Steps for Dragging

Dragging operation can be performed by

1. Touch a particular element in the touch screen monitor.
2. Start dragging the element to the desired location.
3. Take the finger off from the screen after it is reached to the desired location.

This dragging gesture is used to perform the following operation.

Drag and Drop from the SymbolPalette

4. Touch the SymbolPalettItem for Drag and Drop.
5. Drag the SymbolPalettItem and Drop onto DiagramView by touching the finger in DiagramView.

Dragging the Node and LineConnector on DiagramView

6. Touch the Node or Lineconnector for Dragging.
7. The selected item can be moved around using your finger.

Panning

1. Please enable the IsPanEnabled property of the DiagramView.
2. Drag the finger over DiagramView, the DiagramView gets panned along the finger.

Multiple selections

1. Touch the empty space of the DiagramView.
2. Drag the finger in DiagramView, the Selection Adorner will be visible.
3. Objects which are getting intersected while dragging will get selected.



Drag

MultiTouch Support

MultiTouch support has been provided to enable zoom the page and resizing a Node.

Steps for Spreading and Pinching

1. Use two fingers to touch the monitor.
2. Move the fingers away to perform spread operation.
3. Move the fingers close to each other to perform pinch operation.
4. Take the fingers off from the screen after the required size is achieved.

This spread and pinch gesture is used to perform, zooming and resizing operation.

- Zooming the DiagramView with two fingers can be done
- Resizing the Node with two fingers can be done



Spread



Pinch

Steps for Zooming

1. Touch the DiagramView with two fingers.
2. Pinch represents the ZoomOut.
3. Spread represents the ZoomIn.

Steps for Resizing

1. Touch the Node for resizing with two Fingers.
2. Pinch Represents - Increase the size of the Node.
3. Spread Represents - Decrease the size of the Node.

Overview Control

The Overview control enables you to focus on a particular area of the entire diagram page. The Overview control displays a preview of the entire diagram page with a ZoomPan window. The ZoomPan window is a resizable red rectangle which allows you to focus on the desired area.

For more details about the Overview control refer to the user guide of Essential Tools for WPF>Tools WPF Controls>Overview

Use Case Scenarios

This control helps you view huge content hosted within a scroll viewer.

Sample Link

WPF > Diagram > Performance Samples > Overview Demo

Using Overview Control in Diagram Control

Follow the steps below to use the Overview control for DiagramControl:

4. Create an instance of DiagramControl.

HTML

```

<!-- Diagram Control -->
<syncfusion:DiagramControl Name="diagramControl"
IsSymbolPaletteEnabled="True">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel"/>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView"/>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>

```

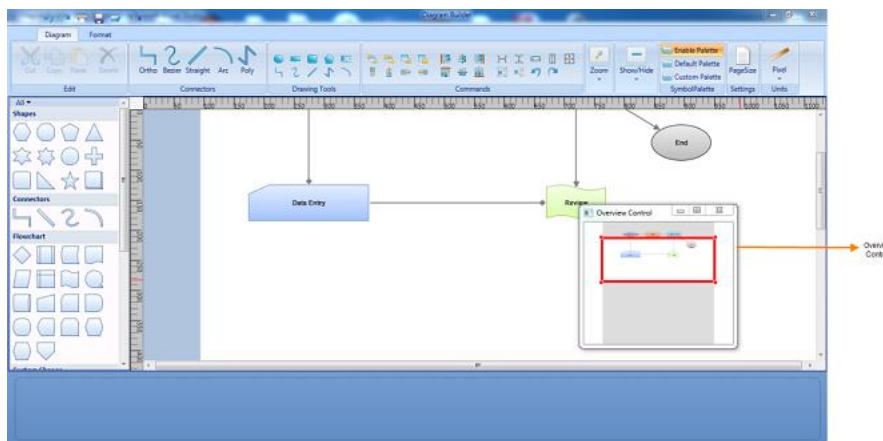
5. Create an instance of Overview and set the OverviewSourceAncestor property to DiagramControl.

HTML

```

<!-- Overview Control -->
<shared:Overview Grid.Column="1" OverviewSourceAncestor="{Binding
ElementName=diagramView}"/>
</Grid>

```



Overview Control in a Diagram

BringIntoCenter

The bring-into-center feature provides the option to bring the element or Rect area into center of the viewport.

This can be used to clearly view the particular element and its neighbor elements in the center of the viewport.

Methods

Method	Description	Parameters	Type	Return Type
BringIntoCenter	Used to bring the element into the center of the page.	BringIntoCenter(Object pageElement) This method only accepts Node, LineConnector, or Rect type objects.	WPF	Void

Adding BringIntoCenter to an Application

C#

```
DiagramView diagramView=new DiagramView();
//Call the Bring IntoCenter method to bring the "node" object into center of the viewport.
diagramView.BringIntoCenter(node);
```

VB.NET

```
Dim diagramView As New DiagramView()
'Call the Bring IntoCenter method to bring the "node" object into center of the viewport.
diagramView.BringIntoCenter(node)
```

The BringIntoCenter method of the DiagramView class is used to bring the object into center of the viewport. This method will accept only parameters of type Node, LineConnector, or Rect.

Bring Into Viewport

The Bring Into ViewPort feature provides option to bring the element or Rect area into the Viewport.

This feature can be applied to the Nodes and Connectors.

Method	Description	Parameters	Return Type
BringIntoViewPort	brings the element into the Viewport of the page.	Bring Into ViewPort(Object pageElement) This method accepts the Node and LineConnector type objects.	Void

Adding BringIntoViewPort to an Application

The BringIntoViewPort method of the DiagramView class is used to bring the object into the Viewport. This method accepts only the parameters of type Node, LineConnector, or Rect.

C#


```
DiagramView diagramview = new DiagramView();
//Calls the BringIntoViewPort method to bring the "node" object to viewport
of the diagrampage.
diagramview.BringIntoViewPort(node);
```

VB.NET

```
Dim diagramview As New DiagramView()
//Calls the BringIntoViewPort method to bring the "node" object to viewport
of the diagrampage.
diagramview.BringIntoViewPort(node)
```

Item Selection Mode

Essential Diagram for WPF provides support for the Item selection mode for diagram view elements. This determines whether the user can select single or multiple elements at a time.

Properties

Property	Description	Type of property	Value it Accepts	Any other dependencies/ sub properties associated
ItemSelectionMode	Specifies the item selection mode for diagram view elements. The default value is Multiple.	Dependency property	ItemSelectionMode.SingleItemSelectionMode.Multiple	No

This property is available in DiagramView and can be set in the following ways:

- Through XAML
- Through code behind

The following code illustrates how to set the ItemSelectionMode as Single:

HTML

```
<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
  <!-- Model to add nodes and connections.-->
  <syncfusion:DiagramControl.Model>
    <syncfusion:DiagramModel x:Name="diagramModel">
    </syncfusion:DiagramModel>
```

```

</syncfusion:DiagramControl.Model>
<!--View to display nodes and connections added through the model.-->
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView" ItemSelectionMode="Single">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>

```

C#

```

DiagramView diagramView = new DiagramView();
diagramview.ItemSelectionMode = ItemSelectionMode.Single;

```

VB.NET

```

Dim diagramView As New DiagramView()
diagramview.ItemSelectionMode = ItemSelectionMode.Single;

```



ItemSelectionMode is Single

The following code illustrates how to set the ItemSelectionMode as Multiple:

HTML

```

<!--Diagram Control-->
<syncfusion:DiagramControl Name="diagramControl">
<!-- Model to add nodes and connections.-->
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="diagramModel">
</syncfusion:DiagramModel>
</syncfusion:DiagramControl.Model>
<!--View to display nodes and connections added through the model.-->
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView Name="diagramView" ItemSelectionMode="Multiple">
</syncfusion:DiagramView>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>

```

C#

```

DiagramView diagramView = new DiagramView();
diagramview.ItemSelectionMode = ItemSelectionMode.Multiple;

```

VB.NET

```

Dim diagramView As New DiagramView()
diagramview.ItemSelectionMode = ItemSelectionMode.Multiple;

```



ItemSelectionMode is Multiple

Context View in WPF Diagram (classic)

A context view is a small view of an entire diagram with a new arrangement, where the small view and the arrangement are based on a particular view of the selected node. It provides three views:

- Successors view—shows all child nodes of the selected node.
- Predecessors view—displays all parent nodes of the selected node.
- Neighborhood view—displays the immediate child and parent nodes of the selected node.

Tables for Properties and Methods

Properties

Property	Description	Type	Data Type
ContextViewMode	This property is used to get or set the context view mode of the context view manager.	Dependency property	ContextViewMode
Layout	This property is used to manage layout information of the context view manager.	CLR property	ILayout
SourceControl	This property is used to get a Diagram control to be a source of the context view manager.	CLR property	DiagramControl
TargetControl	This property is used to get a Diagram control to be a target of the context view manager.	CLR property	DiagramControl

Methods

Method	Description	Parameters	Type
public ContextViewManager(DiagramControl source, DiagramControl target)	Constructor of the context view manager.	sourceâ€”a Diagram control in which selection changes will be monitored.targetâ€”a Diagram control to which a new diagram will be created based on the selected node	In WPF

		and the context view chosen.	
RefreshLayout	This method is used to update the layout. It will be called automatically and can also be forced to be updated.	NA	In WPF

Adding ContextViewManager to an Application

ContextViewManager is a class that helps to communicate between a source diagram and a target diagram.

- Source—it is a Diagram control in which selection changes will be monitored.
- Target—it is a Diagram control to which a new diagram will be created based on the selected node and the context view chosen.

Steps to create ContextViewManager

1. Create a Diagram control to be used as a target to show the context view of another Diagram control.

HTML

```
<syncfusion:DiagramControl Height="235" Width="250"
Name="targetDiagramControl">
<syncfusion:DiagramControl.Model>
<syncfusion:DiagramModel x:Name="targetDiagramModel"/>
</syncfusion:DiagramControl.Model>
<syncfusion:DiagramControl.View>
<syncfusion:DiagramView EnableFitToPage="True" Name="targetDiagramView"/>
</syncfusion:DiagramControl.View>
</syncfusion:DiagramControl>
```

2. Create another Diagram control to be used as a source.
3. Create a ContextViewManager instance to synchronize the source and target diagrams.

C#

```
// Create ContextViewManager to attach the source and target diagrams.
ContextViewManager ContextView = new ContextViewManager(source, target);
//Set ContextViewMode as Predecessors.
ContextView.ContextViewMode =ContextViewMode.Predecessors;
// Create a layout for ContextView.
DirectedTreeLayout layout = new
DirectedTreeLayout (diagramModel,diagramView);
diagramModel.HorizontalSpacing = 10;
diagramModel.VerticalSpacing = 80;
diagramModel.SpaceBetweenSubTrees = 10;
// Set the layout of the ContextView.
```

```
ContextView.Layout = layout;
```

VB.NET

```
' Create ContextViewManager to attach source and target diagrams.
Dim ContextView As New ContextViewManager(source, target)
' Set ContextViewMode as Predecessors.
ContextView.ContextViewMode = ContextViewMode.Predecessors
' Create a layout for ContextView.
Dim layout As New DirectedTreeLayout(diagramModel, diagramView)
diagramModel.HorizontalSpacing = 10
diagramModel.VerticalSpacing = 80
diagramModel.SpaceBetweenSubTrees = 10
' Set the layout of the ContextView.
ContextView.Layout = layout
```

Creating Different Views in the ContextViewManager in WPF Diagram

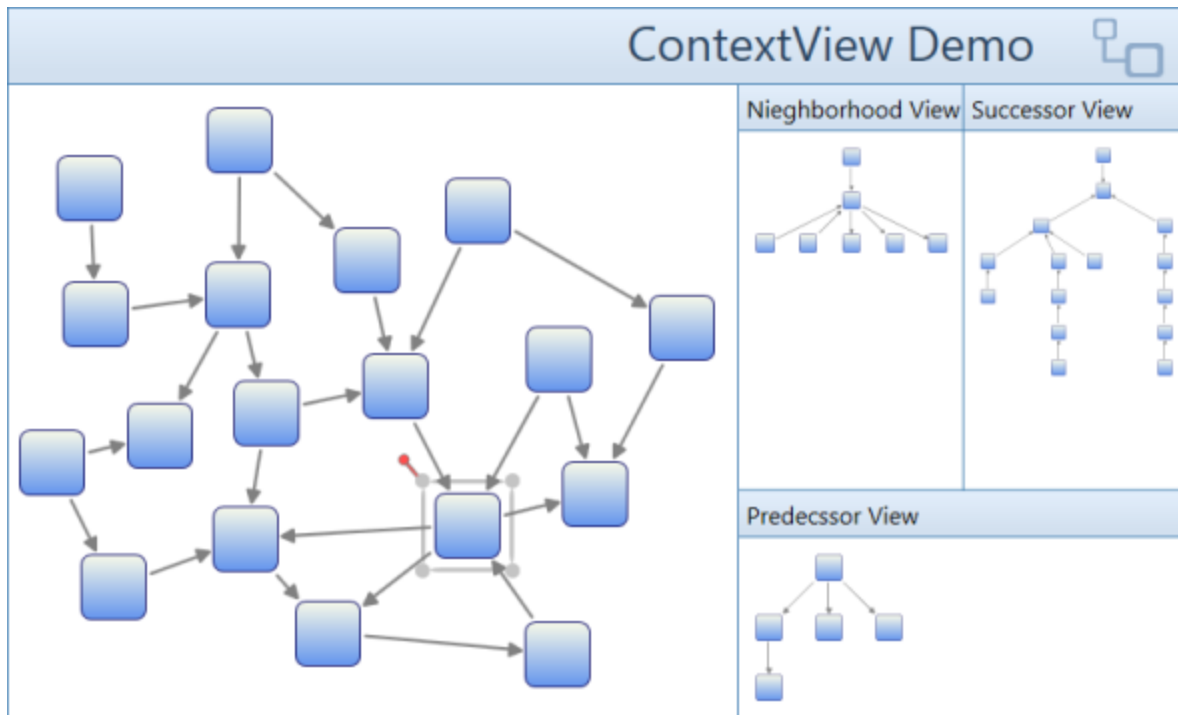
ContextViewManager can be used to choose any one of the three different views. The nodes and connection will be created based on the view chosen.

1. Predecessors View: The predecessors view is used to display all hierarchical parent nodes of the selected node in a layout manner.
2. Successors View: The successor view displays all the hierarchical child nodes of the selected node.
3. Neighborhood View: This view will be used to show immediate child and parent nodes of the selected node.

The following code example explains how to choose the predecessors view

VB.NET

```
' Create ContextViewManager to attach source and target diagrams.
Dim ContextView As New ContextViewManager(source, target)
' Set ContextViewMode as Predecessors.
ContextView.ContextViewMode = ContextViewMode.Predecessors
```



Context View

Note: By default, the context view manager will show the neighborhood view.

SymbolPalette in WPF Diagram (classic)

The SymbolPalette control displays node shapes and allows you to drag and drop symbols onto diagrams. It supports grouping and filtering symbols. It allows you to classify items as groups so they can be navigated easily. Also, custom shapes can be added to the SymbolPalette.

Methods for SymbolGroups in SymbolPalette

Name	Parameters	Return Type	Description	Reference Links
Add(SymbolPaletteGroup)	SymbolPaletteGroup	Void	Adds the SymbolPaletteGroup to the SymbolPalette.	Symbol Groups
Remove(SymbolPaletteGroup)	SymbolPaletteGroup	Void	Removes the SymbolPaletteGroup from SymbolPalette.	Symbol Groups
RemoveAt(int)	Int	Void	Removes the SymbolPaletteGroup from SymbolPalette at the given index.	Symbol Groups
Clear()	Null	Void	Clears all the SymbolPaletteGroups from the SymbolPalette.	Symbol Groups

Methods for SymbolFilters in SymbolPalette

Name	Parameters	Return Type	Description	Reference Links
Add(SymbolPaletteFilter)	SymbolPaletteFilter	Void	Adds the SymbolPaletteFilter to the SymbolPalette.	Symbol Filters
Remove(SymbolPaletteFilter)	SymbolPaletteFilter	Void	Removes the SymbolPaletteFilter from SymbolPalette.	Symbol Filters
RemoveAt(int)	Int	Void	Removes the SymbolPaletteFilter from SymbolPalette at the given index.	Symbol Filters SymbolGroups
Clear()	Null	Void	Clears all the SymbolPaletteFilter from the SymbolPalette.	Symbol Filters

Enable/Disable SymbolPalette

The SymbolPalette can be displayed by setting the IsSymbolPaletteEnabled property to True. By default, this property is disabled.

The following code can be used to enable the SymbolPalette.

XML

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EssentialDiagramWPF" Height="400" Width="600"
xmlns:sfdiagram="clr-namespace:Syncfusion.Windows.Diagram;assembly=Syncfusion.Diagram.WPF"
xmlns:local="clr-namespace:WpfApplication1">
  <Grid Name="diagramgrid">
    <sfdiagram:DiagramControl IsSymbolPaletteEnabled="True">
    </sfdiagram:DiagramControl>
  </Grid>
</Window>
```

C#

```
DiagramControl diagramcontrol = new DiagramControl();
diagramcontrol.IsSymbolPaletteEnabled = true;
```

VB.NET

```
Dim diagramcontrol As New DiagramControl()
diagramcontrol.IsSymbolPaletteEnabled = True
```



[Preview for Symbol Palette Item](#)

Essential Diagram for WPF provides preview support for Symbol Palette. When you drag an item from Symbol Palette to Diagram View, Preview of the dragged item will be displayed. You can enable or disable the preview support. You can also customize the preview.

[Use Case Scenario](#)

This feature displays a preview of the item you drag from Symbol Palette, thus enables you to identify the item you are dragging from the symbol palette to Diagram view.

[Properties](#)

Property	Description	Type	Data Type	Reference links
----------	-------------	------	-----------	-----------------

ShowPreview	Gets or sets a value indicating whether preview is enabled. The default value is true.	Dependency property	Boolean	NA
PreviewBrush	Gets or sets a value for preview content.	Dependency property	Brush	NA

Enabling Preview Support

To enable preview for the dragged item from Symbol Palette, set the ShowPreview property of SymbolPalette to true. To disable preview set this to false. By default this is set to true.

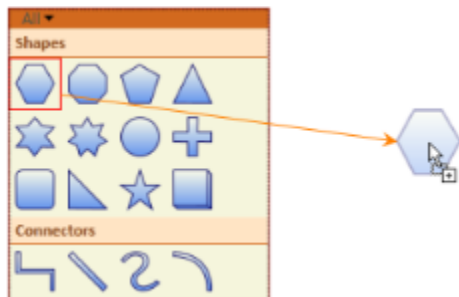
Following code example illustrates how to enable preview support:

C#

```
DiagramControl diagramControl1 = new DiagramControl();
diagramControl1.SymbolPalette.ShowPreview = true;
```

VB.NET

```
Dim diagramControl1 As New DiagramControl()
diagramControl1.SymbolPalette.ShowPreview = true
```



Preview of Dragged Item

Change the preview content using PreviewBrush

You can customize the preview content using the PreviewBrush property of SymbolPaletteItem.

Following code example illustrates how to customize preview content:

C#

```
DiagramControl diagramControl1 = new DiagramControl();
(diagramControl1.SymbolPalette.SymbolGroups[0].Items[0] as
SymbolPaletteItem).PreviewBrush = Brushes.CornflowerBlue;
```

VB.NET

```
Dim diagramControl1 As New DiagramControl()
TryCast(diagramControl1.SymbolPalette.SymbolGroups(0).Items(0),
SymbolPaletteItem).PreviewBrush = Brushes.CornflowerBlue
```



Customized Preview Content

Symbol Filters

A SymbolPalette filter can be added to the SymbolPalette control, using the SymbolFilters property, so that only desired SymbolPalette groups get displayed. The SetFilterIndexes property is used to specify the index value of the filters for which the group is to be displayed. The filter names are specified integer values, with the first filter index starting from 0. Based on the filter indexes specified for that particular group, the visibility of the group is controlled. So the group gets displayed only when any of the specified filter names are selected.

The following lines of code can be used to specify the SymbolPalette filter of the SymbolPalette Group.

C#

```
SymbolPaletteFilter sfilter = new SymbolPaletteFilter();
sfilter.Label = "Custom";
dc.SymbolPalette.SymbolFilters.Add(sfilter);
SymbolPaletteGroup group = new SymbolPaletteGroup();
group.Label = "Custom";
SymbolPalette.SetFilterIndexes(group, new Int32Collection(new int[] { 0, 6 }));
dc.SymbolPalette.SymbolGroups.Add(s);
```

VB.NET

```
Dim sfilter As New SymbolPaletteFilter()
sfilter.Label = "Custom"
dc.SymbolPalette.SymbolFilters.Add(sfilter)
Dim group As New SymbolPaletteGroup()
group.Label = "Custom"
SymbolPalette.SetFilterIndexes(group, New Int32Collection(New Integer() { 0, 6 }))
dc.SymbolPalette.SymbolGroups.Add(s)
```

This adds a new empty group named "Custom" and creates a filter for it.



SymbolPalette Filter

The `SetFilterIndexes` property specifies the index value for the group as 0,4 which implies that this group should be displayed when the filter index is 0 ("All") or 4 ("Custom").

Remove SymbolPaletteFilters

Like `SymbolPaletteGroups`, the `SymbolPaletteFilters` are also indexed from 0. The index 0 refers to the filter All. The index 1 refers to the filter Shapes and so on. The following table lists the filters with their index numbers.

Filter name	Index
All	0
Shapes	1
Connectors	2
Flowchart	3
Custom Shapes	4
Electrical Shapes	5

a) Removing filters and groups named Shapes, Custom Shapes and Electrical Shapes

Use the following code to remove the filters and groups named Shapes, Custom Shapes and Electrical Shapes:

C#

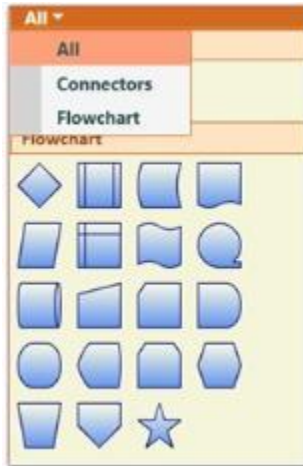
```
DiagramControl diagramControl = new DiagramControl();
diagramControl.SymbolPalette.SymbolGroups.Remove(diagramControl.SymbolPalett
e.SymbolGroups[4]);
diagramControl.SymbolPalette.SymbolGroups.Remove(diagramControl.SymbolPalett
e.SymbolGroups[3]);
diagramControl.SymbolPalette.SymbolGroups.Remove(diagramControl.SymbolPalett
e.SymbolGroups[0]);
diagramControl.SymbolPalette.SymbolFilters.Remove(diagramControl.SymbolPalett
e.SymbolFilters[5]);
diagramControl.SymbolPalette.SymbolFilters.Remove(diagramControl.SymbolPalett
e.SymbolFilters[4]);
diagramControl.SymbolPalette.SymbolFilters.Remove(diagramControl.SymbolPalett
e.SymbolFilters[1]);
```

VB.NET

```
Dim diagramControl As New DiagramControl()
diagramControl.SymbolPalette.SymbolGroups.Remove(diagramControl.SymbolPalett
e.SymbolGroups(4))
diagramControl.SymbolPalette.SymbolGroups.Remove(diagramControl.SymbolPalett
e.SymbolGroups(3))
diagramControl.SymbolPalette.SymbolGroups.Remove(diagramControl.SymbolPalett
e.SymbolGroups(0))
diagramControl.SymbolPalette.SymbolFilters.Remove(diagramControl.SymbolPalett
e.SymbolFilters(5))
```

```
diagramControl.SymbolPalette.SymbolFilters.Remove(diagramControl.SymbolPalet
te.SymbolFilters(4))
diagramControl.SymbolPalette.SymbolFilters.Remove(diagramControl.SymbolPalet
te.SymbolFilters(1))
```

Run the application. The following output is displayed and the groups and filters are removed.



Palette with Groups and Filters removed

Symbol Groups

A `SymbolPaletteGroup` is a collection of `SymbolPalette` items. It is used to group the items in the `SymbolPalette` control based on classifications provided. The `SymbolPalette` group can be added to the `SymbolPalette` using the `SymbolGroups` property. The filter index for the new groups should always start from 6 as the first five indices are predefined for the existing groups.

Use the following code to add a group to `SymbolPalette`:

C#

```
SymbolPaletteGroup s = new SymbolPaletteGroup();
s.Label = "Custom";
SymbolPalette.SetFilterIndexes(s, new Int32Collection(new int[] { 0, 6 }));
diagramControl.SymbolPalette.SymbolGroups.Add(s);
```

VB.NET

```
Dim s As New SymbolPaletteGroup()
s.Label = "Custom"
SymbolPalette.SetFilterIndexes(s, New Int32Collection(New Integer() { 0, 6
}))
diagramControl.SymbolPalette.SymbolGroups.Add(s)
```

Run the application. A new empty group named `Custom` is added to the `SymbolPalette`.

Remove SymbolPaletteGroups

The `SymbolPaletteGroups` are indexed from 0. Therefore, the group with name `Shapes` is indexed as 0, the group with name `Connectors` is indexed as 1 and so on. The groups can be removed using their corresponding index values. The following table lists the groups with their index numbers.

Group Name	Index
Shapes	0
Connectors	1
Flowchart	2
Custom Shapes	3
Electrical Shapes	4

a) Removing filter and group named Electrical Shapes and refreshing the Filters

Use the following code to remove the filter and group named Electrical Shapes:

C#

```
DiagramControl diagramControl = new DiagramControl();
diagramControl.SymbolPalette.SymbolGroups.Remove(diagramControl.SymbolPalett
e.SymbolGroups[4]);
diagramControl.SymbolPalette.SymbolFilters.Remove(diagramControl.SymbolPalett
e.SymbolFilters[5]);
//Refreshing the filters
foreach (SymbolPaletteGroup group in
diagramControl.SymbolPalette.SymbolGroups)
{
    Int32Collection indices = SymbolPalette.GetFilterIndexes(group);
    if (indices.Contains(5))
    {
        indices.Remove(5);
        SymbolPalette.SetFilterIndexes(group, indices);
    }
    if (indices.Contains(6))
    {
        indices.Remove(6);
        indices.Add(5);
        SymbolPalette.SetFilterIndexes(group, indices);
    }
}
```

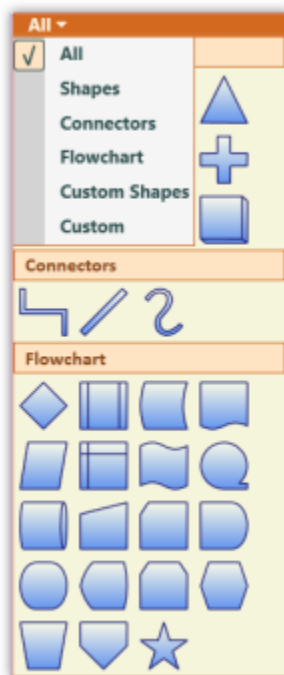
VB.NET

```
Dim diagramControl As New DiagramControl()
diagramControl.SymbolPalette.SymbolGroups.Remove(diagramControl.SymbolPalett
e.SymbolGroups(4))
diagramControl.SymbolPalette.SymbolFilters.Remove(diagramControl.SymbolPalett
e.SymbolFilters(5))
'Refreshing the filters
For Each group As SymbolPaletteGroup In
diagramControl.SymbolPalette.SymbolGroups
    Dim indices As Int32Collection = SymbolPalette.GetFilterIndexes(Group)
    If indices.Contains(5) Then
        indices.Remove(5)
        SymbolPalette.SetFilterIndexes(group, indices)
    End If
```

```
If indices.Contains(6) Then
indices.Remove(6)
indices.Add(5)
SymbolPalette.SetFilterIndexes(group, indices)
End If
```

Next group

Run the application. The following output is displayed and the groups and filters are removed.



Palette with Groups and Filters removed

Note: Whenever a filter is removed the group containing the next filter index must be decremented by one to get the proper output as mentioned in the above code snippet.

b) Removing all the groups from the palette

To remove all the groups from the palette, the Clear method can be used. The following code illustrates the usage of Clear method.

C#

```
DiagramControl diagramControl = new DiagramControl();
diagramControl.SymbolPalette.SymbolGroups.Clear();
```

VB.NET

```
Dim diagramControl As New DiagramControl()
diagramControl.SymbolPalette.SymbolGroups.Clear()
```

Run the application. All the groups are removed from the SymbolPalette.

Symbol Designer

Symbol Designer application allows you to create new palettes with symbols, and also modify the existing palettes. You can use these palettes in your applications, and also in the Diagram Builder.

Software Path

..\..\Syncfusion\Essential Studio\<Version Number>\utilities\Diagram WPF\Symbol Designer

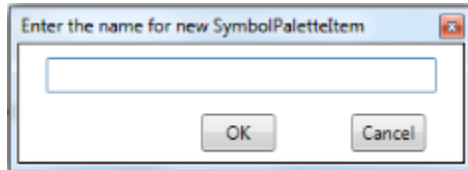
(or)

Start Menu > All Programs > Syncfusion > Utilities > Diagram > WPF > Symbol Designer

Creating SymbolPalettetItem

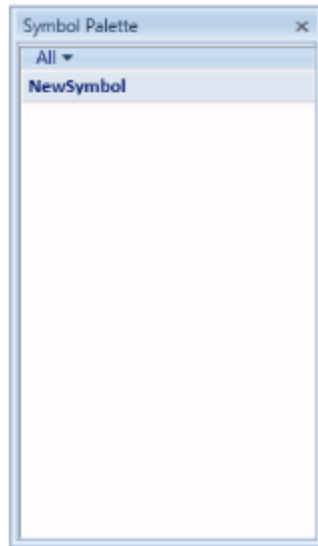
To create your own custom symbols in the symbol designer, follow the procedure given below:

1. Open the Symbol Designer tool which is available in the software path given above.
2. If you want to create a new symbol, select New option in the File menu. Type a name for the palette as shown in the below sample and click OK.



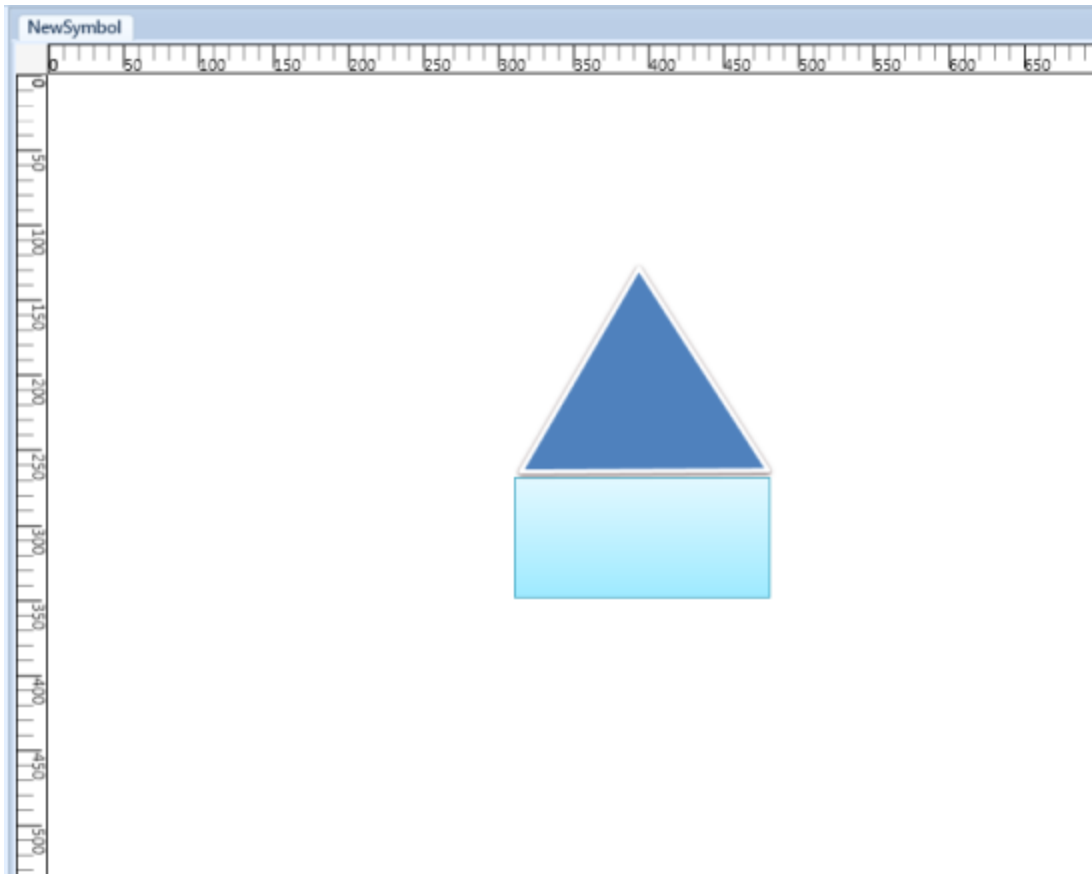
New Symbol PalettetItem Dialog Box

3. A new symbol palette is created with the given name after which, you can design your own symbol.



Flow Diagram Symbol Palette

4. Draw the desired shapes using drawing tool in the work area.



Symbol

The Shape of the SymbolPalettetItem can be customized further using format tab in Symbol Designer:

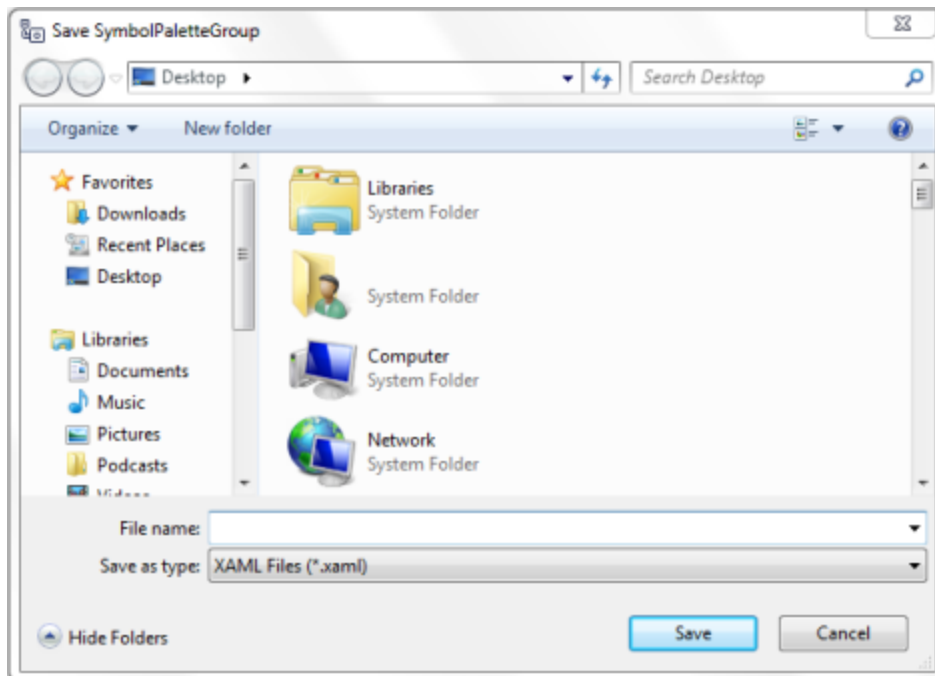
- Background
- Outline
- Dash array
- Shadow
- Soft edges
- Glow visual effect

The node label can also be customized.



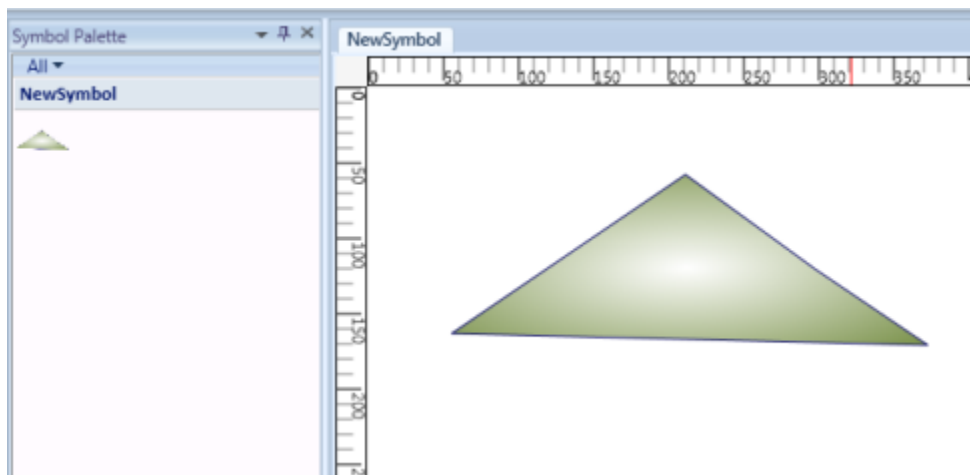
Format tab

5. After creating required symbols, we have to save this symbol into the symbol palette. Go to the File menu and click Save. A Save SymbolPaletteGroup dialog will appear as in the following screen shot.



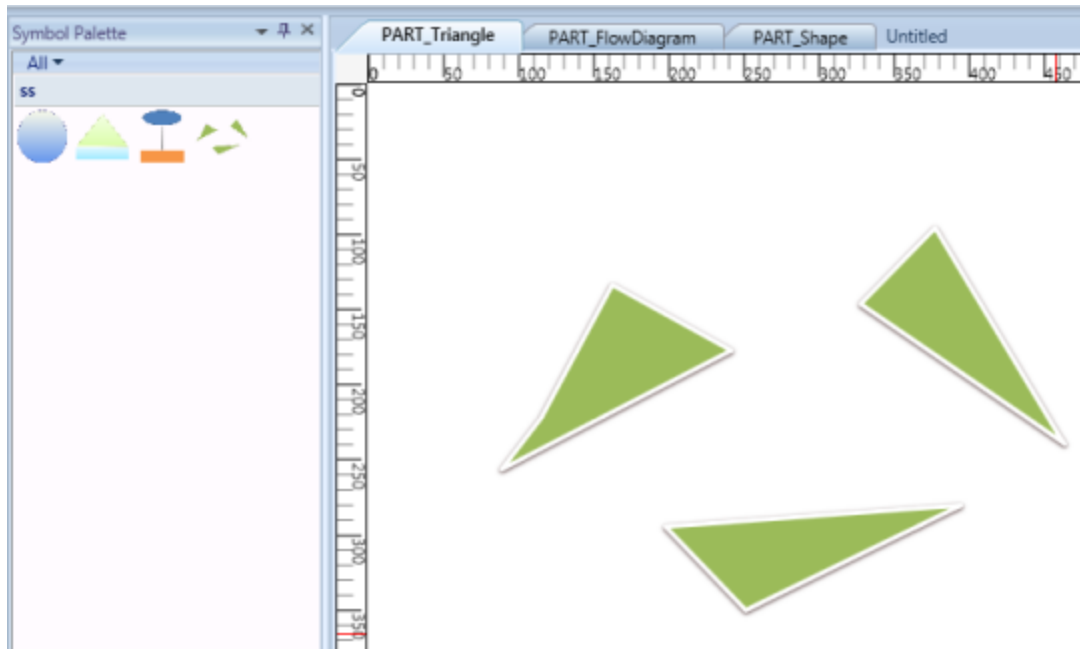
Save SymbolPalette Group Dialog

6. Give a relevant file name for the palette and click Save. As the PaletteGroup is saved, preview of the symbol will be shown in Symbol Palette.

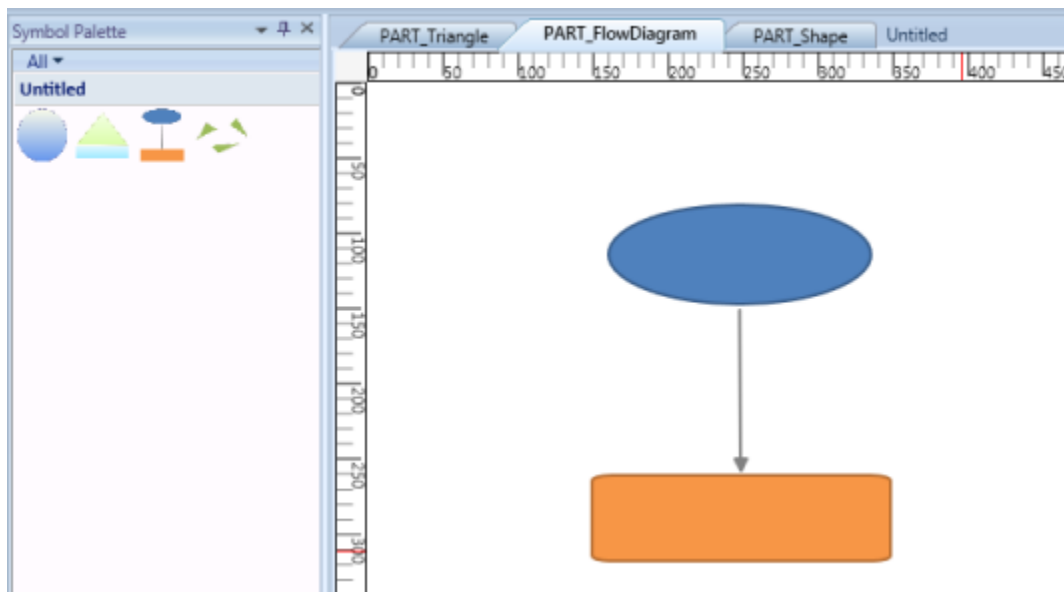


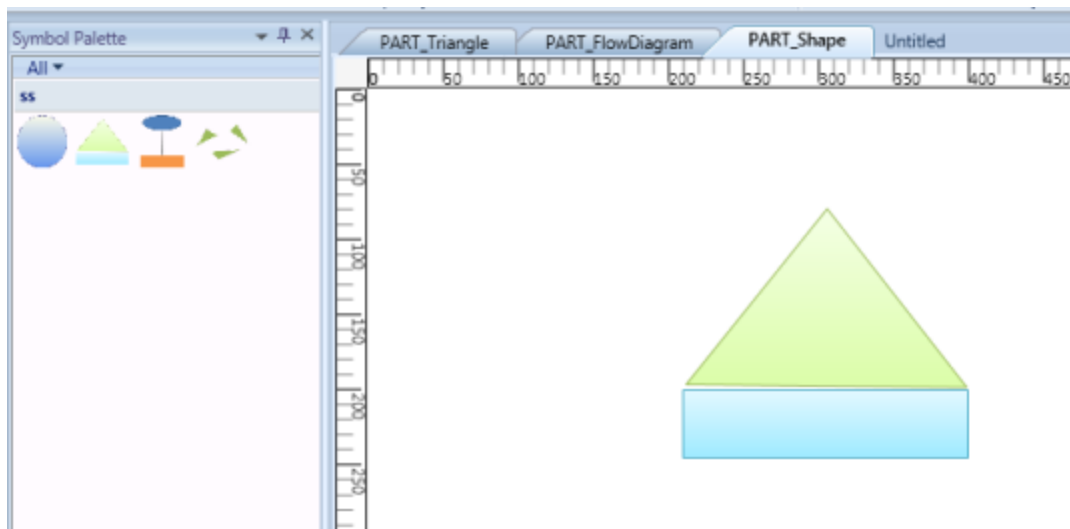
Symbol Palette With New Symbol

7. Repeat the steps 3 to 7 for creating more symbols.



Symbol pallet with PART_Triangle





Different Symbols

8. If you create symbols using more than one shape, you need to group all the shapes into a single symbol using the Group option in Symbol Designer.
9. Finally, Save the NewSymbol Group. Now the above symbols will be available in the SymbolPaletteGroup. The saved SymbolPaletteGroup can be loaded back later.

SymbolPalette in WPF Diagram (classic) Item

SymbolPaletteItems are contained in the SymbolPalette group. A SymbolPalette item does not restrict users to the type of content that can be added to it. A SymbolPalette item can be a text box, combo box, image, button, and so on.

The Name property of the SymbolPaletteItem can be used to refer to the custom item being added in the NodeDrop event. The name of the SymbolPaletteItem becomes the name of the node.

The following code example can be used to add a SymbolPalette item that has an image as its content.

C#

```
SymbolPaletteGroup group = new SymbolPaletteGroup();
group.Label = "Custom";
SymbolPalette.SetFilterIndexes(group, new Int32Collection(new int[] { 0, 6 }));
dc.SymbolPalette.SymbolGroups.Add(group);
SymbolPaletteItem item = new SymbolPaletteItem();
Image i = new Image();
BitmapImage bi3 = new BitmapImage();
bi3.BeginInit();
bi3.UriSource = new Uri("Custom.png", UriKind.RelativeOrAbsolute);
bi3.EndInit();
i.Stretch = Stretch.Fill;
i.Source = bi3;
item.Content = i;
group.Items.Add(item);
```

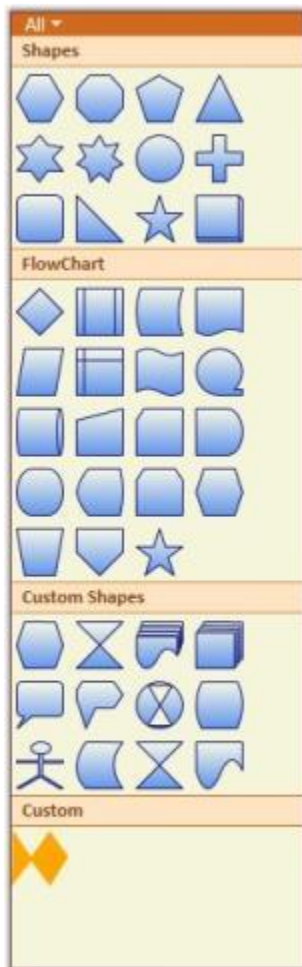
VB.NET

```

Dim group As New SymbolPaletteGroup()
group.Label = "Custom"
SymbolPalette.SetFilterIndexes(group, New Int32Collection(New Integer() { 0, 6 }))
dc.SymbolPalette.SymbolGroups.Add(group)
Dim item As New SymbolPaletteItem()
Dim i As New Image()
Dim bi3 As New BitmapImage()
bi3.BeginInit()
bi3.UriSource = New Uri("Custom.png", UriKind.RelativeOrAbsolute)
bi3.EndInit()
i.Stretch = Stretch.Fill
i.Source = bi3
item.Content = i
group.Items.Add(item)

```

This adds the image content to the newly created `SymbolPaletteItem` that belongs to the `SymbolPaletteGroup` named "Custom".



Custom Group and Item

Create `SymbolPaletteItem`

You can create `SymbolPaletteItem` in two methods. They are:

- Using Symbol Designer
- Using Microsoft Expression Blend 4

Using Symbol Design

To create SymbolPaletteltem using Symbol Designer, refer to the Symbol Designer section.

Using Microsoft Expression Blend 4

Essential Diagram provides an add-on for creating SymbolPaletteltem's content. You can also export the created content as SymbolPaletteltem using this add on. The exported content can be imported into the diagram control for later.

To enable this you need to add the Syncfusion.Diagram.SymbolExporter.dll in the Extension folder.

When Microsoft Expression Blend is installed before installing Essential Studio, the exporter assembly will automatically be placed in the Extensions folder.

When Microsoft Expression Blend is installed after installing Essential Studio, then you need to run the SyncfusionExpressionBlendAddins EXE, to place assembly in the correct location.

The SyncfusionExpressionBlendAddins EXE is available in the following location:

[{\$Essential Studio}][{\$Current Version}\Utilities\Diagram\WPF

Or you can also place the assembly manually in the following location:

{ \$Microsoft Expression } \Blend 4 \Extensions

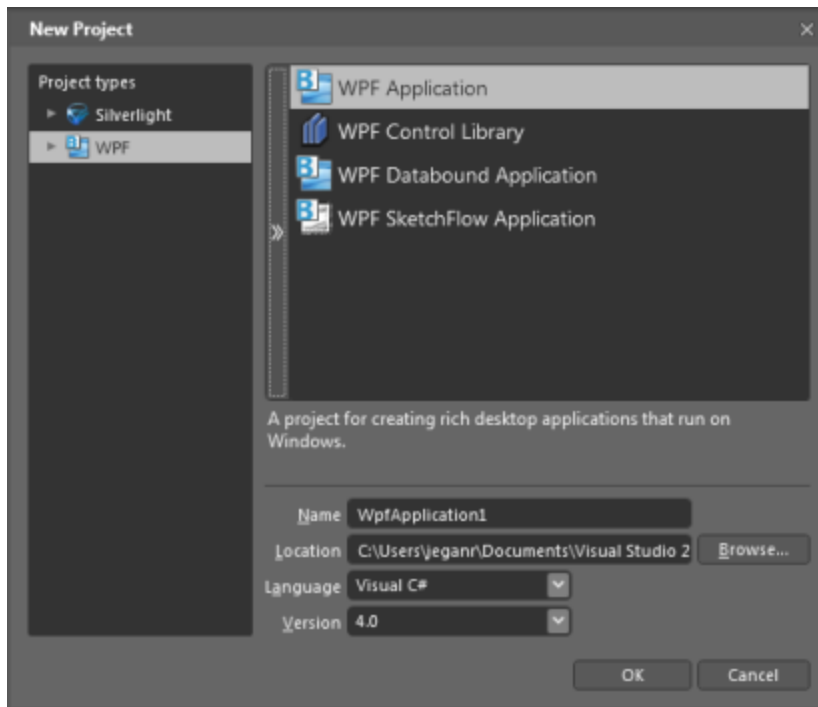
Note: currently exporter support is available only for the Blend 4 stable release(4.0.20525.0) and Blend 4 service packs 1 (version 4.0.20901.0).

To Create SymbolPaletteltem Content

1. Create a symbol that has to be exported.
2. Select the element that has to be exported as a SymbolPaletteltem.
3. The Name property has to be provided for the element that has to be exported as the symbol content.
4. The selected SymbolPalette content will be exported as a XAML file using the add-on.
5. The exported XAML file can be imported in the DiagramControl as SymbolPaletteltem.

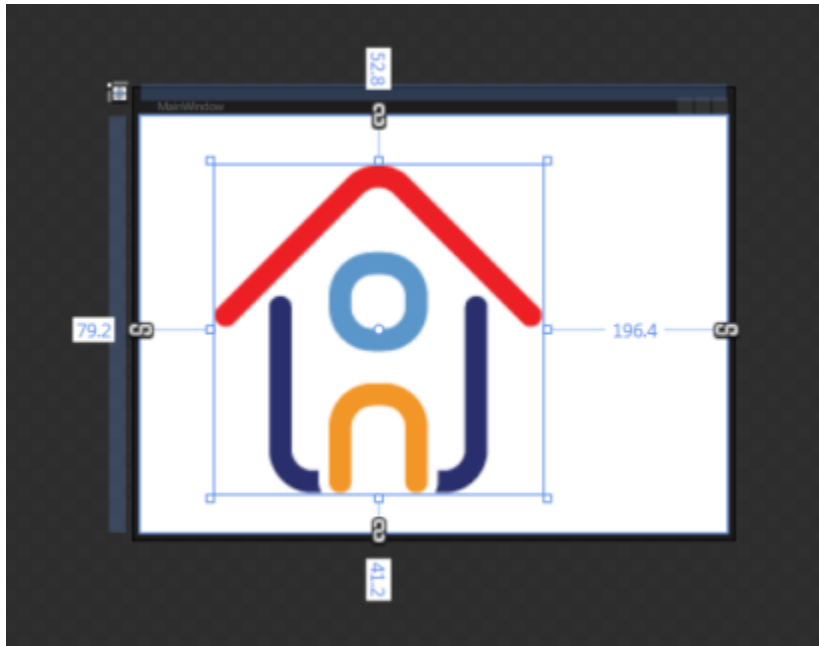
The following are the steps to create a SymbolPaletteltem and import symbols from Blend:

6. Create a WPF application.



New Project

7. Design the content as required.



Design Content

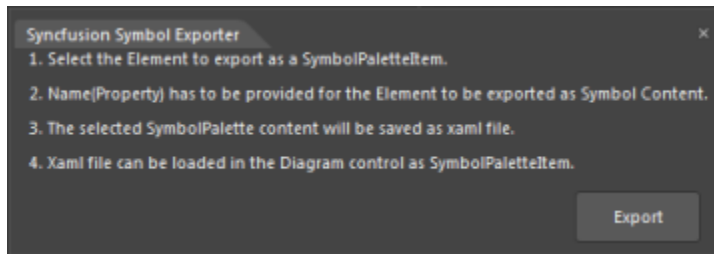
8. Select the part of the element, which you want to export as symbol content. You can also select this from the Object and Timeline.

Note: The selected part will act as the parent element, name the element as desired.

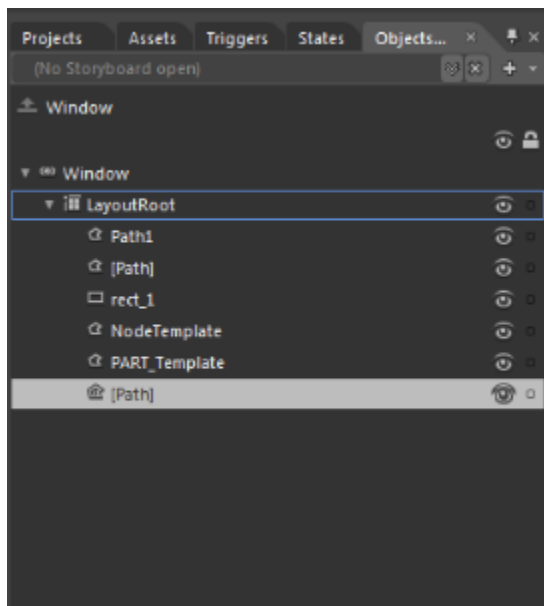
9. Specify the Name property for parent element.

This name property is required for serialization. So this property cannot be set to null.

10. Click Window option in the blend menu. 11. Select the Syncfusion Symbol Exporter add-on. 12. A popup window will be displayed. You can dock this within the blend as shown in following screenshot:

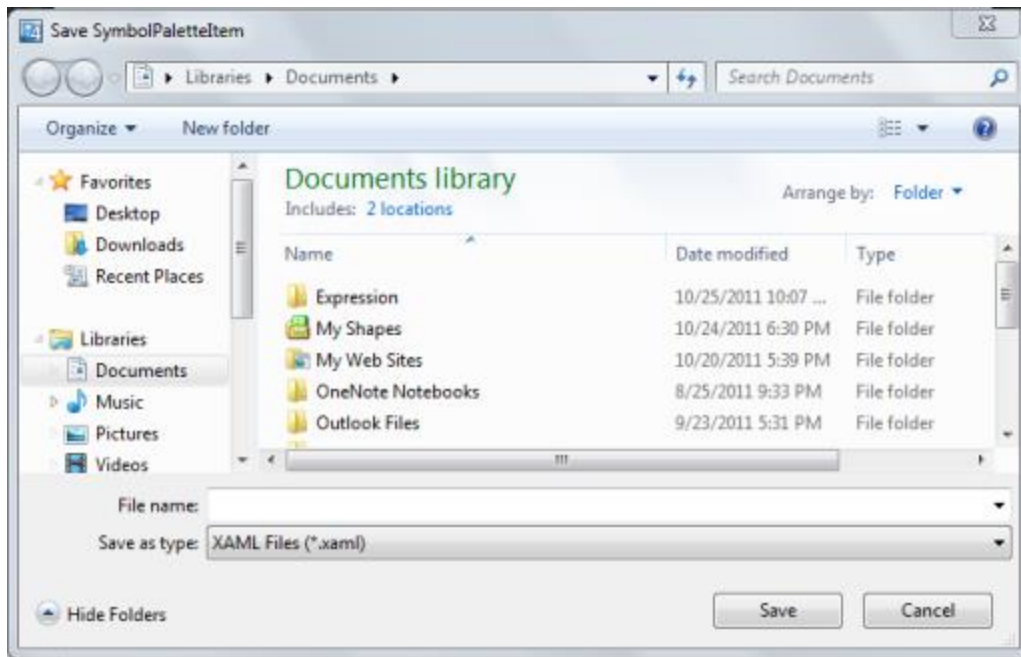


Syncfusion Symbol Exporter



Object and Timeline Window

13. Click Export.
14. The Save SymbolPaletteItem dialog opens.
15. Name the file and save this as XAML file in the desired location.



Save Dialog Box

16. The selected part will be exported as the SymbolPaletteItem content.

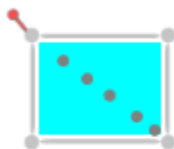
The exported file can be imported in SymbolPalette. To import symbols from XAML file, refer to the SymbolPaletteSerialization.

[Define Node, Port, Group definitions in SymbolPalette](#)

The following steps demonstrate how to specify a Node or Node with Ports in SymbolPaletteItem:

1. To add more than one port, a node is created.
2. Then several ports are added to it.
3. Create a SymbolPaletteItem and add the node as content for the SymbolPaletteItem.

At runtime, Nodes that are added in SymbolPalette can be dragged and dropped on the page. All the ports, and their properties will be cloned and a new copy of the node will be created.



Node with several Ports

To create a new Node

C#


```

Node node = new Node();
node.Width = 100;
node.Height = 100;
node.OffsetX = 200;
node.OffsetY = 200;
node.Background = new SolidColorBrush(Colors.Aqua);
AddMorePorts(node);
#### To add more Ports
private void AddMorePorts(Node node)
{
    Left=10;
    Top=10;
    for (int i = 0; i < 4; i++)
    {
        ConnectionPort port = new ConnectionPort();
        port.Left = Left;
        port.Top = Top;
        port.Node = node;
        node.Ports.Add(port);
        Left += 10;
        Top += 10;
    }
}

```

Creating Groups and SymbolPaletteItems

C#

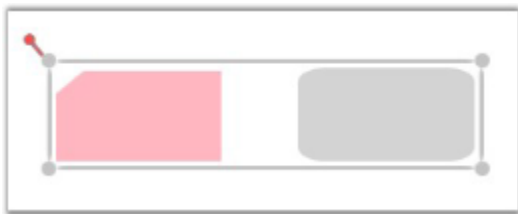
```

SymbolPaletteGroup group = new SymbolPaletteGroup();
group.Label = "Custom";
SymbolPalette.SetFilterIndexes(group, new List<int>() { 0, 6 });
dc.SymbolPalette.SymbolGroups.Add(group);
SymbolPaletteItem item = new SymbolPaletteItem();
// Node added as SymbolPaletteItem's Content
item.Content = node;
group.Items.Add(item);

```

Define Group definition in SymbolPalette

The groups can be given as SymbolPaletteItem's content. At runtime, they can be Dragged and Dropped to create a clone of the group and its children will be added on the page.



To Drag and Drop Groups

To create new Node and Groups

C#

```

public DiagramControl Control;
public DiagramModel Model;

```

```

public DiagramView View;
public Window1 ()
{
    Control = new DiagramControl ();
    Model = new DiagramModel ();
    View = new DiagramView ();
    Control.View = View;
    Control.Model = Model;
    View.Bounds = new Thickness(0, 0, 1000, 1000);
    Node n = new Node(Guid.NewGuid(), "Start");
    n.Shape = Shapes.FlowChart_Card;
    n.Level = 1;
    n.OffsetX = 150;
    n.OffsetY = 25;
    n.Width = 150;
    n.Height = 75;
    Node n1 = new Node(Guid.NewGuid(), "End");
    n1.Shape = Shapes.RoundedRectangle;
    n1.Level = 1;
    n1.OffsetX = 350;
    n1.OffsetY = 325;
    n1.Width = 100;
    n1.Height = 75;
    Model.Nodes.Add(n);
    Model.Nodes.Add(n1);
    Group g = new Group(Guid.NewGuid(), "group1");
    g.AddChild(n);
    g.AddChild(n1);
    Model.Nodes.Add(g);
}

```

To create new SymbolPalette Group and Item

C#

```

SymbolPaletteGroup group = new SymbolPaletteGroup();
group.Label = "Custom";
SymbolPalette.SetFilterIndexes(group, new List<int> ()[] { 0, 6 }));
dc.SymbolPalette.SymbolGroups.Add(group);
SymbolPaletteItem item = new SymbolPaletteItem();
// Group added as SymbolPaletteItem's Content
item.Content = g;
group.Items.Add(item);

```

Adding Through SymbolPalette

Customize the SymbolPalette

The appearance of the SymbolPalette can be customized to suit any application. Several properties have been provided in the SymbolPalette class to enable its customization.

The following properties can be used to customize the SymbolPalette in your application.

Property	Description	Type of the property	Value it accepts	Any other dependencies/
----------	-------------	----------------------	------------------	-------------------------

				sub properties associated
Background	Specifies the background color of the SymbolPalette. The default color is Beige.	Dependency property	Brush	No
BorderThickness	Gets or sets the border thickness of the SymbolPalette. The default value is 1.	Dependency property	Thickness	No
BorderBrush	Specifies the border color of the SymbolPalette. The default color is Brown.	Dependency property	Brush	No
SymbolPaletteGroupBackground	Specifies the background color of the SymbolPalette Group. The default color is Bisque.	Dependency property	Brush	No
SymbolPaletteGroupForeground	Specifies the foreground color of the SymbolPalette Group. The default color is SaddleBrown.	Dependency property	Brush	No
SymbolPaletteGroupBorderBrush	Specifies the border color of the SymbolPalette Group. The default color is Chocolate.	Dependency property	Brush	No
ItemBorderThickness	Gets or sets the border thickness of the SymbolPalette Item. The default value is 1.	Dependency property	Thickness	No

ItemCornerRadius	Gets or sets the corner radius of the SymbolPalette Item. The default value is 2.	Dependency property	CornerRadius	No
ItemMouseOverBorderBrush	Specifies the border color of the SymbolPalette Item over which the mouse pointer rests. The default value is Orange.	Dependency property	Brush	No
ItemCheckedBorderBrush	Specifies the border color of the SymbolPalette Item that is selected. The default value is Red.	Dependency property	Brush	No
ItemCheckedMouseOverBorderBrush	Specifies the border color of the selected SymbolPalette Item over which the mouse pointer rests. The default value is Green.	Dependency property	Brush	No
FilterSelectorBackground	Specifies the background color of the SymbolPalette Filter. The default value is Chocolate.	Dependency property	Brush	No
FilterSelectorForeground	Specifies the foreground color of the SymbolPalette Filter. The default value is DarkSlateGray.	Dependency property	Brush	No
FilterSelectorBorderThickness	Gets or sets the border thickness of the SymbolPalette	Dependency property	Thickness	No

	Filter.The default value is (0,0,0,1).			
FilterSelectorMouseOverForeground	Specifies the foreground color of the SymbolPalette Filter over which the mouse pointer rests.The default value is OldLace.	Dependency property	Brush	No
FilterSelectorBorderBrush	specifies the border color of the SymbolPalette FilterThe default value is Chocolate.	Dependency property	Brush	No
PopUpBackground	Specifies the background color of the SymbolPalette Pop-up.The default value is WhiteSmoke.	Dependency property	Brush	No
PopUpForeground	Specifies the foreground color of the SymbolPalette Pop-up.The default value is DarkSlateGray.	Dependency property	Brush	No
PopUpBorderThickness	Gets or sets the border thickness of the SymbolPalette Pop-upThe default value is (0,1,1,1).	Dependency property	Thickness	No
PopUpMouseOverBrush	Specifies the background color of the SymbolPalette pop-up Item over which the mouse pointer rests.The	Dependency property	Brush	No

	default value is LightSalmon.			
PopUpBorderBrush	Specifies the border color of the SymbolPalette pop-up.The default value is Chocolate.	Dependency property	Brush	No
PopUpLeftColumnBackground	Specifies the background color of the Check Box Column in the SymbolPalette pop-up.The default value is LightGray.	Dependency property	Brush	No
CheckerBackground	Specifies the the background color of the Check Boxes in the SymbolPalette pop-up.The default value is Bisque.	Dependency property	Brush	No
CheckerBorderBrush	Specifies the the border color of the Check Boxes in the SymbolPalette pop-up.The default value is DarkSlateGray.	Dependency property	Brush	No
CheckerTickBrush	Specifies the Tick color of the selected Check Box in the SymbolPalette pop-up.The default value is DarkSlateGray.	Dependency property	Brush	No

The following code example illustrates how to set some of the SymbolPalette properties.

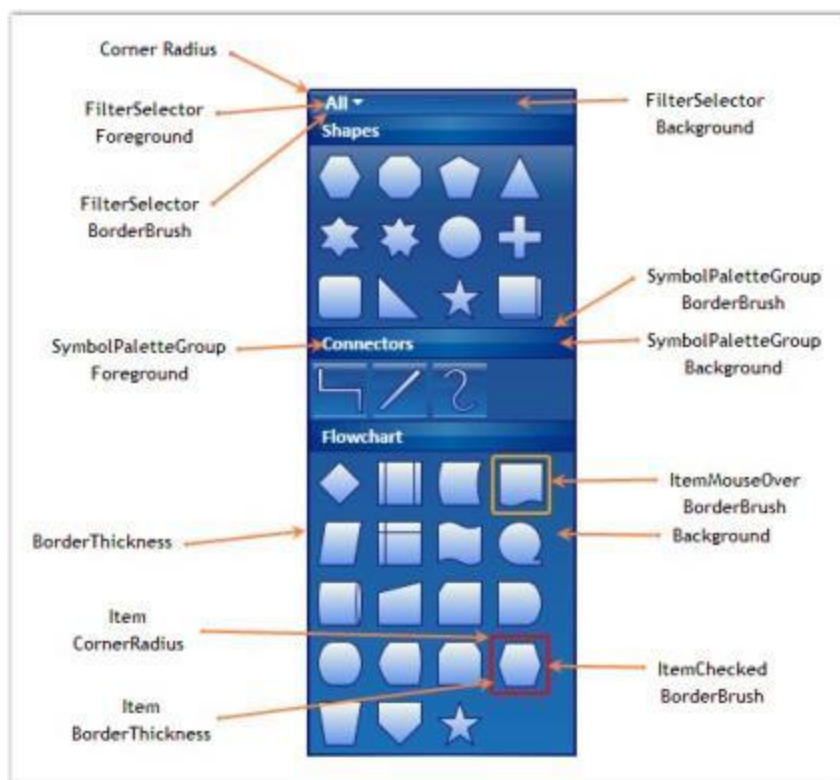
C#

```
DiagramControl diagramControl = new DiagramControl();
diagramControl.SymbolPalette.BorderThickness = new Thickness(2);
diagramControl.SymbolPalette.BorderBrush = Brushes.MidnightBlue;
diagramControl.SymbolPalette.Background = Brushes.Blue;
diagramControl.SymbolPalette.SymbolPaletteGroupBackground =
Brushes.DarkBlue; diagramControl.SymbolPalette.SymbolPaletteGroupForeground
= Brushes.White;
diagramControl.SymbolPalette.SymbolPaletteGroupBorderBrush =
Brushes.SlateBlue;
diagramControl.SymbolPalette.FilterSelectorBackground = Brushes.SkyBlue;
diagramControl.SymbolPalette.FilterSelectorForeground = Brushes.White;
diagramControl.SymbolPalette.FilterSelectorBorderBrush = Brushes.Blue;
diagramControl.SymbolPalette.FilterSelectorBorderThickness = new
Thickness(0);
diagramControl.SymbolPalette.CheckerTickBrush = Brushes.White;
diagramControl.SymbolPalette.CheckerBorderBrush = Brushes.MidnightBlue;
diagramControl.SymbolPalette.CheckerBackground = Brushes.LightBlue;
diagramControl.SymbolPalette.PopUpItemMouseOverBrush =
Brushes.CornflowerBlue;
diagramControl.SymbolPalette.PopUpBorderBrush = Brushes.MidnightBlue;
diagramControl.SymbolPalette.ItemBorderThickness = new Thickness(2);
```

VB.NET

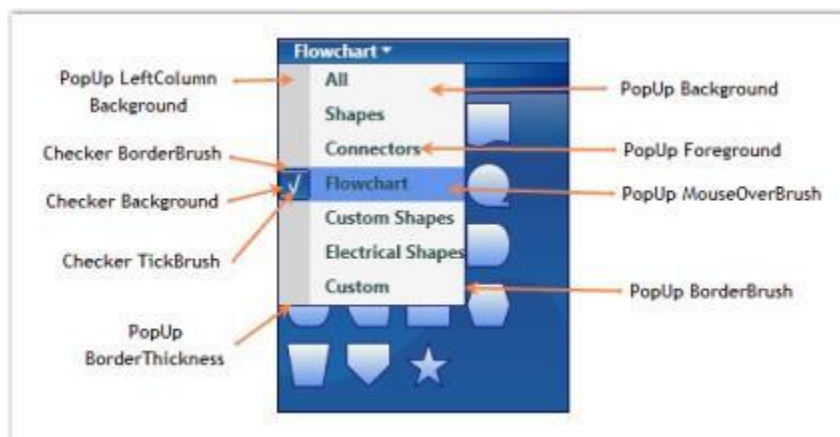
```
Dim diagramControl As New DiagramControl()
diagramControl.SymbolPalette.BorderThickness = New Thickness(2)
diagramControl.SymbolPalette.BorderBrush = Brushes.MidnightBlue
diagramControl.SymbolPalette.Background = Brushes.Blue
diagramControl.SymbolPalette.SymbolPaletteGroupBackground = Brushes.DarkBlue
diagramControl.SymbolPalette.SymbolPaletteGroupForeground = Brushes.White
diagramControl.SymbolPalette.SymbolPaletteGroupBorderBrush =
Brushes.SlateBlue
diagramControl.SymbolPalette.FilterSelectorBackground = Brushes.SkyBlue
diagramControl.SymbolPalette.FilterSelectorForeground = Brushes.White
diagramControl.SymbolPalette.FilterSelectorBorderBrush = Brushes.Blue
diagramControl.SymbolPalette.FilterSelectorBorderThickness = New
Thickness(0)
diagramControl.SymbolPalette.CheckerTickBrush = Brushes.White
diagramControl.SymbolPalette.CheckerBorderBrush = Brushes.MidnightBlue
diagramControl.SymbolPalette.CheckerBackground = Brushes.LightBlue
diagramControl.SymbolPalette.PopUpItemMouseOverBrush =
Brushes.CornflowerBlue
diagramControl.SymbolPalette.PopUpBorderBrush = Brushes.MidnightBlue
diagramControl.SymbolPalette.ItemBorderThickness = New Thickness(2)
```

The following screenshot illustrates the various customization options that are available for the SymbolPalette Item, Group and Filter Selector.



SymbolPalette Item, Group and Filter Selector Customization Properties

The following screenshot illustrates the various customization options available for the SymbolPalette PopUp.



SymbolPalette PopUp Customization Properties

[Customize the SymbolPaletteItem](#)

This feature improves the performance of SymbolPaletteItem dragging, by avoiding serialization and deserialization of the item.

Method	Description	Parameters	Return Type
--------	-------------	------------	-------------

CloneContent	This method assigns the customized and serialized content to the symbol palette item, while performing drag and drop.	No parameters	object
--------------	---	---------------	--------

Adding this Feature to an Application

Creating Custom SymbolPalettetItem

The content for the custom SymbolPalettetItem is assigned by overriding the CloneContent method.

C#

```
public class customItem : SymbolPaletteItem
{
    public customItem()
    { }
    public override object CloneContent()
    {
        return new Button() { Height = 50, Width = 50, Content = "Custom" };
    }
}
```

VB.NET

```
Public Class customItem
    Inherits SymbolPaletteItem
    Public Sub New()
    End Sub
    Public Overrides Function CloneContent() As Object
    Dim button As New Button()
    button.Content = "asas"
    button.Width = 50
    button.Height = 50
    Return button
    End Function
End Class
```

Adding CustomItem to SymbolPalette

To add CustomItem to SymbolPalette, use the code example given below:

C#

```
customItem custom = new customItem();
custom.Content = "asd";
(diagramControl.SymbolPalette.SymbolGroups[0] as
SymbolPaletteGroup).Items.Add(custom);
```

VB.NET

```
Dim custom As New customItem()
custom.Content = "asd"
TryCast(customs.SymbolGroups(0), SymbolPaletteGroup).Items.Add(custom)
```

When you drag and drop the custom SymbolPaletteltem, the content that is assigned in the CloneContent method is applied to the content of the dropped item.

Thereby, the performance of dragging the SymbolPaletteltem is improved by 40%. If the default base method is returned in the CloneContent method, the default content is dragged and dropped.

SymbolPalette in WPF Diagram (classic) Serialization

Serialization is the process of saving and retrieving the SymbolPalette groups and items. Essential DiagramWPF supports saving the SymbolPalette as an XAML file. This load and save feature allows you to save the SymbolPalette for future use. You can continue working on their page by loading the appropriate XAML file.

SymbolPaletteSerialization feature provides an option to save and load the SymbolPalette, SymbolPalette groups, elements and items in diagram control. So any item can be customized and imported onto the SymbolPalette.

- User can easily Save/Load the SymbolPalette
- User can Save/Load the SymbolPaletteGroup
- User can Save/Load the SymbolPaletteltem

Methods

Method	Description	Parameters	Return Type	Reference links
SaveSymbolPalette	Displays the save dialog box to save the entire SymbolPalette(including all SymbolPalette groups) into XAML file.	NA	Void	NA
LoadSymbolPalette	The existing SymbolPalette groups will be cleared and new groups will be added from selected XAML file.	NA	Void	NA
SaveSymbolPaletteGroup	Saves the Symbol Palette Group into XAML file using the given SymbolPaletteGroup parameter.	SymbolPaletteGroup	Void	NA
LoadSymbolPaletteGroup	Displays the Load Dialog Box to load the Symbol Palette Group from the selected XAML file.	NA	Void	NA
SaveSymbolPaletteltem	Saves the Symbol Palette Item into XAML file using the given	SymbolPaletteltem	Void	NA

	SymbolPalettItem parameter.			
LoadSymbolPalettItem	Loads the SymbolPalette Item from the XAML file. The Items are loaded in any given Symbol Palette Group using the SymbolPaletteGroup parameter.	SymbolPaletteGroup	Void	NA

Bind to ItemSource

The symbol palette supports binding with business objects. A symbol group will be created and added to the symbol palette depending upon the business objects and template provided. The symbol group itself supports binding with business objects. Symbol palette items will be created and added to it depending upon the business objects and template.

Properties

Property	Description	Type	Data Type	Reference links
ItemGenerateMode	Specifies the ItemGenerate Mode for symbol palette. Default is Manual.	Dependency property	ItemGenerateMode.ItemsSourceItemGenerateMode.Manual	No
ItemsSource	Gets or sets the source for the list of the items, the containers about to represent.	DependencyProperty	IEnumerable	No

ItemSource property gets the source for the list of Symbol groups to be added to symbol palette.

The following code samples illustrate this.

HTML

```
<Window.Resources>
<Style TargetType="syncfusion:SymbolPaletteGroup">
<Setter Property="Label" Value="{Binding Label}"></Setter>
<Setter Property="syncfusion:SymbolPalette.FilterIndexes" Value="{Binding Filter}">
```

```

</Setter>
<Setter Property="ItemTemplate">
<Setter.Value>
<DataTemplate>
<syncfusion:SymbolPaletteItem Height="50" Width="50" Content="{Binding
Content}">
</syncfusion:SymbolPaletteItem>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>

```

C#

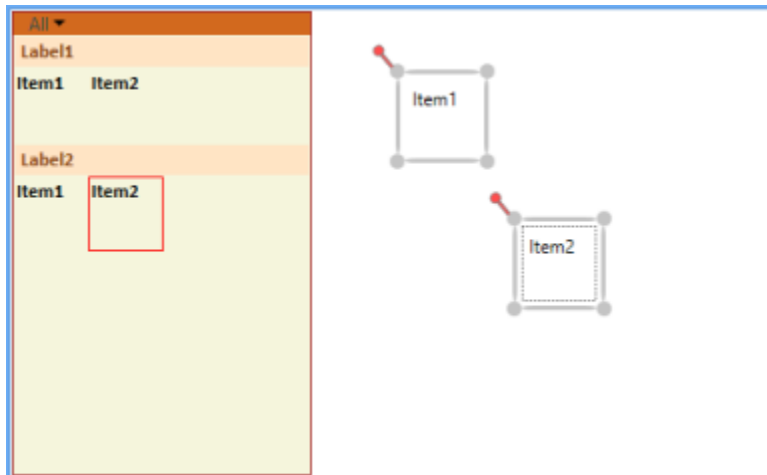
```

symbol = diagramControl.SymbolPalette;
// Set the ItemGenerateMode property value to ItemsSource
symbol.ItemGenerateMode = ItemGenerateMode.ItemsSource;
//Clear the default filter item from the SymbolFilter collections.
symbol.SymbolFilters.Clear();
//Clear default items added in SymbolGroups
symbol.SymbolGroups.Clear();
//Then assign ItemsSource for the SymbolPalette
symbol.ItemsSource = CreatingBusinessObjForSymbolPalette();
//ItemsSource for SymbolPalette
private System.Collections.IEnumerable CreatingBusinessObjForSymbolPalette()
{
ObservableCollection<SymbGroup> grp = new ObservableCollection<SymbGroup>();
grp.Add(new SymbGroup() { Label = "Label1", Filter = new Int32Collection(new
int[] { 0, 1 }) });
grp.Add(new SymbGroup() { Label = "Label2", Filter = new Int32Collection(new
int[] { 0, 2 }) });
return grp;
}
//ItemsSource for SymbolPaletteGroup assigned when group collection changed.
void SymbolGroups_CollectionChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
if (e.NewItems != null)
{
ObservableCollection<SymbolItem> symbolItemCollection = new
ObservableCollection<SymbolItem>();
symbolItemCollection.Add(new SymbolItem() { Content = "Item1" });
symbolItemCollection.Add(new SymbolItem() { Content = "Item2" });
(e.NewItems[0] as SymbolPaletteGroup).ItemsSource = symbolItemCollection;
}
}
//Business object for symbol group and symbol palette item.
public class SymbGroup
{
public String Label { get; set; }
public Int32Collection Filter { get; set; }
}
public class SymbolItem
{
public string Content { get; set; }
}

```

```
}

```



Symbol palette ItemSource

Note: Symbol groups are not allowed to be added manually when ItemSource is used.

Serialization in WPF Diagram (classic)

Serialization is the process of saving and retrieving the Essential Diagram file. Essential Diagram WPF supports saving the diagram page as an XAML file. The page and all its properties get saved. On loading, the page gets loaded in the current view with all its nodes and connections. This load and save feature allows you to save their diagram page for future use. You can continue working on their page by loading the appropriate XAML file.

Name	Parameters	Return Type	Description	Reference Links
Save()	Null	Void	Displays the Save Dialog Box to save the DiagramPage into XAML file.	Save Diagram Page
Save(string)	String	Void	Saves the DiagramPage into XAML file whose file name is specified.	Save Diagram Page
Save(Stream)	System.IO.Stream	Void	Saves the DiagramPage into memory stream.	Save Diagram Page
Load()	Null	Void	Displays the Load Dialog Box to load the DiagramPage from selected XAML file.	Load Diagram Page
Load(string)	String	Void	Loads the DiagramPage from the file name mentioned.	Load Diagram Page
Load(Stream)	System.IO.Stream	Void	Loads the DiagramPage from the memory stream.	Load Diagram Page

This process is explained in the following topic:

[Save Diagram Page](#)

Save operation can be done in three ways,

- Using the Save Dialog Box.
- File name with full path.
- Using Memory Stream

Using the Save Dialog Box

To save the page, the following code can be used.

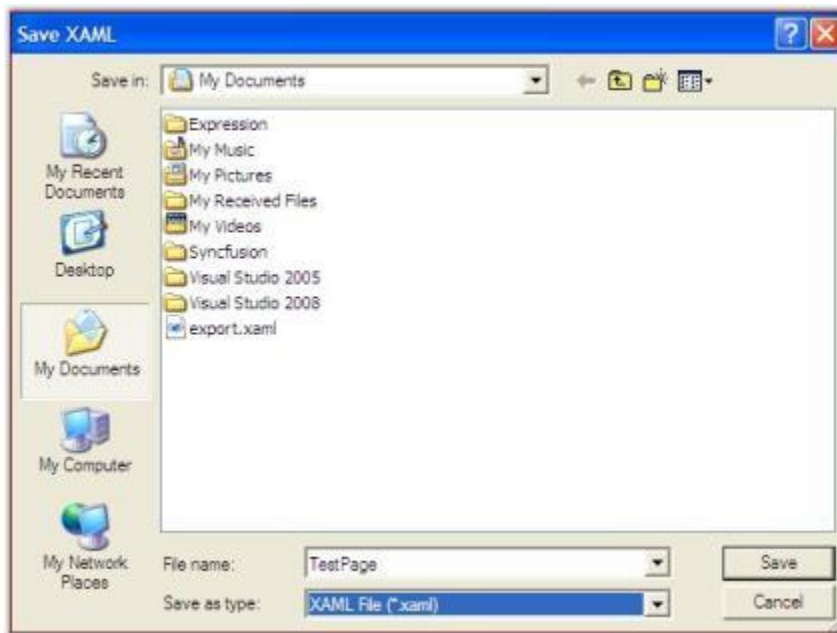
C#

```
DiagramControl dc = new DiagramControl();  
dc.Save();
```

VB.NET

```
Dim dc As New DiagramControl()  
dc.Save()
```

The Save Dialog box will appear. Select the 'Save as Type' as XAML and select the location at which the file is to be saved and click the save button in the dialog box after specifying a name for the file.



Save Dialog Box

File name with path

You can also specify the name of the file directly in the Save method.

C#

```
DiagramControl dc = new DiagramControl();
```

```
dc.Save(@"C:\TestPage.xaml");
```

VB.NET

```
Dim dc As New DiagramControl()  
dc.Save("C:\TestPage.xaml")
```

Note: Essential Diagram WPF does not support serializing bindings and bitmap Images.

Saving to a stream

You can also save to a stream.

The following code example shows how it can be done.

C#

```
DiagramControl dc = new DiagramControl();  
System.IO.MemoryStream stream = new System.IO.MemoryStream();  
dc.Save(stream as System.IO.Stream);
```

VB.NET

```
Dim dc As New DiagramControl()  
Dim stream As New System.IO.MemoryStream()  
dc.Save(TryCast(stream, System.IO.Stream))
```

Loading the Diagram Page

Load operation can be done in three ways,

- Using the Load Dialog Box.
- File name with full path.
- Using Memory Stream

Load using the Load Dialog Box

To load the page, the following code can be used.

C#

```
DiagramControl dc = new DiagramControl();  
dc.Load();
```

VB.NET

```
Dim dc As New DiagramControl()  
dc.Load()
```

The Load Dialog box will appear. Select the 'Files of Type' as XAML and specify the path of the file to be loaded and click the Open button in the dialog box. The selected page gets loaded in the current view and the page is ready to be edited.



Load Dialog Box

File name with path

You can also specify the name of the file directly in the Load method.

C#

```
DiagramControl dc = new DiagramControl();  
dc.Load(@"C:\TestPage.xaml");
```

VB.NET

```
Dim dc As New DiagramControl()  
dc.Load("C:\TestPage.xaml")
```

Note: Essential Diagram WPF does not support serializing bindings and bitmap Images.

Loading from a stream

You can also load from a stream.

To load from the stream use the following code snippet.

C#

```
stream.Position = 0;  
dc.Load(stream as System.IO.Stream);
```

VB.NET

```
stream.Position = 0  
dc.Load(TryCast(stream, System.IO.Stream))
```

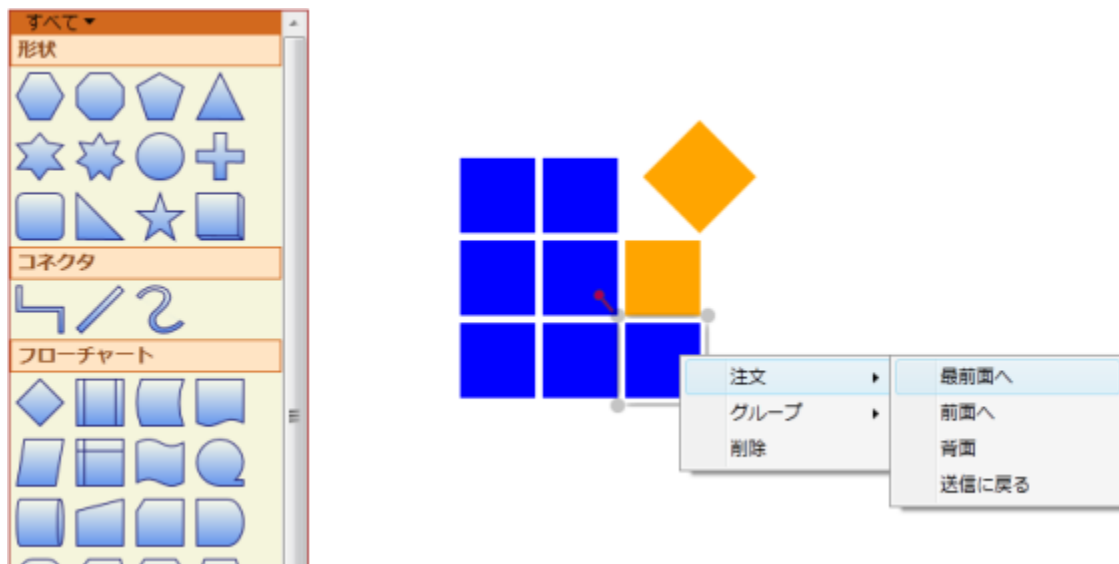
Note: While loading from memory stream please make sure the stream's Position property is set to 0.

Localization

Localization is the process of providing controls in different cultures to help users to easily set their own culture.

Use Case Scenarios

Localization is the process of customizing the User Interface (UI) in a language and culture specific to a particular country or region to display regional data. Localization is the key feature that provides solutions to global customers with the help of localized resource files provided by controls.



Localization Sample in Japanese Language

Localizing the Application

Adding Resource Files

To localize the Syncfusion Diagram WPF control, you need to create a resource file for each culture. The following steps should be performed when localizing strings for your culture:

1. Add the resource (.resx) files in the Resources folder for different cultures. The .resx files for the different cultures or invariant cultures should be placed in the Resources folder of your project.
2. Name the resource files according to the formats specified, namely `AssemblyName.CultureName.resx` and `AssemblyName.resx` for the invariant cultures. Here, `AssemblyName` is the Syncfusion WPF control assembly name and `CultureName` is the culture code of the resource file that you want to show in the UI. If your conversion is only for the invariant culture, then the .resx file does not require a culture suffix.

Examples

`Syncfusion.Diagram.Wpf.ja.resx` - A Japanese resource file for the `Syncfusion.Diagram.Wpf` assembly.

`Syncfusion.Diagram.Wpf.resx` - An invariant culture resource file for the `Syncfusion.Diagram.Wpf` assembly.

Assigning the Current UI Culture to the Application

By default, the current culture is set to “en-US”. You can check the current culture from “System.Threading.Thread.CurrentThread.CurrentUICulture”. CurrentUICulture can be changed, as shown in the following code snippets.

In the following example, CurrentUICulture is set before InitializeComponent in the StartUp page (MainPage.xaml.cs).

C#

```
public MainPage()
{
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("ja");
    InitializeComponent();
}
```

VB.NET

```
'INSTANT VB WARNING: The following constructor is declared outside of its
associated class:
'ORIGINAL LINE: public MainPage()
Public Sub New()
    System.Threading.Thread.CurrentThread.CurrentUICulture = New
    System.Globalization.CultureInfo("ja")
    InitializeComponent()
End Sub
```

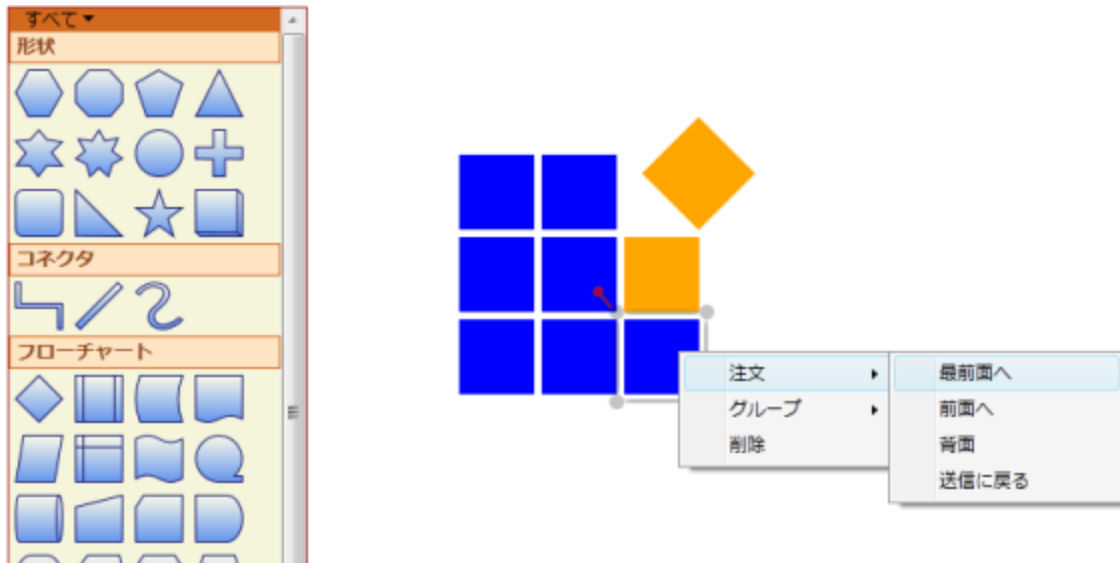
Else, CurrentUICulture is set in the Application_Startup event in the App.xaml.cs file, as shown in the following example.

C#

```
private void Application_Startup(object sender, StartupEventArgs e)
{
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("ja");
    this.RootVisual = new MainPage();
}
```

VB.NET

```
Private Sub Application_Startup(ByVal sender As Object, ByVal e As
StartupEventArgs)
    System.Threading.Thread.CurrentThread.CurrentUICulture = New
    System.Globalization.CultureInfo("ja")
    Me.RootVisual = New MainPage()
End Sub
```



Localization Sample in Japanese Language

Specifying the Directory Location of the Resource File

By default, the resource file for a specific culture is obtained from the Resources directory. However, the location of the resource file can be changed by using DiagramControl's LocalizationPath property, as shown in the following code snippet.

C#

```
// The location of the localized resource file is stored in the
// \Resources\Controls directory.
diagramControl.LocalizationPath = "Resources.Controls";
```

VB.NET

```
'The location of the localized resource file is stored in the
'\Resources\Controls directory.
diagramControl.LocalizationPath = "Resources.Controls"
```



Customized LocalizationPath

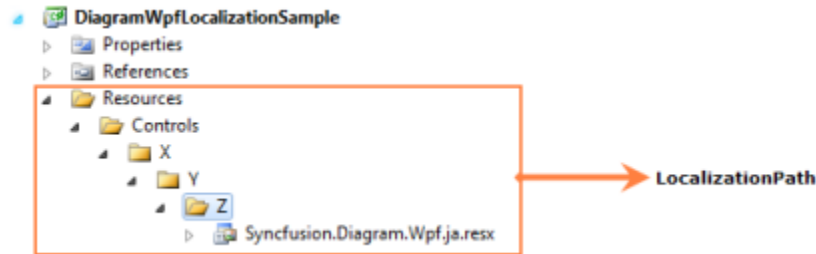
Example

C#

```
// The location of the localized resource file is stored in the
// \Resources\X\Y\Z directory.
diagramControl.LocalizationPath = "Resources.X.Y.Z";
```

VB.NET

```
'The location of the localized resource file is stored in the
\Resources\X\Y\Z directory.
diagramControl.LocalizationPath = "Resources.X.Y.Z"
```

*Customized LocalizationPath*

Note: LocalizationPath for the resource file should be specified before DiagramControl's Template is applied.

Properties

The property of the Localization feature is described in the following tabulation:

Property	Description	Type	Data Type	Reference links
LocalizationPath	Indicates the directory in which the resource files are located.	CLR	string	Not applicable

Sample Link

To view a sample

1. Open the Diagram Sample Browser from the dashboard. (Refer to the Samples and Location chapter.)
2. Navigate to Editable Diagram -> Localization Demo.

Event Mechanism in WPF Diagram (classic)

This section describes several events triggered and handled while using Essential Diagram WPF in the following topic:

Events for Nodes and Connections

Diagram control has several events which respond to several actions performed on nodes and connections.

The various events and their descriptions are explained in the following table.

Event	Description	Arguments
NodeClick	Raised when the node is clicked. Event cannot be canceled.	Node – Node on which event is raised.

NodeDoubleClick	Raised when the node is clicked twice in succession. Event cannot be canceled.	Node â€œ Node on which event is raised.
NodeStartLabelEdit	Raised when the label editing on the node is started. Event cannot be canceled.	NewLabelValue â€œ The new label value.OldLabelValue â€œ the old label value.Node - Node on which event is raised.
NodeLabelChanged	Raised when the node's label value is changed. Event cannot be canceled.	NewLabelValue â€œ The new label value.OldLabelValue â€œ the old label value.Node - Node on which event is raised.
NodeDragStart	Raised when the node is dragged. Event cannot be canceled.	Node â€œ Node on which event is raised.
NodeDragEnd	Raised when the drag operation on node is complete.Event cannot be canceled.	Node â€œ Node on which event is raised.
NodeResizing	Raised when the resize operation is being performed. Event cannot be canceled.	Node â€œ Node on which event is raised.
NodeResized	Raised after the node is resized. Event cannot be canceled.	Node â€œ Node on which event is raised.
NodeRotationChanging	Raised when the node is being rotated. Event cannot be canceled.	Node â€œ Node on which event is raised.
NodeRotationChanged	Raised after the node is rotated. Event cannot be canceled.	Node â€œ Node on which event is raised.
ConnectorDoubleClick	Raised when the Connector is clicked twice in succession. Event cannot be canceled.	Connector â€œ Connector on which the event is raised.Head Node â€œ Head Node of the connector.Tail Node â€œ Tail Node of the connector.
ConnectorStartLabelEdit	Raised when the label editing on the Connector is started. Event cannot be canceled.	Connector â€œ Connector on which the event is raised.Head Node â€œ Head Node of the connector.Tail Node â€œ Tail Node of the connector.OldLabelValue â€œ the old label value.

ConnectorLabelChanged	Raised when the connector's label value is changed. Event cannot be canceled.	Connector â€œ Connector on which the event is raised.Head Node â€œ Head Node of the connector.Tail Node â€œ Tail Node of the connector.OldLabelValue â€œ the old label value.NewLabelValue â€œ The new label value.
ConnectorDragStart	Raised when either ends of the connector is dragged. Event cannot be canceled.	Connector â€œ Connector on which the event is raised.FixedNodeEnd â€œ Node on which the connection is fixed.MovableNodeEnd â€œ The old Node on which the Connector was connected.
ConnectorDragEnd	Raised when the drag operation is complete. Event cannot be canceled.	Connector â€œ Connector on which the event is raised.FixedNodeEnd â€œ Node on which the connection is fixed.HitNodeEnd â€œ The new node on which the Connector is getting connected.
NodeDrop	Raised when a shape from the SymbolPalette is dropped on the page. Event cannot be canceled.	DroppedNode â€œ The new node just dropped from SymbolPalette.SymbolPalettetItemName â€œ The name of the SymbolPalette item, which is dropped on the page.
HeadNodeChanged	Raised when the HeadNode of the connector is changed Event cannot be canceled.	Connector â€œ The connector whose HeadNode is changed.PreviousNode â€œ The old Node on which the HeadNode of the Connector was connector.CurrentNode - The new Node on which the HeadNode of the Connector is connector.
TailNodeChanged	Raised when the TailNode of the connector is changed. Event cannot be canceled.	Connector â€œ The connector whose HeadNode is changed.PreviousNode â€œ The old Node on which the TailNode of the Connector was connector.CurrentNode - The new Node on which the TailNode of the Connector is connector.
ConnectorDrop	Raised when the connector is dropped on the page. Event cannot be canceled.	DroppedConnector â€œ Connector on which the event is raised.

BeforeConnectionCreate	Raised when a new connection is being made. Event cannot be canceled.	Connector â€œ The connector whose HeadNode is changed.
AfterConnectionCreate	Raised after the connection has been made. Event cannot be canceled.	Connector â€œ Connector on which the event is raised.FixedNodeEnd â€œ Node on which the connection is fixed.HitNodeEnd â€œ The new node on which the Connector is getting connected.
NodeSelected	Raised when a node is selected. Event cannot be canceled.	Node â€œ Node on which event is raised.
NodeUnSelected	Raised when a node is not selected. Event cannot be canceled.	Node â€œ Node on which event is raised.
NodeDeleting	Raised before a node is deleted from the model. Event cannot be canceled.	DeletedNode â€œ Node which is going to get deleted.
NodeDeleted	Raised when a node is deleted from the model. Event cannot be canceled.	DeletedNode â€œ Node which is deleted.
ConnectorDeleting	Raised before a line connector is deleted from the model. Event cannot be canceled.	DeletedLineConnector â€œ LineConnector which is getting deleted.
ConnectorDeleted	Raised when a line connector is deleted from the model. Event cannot be canceled.	DeletedLineConnector â€œ LineConnector which is deleted.
PreviewNodeDrop	Raised before a node is dropped on the page. Event cannot be canceled.	Node â€œ Node on which event is raised.
PreviewConnectorDrop	Raised before a line connector is dropped on the page. Event cannot be canceled.	Connector â€œ Connector on which the event is raised.
NodeMoved(event is fired before nudge operation is completed)	Raised when the nudge operation on node is completed. Event cannot be canceled.	Node â€œ Node on which event is raised.oldOffset â€œ The old offset value before nudge operation.newOffset â€œ The new offset value after performing nudge operation.

The events can be specified using DiagramView object as follows.

- For instance, NodeClick event can be specified in the following way.

HTML

```
<sfDiagram:DiagramView Name="diagramView" NodeClick="diagramView_NodeClick">
</sfDiagram:DiagramView>
```

C#

```
diagramView.NodeClick += new NodeEventHandler(diagramView_NodeClick);
```

VB.NET

```
AddHandler diagramView.NodeClick, AddressOf diagramView_NodeClick
```

- And then the event handler can be specified in the code behind as follows.

C#

```
//Event Handler
void diagramView_NodeClick(object sender, NodeRoutedEventArgs evtArgs)
{
    //user specified code
}
```

VB.NET

```
'Event Handler
Private Sub diagramView_NodeClick(ByVal sender As Object, ByVal evtArgs As
NodeRoutedEventArgs)
    'user specified code
End Sub
```

- As another example, the ConnectorDoubleClick event can be specified in the following way.

HTML

```
<sfDiagram:DiagramView Name="diagramView"
ConnectorDoubleClick="diagramView_ConnectorDoubleClick">
</sfDiagram:DiagramView>
```

C#

```
diagramView.ConnectorDoubleClick += new
ConnChangedEventHandler(diagramView_ConnectorDoubleClick);
```

VB.NET


```
AddHandler diagramView.ConnectorDoubleClick, AddressOf
diagramView.ConnectorDoubleClick
```

And then the event handler can be specified in the code behind as follows.

C#

```
// Event Handler
void diagramView_ConnectorDoubleClick(object sender, ConnRoutedEventArgs
evtArgs)
{
    // user specified code
}
```

VB.NET

```
'Event Handler
Private Sub diagramView_ConnectorDoubleClick(ByVal sender As Object, ByVal
evtArgs As ConnRoutedEventArgs)
    'user specified code
End Sub
```

- NodeMoved and NodeDrop events

C#

```
diagramView.NodeMoved += new NodeNudgeEventHandler(diagramView_NodeMoved);
void diagramView_NodeMoved(object sender, NodeNudgeEventArgs evtArgs)
{
}
diagramView.NodeDrop += new NodeNudgeEventHandler(diagramView_LineMoved);
void diagramView_NodeDrop(object sender, NodeNudgeEventArgs evtArgs)
{
}
```

VB.NET

```
Private diagramView.NodeMoved += New NodeNudgeEventHandler(AddressOf
diagramView_NodeMoved)
Private Sub diagramView_NodeMoved(ByVal sender As Object, ByVal evtArgs As
NodeNudgeEventArgs)
End Sub
Private diagramView.NodeDrop += New NodeNudgeEventHandler(AddressOf
diagramView_LineMoved)
Private Sub diagramView_NodeDrop(ByVal sender As Object, ByVal evtArgs As
NodeNudgeEventArgs)
End Sub
```

General in WPF Diagram (classic)

This section illustrates the general features pertaining to both Node and LineConnector.

Select Nodes and Connectors

The corresponding node or connector is selected by moving the mouse pointer to the desired node or connector and by clicking the left mouse button. This is indicated by an adorer being displayed. It also displays a RotateThumb on the top left corner of the node.

- Items on the drawing area are selected automatically if they fall within the bounds of the drag adorer.
- The drag adorer is displayed when you click anywhere on the page and start dragging the mouse pointer.
- A rectangle formed with a drag start-point as one of its points, and the point where the mouse button is released as its second point, defines the drag adorer's bounds.



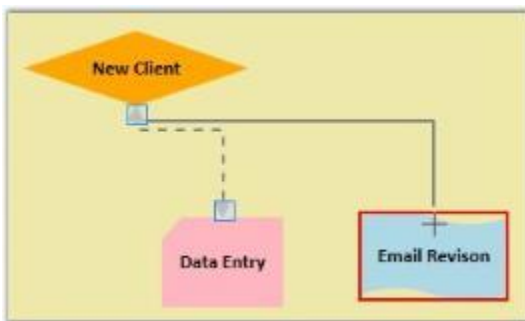
Node Selection

Move Nodes and Connectors

Click and drag the desired node for moving nodes from one position to another.. Connections can also be moved to other nodes and a new node will start acting as the head node or the tail node depending on where the connector was moved from.

To move the connector, follow these steps.

- Click on the connector to be moved.
- An adorer will be displayed on the head and the tail decorators which indicates the selection of the connector.
- Now click and drag the desired decorator shape and drop it on the node to which you want to connect.



Moving Connector

As seen, the connector will then be removed from the old node and added to the node that is currently hit.

Customize the Label of Nodes and Line Connectors

The labels of the nodes and connectors are equipped with Multiline support i.e. you can specify the labels to span multiple lines by setting the LabelTextWrapping property to wrap the text and by

specifying the width of the label. Also, several other customization properties have been added for the labels. These are listed below:

Property	Description	Type of the property	Value it accepts	Any other dependencies/ sub properties associated
LabelTextTrimming	Gets or sets the text trimming style. Default value is CharacterEllipsis.	Dependency property	TextTrimming.CharacterEllipsisTextTrimming.NoneTextTrimming.WordEllipsis	No
LabelForeground	Gets or sets the foreground of the label. Default value is Black.	Dependency property	Brush	No
LabelBackground	Gets or sets the background of the label. Default value is White.	Dependency property	Brush	No
LabelFontStyle	Gets or sets the background of the label. Default value is White.	Dependency property	FontStyles.ObliqueFontStyles.ItalicFontStyles.Normal	No
LabelFontFamily	Gets or sets the font family of	Dependency	FontFamily	No

	the label. Default value is Arial.	property		
LabelTextAlignment	Gets or sets the text alignment of the label. Default value is Center.	Dependency property	TextAlignment.RightTextAlignment.LeftTextAlignment.CenterTextAlignment.Justify	No
LabelFontSize	Gets or sets the font size of the label. Default value is 11.	Dependency property	Double	No
LabelFontWeight	Gets or sets the font weight of the label. Default value is SemiBold.	Dependency property	FontWeights	No
LabelTextWrapping	Gets or sets the text wrapping of the label. Default value is NoWrap.	Dependency property	TextWrapping.NoWrapTextWrapping.WrapTextWrapping.WrapWithOverflow	No
LabelWidth	Gets or sets the width of the label. Default value is <code>nodeâ€™s width</code> .	Dependency property	Double	No

The following code example illustrates the implementation of the properties mentioned in the table above.

C#

```
Node node1 = new Node(Guid.NewGuid(), "Register");
node1.Shape = Shapes.RoundedSquare;
node1.Width = 150;
node1.Height = 50;
node1.OffsetX = 250;
node1.OffsetY = 100;
node1.Label = "This is a Multiline Label ";
node1.LabelWidth = 70;
node1.LabelTextWrapping = TextWrapping.Wrap;
node1.LabelForeground = Brushes.IndianRed;
node1.LabelFontSize = 14;
node1.LabelFontStyle = FontStyles.Italic;
node1.LabelBackground = Brushes.Beige;
Node node2 = new Node(Guid.NewGuid(), "ClientAccountInfo");
node2.Shape = Shapes.FlowChart_Card;
node2.Width = 150;
node2.Height = 50;
node2.OffsetX = 450;
node2.OffsetY = 100;
node2.LabelWidth = 75;
node2.LabelTextWrapping = TextWrapping.Wrap;
node2.LabelForeground = Brushes.White;
node2.LabelFontSize = 16;
node2.LabelBackground = Brushes.Gray;
node2.LabelTextAlignment = TextAlignment.Left;
node2.Label = "Here text is aligned to Left";
LineConnector line = new LineConnector();
line.ConnectorType = ConnectorType.Straight;
line.TailNode = node1;
line.HeadNode = node2;
line.HeadDecoratorShape = DecoratorShape.None;
line.Label = "This is a Multiline Label for Connectors";
line.LabelWidth = 84;
line.LabelTextWrapping = TextWrapping.Wrap;
line.LabelForeground = Brushes.Green;
line.LabelFontSize = 12;
line.LabelFontStyle = FontStyles.Normal;
line.LabelBackground = Brushes.Yellow;
```

VB.NET

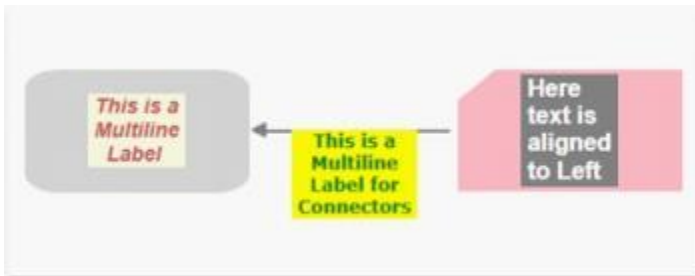
```
Dim node1 As New Node(Guid.NewGuid(), "Register")
node1.Shape = Shapes.RoundedSquare
node1.Width = 150
node1.Height = 50
node1.OffsetX = 250
node1.OffsetY = 100
node1.Label = "This is a Multiline Label "
node1.LabelWidth = 70
node1.LabelTextWrapping = TextWrapping.Wrap
node1.LabelForeground = Brushes.IndianRed
node1.LabelFontSize = 14
node1.LabelFontStyle = FontStyles.Italic
node1.LabelBackground = Brushes.Beige
```

```

Dim node2 As New Node(Guid.NewGuid(), "ClientAccountInfo")
node2.Shape = Shapes.FlowChart_Card
node2.Width = 150
node2.Height = 50
node2.OffsetX = 450
node2.OffsetY = 100
node2.LabelWidth = 75
node2.LabelTextWrapping = TextWrapping.Wrap
node2.LabelForeground = Brushes.White
node2.LabelFontSize = 16
node2.LabelBackground = Brushes.Gray
node2.LabelTextAlignment = TextAlignment.Left
node2.Label = "Here text is aligned to Left"
Dim line As New LineConnector()
line.ConnectorType = ConnectorType.Straight
line.TailNode = node1
line.HeadNode = node2
line.HeadDecoratorShape = DecoratorShape.None
line.Label = "This is a Multiline Label for Connectors"
line.LabelWidth = 84
line.LabelTextWrapping = TextWrapping.Wrap
line.LabelForeground = Brushes.Green
line.LabelFontSize = 12
line.LabelFontStyle = FontStyles.Normal
line.LabelBackground = Brushes.Yellow

```

The following output is generated using the code snippets above:



Customized Multiline Label

Customize the Context Menu of Nodes and Line Connectors

Essential Diagram for WPF provides support for the display of context menus for nodes and connectors on right-clicking the node or connector. The context menu contains the default commands, Z-order BringToFront, SendToBack, MoveForward, SendBackward, Grouping (Group and Ungroup), and Delete. The context menu can be customized so that you can add some custom options to the context menu. This can be done using the ContextMenu property of the nodes and the line connectors.

The following code example illustrates addition of custom options to the context menu.

C#

```

Node node1 = new Node(Guid.NewGuid(), "Register");
node1.Shape = Shapes.RoundedSquare;
node1.Width = 150;
node1.Height = 50;
node1.OffsetX = 250;
node1.OffsetY = 100;

```

```
ContextMenu menu = new ContextMenu();
MenuItem item1 = new MenuItem();
item1.Header = "Custom Option1";
MenuItem item2 = new MenuItem();
item2.Header = "Custom Option2";
MenuItem item3 = new MenuItem();
item3.Header = "Custom Option3";
menu.Items.Add(item1);
menu.Items.Add(item2);
menu.Items.Add(item3);
node1.ContextMenu = menu;
```

VB.NET

```
Dim node1 As New Node(Guid.NewGuid(), "Register")
node1.Shape = Shapes.RoundedSquare
node1.Width = 150
node1.Height = 50
node1.OffsetX = 250
node1.OffsetY = 100
Dim menu As New ContextMenu()
Dim item1 As New MenuItem()
item1.Header = "Custom Option1"
Dim item2 As New MenuItem()
item2.Header = "Custom Option2"
Dim item3 As New MenuItem()
item3.Header = "Custom Option3"
menu.Items.Add(item1)
menu.Items.Add(item2)
menu.Items.Add(item3)
node1.ContextMenu = menu
```

Similarly we can set it for the connectors as follows:

C#

```
LineConnector line = new LineConnector();
line.ContextMenu = menu;
```

VB.NET

```
Dim line As New LineConnector()
line.ContextMenu = menu
```



Custom Context Menu

The context menu can also be specified for all the nodes on the page using the `NodeContextMenu` property. Similarly to specify custom context menu for all the lines on the page, the `LineConnectorContextMenu` property of `DiagramView` can be used as follows:

C#

```
ContextMenu menu1 = new ContextMenu();
MenuItem item11 = new MenuItem();
item11.Header = "Custom Option11";
MenuItem item21 = new MenuItem();
item21.Header = "Custom Option21";
MenuItem item31 = new MenuItem();
item31.Header = "Custom Option31";
menu1.Items.Add(item11);
menu1.Items.Add(item21);
menu1.Items.Add(item31);
diagramView.NodeContextMenu = menu1;
diagramView.LineConnectorContextMenu = menu1;
```

VB.NET

```
Dim menu1 As New ContextMenu()
Dim item11 As New MenuItem()
item11.Header = "Custom Option11"
Dim item21 As New MenuItem()
item21.Header = "Custom Option21"
Dim item31 As New MenuItem()
item31.Header = "Custom Option31"
menu1.Items.Add(item11)
menu1.Items.Add(item21)
menu1.Items.Add(item31)
diagramView.NodeContextMenu = menu1
diagramView.LineConnectorContextMenu = menu1
```

Note: If any node's context menu is assigned using the `ContextMenu` property of that node, then it will take precedence over the `DiagramView`'s `NodeContextMenu` property. The same applies to `Line Connectors`.

Behavior Changes in WPF Diagram (classic)

The following are the changes made from version 10.1.0.44:

- The `Bounds` property of `DiagramView` will not have any effect and instead a new `LayoutBounds` property has been implemented with `Horizontal` and `Vertical` alignment of `Diagram` within rectangular bounds.
- Hereafter `RefreshLayout` has to be used to update the layout instead of calling `StartNodeArrangement` and `PrepareActivity` methods.

How to

Common in WPF Diagram (classic)

This section comprises an assembled list of questions and answers to provide expert solutions on product and its usage. It contains the following:

Common—Answers common questions that arises in minds of fresh users of Essential Diagram WPF.

Advanced—Answers questions that are in an advanced level, meant for experts.

Common in WPF Diagram (classic)

This section answers the following common question that arises in the minds of fresh users of Essential Diagram WPF.

Refresh the Tree Layout while Binding Dynamic Data to the Diagram

Essential Diagram for WPF provides support to bind dynamic data to the diagram.. But once the new data is assigned, the tree needs to be refreshed. This can be done using the RefreshLayout method.

The following code can be used to refresh the layout:

C#

```
diagramModel.ItemsSource = dataobj;  
DirectedTreeLayout tree = new DirectedTreeLayout(diagramModel, diagramView);  
tree.RefreshLayout();
```

VB.NET

```
diagramModel.ItemsSource = dataobj  
Dim tree As New DirectedTreeLayout(DiagramModel, DiagramView)  
tree.RefreshLayout()
```

In case the hierarchical layout is being used, then the following code can be used:

C#

```
diagramModel.ItemsSource = dataobj;  
HierarchicalTreeLayout tree = new HierarchicalTreeLayout(diagramModel,  
diagramView);  
tree.RefreshLayout();
```

VB.NET

```
diagramModel.ItemsSource = dataobj  
Dim tree As New HierarchicalTreeLayout(DiagramModel, DiagramView)  
tree.RefreshLayout()
```

So once data has been assigned, call the RefreshLayout() method of the corresponding tree-layout.

Host an UIElement as Node's Content

You can host any content inside the node using the Content property.

C#

```
Node n = new Node();  
n.Shape = Shapes.FlowChart_Card;  
Button b = new Button();  
b.Content = "Click ME!";  
n.Content = b;  
(n.Content as Button).IsHitTestVisible = true;
```

VB.NET

```
Dim n As New Node()
n.Shape = Shapes.FlowChart_Card
Dim b As New Button()
b.Content = "Click ME!"
n.Content = b
TryCast(n.Content, Button).IsHitTestVisible = True
```

**NodeContent**

Here, Button is a UIElement. Similarly any UIElement can be hosted as Node's Content.

Customize Node's Shape

Users can specify their own custom shapes to be used for the node as follows. First create a style resource that contains your custom shape.

HTML

```
<Style TargetType="{x:Type Path}" x:Key="myNode">
  <Setter Property="Data" Value="M200,239L200,200 240,239 280,202 320,238
  281,279 240,244 198,279z"></Setter>
  <Setter Property="Fill" Value="MidnightBlue" />
</Style>
```

Now use it for the node; the following code can be used as an example.

C#

```
Style s = (Style) this.Resources["myNode"];
Node n = new Node();
n.Shape = Shapes.CustomPath;
n.CustomPathStyle = s;
diagramModel.Nodes.Add(n);
```

VB.NET

```
Dim s As Style = CType(Me.Resources("myNode"), Style)
Dim n As New Node()
n.Shape = Shapes.CustomPath
n.CustomPathStyle = s
diagramModel.Nodes.Add(n)
```



CustomNode

Apply Style with Triggers for all the Nodes

You can specify style with triggers which is applied to all the Nodes as shown in the following code snippet.

HTML

```
<Style TargetType="{x:Type syncfusion:Node}" x:Key="{x:Type
syncfusion:Node}">
<Setter Property="LabelHorizontalAlignment" Value="Center"/>
<Setter Property="LabelVerticalAlignment" Value="Center"/>
<Setter Property="Foreground" Value="Black"/>
<Setter Property="FontWeight" Value="Bold"/>
<Style.Triggers>
<Trigger Property="Level" Value="0">
<Setter Property="CustomPathStyle">
<Setter.Value>
<Style TargetType="{x:Type Path}">
<Setter Property="Fill" Value="LightGray" />
</Style>
</Setter.Value>
</Setter>
</Trigger>
<Trigger Property="Level" Value="2">
<Setter Property="CustomPathStyle">
<Setter.Value>
<Style TargetType="{x:Type Path}">
<Setter Property="Fill" Value="LightBlue" />
</Style>
</Setter.Value>
</Setter>
</Trigger>
</Style.Triggers>
</Style>
```

Restrict Port Connections

Using the events provided for LineConnectors we can restrict making connections to ports by checking the desired condition in the event handler.

Let us consider a case where we want to restrict making connections to the port based on the color of the ports, say connections should only happen when the ports are of same color.

There are two scenarios that we need to take care of:

Scenario 1

In case a new connection is being created by dragging from one port to another, The

AfterConnectionCreate event can be used. This event fires soon after a new connection is created.

The following is the code example for restricting connections between ports and allowing connecting ports of the same color:

C#

```
// Declare the event.
```

```

diagramView.AfterConnectionCreate += new
ConnDragEndChangedEventHandler(diagramView_AfterConnectionCreate);
// Handle the event.
void diagramView_AfterConnectionCreate(object sender,
ConnDragEndRoutedEventArgs evtArgs)
{
    LineConnector line = evtArgs.Connector;
    if (line.ConnectionTailPort != null && line.ConnectionHeadPort != null &&
        line.ConnectionTailPort.PortStyle.Fill ==
        line.ConnectionHeadPort.PortStyle.Fill)
    {
        // Do nothing.
    }
    else
    {
        // Remove the connection.
        diagramModel.Connections.Remove(evtArgs.Connector);
    }
}

```

VB.NET

```

'Declare the event.
Private diagramView.AfterConnectionCreate += New
ConnDragEndChangedEventHandler(AddressOf diagramView_AfterConnectionCreate)
'Handle the event.
Private Sub diagramView_AfterConnectionCreate(ByVal sender As Object, ByVal
evtArgs As ConnDragEndRoutedEventArgs)
Dim line As LineConnector = evtArgs.Connector
If line.ConnectionTailPort IsNot Nothing AndAlso line.ConnectionHeadPort
IsNot Nothing AndAlso line.ConnectionTailPort.PortStyle.Fill =
line.ConnectionHeadPort.PortStyle.Fill Then
'Do nothing.
Else
'Remove the connection.
diagramModel.Connections.Remove(evtArgs.Connector)
End If
End Sub

```

Scenario 2

In case an already existing connection is been dragged to connect to other ports, then the ConnectorDragStart and ConnectorDragEnd events can be used to restrict connections.

As the name implies, the ConnectorDragStart event fires when either ends of the connector is dragged.

The ConnectorDragEnd fires soon after the drag operation is complete.

The following is the code example for restricting connections between ports and allow only red ports to connect.

C#

```

// Declare the event.
diagramView.ConnectorDragStart += new
ConnDragChangedEventHandler(diagramView_ConnectorDragStart);
// To store the previous port to which the line was connected to.

```

```

ConnectionPort oldport;
// Handle the event.
void diagramView_ConnectorDragStart(object sender, ConnDragRoutedEventArgs
evtArgs)
{
    LineConnector line = evtArgs.Connector;
    if (evtArgs.FixedNodeEnd == line.TailNode)
        oldport = line.ConnectionHeadPort;
    else
        oldport = line.ConnectionTailPort;
}

```

VB.NET

```

'Declare the event.
Private diagramView.ConnectorDragStart += New
ConnDragChangedEventHandler(AddressOf diagramView_ConnectorDragStart)
'To store the previous port to which the line was connected to.
Private oldport As ConnectionPort
'Handle the event.
Private Sub diagramView_ConnectorDragStart(ByVal sender As Object, ByVal
evtArgs As ConnDragRoutedEventArgs)
    Dim line As LineConnector = evtArgs.Connector
    If evtArgs.FixedNodeEnd = line.TailNode Then
        oldport = line.ConnectionHeadPort
    Else
        oldport = line.ConnectionTailPort
    End If
End Sub

```

Now once you have stored the old port, you can check for the condition in ConnectorDragEnd event as follows.

C#

```

// Declare the event.
diagramView.ConnectorDragEnd += new
ConnDragEndChangedEventHandler(diagramView_ConnectorDragEnd);
// Handle the event.
void diagramView_ConnectorDragEnd(object sender, ConnDragEndRoutedEventArgs
evtArgs)
{
    LineConnector line = evtArgs.Connector;
    if (line.ConnectionTailPort != null && line.ConnectionHeadPort != null &&
line.ConnectionTailPort.PortStyle.Fill ==
line.ConnectionHeadPort.PortStyle.Fill)
    {
        // Do nothing.
    }
    else
    {
        // Check to which end the old port has to be restored.
        if (evtArgs.HitNodeEnd == line.HeadNode && oldport != null)
        {
            evtArgs.Connector.ConnectionHeadPort = oldport;
            evtArgs.Connector.HeadNode = oldport.Node;
        }
    }
}

```

```

}
else
{
    evtArgs.Connector.ConnectionTailPort = oldport;
    evtArgs.Connector.TailNode = oldport.Node;
}
}
}

```

VB.NET

```

'Declare the event.
Private diagramView.ConnectorDragEnd += New
ConnDragEndChangedEventHandler(AddressOf diagramView_ConnectorDragEnd)
'Handle the event.
Private Sub diagramView_ConnectorDragEnd(ByVal sender As Object, ByVal
    evtArgs As ConnDragEndRoutedEventArgs)
    Dim line As LineConnector = evtArgs.Connector
    If line.ConnectionTailPort IsNot Nothing AndAlso line.ConnectionHeadPort
        IsNot Nothing AndAlso line.ConnectionTailPort.PortStyle.Fill =
        line.ConnectionHeadPort.PortStyle.Fill Then
        'Do nothing.
    Else
        'Check to which end the old port has to be restored.
        If evtArgs.HitNodeEnd = line.HeadNode AndAlso oldport IsNot Nothing Then
            evtArgs.Connector.ConnectionHeadPort = oldport
            evtArgs.Connector.HeadNode = oldport.Node
        Else
            evtArgs.Connector.ConnectionTailPort = oldport
            evtArgs.Connector.TailNode = oldport.Node
        End If
    End If
End Sub

```

Hide Resizer or Rotator's Visibility of a Node

Gripper or Rotator Visibility can be hidden using the following code example.

C#

```

node.Loaded += new RoutedEventHandler(node_Loaded);
//Hide the Node's Resizer and Rotator in the Node's loaded event.
void node_Loaded(object sender, RoutedEventArgs e)
{
    Node node = sender as Node;
    //node.Template will be null if it's template is not applied.
    if (node != null && node.Template != null)
    {
        //To hide the Resizer.
        (node.Template.FindName("PART_Resizer", node) as Control).Template = null;
        //To hide the Rotator.
        (node.Template.FindName("PART_Rotator", node) as Control).Template = null;
    }
}

```

VB.NET

```

Private node.Loaded += New RoutedEventHandler(AddressOf node_Loaded)
'Hide the Node's Resizer and Rotator in the Node's loaded event.
Private Sub node_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
Dim node As Node = TryCast(sender, Node)
'node.Template will be null if it's template is not applied.
If node IsNot Nothing AndAlso node.Template IsNot Nothing Then
'To hide the Resizer.
TryCast(node.Template.FindName("PART_Resizer", node), Control).Template =
Nothing
'To hide the Rotator.
TryCast(node.Template.FindName("PART_Rotator", node), Control).Template =
Nothing
End If
End Sub

```

Note: Node's Template will be available only after its template is applied, so if you try to do these operations before it will not give an expected result.

Hide the Default Center Port of a Node

Each node will have a default center port visibility of this port can be hidden using the following statement.

C#

```

node.Loaded += new RoutedEventHandler(node_Loaded);
//Hide the Node's center port in the Node's loaded event.
void node_Loaded(object sender, RoutedEventArgs e)
{
    Node node = sender as Node;
    if (node.Ports.Count > 0)
    {
        node.Ports[0].Visibility = Visibility.Hidden;
    }
}

```

VB.NET

```

Private node.Loaded += New RoutedEventHandler(AddressOf node_Loaded)
'Hide the Node's center port in the Node's loaded event.
Private Sub node_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
Dim node As Node = TryCast(sender, Node)
If node.Ports.Count > 0 Then
    node.Ports(0).Visibility = Visibility.Hidden
End If
End Sub

```

Note: node.Ports[0] refers to the center port. This default center port will be available only after the Node's Template is applied. So you have to change the Visibility accordingly.

Programmatically Rotate a Node and Keep the Label Horizontal after Rotating

Node can be programmatically rotated and the Label can be kept horizontal after rotation using the following code example.

C#

```
Node NewClient = new Node();
NewClient.Shape = Shapes.FlowChart_Card;
diagramModel.Nodes.Add(NewClient);
double angle = 90;
NewClient.RenderTransform = new RotateTransform(angle);
NewClient.Label = "90 deg rotation";
NewClient.RenderTransformOrigin = new System.Windows.Point(0.5, 0.5);
NewClient.LabelAngle = 360 - angle;
```

VB.NET

```
Dim NewClient As New Node()
NewClient.Shape = Shapes.FlowChart_Card
diagramModel.Nodes.Add(NewClient)
Dim angle As Double = 90
NewClient.RenderTransform = New RotateTransform(angle)
NewClient.Label = "90 deg rotation"
NewClient.RenderTransformOrigin = New System.Windows.Point(0.5, 0.5)
NewClient.LabelAngle = 360 - angle
```

Here diagramModel is an instance of DiagramModel.

Identify the Shapes Dropped on the Page from the Palette

The SymbolPalettItemName property can be used to identify the item dropped on the page in the NodeDrop event. This is particularly useful when you have to identify the item which is dropped and performs an operation on the node before it is added to the View. The Name property of the SymbolPalettItem can be set while adding the item to the palette and then by using SymbolPalettItemName property in the event args of NodeDrop. A reference to the corresponding SymbolPalettItem can be obtained.

C#

```
SymbolPalettItem ss = new SymbolPalettItem();
Label l = new Label();
l.Content = "Label";
ss.Content = l;
ss.Name = "MyItem";
```

VB.NET

```
Dim ss As New SymbolPalettItem()
Dim l As New Label()
l.Content = "Label"
ss.Content = l
ss.Name = "MyItem"
```

The NodeDrop event can be declared:

C#

```
// Declare the event.
diagramView.NodeDrop += new NodeDroppedEventHandler(diagramView_NodeDrop);
// Handle the event.
void diagramView_NodeDrop(object sender, NodeDroppedRoutedEventArgs evtArgs)
```



```
{
  if (evtArgs.SymbolPaletteItemName == "MyItem")
  {
    // User-specified code.
  }
}
```

VB.NET

```
'Declare the event.
Private diagramView.NodeDrop += New NodeDroppedEventHandler(AddressOf
diagramView_NodeDrop)
'Handle the event.
Private Sub diagramView_NodeDrop(ByVal sender As Object, ByVal evtArgs As
NodeDroppedRoutedEventArgs)
If evtArgs.SymbolPaletteItemName = "MyItem" Then
'User-specified code.
End If
End Sub
```

Hide ContextMenu for all Nodes and Connections

ContextMenu for all Nodes and LineConnectors can be hidden using the following code snippet.

C#

```
diagramView.NodeContextMenu = new ContextMenu { Visibility =
Visibility.Collapsed };
diagramView.LineConnectorContextMenu = new ContextMenu { Visibility =
Visibility.Collapsed };
```

VB.NET

```
diagramView.NodeContextMenu = New ContextMenu With {.Visibility =
Visibility.Collapsed}
diagramView.LineConnectorContextMenu = New ContextMenu With {.Visibility =
Visibility.Collapsed}
```

Where, diagramView is an instance of DiagramView

How can the nudge operation be stopped?

The nudge operation can be stopped by handling the DiagramControl's PreviewKeyDown event, as shown in the code example displayed below.

C#

```
//Register the PreviewKeyDown event.
diagramControl.PreviewKeyDown += new
KeyEventHandler(MainWindow_PreviewKeyDown);
//Handle the PreviewKeyDown event for the arrow keys.
void MainWindow_PreviewKeyDown(object sender, KeyEventArgs e)
{
  if (e.Key == Key.Up || e.Key == Key.Down || e.Key == Key.Right || e.Key ==
Key.Left)
  {
    e.Handled = true;
  }
}
```

```
}

```

VB.NET

```
'Register the PreviewKeyDown event.
Private diagramControl.PreviewKeyDown += New KeyEventHandler(AddressOf
MainWindow_PreviewKeyDown)
'Handle the PreviewKeyDown event for the arrow keys.
Private Sub MainWindow_PreviewKeyDown(ByVal sender As Object, ByVal e As
KeyEventArgs)
If e.Key = Key.Up OrElse e.Key = Key.Down OrElse e.Key = Key.Right OrElse
e.Key = Key.Left Then
e.Handled = True
End If
End Sub

```

How can MultipleTrigger be applied for a Node?

MultiTrigger enables you to set the property values or start actions based on a collection of conditions. A condition is met when the element's property value matches the specified value. This comparison is performed by a reference equality check. The following example shows how to use MultipleTrigger.

HTML

```
<Style.Triggers>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="IsSelected" Value="True"/>
<Condition Property="Tag" Value="True"/>
</MultiTrigger.Conditions>
<Setter Property="Effect">
<Setter.Value>
<DropShadowEffect Color="Gray" BlurRadius="5" Direction="325"
ShadowDepth="5"/>
</Setter.Value>
</Setter>
</MultiTrigger>
</Style.Triggers>

```

If a Node has the above Trigger set in its Style, then the Node will have a DropShadowEffect when both the IsSelected property and the Tag property are set to True.

Advanced support in WPF Diagram (Classic)

This section answers the following questions that are in an advanced level, meant for experts.

Animate the Nodes in the Diagram

You can perform many kinds of animations on nodes by using the double animation. Rotation and Translation are some of the basic operations performed on the nodes. You can use double animation to perform these operations on the node in a specific pattern.

To rotate a node, the following code can be used.

C#

```
DoubleAnimation nodeanimation = new DoubleAnimation();
nodeanimation.From = 0;

```

```
nodeanimation.To = 360;
nodeanimation.Duration = new Duration(new TimeSpan(0, 0, 0, 0, 500));
nodeanimation.RepeatBehavior = new RepeatBehavior(15);
RotateTransform rt = new RotateTransform();
nodeObj.RenderTransform = rt;
nodeObj.RenderTransformOrigin = new Point(.5, .5);
rt.BeginAnimation(RotateTransform.AngleProperty, nodeanimation);
```

VB.NET

```
Dim nodeanimation As New DoubleAnimation()
nodeanimation.From = 0
nodeanimation.To = 360
nodeanimation.Duration = New Duration(New TimeSpan(0, 0, 0, 0, 500))
nodeanimation.RepeatBehavior = New RepeatBehavior(15)
Dim rt As New RotateTransform()
nodeObj.RenderTransform = rt
nodeObj.RenderTransformOrigin = New Point(.5, .5)
rt.BeginAnimation(RotateTransform.AngleProperty, nodeanimation)
```

To translate a node with respect to the x-axis, the TranslateTransform can be applied.

C#

```
DoubleAnimation nodeanimation = new DoubleAnimation();
nodeanimation.From = 500;
nodeanimation.To = 0;
nodeanimation.Duration = new Duration(new TimeSpan(0, 0, 0, 0, 500));
nodeanimation.RepeatBehavior = new RepeatBehavior(1);
```

VB.NET

```
Dim nodeanimation As New DoubleAnimation()
nodeanimation.From = 500
nodeanimation.To = 0
nodeanimation.Duration = New Duration(New TimeSpan(0, 0, 0, 0, 500))
nodeanimation.RepeatBehavior = New RepeatBehavior(1)
```

Once you have created the double animation, you can then apply it to the node which we want to translate in the following way.

C#

```
DoubleAnimation nodeanimation = new DoubleAnimation();
nodeanimation.From = 500;
nodeanimation.To = 0;
nodeanimation.Duration = new Duration(new TimeSpan(0, 0, 0, 0, 500));
nodeanimation.RepeatBehavior = new RepeatBehavior(1);
TranslateTransform rt = new TranslateTransform();
nodeObj.RenderTransform = rt;
rt.BeginAnimation(TranslateTransform.XProperty, nodeanimation);
```

VB.NET

```

Dim nodeanimation As New DoubleAnimation()
nodeanimation.From = 500
nodeanimation.To = 0
nodeanimation.Duration = New Duration(New TimeSpan(0, 0, 0, 0, 500))
nodeanimation.RepeatBehavior = New RepeatBehavior(1)
Dim rt As New TranslateTransform()
nodeObj.RenderTransform = rt
rt.BeginAnimation(TranslateTransform.XProperty, nodeanimation)

```

Print DiagramPage in Uniform Print Mode Using Framework Print Dialog

DiagramPage can also be printed using Framework PrintDialog instead of using Syncfusion DiagramControlPrintPreview Dialog, as shown in the following code example.

C#

```

//Create Framwork Print Dialog.
PrintDialog PrintDialog = new PrintDialog();
//Open Print Dialog.
Nullable<Boolean> printClicked = PrintDialog.ShowDialog();
//If Print is clicked.
if (printClicked == true)
{
    //Print the Diagram Page.
    //Get Printer Capabilities.
    PrintCapabilities printCapabilities =
    PrintDialog.PrintQueue.GetPrintCapabilities(PrintDialog.PrintTicket);
    Size pageAreaSize = new
    Size(printCapabilities.PageImageableArea.ExtentWidth,
    printCapabilities.PageImageableArea.ExtentHeight);
    //Visual Brush for the DiagramPage to be printed.
    VisualBrush VisualBrush = new VisualBrush(diagramView.Page);
    VisualBrush.Stretch = Stretch.Uniform;
    VisualBrush.ViewboxUnits = BrushMappingMode.Absolute;
    VisualBrush.Viewbox = new Rect(0, 0, diagramView.Page.ActualWidth,
    diagramView.Page.ActualHeight);
    //Rectangle to contain the VisualBrush.
    Rectangle rect = new Rectangle();
    rect.Fill = VisualBrush;
    rect.Arrange(new Rect(new Point(0, 0), pageAreaSize));
    SetViewport(VisualBrush, new Size(diagramView.Page.ActualWidth,
    diagramView.Page.ActualHeight));
    //Print the Page.
    XpsDocumentWriter writer =
    PrintQueue.CreateXpsDocumentWriter(PrintDialog.PrintQueue);
    writer.Write(rect, PrintDialog.PrintTicket);
}
//Paint the brush to fit uniformly.
private void SetViewport(VisualBrush brush, Size size)
{
    double coefficientHeight = size.Height / brush.Viewbox.Height;
    double coefficientWidth = size.Width / brush.Viewbox.Width;
    if (coefficientHeight < coefficientWidth)
    {
        double width = coefficientHeight * brush.Viewbox.Width / size.Width;
        double x = (1 - width) / 2;
        brush.Viewport = new Rect(new Point(x, 0), new Size(width, 1));
    }
}

```

```

}
else if (coefficientHeight > coefficientWidth)
{
double height = coefficientWidth * brush.Viewbox.Height / size.Height;
double y = (1 - height) / 2;
brush.Viewport = new Rect(new Point(0, y), new Size(1, height));
}
}

```

VB.NET

```

'Create Framwork Print Dialog.
Dim PrintDialog As New PrintDialog()
'Open Print Dialog.
Dim printClicked As Nullable(Of Boolean) = PrintDialog.ShowDialog()
'If Print is clicked.
If printClicked.GetValueOrDefault() = True Then
'Print the Diagram Page.
'Get Printer Capabilities.
Dim printCapabilities As PrintCapabilities =
PrintDialog.PrintQueue.GetPrintCapabilities(PrintDialog.PrintTicket)
Dim pageAreaSize As New
Size(PrintCapabilities.PageImageableArea.ExtentWidth,
PrintCapabilities.PageImageableArea.ExtentHeight)
'Visual Brush for the DiagramPage to be printed.
Dim VisualBrush As New VisualBrush(DiagramView.Page)
VisualBrush.Stretch = Stretch.Uniform
VisualBrush.ViewboxUnits = BrushMappingMode.Absolute
VisualBrush.Viewbox = New Rect(0, 0, diagramView.Page.ActualWidth,
diagramView.Page.ActualHeight)
'Rectangle to contain the VisualBrush.
Dim rect As New Rectangle()
rect.Fill = VisualBrush
rect.Arrange(New Rect(New Point(0, 0), pageAreaSize))
SetViewport(VisualBrush, New Size(diagramView.Page.ActualWidth,
diagramView.Page.ActualHeight))
'Print the Page.
Dim writer As XpsDocumentWriter =
PrintQueue.CreateXpsDocumentWriter(PrintDialog.PrintQueue)
writer.Write(rect, PrintDialog.PrintTicket)
End If
'Paint the brush to fit uniformly.
private void SetViewport(VisualBrush brush, Size size)
Dim coefficientHeight As Double = Size.Height / Brush.Viewbox.Height
Dim coefficientWidth As Double = Size.Width / Brush.Viewbox.Width
If coefficientHeight < coefficientWidth Then
Dim width As Double = coefficientHeight * Brush.Viewbox.Width / Size.Width
Dim x As Double = (1 - Width) / 2
brush.Viewport = New Rect(New Point(x, 0), New Size(width, 1))
ElseIf coefficientHeight > coefficientWidth Then
Dim height As Double = coefficientWidth * Brush.Viewbox.Height / Size.Height
Dim y As Double = (1 - Height) / 2
brush.Viewport = New Rect(New Point(0, y), New Size(1, height))
End If

```

Save the Current Zoom Settings and Load the Settings Back

Zoom settings can be saved into variables and this saved settings can be applied back again using the following code example.

C#

```
//Save current zoom setting.
double SavedZoomFactor = diagramView.ZoomFactor;
double SavedCurrentZoom = (double)
diagramView.GetValue(DiagramView.CurrentZoomProperty);
//Load the saved zoom settings.
//Reset the current zoom
ZoomCommands.Reset.Execute(diagramView.Page, diagramView);
//Set the zoom factor temporarily to the stored CurrentZoomProperty
diagramView.ZoomFactor = SavedCurrentZoom - 1;
//Now if a zoom operation is performed, we will get the stored zoom setting.
ZoomCommands.ZoomIn.Execute(diagramView.Page, diagramView);
//Change the Zoom factor to the required value.
diagramView.ZoomFactor = SavedZoomFactor;
```

VB.NET

```
'Save current zoom setting.
Dim SavedZoomFactor As Double = DiagramView.ZoomFactor
Dim SavedCurrentZoom As Double =
Cdbl(DiagramView.GetValue(DiagramView.CurrentZoomProperty))
'Load the saved zoom settings.
'Reset the current zoom
ZoomCommands.Reset.Execute(diagramView.Page, diagramView)
'Set the zoom factor temporarily to the stored CurrentZoomProperty
diagramView.ZoomFactor = SavedCurrentZoom - 1
'Now if a zoom operation is performed, we will get the stored zoom setting.
ZoomCommands.ZoomIn.Execute(diagramView.Page, diagramView)
'Change the Zoom factor to the required value.
diagramView.ZoomFactor = SavedZoomFactor
```

Label Alignment

You are provided with a lot more alignment options to customize Label of Node and Connector.

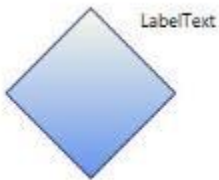
Property	Description	Property Type	Accepted Value	Any other dependencies/sub-properties associated
LabelHorizontalTextAlignment	Specifies the Horizontal	Dependency property	HorizontalAlignment.CenterHorizontalAlignment.LeftHorizontalAlignment.RightHorizontalAlignment.Stretch;	No

	Text alignment of the Label.			
LabelVerticalTextAlignment	Specifies the Vertical Text alignment of the Label.	Dependency property	VerticalAlignment.Center;VerticalAlignment.Top;VerticalAlignment.Bottom;VerticalAlignment.Stretch;	No

The following code snippet depicts the behavior of LabelHorizontalTextAlignment and LabelVerticalTextAlignment.

C#

```
Node node = new Node();
node.Label = "LabelText";
node.Shape = Shapes.FlowChart_Decision;
node.LabelHorizontalTextAlignment = HorizontalAlignment.Right;
node.LabelVerticalTextAlignment = VerticalAlignment.Top;
```



LabelAlignment

Label Resizer

You are provided with support for Label Selection and Resizing at runtime. This feature can be enabled by using `IsEnableLabelSelection` property in `DiagramView`. By enabling this property you are able to resize, drag and rotate the `LabelEditor` like Nodes. The default value is "False".

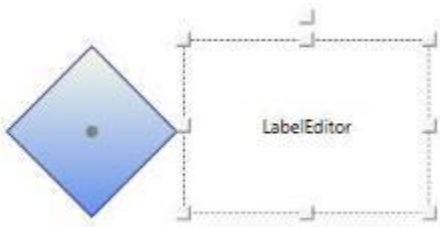
Property	Description	Property Type	Accepted Value	Any other dependencies/ sub-properties associated
IsEnableLabelSelection	Gets or sets a value indicating whether to select the node	Dependency property	Boolean (True/False)	No

	or not. The default value is 'False'.			
--	---------------------------------------	--	--	--

The following code example is used to enable LabelEditor for Nodes.

C#

```
DiagramView view = new DiagramView();
view.IsEnableLabelSelection = true;
Node node = new Node();
node.LabelWidth = 100;
node.LabelHeight = 100;
node.Label = "LabelEditor";
```

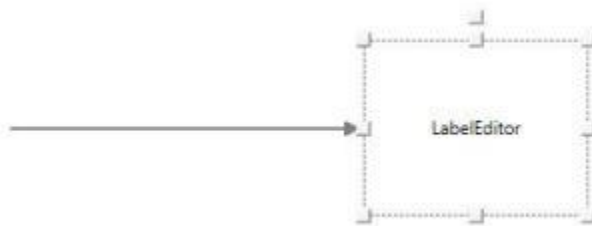


LabelEditor for Node

The following code example is used to enable LabelEditor for Connectors.

C#

```
DiagramView view = new DiagramView();
view.IsEnableLabelSelection = true;
LineConnector conn = new LineConnector();
conn.LabelWidth = 100;
conn.LabelHeight = 100;
conn.Label = "LabelEditor";
```



LabelEditor for Connector

GridDataControl (Classic)

WPF GridDataControl (Classic) 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.

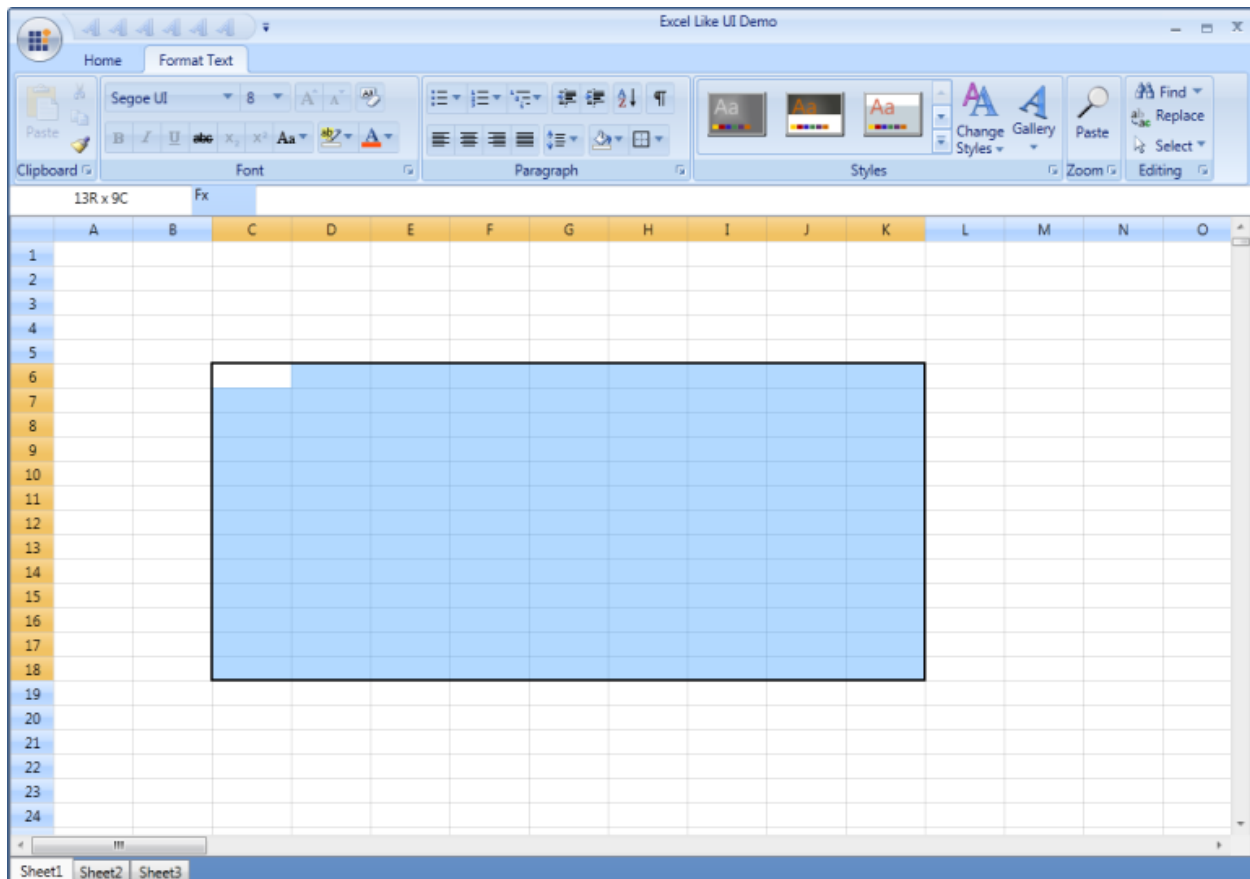
The display system also hosts a powerful and complete styles architecture. Settings can be specified at the cell level or higher levels using parent styles that are referred to as base styles. Base styles can affect groups of cells. Cell level settings override any higher-level settings and enable easy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

Use Case Scenarios

EssentialGrid for WPF can be applied to a variety of industries such as finance, banking, software, etc. Some of its important features are:

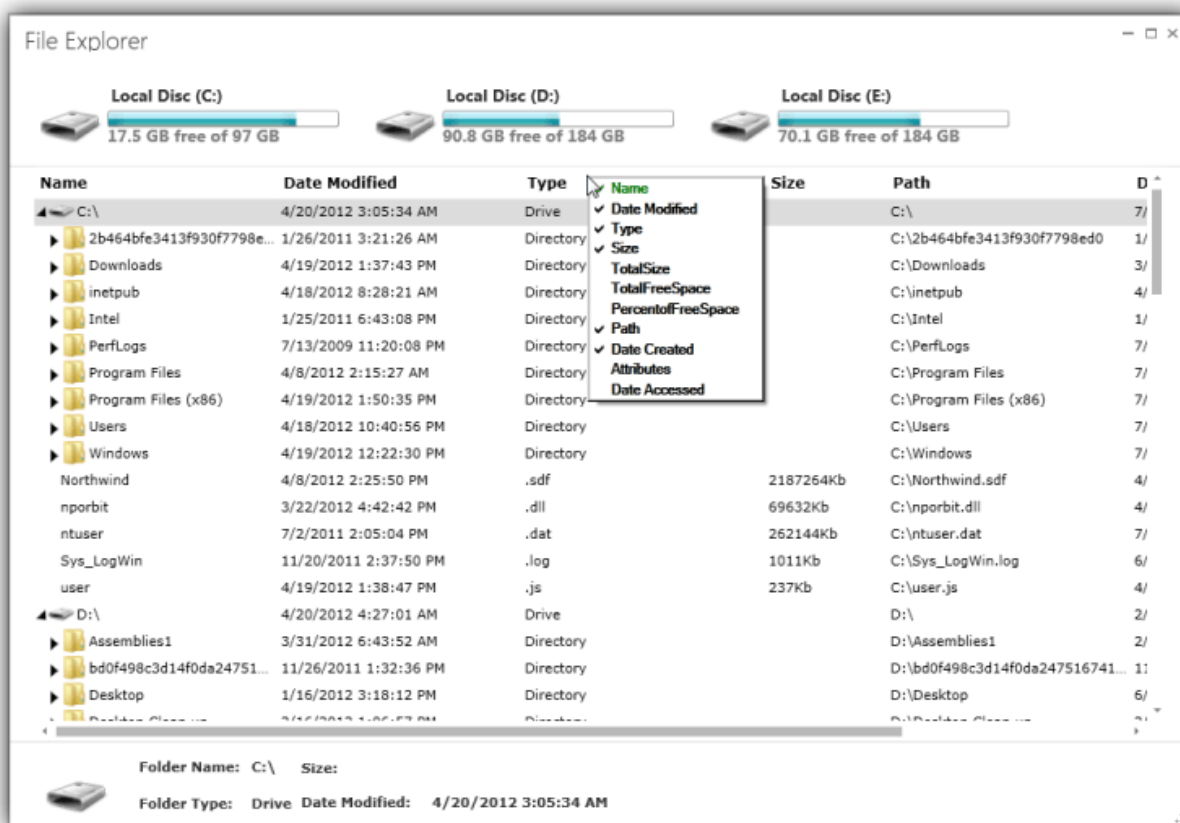
Excel-Like UI: Essential Grid's rich feature set allows you to build Excel-like UI applications.



High Performance - EssentialGrid is a great asset to high-performance applications, as it can display large amounts of real-time data that tends to periodic changes without any performance hits. Following is an illustration of a stock portfolio application using the GridData control.

Quotes Symbol ▲		Account ID ▲					
	Holding ID	Quotes Symbol ▲	Quantity	Price Paid	Purchase Date	Account ID ▲	Stock Exchange ID ▲
▲	Total Quantity		Qty - (421)	Price - (3.550)			
▲	Total Quantity		Qty - (1)	Price - (.830)			
	6463	AAME	1	Rs. 0.83	23-12-2008	1	2
▲	Total Quantity		Qty - (193)	Price - (.950)			
	7293	AAME	193	Rs. 0.95	23-12-2008	5	4
▲	Total Quantity		Qty - (227)	Price - (1.770)			
	7415	AAME	110	Rs. 0.80	23-12-2008	6	7
	7450	AAME	117	Rs. 0.97	23-12-2008	6	6
▲	Total Quantity		Qty - (397)	Price - (66.240)			
▲	Total Quantity		Qty - (397)	Price - (66.240)			
	7656	AAON	34	Rs. 16.56	09-01-2009	7	7
	7657	AAON	121	Rs. 16.56	11-02-2009	7	7
	7658	AAON	121	Rs. 16.56	11-02-2009	7	7
	7659	AAON	121	Rs. 16.56	11-02-2009	7	7
▶	Total Quantity		Qty - (186)	Price - (185.090)			
▶	Total Quantity		Qty - (234)	Price - (19.520)			
▶	Total Quantity		Qty - (279)	Price - (3391.460)			
▶	Total Quantity		Qty - (211)	Price - (86.070)			
▶	Total Quantity		Qty - (242)	Price - (28.000)			

File Explorer - Applications that deal with hierarchical data can make use of Essential Grid's file explorer feature, which allows child items to be displayed on-demand by using the GridTree control.



Key Features

You can find the following features of EssentialGrid for WPF:

- 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 you to copy and paste grid cell content to text or any format.
- Frozen Rows and Columns – Essential Grid allows you 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.

User Guide Organization

EssentialGrid for WPF comes with numerous samples as well as extensive documentation for your reference. This user guide provides detailed information on features and functionalities. It is organized into the following sections:

- Overview – This section provides a brief introduction to Essential Grid and its key features.
- Getting Started – This section guides you on getting started with a WPF application and WPF controls.
- Concepts and Features – Under this section, the features of individual controls are illustrated with use-case scenarios, code examples, and screenshots.
- Frequently Asked Questions – This section contains answers to frequently asked questions about Essential Grid.

Document Conventions

The conventions below help you quickly identify important sections of information when using this user guide:

Convention	Description
Note	Represents important information to be noted.
Example	Represents an example.
Tip	Represents useful hints that help you use the controls and features.
Additional information	Represents additional information on the corresponding topic.

Feature Summary

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

GridData Control

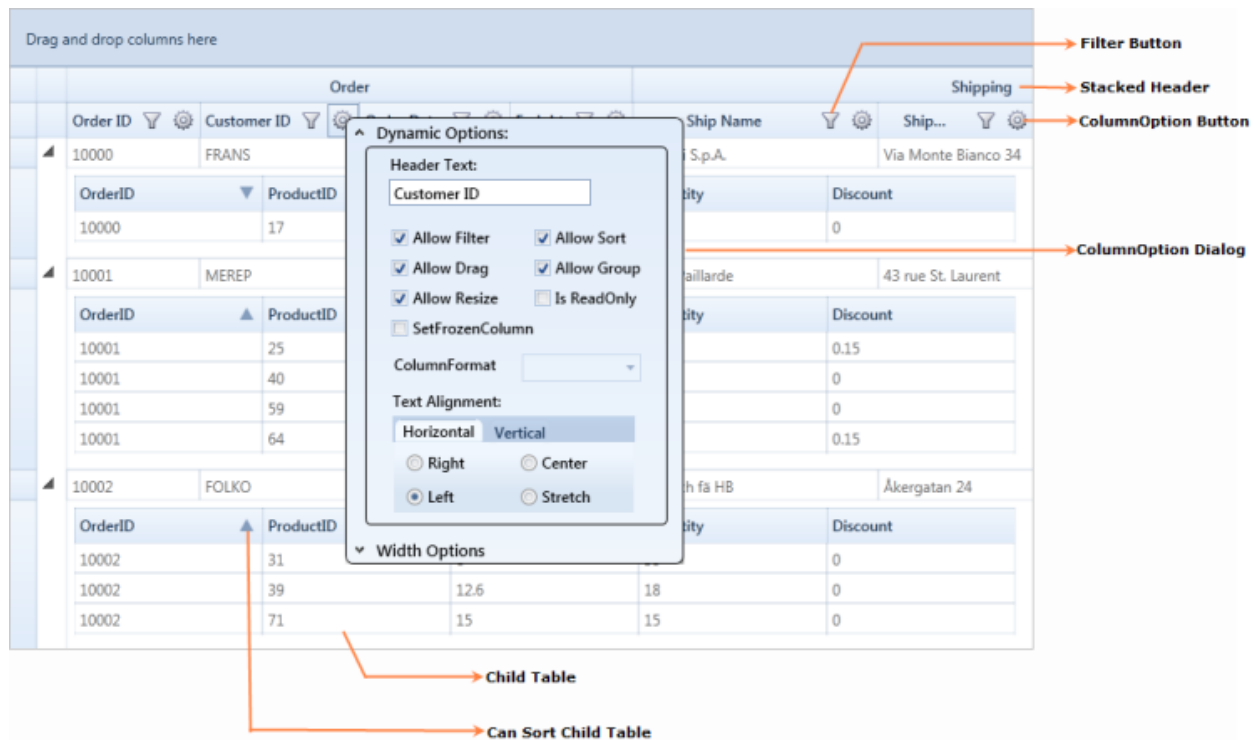
The GridData control is a data-bound control that supports editing, sorting, and grouping. The main features of the GridData control are explained in the following list.

Working with Data:

The GridData control supports all popular data sources including the following:

- Observable collections.
- Data tables.
- Collection views.
- Business objects.
- ADO.NET Entity Framework.
- LINQ to SQL.
- ORM.ADO.NET.
- DataView.
- ICollectionView.
- ObjectDataProvider.
- IEnumerable.
- IList.
- IBindingList.
- IQueryable.
- IListed.

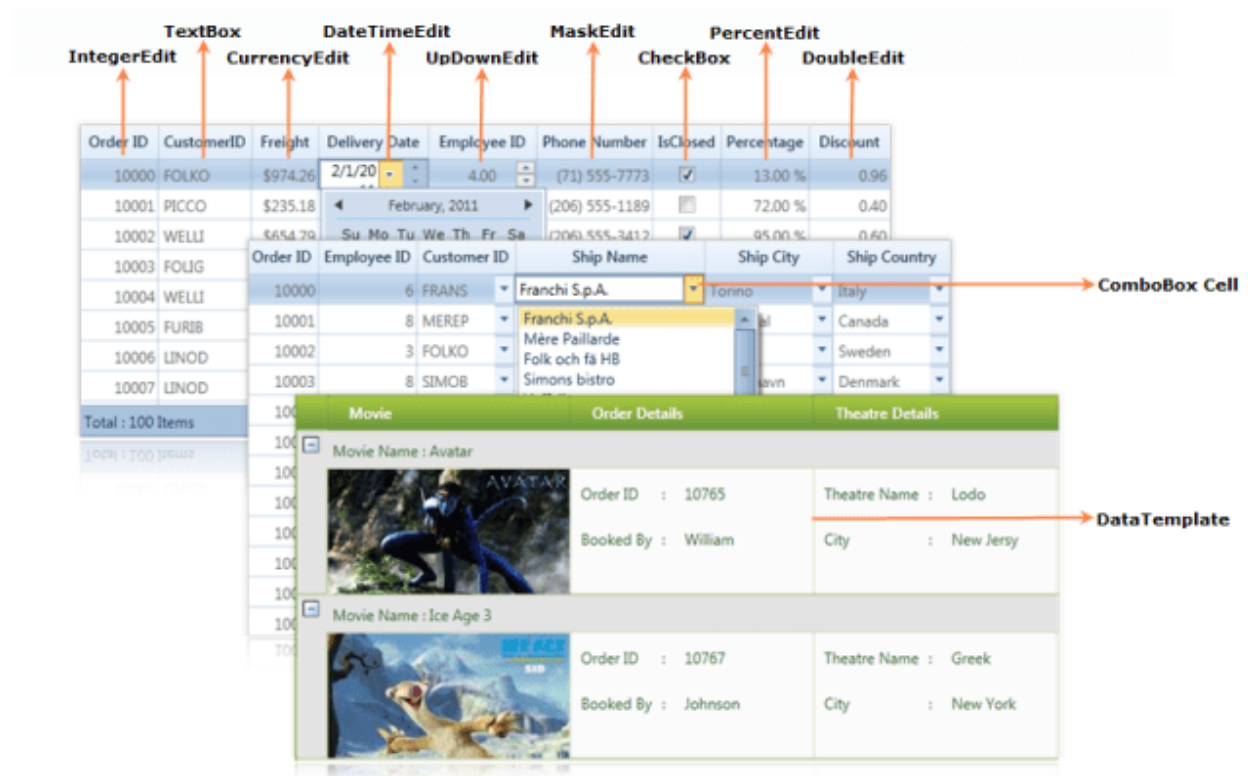
Complex objects can bind with the GridData control and the GridData control can display related information through hierarchies. The master-detail relationship can be represented through nested grids, which can be expanded and collapsed as required. Multilevel nesting is also supported.



Data Presentation

Cell Types - Several built-in cell types can be used to display and edit any underlying data type. The cell types include the following:

- Static-text cells.
- Check-box cells.
- Button cells.
- Image cells.
- Combo-box cells.
- Drop-down lists.
- Currency cells.
- Date-time cells.
- Double edit cells.
- Integer edit cells.
- Mask edit cells.
- Percent edit cells.
- Rich-text-box cells.
- Up-down edit cells.
- Nested grid cells.



Interactive Features

The GridData control contains a number of features for managing data effectively, including grouping, sorting, and Excel-like filtering with filter bars and advanced filtering. You can choose a column with the ColumnChooser feature, and column-related options such as dragging, sorting, grouping, and resizing can be enabled or disabled dynamically by using a separate dialog called ColumnOptions. Other specialized interactive features include stacked headers, context menus, ToolTips, and paging.

Visual Styles and Expression Blend

The GridData control has a rich selection of over 14 built-in styles that provide an attractive look and feel for the grid. The GridData control also allows you to customize all aspects of grid appearance by using Microsoft Expression Blend.

The image displays two screenshots of the WPF GridDataControl (Classic) showing movie data. The left screenshot shows a grouped view with expandable sections for each movie. The right screenshot shows a flat view of the same data.

Orderid	Name	SeatNo	City	Theatre
Movie : Avatar - 3 Items				
10765	William	F23,24,25,26	UK	Lodo
10945	Smith	A-12	Canada	Modern
10967	Johnson	B-16, 17	Canada	Greek
Total - 3 Items				
Movie : Ice Age3 - 3 Items				
10767	Johnson	B-16, 17	Canada	Greek
10765	William	F23,24,25,26	UK	Lodo
10978	William	F23,24,25,26	UK	Lodo
Total - 3 Items				
Movie : Kung Fu Panda 2 - 2 Items				
10062	Smith	A-12	Canada	Modern
10978	William	F23,24,25,26	UK	Lodo
Total - 2 Items				
Movie : Shrek - 4 Items				
10643	Johnson	B-16, 17	Canada	Greek
10942	William	F23,24,25,26	UK	Lodo
10942	William	F23,24,25,26	UK	Lodo
10967	Johnson	B-16, 17	Canada	Greek
Total - 4 Items				
Movie : The Hangover Part II - 4 Items				
10065	Smith	A-12	Canada	Modern
10942	William	F23,24,25,26	UK	Lodo
10942	William	F23,24,25,26	UK	Lodo
10967	Johnson	B-16, 17	Canada	Greek
Total - 4 Items				
Total : 27 Items				

Getting Started with WPF GridDataControl (Classic)

This section is designed to help you understand and quickly get started using Essential Grid in your WPF application. Control appearance and structure are defined and the Essential Grid's relevant classes are depicted.

The following sections comprise the Getting Started section:

Appearance and Structure of the Grid

EssentialGrid for WPF is 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 EssentialGrid package is comprised of following three types of grid controls:

- Grid Control
- GridData Control
- GridTree Control

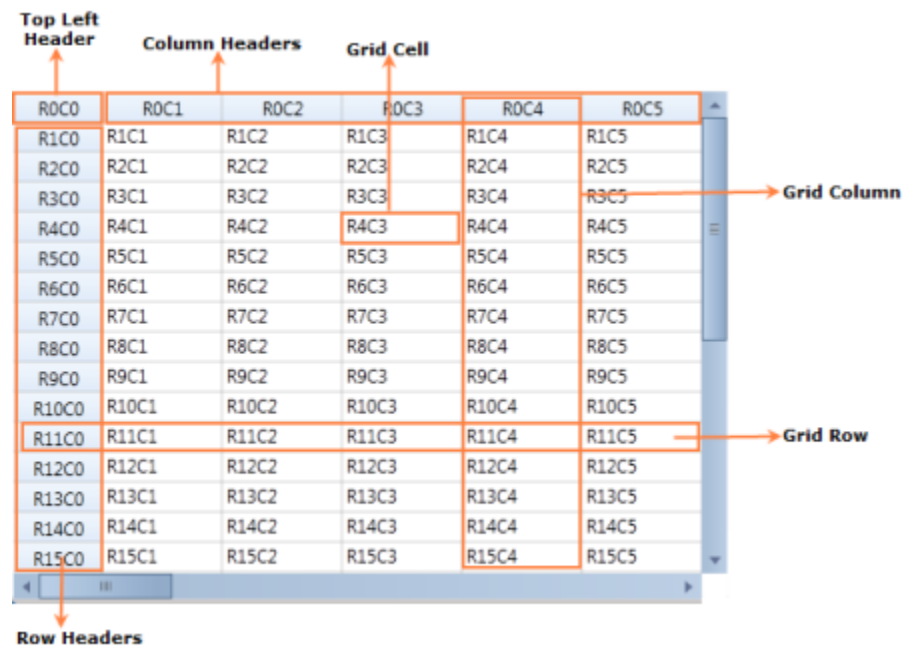
Now, take closer look at the characteristics of each of these controls.

Grid Control

This is a general-purpose grid that can be used in any form, either holding its own data or virtually bound to an external data source. It acts as a base grid for the other two types of grids (the GridData and GridTree controls). Most features are shared among the three grid types.

In the Grid control, each cell acts as a single entity, which is suitable for applications such as Excel simulator, where the data in the grid cells are not interrelated and need to be maintained in the specific

cells themselves. You can also operate this control in virtual mode, where data is not stored in the grid's internal data structure but comes from an external source like a data table, for example. In virtual mode, data is dynamically loaded into the grid on demand or when users need to view the data.



GridData Control

The GridData control is designed to be bound with a data source. In the GridData control, each column behaves as a single entity. This grid is more column-centric and can be used to display interrelated tabular data. Unlike the base grid, this grid does not store data values in its data structures; instead, it is connected to an external data source.

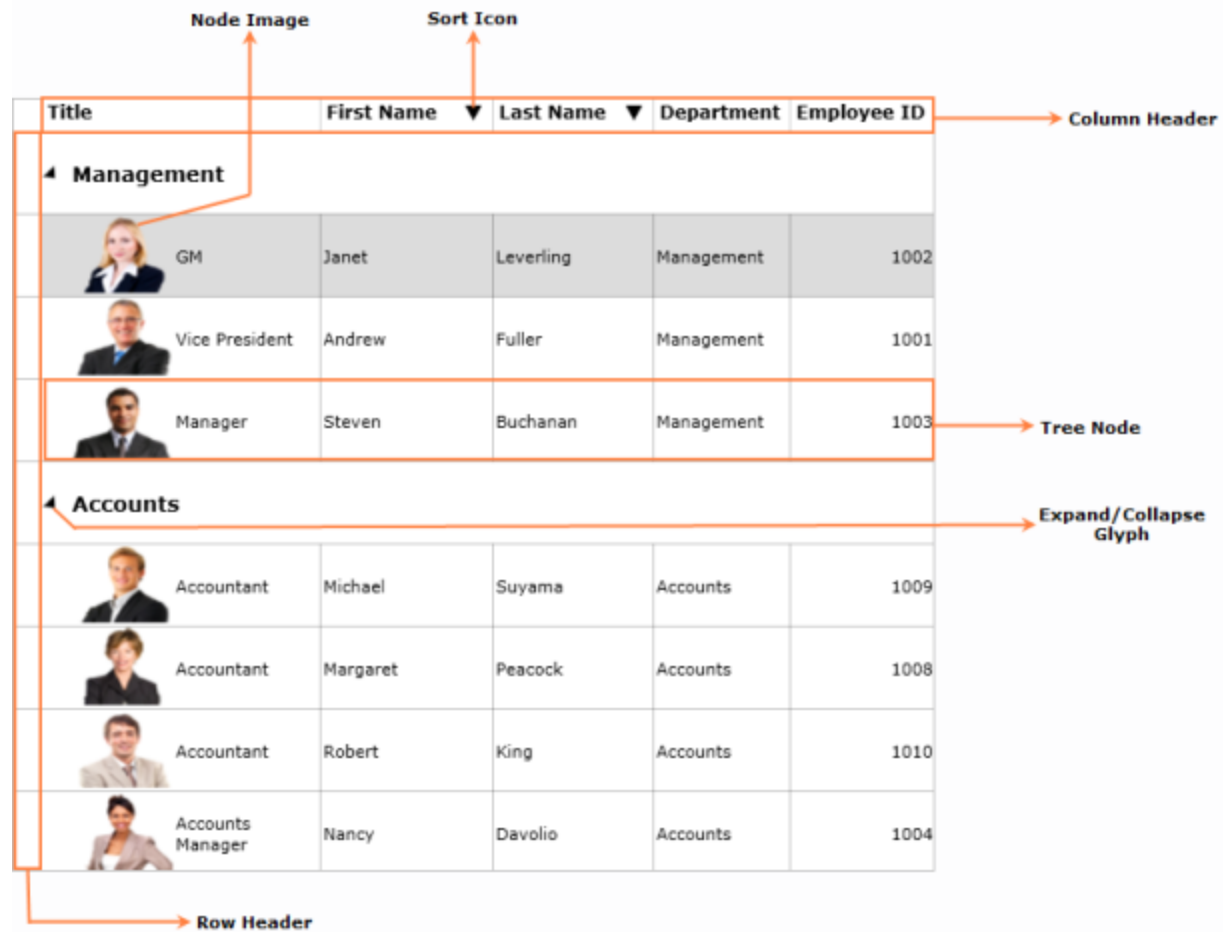
For more detailed information about data source connections, refer to the Data Binding section.

The screenshot displays a WPF GridDataControl (Classic) with a grouped data table. The table is organized into three groups based on the 'Customer ID' column. The first group, '7 Items', contains 7 rows of data. The second group, '4 Items', contains 4 rows of data. The third group, '13 Items', contains 2 rows of data. The table has columns for Order ID, Customer ID, Ship Name, Freight, and Order Date. The 'Customer ID' column is highlighted as the 'Grouped by header'. The 'Group Drop Area' is indicated by an arrow pointing to the 'Customer ID' column header. The 'Column Options Icon' is indicated by an arrow pointing to the gear icon in the 'Customer ID' column header. The 'Sort Icon' is indicated by an arrow pointing to the downward arrow icon in the 'Ship Name' column header. The 'Filter Icon' is indicated by an arrow pointing to the funnel icon in the 'Order Date' column header. The 'Group Caption' is indicated by an arrow pointing to the '7 Items' caption. The 'Record Row' is indicated by an arrow pointing to a row in the first group. The 'Summary Row' is indicated by an arrow pointing to the 'Total - 7 Items' row. The 'Expander Cell' is indicated by an arrow pointing to the plus-minus glyph in the 'Customer ID' column header of the first group.

Order ID	Customer ID	Ship Name	Freight	Order Date
7 Items				
10835	ALFKI	Alfred's Futterkiste	Rs. 69.53	09-12-1993
10952	ALFKI	Alfred's Futterkiste	Rs. 40.42	07-02-1994
11011	ALFKI	Alfred's Futterkiste	Rs. 1.21	03-03-1994
10702	ALFKI	Alfred's Futterkiste	Rs. 23.94	06-09-1993
10062	ALFKI	Alfred's Futterkiste	Rs. 47.22	22-08-1991
10643	ALFKI	Alfred's Futterkiste	Rs. 29.46	19-07-1993
10692	ALFKI	Alfred's Futterkiste	Rs. 61.02	27-08-1993
Total - 7 Items			273.00	
4 Items				
10759	ANATR	Ana Trujillo Emparedados y...	Rs. 11.99	22-10-1993
10926	ANATR	Ana Trujillo Emparedados y...	Rs. 39.92	26-01-1994
10308	ANATR	Ana Trujillo Emparedados y...	Rs. 1.61	12-08-1992
10625	ANATR	Ana Trujillo Emparedados y...	Rs. 43.90	02-07-1993
Total - 4 Items			97.00	
13 Items				
10573	ANTON	Antonio Moreno Taquería	Rs. 84.84	13-05-1993
10535	ANTON	Antonio Moreno Taquería	Rs. 15.64	06-04-1993

GridTree Control

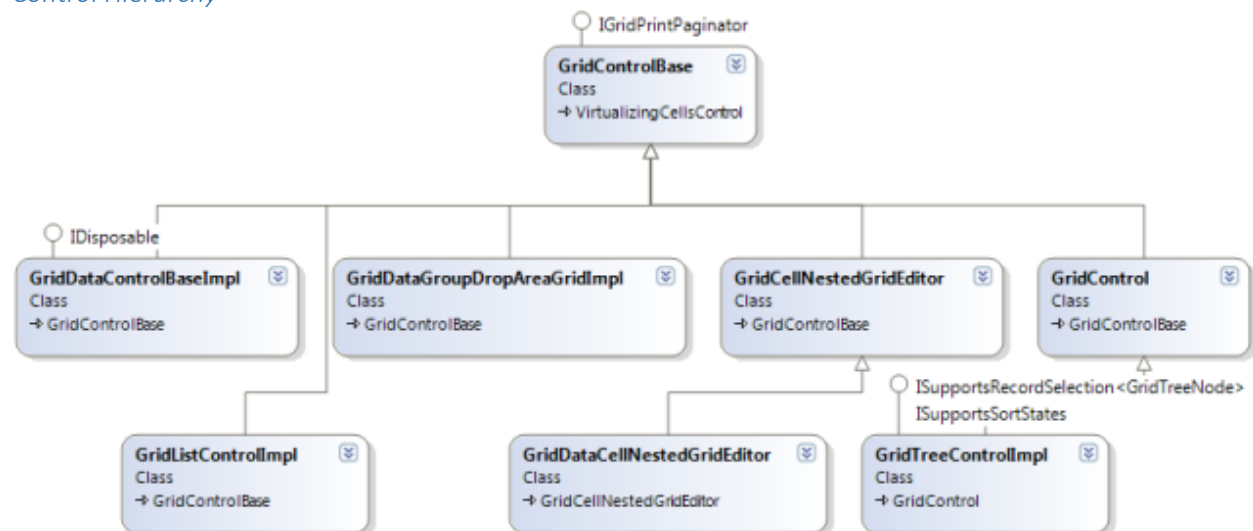
The GridTree control serves as a multicolumn tree control that is optimized to display thousands of items. This control uses a load-on-demand architecture to quickly generate a tree view. You can toggle the view of the underlying nodes by clicking the plus-minus glyphs of a root node. This control provides complete customization options such as custom level styles, glyphs, node images, and more.



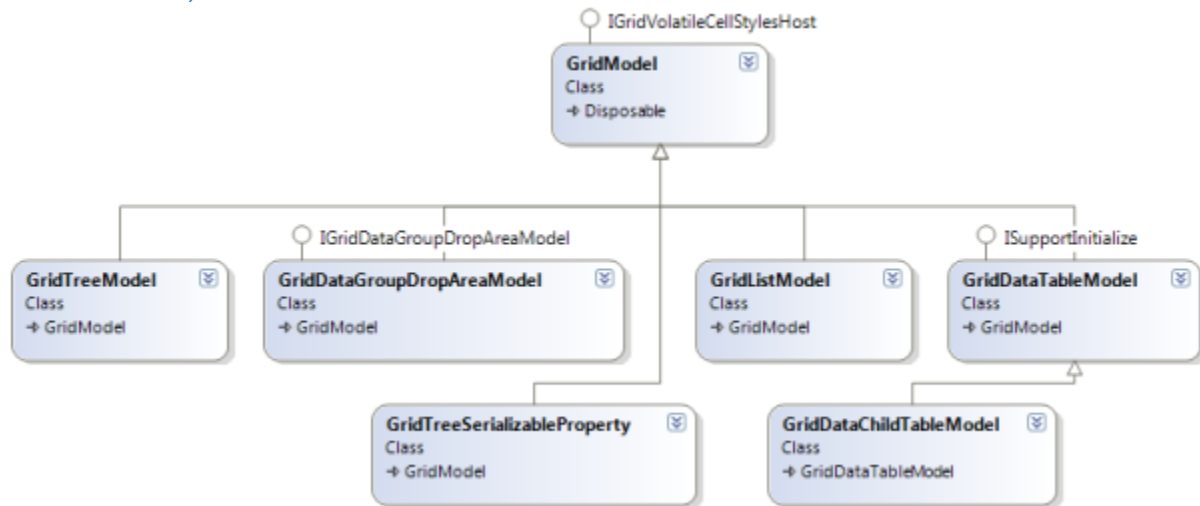
Class Diagram

The following illustration depicts the Class Diagram for Essential Grid for WPF.

Control Hierarchy



Model Hierarchy



Add Essential Grid to an Application

This section serves as a guide on how to deploy EssentialGrid in an application.

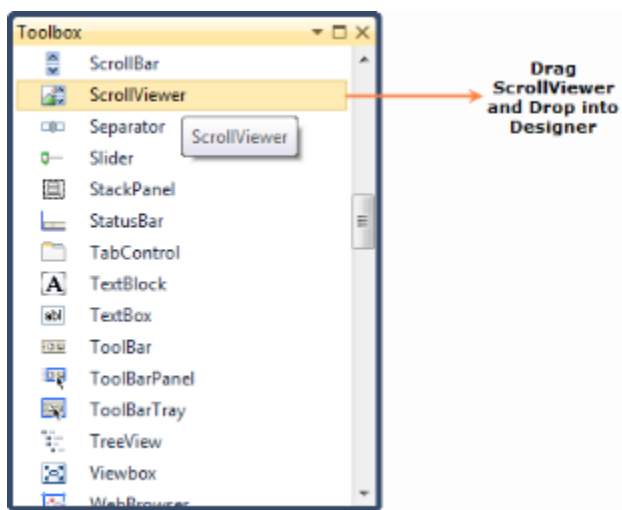
Add the Grid Control to a WPF Application

In this section, you can 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.

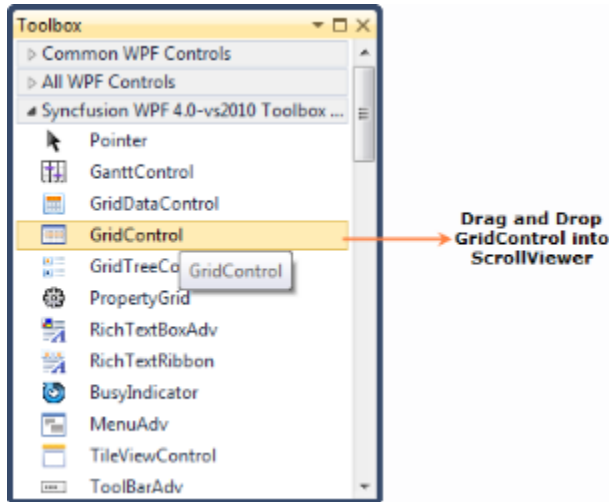
Adding the Grid Control through a Designer

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

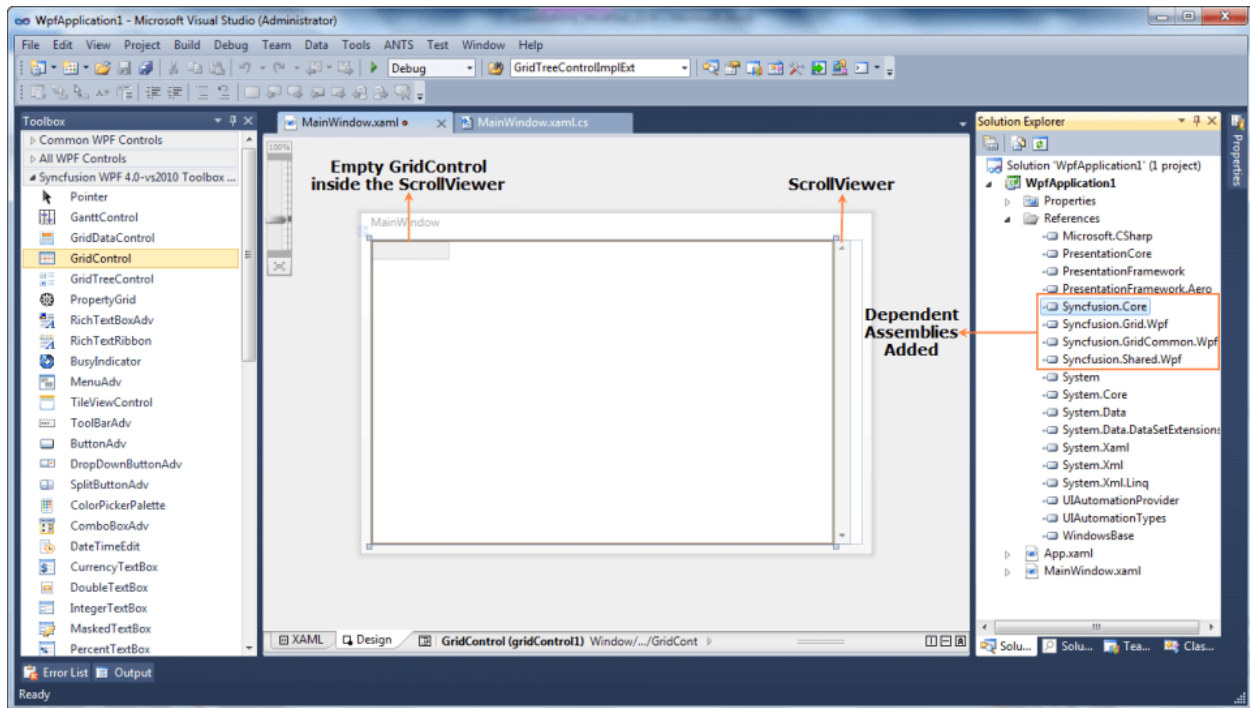
1. Create new WPF application.
2. Open the Designer window.
3. Drag ScrollViewer from the Toolbox and drop it in the Designer window (Since the Grid control doesn't have a built-in ScrollViewer, to make the grid flow based on data, the grid should be placed inside the ScrollViewer control).



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 is added to the designer and its dependent assemblies are added to the project.

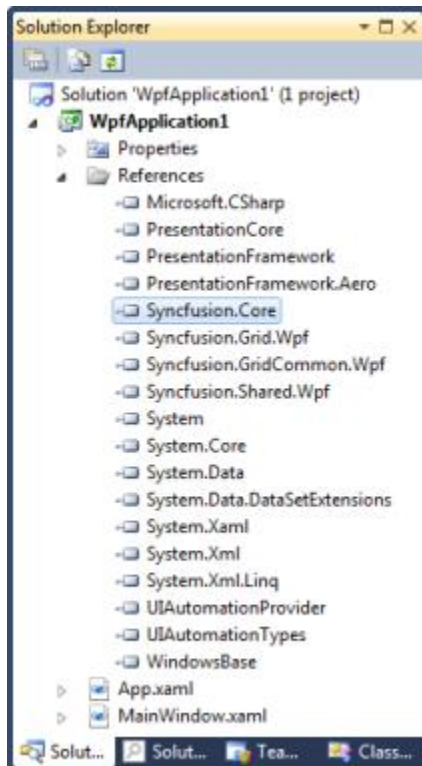


Programmatically Adding the Grid Control

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 Synfusion assemblies to the project.
3. Synfusion.Core.dll

4. Syncfusion.Grid.Wpf.dll
5. Syncfusion.GridCommon.Wpf.dll
6. Syncfusion.Shared.Wpf.dll



3. Name the root Grid as layoutRoot in the application's XAML page.

XML

```
<Grid Name="layoutRoot"/>
```

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

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.

CSHARP

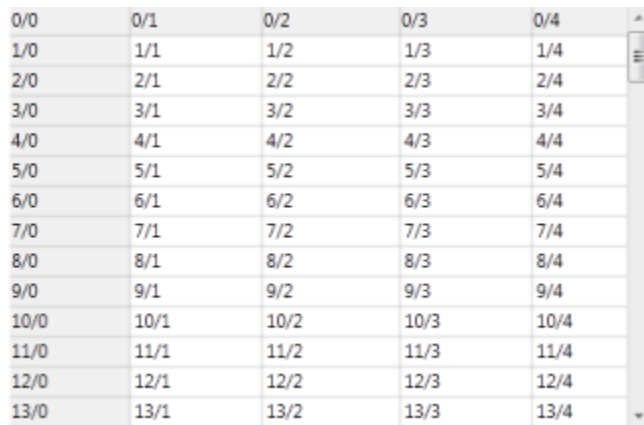
```
//Specifying row and column count
gridControl.Model.RowCount = 100;
gridControl.Model.ColumnCount = 20;
//Looping through the cells and assigning the values based on row and column
index
for (int i = 0; i < 100; i++)
{
    for (int j = 0; j < 20; j++)
    {
        gridControl.Model[i, j].CellValue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

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

CSHARP

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

3. Now, run the application. The grid appears as follows.



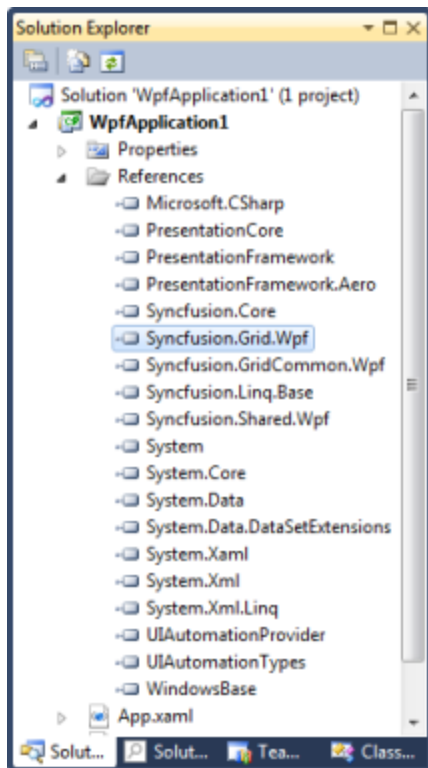
0/0	0/1	0/2	0/3	0/4
1/0	1/1	1/2	1/3	1/4
2/0	2/1	2/2	2/3	2/4
3/0	3/1	3/2	3/3	3/4
4/0	4/1	4/2	4/3	4/4
5/0	5/1	5/2	5/3	5/4
6/0	6/1	6/2	6/3	6/4
7/0	7/1	7/2	7/3	7/4
8/0	8/1	8/2	8/3	8/4
9/0	9/1	9/2	9/3	9/4
10/0	10/1	10/2	10/3	10/4
11/0	11/1	11/2	11/3	11/4
12/0	12/1	12/2	12/3	12/4
13/0	13/1	13/2	13/3	13/4

Add the GridData Control to a WPF Application

This section shows you how to add a GridData control to a WPF application and how to bind data from a database and an IEnumerable Collection to the GridData control. The GridData control can be added to an application through programmatically.

Programmatically Adding the GridData Control:

1. Create a new WPF application.
2. Add the following Syncfusion assemblies to the project.
3. Syncfusion.Core.dll
4. Syncfusion.Grid.Wpf.dll
5. Syncfusion.Linq.Base.dll
6. Syncfusion.GridCommon.Wpf.dll
7. Syncfusion.Shared.Wpf.dll



3. Name the root grid as layoutRoot in the application's XAML page.

CSHARP

```
<Grid Name="layoutRoot"/>
```

4. Create a new GridDataControl in code and add it as a child of layoutRoot (Grid). This adds GridDataControl to the view.

CSHARP

```
//GridDataControl defined here  
GridDataControl dataGrid = new GridDataControl();  
//To bring GridDataControl to the view, it should be  
added to the children of  
layoutRoot.  
layoutRoot.Children.Add(dataGrid);
```

Binding a Data Source to the GridData Control

The previous section explained how to add a GridData control to an application. This section explains how to bind a data source with the GridData control after it is added.

The GridData control supports all popular data sources including observable collections, data tables, collection view sources, business objects generated by the ADO.NET Entity Framework, LINQ to SQL or any other ORM.ADO.NET, data views, ICollection views, object data providers, IEnumerable, IList, IBindingList, IQueryable, and ITypedList.

This section explains how to bind data tables and IEnumerable collections to the GridData control.

Binding Data Tables to the GridData Control

This section explains how to bind a data table from a database to the GridData control.

1. Connect a database to the current application. The database can be connected several ways, such as ADO.NET, LNQ to SQL, classes, etc. In this example, we have directly established a connection to a simple Northwind.sdf database.
2. Once the connection is established, fetch the required data table from the database. Below, a data table has been fetched from the connected Northwind.sdf by using SQL DataAdapter.

Note: Check that System.data.SqlServerCe.dll has been added to your project before using this procedure.

CSHARP

```
public class Data
{
    public static DataTable GetDataTable()
    {
        //Connection string
        using (SqlCeConnection con = new SqlCeConnection(string.Format(@"Data Source
            = {0}", "C:\\Northwind.sdf"))) //Path where .sdf file is placed.
        {
            //Establish the connection
            con.Open();
            SqlCeDataAdapter sda = new SqlCeDataAdapter("SELECT * FROM Customers", con);
            DataTable dt = new DataTable();
            sda.Fill(dt);
            return dt;
        }
    }
}
```

3. Now bind the data table as an ItemsSource of GridDataControl, either in XAML or in code.

XML

```
<syncfusion:GridDataControl Name="dataGrid"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    ItemsSource="{Binding GDCSource}"/>
```

CSHARP

```
public MainWindow()
{
    InitializeComponent();
    this.DataContext = this;
}

public DataTable GDCSource
{
    get
```

```

{
    return Data.GetDataTable();
}
}
Assigning the Items' Source in Code
public MainWindow()
{
    InitializeComponent();
    //ItemsSource set to GridDataControl
    this.dataGrid.ItemsSource = Data.GetDataTable();
}

```

Once the application runs, the following output is generated.

Customer ID	Company Name	Contact Name	Contact Title
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedado...	Ana Trujillo	Owner
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager
BOLID	Bólido Comidas preparad...	Martin Sommer	Owner
BONAP	Bon app'	Laurence Lebihan	Owner

Samples

To view samples, follow these steps:

1. Select Start > Programs > Syncfusion > Essential Studio xx.x.x.xx > Dashboard.
2. Click Run Samples for WPF under the User Interface Edition panel.
3. Select GridDataControl.
4. Expand the Data Binding Features item in the Sample Browser.
5. Choose the Data Table Demo sample to launch.

Binding an IEnumerable Collection of GridDataControl

1. Create a collection of objects to bind with the GridData control. In this sample, we have created a collection of objects containing personnel information.

CSHARP

```

//The following is used to create a simple list collection
public class PersonDetails : List<Person>
{
    public PersonDetails()
    {
        this.Add(new Person() { CustomerID = "ALKI", CompanyName = "Alfreds Futterki
ste", ContactName = "Maria Anders", ContactTitle = "Sales Representative" });
    }
}

```

```

this.Add(new Person() { CustomerID = "ANATR", CompanyName = "Ana Trujillo Em
paredado", ContactName = "Ana Trujillo", ContactTitle = "Owner" });
this.Add(new Person() { CustomerID = "ANTON", CompanyName = "Antonio Moreno
Taquarea", ContactName = "Antonio Moreno", ContactTitle = "Owner" });
this.Add(new Person() { CustomerID = "ALKI", CompanyName = "Alfreds Futterki
ste", ContactName = "Maria Anders", ContactTitle = "Sales Representative" })
;
this.Add(new Person() { CustomerID = "AROUT", CompanyName = "Around the Horn
", ContactName = "Thomas Hardy", ContactTitle = "Sales Representative" });
this.Add(new Person() { CustomerID = "BERGS", CompanyName = "Berglunds snabb
kop", ContactName = "christinaBerglund", ContactTitle = "Sales Representativ
e" });
this.Add(new Person() { CustomerID = "BLONP", CompanyName = "Blondel pere et
fils", ContactName = "Frederique Citeaux", ContactTitle = "Marketing Manage
r" });
this.Add(new Person() { CustomerID = "BOLID", CompanyName = "Bolido Comidas
preparad", ContactName = "Martin Sommer", ContactTitle = "Owner" });
this.Add(new Person() { CustomerID = "BONAP", CompanyName = "Bon app", Conta
ctName = "Laurence Lebihan", ContactTitle = "Owner" });
}
}
public class Person
{
    public string CustomerID
    { get; set; }
    public string CompanyName
    { get; set; }
    public string ContactName
    { get; set; }
    public string ContactTitle
    { get; set; }
    public string Address
    { get; set; }
    public string City
    { get; set; }
    public string Region
    { get; set; }
}

```

2. Bind the collection as item sources of GridDataControl, either in XAML or in code.

XAML

```

<syncfusion:GridDataControl Name="dataGrid"
AutoPopulateColumns="True" ItemsSource="{Binding GDCSource}"/>

```

CSHARP

```

public MainWindow()
{
    InitializeComponent();
    this.DataContext = this;
}
private PersonDetails _gdcSource = new PersonDetails();

```

```

//This property sets as ItemsSource of GridDataControl
public PersonDetails GDCSource
{
    get
    {
        return _gdcSource;
    }
    set
    {
        _gdcSource = value;
    }
}
Assigning the Items Source in Code
public MainWindow()
{
    InitializeComponent();
    //ItemsSource set to GridDataControl
    this.dataGrid.ItemsSource = this.GDCSource;
}

```

When the application runs, the following output is generated.

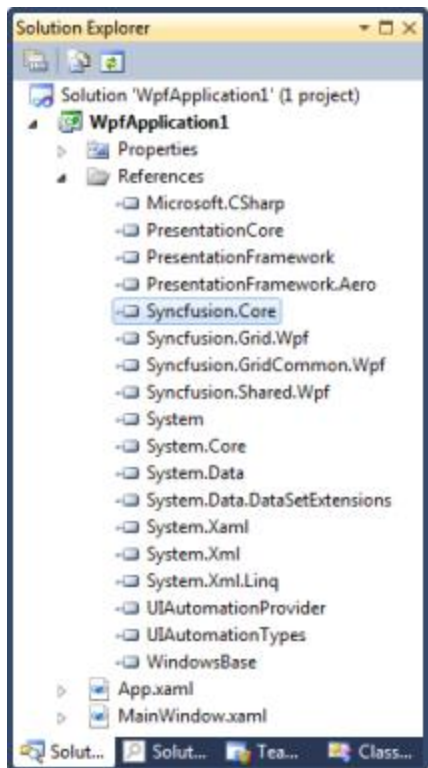
CustomerID	CompanyName	ContactName	ContactTitle
ALFK	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedado	Ana Trujillo	Owner
ANTON	Antonio Moreno Taqur...	Antonio Moreno	Owner
ALFK	Alfreds Futterkiste	Maria Anders	Sales Representative
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbkop	christinaBerglund	Sales Representative
BLONP	Blondel pere et fils	Frederique Citeaux	Marketing Manager
BOLID	BOlido Comidas prepar...	Martin Sommer	Owner
BONAP	Bon app	Laurence Lebihan	Owner

[Add the GridTree Control to a WPF Application](#)

This section demonstrates how to add a GridTree control to a WPF application and how to load the grid with a data source. The GridTree control can be added to an application through programmatically.

[Programmatically Adding GridTree Control](#)

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



3. Name the root grid as layoutRoot in the application's XAML page.

XML

```
<Grid Name="layoutRoot"/>
```

4. Create a new GridTreeControl in code and add it as a child of layoutRoot (Grid). Now GridTreeControl is added to the view.

XML

```
//GridTreeControl defined here  
GridTreeControl treeGrid = new GridTreeControl();  
//To bring the GridTreeControl to the view, GridTreeControl should be  
added to the children of layoutRoot.  
layoutRoot.Children.Add(treeGrid);
```

Data Population in the GridTree Control

The previous section explained how to add the GridTree control to an application. This section explains how to populate data in the GridTree control. There are three approaches to populating data:

- With the RequestTreeItems event.
- By self-relational collection binding.
- Using data-view binding.

The RequestTreeItems Event

The GridTree control can populate data on demand by handling the RequestTreeItems event. GridTreeControl receives the source of root and child nodes through this event handler. This event is triggered when initially loading and expanding nodes.

To populate data using this event, follow these steps:

1. Create a collection of objects to bind with the GridTree control. In this example, a collection of objects containing employee information has been created.

CSHARP

```
public class EmployeesCollection:List<Employee>
{
    public EmployeesCollection()
    {
        this.Add(new Employee() { Title = "Management", ReportsTo = -1, ID = 2 });
        this.Add(new Employee() { Title = "Accounts", ReportsTo = -1, ID = 3 });
        this.Add(new Employee() { Title = "Sales", ReportsTo = -1, ID = 4 });
        //Management
        this.Add(new Employee() { FirstName = "Andrew", LastName = "Fuller", Department = "Management", EmpID = 1001, ID = 9, Salary = 1200000, ReportsTo = 2, Title = "Vice President" });
        this.Add(new Employee() { FirstName = "Janet", LastName = "Leverling", Department = "Management", EmpID = 1002, ID = 10, Salary = 1000000, ReportsTo = 2, Title = "GM" });
        //Accounts
        this.Add(new Employee() { FirstName = "Nancy", LastName = "Davolio", Department = "Accounts", EmpID = 1004, ID = 12, Salary = 850000, ReportsTo = 3, Title = "Accounts Manager" });
        this.Add(new Employee() { FirstName = "Margaret", LastName = "Peacock", Department = "Accounts", EmpID = 1008, ID = 13, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
        //Sales
        this.Add(new Employee() { FirstName = "Laura", LastName = "Callahan", Department = "Sales", EmpID = 1005, ID = 16, Salary = 900000, ReportsTo = 4, Title = "Sales Manager" });
        this.Add(new Employee() { FirstName = "Anne", LastName = "Dodsworth", Department = "Sales", EmpID = 1011, ID = 17, Salary = 800000, ReportsTo = 4, Title = "Sales Representative" });
    }
}

public class Employee
{
    int id;
    public int ID
    {
        get { return id; }
        set { id = value; }
    }
    int? empId;
    public int? EmpID
    {
        get { return empId; }
        set { empId = value; }
    }
}
```

```
string firstName;
public string FirstName
{
    get { return firstName; }
    set { firstName = value; }
}
string lastName;
public string LastName
{
    get { return lastName; }
    set { lastName = value; }
}
string department;
public string Department
{
    get { return department; }
    set { department = value; }
}
private string title;
public string Title
{
    get { return title; }
    set { title = value; }
}
double? salary;
public double? Salary
{
    get { return salary; }
    set { salary = value; }
}
int reportsTo;
public int ReportsTo
{
    get { return reportsTo; }
    set { reportsTo = value; }
}
}
```

2. The RequestTreeItems event can hook in either XAML or code.

Hooking RequestTreeItems Event in XAML

XML

```
<syncfusion:GridTreeControl Name="treeGrid"
RequestTreeItems="treeGrid_RequestTreeItems"
EnableNodeSelection="False"
AutoPopulateColumns="True"
PercentSizingBehavior="SizeUntouchedColumns" >
```

Hooking RequestTreeItems Event in Code

C#

```
this.treeGrid.RequestTreeItems += new GridTreeRequestTreeItemsHandler(treeGrid
    RequestTreeItems);
```

3. Handle the RequestTreeItems event to pass the source to the root and child nodes dynamically.

CSHARP

```
EmployeesCollection employees;
public MainWindow()
{
    InitializeComponent();
    this.gridTreeControl1.RequestTreeItems += new Syncfusion.Windows.Controls.Gr
        id.GridTreeRequestTreeItemsHandler(treeGrid_RequestTreeItems);
    employees = new EmployeesCollection();
}
private void treeGrid_RequestTreeItems(object sender, GridTreeRequestTreeIte
    msEventArgs args)
{
    //When ParentItem is null, you need to set args.ChildList to be the root ite
    ms
    if (args.ParentItem == null)
    {
        //Get the root list-get all employees who have no boss
        //Get all employees whose boss' id is -1 (no boss)
        args.ChildList = employees.Where(x => x.ReportsTo == -1);
    }
    else //If ParentItem not null, then set args.ChildList to the child items fo
        r the given ParentItem.
    {
        //Get the children of the parent object
        Employee emp = args.ParentItem as Employee;
        if (emp != null)
        {
            //Get all employees that report to the parent employee
            args.ChildList = employees.Where(x => x.ReportsTo == emp.ID);
        }
    }
}
```

When the application runs, the following output is generated.

ID	EmpID	FirstName	LastName	Department	Title	Salary
2					Management	
9	1001	Andrew	Fuller	Management	Vice President	1200000
10	1002	Janet	Leverling	Management	GM	1000000
3					Accounts	
12	1004	Nancy	Davolio	Accounts	Accounts Ma...	850000
13	1008	Margaret	Peacock	Accounts	Accountant	700000
4					Sales	
16	1005	Laura	Callahan	Sales	Sales Manager	900000
17	1011	Anne	Dodsworth	Sales	Sales Represe...	800000

Samples

To view samples:

1. Select Start > Programs > Syncfusion > Essential Studio xx.x.x.xx > Dashboard.
2. Click Run Samples for WPF under the User Interface Edition panel.
3. Select GridTreeControl.
4. Expand the Data Population Features item in the Sample Browser.
5. Select On-Demand Loading Demo to launch the sample.

Binding a Self-Relational Collection to the GridTree Control

A self-relational collection is a collection of objects in which each object has a hierarchy within. Each object acts as a parent and hold its children in an attribute. Each child acts as the next-level parent and holds children in an attribute, and so on. In this example, both child and parent are of the same type (data type/object type). Specifying the child attribute name in ChildPropertyName of GridTreeControl automatically fetches the hierarchy and populate it.

1. Create a self-relational collection of objects to bind with the GridTree control. In this example, we have created a collection of objects containing employee information.

CSHARP

```
//This code is used to create a list collection of hierarchical data
public class EmployeeDetails : List<Employee>
{
    public EmployeeDetails()
    {
        //Management
        //The child list is the ChildCollection of the node
        List<Employee> childList = new List<Employee>();
        childList.Add(new Employee() { FirstName = "Andrew", LastName = "Fuller", Department = "Management", EmpID = 1001, ID = 9, Salary = 1200000, ReportsTo = 2, Title = "Vice President" });
        childList.Add(new Employee() { FirstName = "Janet", LastName = "Leverling", Department = "Management", EmpID = 1002, ID = 10, Salary = 1000000, ReportsTo = 2, Title = "GM" });
        childList.Add(new Employee() { FirstName = "Steven", LastName = "Buchanan", Department = "Management", EmpID = 1003, ID = 11, Salary = 900000, ReportsTo = 2, Title = "Manager" });
        this.Add(new Employee() { Title = "Management", ReportsTo = 1, ID = 2, Child = childList });
        //Accounts
        childList = new List<Employee>();
        childList.Add(new Employee() { FirstName = "Nancy", LastName = "Davolio", Department = "Accounts", EmpID = 1004, ID = 12, Salary = 850000, ReportsTo = 3, Title = "Accounts Manager" });
        childList.Add(new Employee() { FirstName = "Margaret", LastName = "Peacock", Department = "Accounts", EmpID = 1008, ID = 13, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
        childList.Add(new Employee() { FirstName = "Michael", LastName = "Suyama", Department = "Accounts", EmpID = 1009, ID = 14, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
    }
}
```

```
childList.Add(new Employee() { FirstName = "Robert", LastName = "King", Department = "Accounts", EmpID = 1010, ID = 15, Salary = 650000, ReportsTo = 3, Title = "Accountant" });
this.Add(new Employee() { Title = "Accounts", ReportsTo = 1, ID = 3, Child=hildList });
}
}
public class Employee
{
    int id;
    public int ID
    {
        get { return id; }
        set { id = value; }
    }
    int? empId;
    public int? EmpID
    {
        get { return empId; }
        set { empId = value; }
    }
    string firstName;
    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }
    string lastName;
    public string LastName
    {
        get { return lastName; }
        set { lastName = value; }
    }
    string department;
    public string Department
    {
        get { return department; }
        set { department = value; }
    }
    private string title;
    public string Title
    {
        get { return title; }
        set { title = value; }
    }
    double? salary;
    public double? Salary
    {
        get { return salary; }
        set { salary = value; }
    }
    int reportsTo;
    public int ReportsTo
    {
        get { return reportsTo; }
        set { reportsTo = value; }
    }
}
```

```
public List<Employee> Child
{
    get;
    set;
}
```

2. Bind ItemsSource of GridTreeControl and assign ChildPropertyName—these can be set in either XAML or code.

XML

```
<syncfusion:GridTreeControl Name="treeGrid"
AutoPopulateColumns="True"
ExpandStateAtStartup="AllNodesExpanded"
ChildPropertyName="Child"
ItemsSource="{Binding GTCSource}"/>
```

CSHARP

```
public MainWindow()
{
    InitializeComponent();
    this.DataContext = this;
    _gtcSource = new EmployeeDetails();
}
//This property is set as ItemsSource of GridTreeControl
private EmployeeDetails _gtcSource;
public EmployeeDetails GTCSource
{
    get
    {
        return _gtcSource;
    }
    set
    {
        _gtcSource = value;
    }
}
Assigning Items' Source Code
public MainWindow()
{
    InitializeComponent();
    //ItemsSource set to GridTreeControl
    this.treeGrid.ItemsSource = new EmployeeDetails();
}
```

When the application runs, the following output is generated.

ID	EmpID	FirstName	LastName	Department	Title
2					Management
9	1001	Andrew	Fuller	Management	Vice President
10	1002	Janet	Leverling	Management	GM
11	1003	Steven	Buchanan	Management	Manager
3					Accounts
12	1004	Nancy	Davolio	Accounts	Accounts Ma...
13	1008	Margaret	Peacock	Accounts	Accountant
14	1009	Michael	Suyama	Accounts	Accountant
15	1010	Robert	King	Accounts	Accountant

Samples

To view samples:

1. Select Start > Programs > Syncfusion > Essential Studio xx.x.x.xx > Dashboard.
2. Click Run Samples for WPF under User Interface Edition panel.
3. Select GridTreeControl.
4. Expand the Data Population Features item in the Sample Browser.
5. Select Self-Relational Data Binding Demo to launch the sample.

Binding a Data View to the GridTree Control

The following steps explain how to bind a data view from a database to the GridTree control.

1. Connect a database to the current application. The database can be connected several ways, such as ADO.NET, LINQ to SQL, classes, and so on. In this example, a connection has been directly established to a simple northwind.sdf database.
2. Once the connection is established, fetch the required data table from the database. In this example, the data table has been fetched from the northwind.sdf by using SQL DataAdapter.

Note: Before using this procedure, check that System.data.SqlServerCe.dll has been added to your project.

C# SHARP

```
// Connect to a data table
public DataTable GetDataTable()
{
    DataSet ds = new DataSet();
    if (!LayoutControl.IsInDesignMode)
    {
        using (SqlCeConnection con = new SqlCeConnection(connectionString))
        {
            con.Open();
            SqlCeDataAdapter sda = new SqlCeDataAdapter("SELECT * FROM Employees", con);
            sda.Fill(ds, "Employee");
        }
        //The following line is used to create the hierarchical relations
    }
}
```

```

ds.Relations.Add(new DataRelation("Employee_Relation", ds.Tables["Employee"]
.Columns["Employee ID"], ds.Tables["Employee"].Columns["Reports To"], false))
;
}
if (ds.Tables.Count > 0)
return ds.Tables[0];
else
return null;
}

```

3. Now bind the data table as an ItemsSource of GridTreeControl in either XAML or code.

XML

```

<syncfusion:GridTreeControl Name="treeGrid"
AutoPopulateColumns="True"
ItemsSource="{Binding GTCSource}"
ExpandStateAtStartup="AllNodesExpanded"
ChildPropertyName="Employee_Relation" />

```

CSHARP

```

public MainWindow()
{
InitializeComponent();
this.DataContext = this;
dataTable = GetDataTable();
}
DataTable dataTable;
public DataView GTCSource
{
get
{
if (dataTable == null)
return null;
DataView dataView = new DataView(dataTable);
//RowFilter is applied to form the hierarchy
dataView.RowFilter = "[Reports To] Is NULL";
return dataView;
}
}
Assigning the Items' Source in Code
public MainWindow()
{
InitializeComponent();
//Relation name set as ChildPropertyName
this.treeGrid.ChildPropertyName = "Employee_Relation";
dataTable = GetDataTable();
//ItemsSource set to GridTreeControl
this.treeGrid.ItemsSource = GTCSource;
}
DataTable dataTable;
public DataView GTCSource
{

```

```

get
{
    if (dataTable == null)
        return null;
    DataView dataView = new DataView(dataTable);
    dataView.RowFilter = "[Reports To] Is NULL";
    return dataView;
}
}

```

When the application runs, the following output is generated.

Employee ID	Last Name	First Name	Title	Birth Date	Hire Date
2	Fuller	Andrew	Vice Preside...	19-02-1942 0...	12-07-1991 0...
1	Davolio	Nancy	Sales Repres...	08-12-1948 0...	29-03-1991 0...
3	Leverling	Janet	Sales Repres...	30-08-1963 0...	27-02-1991 0...
11	Smith	Tim	Mail Clerk	06-06-1973 0...	15-01-1993 0...
12	Patterson	Caroline	Receptionist	11-09-1972 0...	15-05-1993 0...
4	Peacock	Margaret	Sales Repres...	19-09-1937 0...	30-03-1992 0...
5	Buchanan	Steven	Sales Manag...	04-03-1955 0...	13-09-1992 0...
6	Suyama	Michael	Sales Repres...	02-07-1963 0...	13-09-1992 0...
7	King	Robert	Sales Repres...	29-05-1960 0...	29-11-1992 0...
9	Dodsworth	Anne	Sales Repres...	27-01-1966 0...	12-10-1993 0...
8	Callahan	Laura	Inside Sales...	09-01-1958 0...	30-01-1993 0...
10	Hellstern	Albert	Business Ma...	13-03-1960 0...	01-03-1993 0...

Sample

To view samples:

1. Select Start > Programs > Syncfusion > Essential Studio xx.x.x.xx > Dashboard.
2. Click Run Samples for WPF under User Interface Edition panel.
3. Select GridTreeControl.
4. Expand the Data Population Features item in the Sample Browser.
5. Select Data View Binding Demo to launch the sample.

GridDataControl in WPF GridDataControl (Classic)

The GridData control is specifically designed for the scenarios, where you need to bound the grid to an external data source and customize the data view by performing the operations such as grouping, sorting, summarizing, filtering, conditional formats and unbound fields. It attains the fundamental features by deriving from the GridControlBase class and hence adapts most of the features of Grid control, which are discussed in the previous sections. It can display nested grids with hierarchical data and can also display multiple unrelated tables in one grid.

Major Control Classes

- GridDataControl (GDC) is the main control class of this control. It is templated with GridDataControlBaselImpl class that is based on GridControlBase class. GDC exposes numerous properties and methods that are available for the end-user to setup the grid in the desired manner.

- The GridDataTableModel serves as the model class for GDC. It stores all the data information of the grid and provides methods to completely initialize the grid. It also provides methods to attach the grid later to the GDC, for rendering.
- The GridDataTableProperties defines property values for the Grid Table that lets you customize the appearance and behavior of the GDC.
- As a data bound grid, GDC displays tabular data where each row corresponds to a data record and every column stands for data field of the data source. You wrap all the columns in a group together and manipulate them. The GridDataVisibleColumn class is used for this purpose. It groups the columns that are visible in the screen and holds information about each column.
- The GridDataStyleInfo class, which is derived from GridStyleInfo, provides user-friendly access to all the cell level properties that control the appearance of the cell. It holds all the information of the cell.
- GridDataTable provides the virtual representation of the entity set and manages all the underlying records in the data source.
- GridDataGroupDropAreaGridImpl is the class that defines the group drop area for a GDC. It displays a panel on the top of the GDC, where the user can drag-and-drop any column header in order to group the grid against that column. GridDataGroupDropAreaModel is the model class for this purpose.

Note: There are several other classes that assist the user in creating groups, summaries, filters, sorted columns, etc.

The following sections elaborate the properties of the GridData control:

- Data Binding-Elaborates on the data binding concept in GDC
- Data Presentation-Discusses different data presentation techniques
- Events- Events that are handled in GDC are discusses here
- Look and Feel-Different appearance settings are elaborated here
- Exporting GDC to Excel-Discusses the steps to export a GDC into the Excel
- Performance-High performance of the Grid with large amount of data is discussed in this section
- Real Time Application-Illustrates how to employ the grid in portfolio applications

Data Model in WPF GridDataControl (Classic)

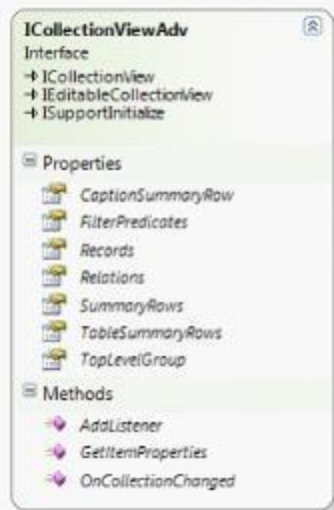
This section has the following topics:

ICollectionViewAdv

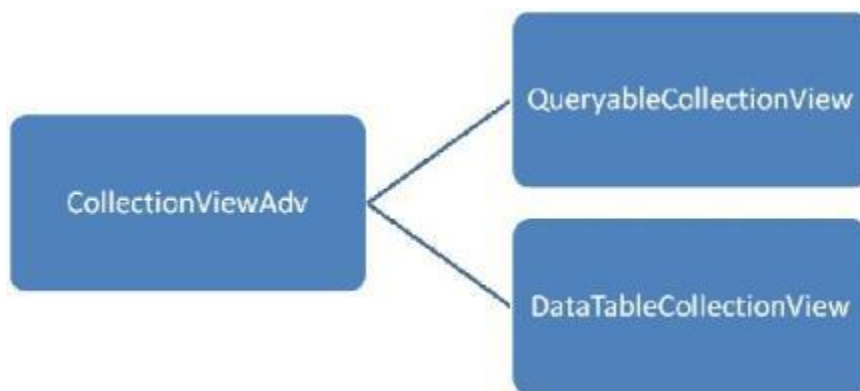
The ICollectionViewAdv interface is an extended ICollectionView interface that includes support for the following:

- Grouping structure—Binary tree data structure to maintain the groups.
- Group summaries—Collection to specify group summaries.
- Caption summary—Specifies caption summary row.
- Records structure—Flat data structure to maintain the list of internal records.
- Table summaries—Collection to specify table summaries.
- Filter definitions—Collection to specify filter descriptors.

ICollectionViewAdv interface implements the following:



ICollectionViewAdv is implemented as two parts in Syncfusion.Linq.Base library, as shown below:



The CollectionViewAdv is an abstract class implementing ICollectionViewAdv. By sub-classing the CollectionViewAdv, the specific actions for the following can be defined:

- Sorting
- Filtering
- Grouping
- Summaries

QueryableCollectionView - Implements an IQueryable way to provide Sorting, Filtering, Grouping and Summaries.

DataTableCollectionView – Uses the DataView to provide Sorting/Filtering and uses custom logics to implement grouping and calculating summaries.

Grouping in ICollectionViewAdv

To specify groups, add GroupDescriptions to ICollectionViewAdv.GroupDescriptions. The following code example shows adding groups to Northwind database.

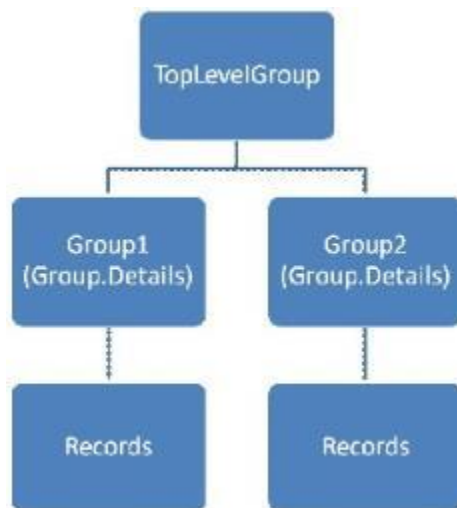
C#

```
var northwind = new Northwind("Data Source = Northwind.sdf");
var orders = northwind.Orders;
```

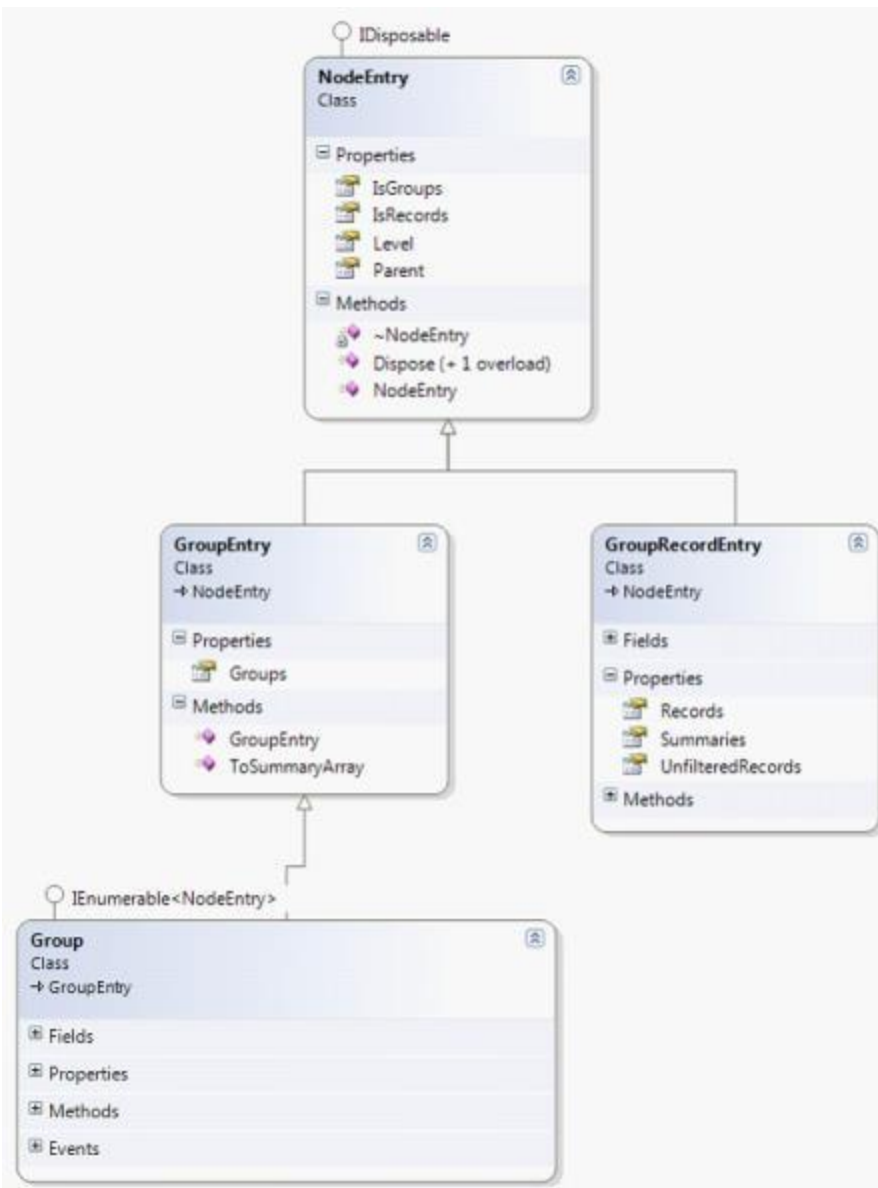


```
var queryableCollectionView = new QueryableCollectionView(orders);  
queryableCollectionView.GroupDescriptions.Add(new  
PropertyGroupDescription("ShipCountry"));  
queryableCollectionView.GroupDescriptions.Add(new  
PropertyGroupDescription("ShipCity"));
```

The grouping values are stored in a binary tree structure in the `ICollectionViewAdv.TopLevelGroup`. The visual graph of the `TopLevelGroup` is as follows:



The class diagram for the structure above is as follows:



- **NodeEntry** – It is the base class for all the nodes in the binary tree.
- **GroupEntry** – Contains a list of groups.
- **GroupRecordEntry** – Contains a list of records.
- **Group** – Extended Group entry that has implementation for populating and structuring the groups and its sub- groups. It can store both a list of Groups or Records. If the Group is a **BottomLevelGroup** then the **Group.Details** would contain the **GroupRecordEntry**.

Iterating through the whole structure

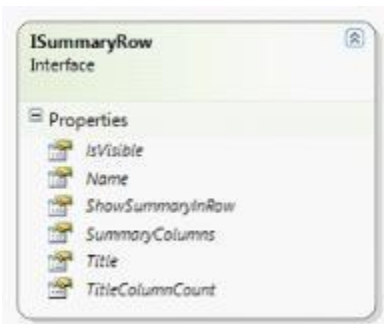
The **TopLevelGroup** is an extended **Group** class. Iteration through the whole structure can be performed by looping through the structure using a **foreach** loop, since the **Group** already has an **Enumerator** implemented.

C#

```
var northwind = new Northwind("Data Source = Northwind.sdf");
var orders = northwind.Orders;
var queryableCollectionView = new QueryableCollectionView(orders);
queryableCollectionView.GroupDescriptions.Add(new
PropertyGroupDescription("ShipCountry"));
foreach (var nodeEntry in queryableCollectionView.TopLevelGroup)
{
    Console.WriteLine(nodeEntry);
}
```

Summaries in Grouping

The ICollectionViewAdv interface exposes an ObservableCollection<ISummaryRow> and CaptionSummary.



Any class implementing this interface should be able to interact with the TopLevelGroup for specifying summaries.

Note: The ICollectionViewAdv interface was made as an interface since the WPF properties had to be DependencyProperty to enable binding in XAML. With ICollectionViewAdv, we can specify two kinds of summaries:

- Group summaries.
- Caption summary.

Group Summaries

Specifies summaries added for each bottom level group. Access each summary node from GridRecordEntry.Summaries.

Caption Summary

Specifies a caption summary row required by an UI control to display the group caption. Access the caption summary as Group.SummaryRecordEntry.

Sorting in ICollectionViewAdv

Adding SortDescriptions to the ICollectionView would sort the internal data.

C#

```
var orders = northwind.Orders;
var queryableCollectionView = new QueryableCollectionView(orders);
queryableCollectionView.SortDescriptions.Add(new
System.ComponentModel.SortDescription("CustomerID",
System.ComponentModel.ListSortDirection.Descending));
foreach (var record in queryableCollectionView.Records)
```

```
{  
var order = (Orders)record.Data;  
Console.WriteLine("OrderID - {0} / CustomerID - {1}", order.OrderID,  
order.CustomerID);  
}
```

Sorting with Grouping

The `TopLevelGroup` works in conjunction with the sort descriptions present in the `ICollectionView`. It would automatically sort the groups and its bottom level group (records).

C#

```
var orders = northwind.Orders;  
var queryableCollectionView = new QueryableCollectionView(orders);  
queryableCollectionView.SortDescriptions.Add(new  
System.ComponentModel.SortDescription("CustomerID",  
System.ComponentModel.ListSortDirection.Descending));  
queryableCollectionView.GroupDescriptions.Add(new  
PropertyGroupDescription("ShipCountry"));  
queryableCollectionView.GroupDescriptions.Add(new  
PropertyGroupDescription("ShipCity"));  
foreach (var nodeEntry in queryableCollectionView.TopLevelGroup)  
{  
Console.WriteLine(nodeEntry);  
}  
foreach (var record in queryableCollectionView.Records)  
{  
var order = (Orders)record.Data;  
Console.WriteLine("OrderID - {0} / CustomerID - {1}", order.OrderID,  
order.CustomerID);  
}
```

Note: When applying sorting with grouping, both the `ICollectionViewAdv.TopLevelGroup` and the `ICollectionViewAdv.Records` is in sync.

Data Binding in WPF GridDataControl (Classic)

Data binding is the master feature of the GridData control. Grid must be bound to an external data source to display the data. GDC supports the following data sources such as, Data Tables, Data Sets or Custom collections of type List, Binding List, Observable Collection or Collection View Source. These data source can have multiple nested tables that is displayed hierarchically by the grouping grid.

Data Binding mechanisms

Following are the data binding mechanisms:

- Using Data Providers-Object Data Provider, XML Data Provider and usage of Data Context
- Using ADO.NET Data-Data Table, Data Set and Data Row
- Using Business Objects-List, Binding List, Observable Collection, Collection View Source
- XAML Binding-Elaborates on data binding using XAML code
- Notify Property Changes-Elaborates on notifying the underlying data source changes to the grid
- Data Error Validation-Discusses on the support to validate the grid data and display error information

- Synchronize Current Selection-Discusses about synchronization of changes in the current with another control
- Unbound Columns-Discusses on the addition of unbound columns to the grid

Important Data Binding Properties

The following table contains some data binding properties and their corresponding descriptions:

Property

Property	Description
DataContext	Gets or sets the data context for binding. It simplifies the data binding.
ItemsSource	Binds the grid to a collection object.
AutoPopulateColumns	When set to true, it extracts the column from the data set and populates the Grid automatically.
AutoPopulateRelations	When set to true, it extracts the relation from the data set and populates the Grid automatically.
AutoGenerateColumnsInfo	When set to true, this property assigns the cell editor to the column depending on the content. For example, some types and its corresponding cell types are listed below:String â€œ Text BoxBool â€œ CheckboxUri â€œ HyperlinkDouble â€œ DoubleEditDatetime â€œ DateTimeEditEnums â€œ ComboBoxTimespan â€œ TimeSpanEdit

Data Providers

An ObjectDataProvider is a class which creates an object that you can use as a binding source. The GridData control supports this class (offered by WPF platform) that creates an object in the XAML code and can be used for data binding. The ObjectDataProvider allows you to specify binding expressions against an object and its methods. You can also write custom data providers, if required.

Example

Here is an example that illustrates how to use Object Data Provider with GDC.

Say, your data source is defined in C# class named Order, and it queries the records from Northwind Orders table. Then the respective Object Data Provider definition is given by the code below:

Note: The ObjectType attribute of ObjectDataProvider should point to the data source you defined.

XML

```
<Window.Resources>
<ObjectDataProvider x:Key="order" ObjectType="{x:Type local:Order}">
</Window.Resources>
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource order}" >
```

Note: The following line in the above code references the Object (Order), which returns the data for binding.

XML

```
<ObjectDataProvider x:Key="order" ObjectType="{x:Type local:Order}"
```

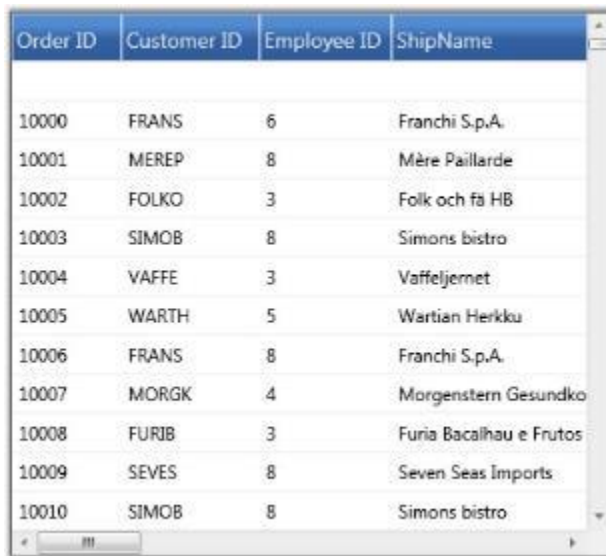
Defining the Order Class

The Order class returns the data to be bound to the Grid as shown in the following code:

C#

```
public class Order : ObservableCollection<Orders>
{
    Northwind northWind;
    public Order()
    {
        string connectionString = string.Format(@"Data Source = {0}",
            "Northwind.sdf");
        northWind = new Northwind(connectionString);
        var order = northWind.Orders;
        foreach (var o in order)
        {
            this.Add(o);
        }
    }
}
```

The following screenshot shows a GDC which bound with an Object data using the Object Data Provider.



Order ID	Customer ID	Employee ID	ShipName
10000	FRANS	6	Franchi S.p.A.
10001	MEREP	8	Mère Paillarde
10002	FOLKO	3	Folk och få HB
10003	SIMOB	8	Simons bistro
10004	VAFFE	3	Vaffeljernet
10005	WARTH	5	Wartian Herkku
10006	FRANS	8	Franchi S.p.A.
10007	MORGK	4	Morgenstern Gesundko
10008	FURJB	3	Furia Bacalhau e Frutos
10009	SEVES	8	Seven Seas Imports
10010	SIMOB	8	Simons bistro

The GDC is bound with a data source provided by an object.

Database Data

ADO.NET is an object-oriented set of libraries that allows you to interact with different types of data sources and different types of databases. It is used by data-oriented applications for connecting to the data sources and manipulating data. It owns a set of data providers that retrieves data from the underlying sources and place it in an ADO.NET DataSet object. The DataSet object, which holds a collection of ADO.NET DataTable objects (represents the tables of data source), is created to populate a data-aware control like the grid with database data. These DataSet objects can operate independent of the .NET data providers.

Here is an example that illustrates the binding of ADO.NET Data Table with the GridData control.

Code that Sets Up a DataTable

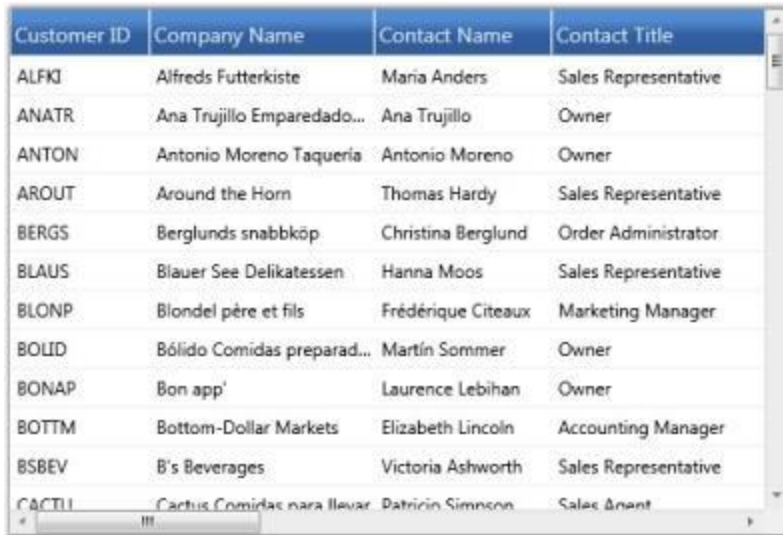
C#

```
public class Data
{
    public static DataTable
    GetDataTable()
    {
        using (SqlCeConnection con = new SqlCeConnection(string.Format(@"Data Source
= {0}", LayoutControl.FindFile("Northwind.sdf"))))
        {
            con.Open();
            SqlCeDataAdapter sda = new SqlCeDataAdapter("SELECT * FROM Customers", con);
            DataSet ds = new DataSet();
            sda.Fill(ds);
            return ds.Tables[0];
        }
        return new DataTable();
    }
}
```

Code to Bind the Above DataTable to GDC.

XML

```
<Window.Resources>
<ObjectDataProvider x:Key="CustomerTable" MethodName="GetDataTable"
ObjectType="{x:Type local:Data}" />
<CollectionViewSource x:Key="orderSource" Source="{StaticResource
CustomerTable}" >
</CollectionViewSource>
</Window.Resources>
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ShowAddNewRow="False"
ItemsSource="{StaticResource orderSource}">
</syncfusion:GridDataControl>
```



Customer ID	Company Name	Contact Name	Contact Title
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedado...	Ana Trujillo	Owner
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager
BOLID	Bólido Comidas preparad...	Martín Sommer	Owner
BONAP	Bon app'	Laurence Leblan	Owner
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative
CACTI	Cactus Comidas para llevar	Patricio Simmon	Sales Agent

The GDC is bound with a data source provided by Data Table.

Business Objects

The grid can be bound to user-defined collections that let you store arbitrary objects in a structured fashion. The data binding is based on a set of interfaces that differ in the context of accessing and navigating through data and of course manipulating the data. These interfaces set up a two-way communication between the bound grid and the objects collection used by the same grid. Such object collections are called as custom business collections.

The data binding interfaces allow you to create collection of custom objects where you want to present those collection of objects together, through the grid. You can also navigate through the objects to view them through the same grid and interact with them. Some of these interfaces are *IList*, *IBindingList*, and so on. For ease of use, .NET provides built-in, ready-to-use collection classes that internally implements these collection interfaces.

Some of the collection classes are as follows:

- *List* that implements *IList*
- *BindingList* that implements *IBindingList*
- *ObservableCollection*.

Of the above classes, the *ObservableCollection* is widely preferred as it is more user-friendly to use in a WPF application, and can be easily created by using XAML.

Let us see an example usage of this collection class with our GridData control.

Example

Defining Observable Collection

C#

```
public class Customer : ObservableCollection<Customers>
{
    Northwind northWind;
    public Customer()
    {
```



```

string connectionString = string.Format(@"Data Source = {0}",
Northwind.sdf");
northWind = new Northwind(connectionString);
var customer = northWind.Customers.Skip(0).Take(100).ToList();
foreach (var o in customer)
{
    this.Add(o);
}
}
}

```

Binding the Above Collection to GDC.

XML

```

<Window.Resources>
<ObjectDataProvider x:Key="customer" ObjectType="{x:Type local:Customer}" />
</Window.Resources>
<syncfusion:GridDataControl x:Name="grid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource customer}">
</syncfusion:GridDataControl>

```

The following image shows a Grid bound with an Observable collection.



CustomerID	CompanyName	ContactName	ContactTitle
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedado...	Ana Trujillo	Owner
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager
BOLID	Bólido Comidas preparad...	Martin Sommer	Owner

The GDC is bound with a data source provided by Observable Collection.

Collection View Source

The CollectionViewSource[CVS] serves as a wrapper for the custom collections. It acts as a filter between the source collection (ObservableCollection or List or BindingList) and the grid. Once your grid is bound to CollectionViewSource-driven source list, the CVS manages all the data related operations such as grouping, sorting, filtering and so on. This relieves you of writing custom code for these operations. CVS does this by internally implementing the ICollectionView interface that understands the source type and manages the data operations. It also implements INotifyCollectionChanged interface. This means that if CVS is linked to a source collection (ObservableCollection), then all the updates to the source list is transmitted to the grid.

Example

Let us see an example usage of this Collection View Source with our GDC.

XML

```

<Window.Resources>
<ObjectDataProvider x:Key="customer" ObjectType="{x:Type local:Customer}" />
<CollectionViewSource x:Key="customerSource" Source="{StaticResource
customer}" >
</CollectionViewSource>
</Window.Resources>
<syncfusion:GridDataControl x:Name="grid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource customerSource}">
</syncfusion:GridDataControl>

```

Defining the Source Collection

C#

```

public class Customer : ObservableCollection<Customers>
{
    Northwind northWind;
    public Customer()
    {
        string connectionString = string.Format(@"Data Source = {0}",
"Northwind.sdf");
        northWind = new Northwind(connectionString);
        var customer = northWind.Customers.Skip(0).Take(100).ToList();
        foreach (var o in customer)
        {
            this.Add(o);
        }
    }
}

```

The following image shows the output of the above given code:



CustomerID	CompanyName	ContactName	ContactTitle
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedado...	Ana Trujillo	Owner
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager
BOLID	Bólido Comidas preparad...	Martin Sommer	Owner

The GDC is bound with a data source provided by Collection View Source.

Entity Collection

GridDataControl confirms that it is an EntityFramework-aware control by providing support to EF-driven data sources. The EF model offers several architectural benefits that can be experienced in the networking applications.

To bind your grid to an EF-driven data source, use the below code.

C#

```

string connectionString = string.Format(@"Data Source = {0}",
LayoutControl.FindFile("NorthwindGrid.sdf"));
EntityConnectionStringBuilder entityBuilder = new
EntityConnectionStringBuilder();
entityBuilder.Metadata =
"res://*/NorthWind.csdl|res://*/NorthWind.ssdl|res://*/NorthWind.msl";
entityBuilder.Provider = "System.Data.SqlClient";
entityBuilder.ProviderConnectionString = connectionString;
NorthWind northwind = new NorthWind(entityBuilder.ToString());
this.gridDataControl1.ItemsSource = northwind.Orders;

```

OrderID	OrderDate	RequiredDate	ShippedDate
10248	7/4/1996 12:00:00 AM	8/1/1996 12:00:00 AM	7/16/1996 12:00:00 AM
10249	7/5/1996 12:00:00 AM	8/16/1996 12:00:00 AM	7/10/1996 12:00:00 AM
10250	7/8/1996 12:00:00 AM	8/5/1996 12:00:00 AM	7/12/1996 12:00:00 AM
10251	7/8/1996 12:00:00 AM	8/5/1996 12:00:00 AM	7/15/1996 12:00:00 AM
10252	7/9/1996 12:00:00 AM	8/6/1996 12:00:00 AM	7/11/1996 12:00:00 AM
10253	7/10/1996 12:00:00 AM	7/24/1996 12:00:00 AM	7/16/1996 12:00:00 AM
10254	7/11/1996 12:00:00 AM	8/8/1996 12:00:00 AM	7/23/1996 12:00:00 AM
10255	7/12/1996 12:00:00 AM	8/9/1996 12:00:00 AM	7/15/1996 12:00:00 AM
10256	7/15/1996 12:00:00 AM	8/12/1996 12:00:00 AM	7/17/1996 12:00:00 AM
10257	7/16/1996 12:00:00 AM	8/13/1996 12:00:00 AM	7/22/1996 12:00:00 AM
10258	7/17/1996 12:00:00 AM	8/14/1996 12:00:00 AM	7/23/1996 12:00:00 AM
10259	7/18/1996 12:00:00 AM	8/15/1996 12:00:00 AM	7/25/1996 12:00:00 AM

XAML Binding

Essential Grid allows you to specify the data source in XAML. XAML binding can be established with the help of Binding class, which provides high level access to the definition of binding to connect the properties of the target binding object to any data source. The specification of a binding in XAML is referred to as Binding Expression. Each binding typically has four components:

- Target object-It corresponds to GridDataControl object
- Target Property-It corresponds to GridDataControl.ItemsSource property
- Binding Source-It can be a list object, a UI element or an object data provider
- Path-It specifies a value in the binding source to be used

Binding Modes

The Binding class also provides you with various modes that let you control how source and target objects of the binding can update each other. Following are the different modes of binding:

- OneWay-Updates the target property only when the source property changes
- TwoWay-Updates the target property when the source property changes and vice versa
- OneTime-Updates the target property only when the application starts or when the Data Context undergoes a change

- OneWayToSource-Updates the source property when the target property changes

Example

Following is a simple XAML binding code that specifies a data source for the grid. It binds the GridDataControl.ItemsSource property to a CollectionViewSource object, which in turn points to an ObservableCollection.

1. Code behind to Create an Observable Collection

CSHARP

```
public class Customer : ObservableCollection<Customers>
{
    Northwind northWind;
    public Customer()
    {
        string connectionString = string.Format(@"Data Source = {0}",
        LayoutControl.FindFile("Northwind.sdf"));
        northWind = new Northwind(connectionString);
        var customer = northWind.Customers.Skip(0).Take(100).ToList();
        foreach (var o in customer)
        {
            this.Add(o);
        }
    }
}
```

2. Data Source Definition

XML

```
<Window.Resources>
<ObjectDataProvider x:Key="customer" ObjectType="{x:Type local:Customer}" />
<CollectionViewSource x:Key="customerSource" Source="{StaticResource
customer}" >
</CollectionViewSource>
</Window.Resources>
```

3. XAML Binding

XML

```
<syncfusion:GridDataControl x:Name="grid" AutoPopulateColumns="True"
AutoPopulateRelations="False"
ItemsSource="{Binding Source={StaticResource customerSource}}">
</syncfusion:GridDataControl>
```

The following image shows the output of the above given code:

CustomerID	CompanyName	ContactName
ALFKI	Alfreds Futterkiste	Maria Anders
ANATR	Ana Trujillo Emparedado...	Ana Trujillo
ANTON	Antonio Moreno Taquería	Antonio Moreno
AROUT	Around the Horn	Thomas Hardy
BERGS	Berglunds snabbköp	Christina Berglund
BLAUS	Blauer See Delikatessen	Hanna Moos
BLONP	Blondel père et fils	Frédérique Citeaux
BOLID	Bólido Comidas preparad...	Martin Sommer
BONAP	Bon app'	Laurence Lebihan
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln

The GDC is bound with data using XAML code.

Complex Property Binding

GridData control provides support to bind to complex properties through the Complex Property Binding feature. A Complex Property is one that contains multiple values, for example, an object of type "size" which contains two values – Width and Height, is represented as one property. Such complex properties are displayed in grid columns in the "Property.Member" format. For Size property, for example, the grid populates two columns namely, "Size.Width" and "Size.Height".

Complex Property Binding feature is useful when the user has nested data, i.e., when data of one table is mapped to another table.

Example

You can bind data columns from multiple data tables which are inter-related to the same grid. The following code example illustrates how to bind 'OrderID' and 'CustomerID' columns from the 'Orders' table, and 'CompanyName' and 'ContactTitle' columns from the 'Customers' table.

XML

```
<syncfusion:GridDataControl.VisibleColumns>
<syncfusion:GridDataVisibleColumn MappingName="OrderID" HeaderText="Order
ID">
</syncfusion:GridDataVisibleColumn>
<syncfusion:GridDataVisibleColumn MappingName="CustomerID"
HeaderText="Customer ID">
</syncfusion:GridDataVisibleColumn>
<syncfusion:GridDataVisibleColumn MappingName="Customers.CompanyName"
HeaderText="Customers.CompanyName"/>
<syncfusion:GridDataVisibleColumn MappingName="Customers.ContactTitle"
HeaderText="Customers.ContactTitle"/>
</syncfusion:GridDataControl.VisibleColumns>
```

Customers.ContactTitle ▲			
Order ID	Customer ID	Customers.CompanyName	Customers.Conta... ▲
+ Customers.ContactTitle : Accounting Manager - 147 Items			
+ Customers.ContactTitle : Assistant Sales Agent - 22 Items			
+ Customers.ContactTitle : Assistant Sales Representative - 25 Items			
- Customers.ContactTitle : Marketing Assistant - 57 Items			
10001	MEREP	Mère Paillarde	Marketing Assistant
10007	MORGK	Morgenstern Gesundkost	Marketing Assistant
10026	MORGK	Morgenstern Gesundkost	Marketing Assistant
10039	MEREP	Mère Paillarde	Marketing Assistant
10047	MORGK	Morgenstern Gesundkost	Marketing Assistant
10057	THECR	The Cracker Box	Marketing Assistant
Total : 1078 Items			

Order ID ▼	Customers.ContactTitle ▼	
10000	Sales Representative	<input checked="" type="checkbox"/> (Select All)
10001	Marketing Assistant	<input checked="" type="checkbox"/> Accounting Manager
10002	Owner	<input checked="" type="checkbox"/> Assistant Sales Agent
10003	Owner	<input checked="" type="checkbox"/> Assistant Sales Representative
10004	Sales Manager	<input checked="" type="checkbox"/> Marketing Assistant
10005	Accounting Manager	<input checked="" type="checkbox"/> Marketing Manager
10006	Sales Representative	<input checked="" type="checkbox"/> Order Administrator
10007	Marketing Assistant	<input checked="" type="checkbox"/> Owner
10008	Sales Manager	<input checked="" type="checkbox"/> Owner/Marketing Assistant
10009	Sales Manager	<input checked="" type="checkbox"/> Sales Agent
		<input checked="" type="checkbox"/> Sales Associate
		<input checked="" type="checkbox"/> Sales Manager
		<input checked="" type="checkbox"/> Sales Representative

You can also apply grouping, sorting, filtering, summaries and conditional formats to these complex property bound columns. The following code example illustrates this.

XML

```
// Applying Conditional Formats.
<syncfusion:GridDataControl.ConditionalFormats>
  <syncfusion:GridDataConditionalFormat Name="C1"
  ApplyStyleToColumn="Customers.ContactTitle">
    <syncfusion:GridDataConditionalFormat.Style>
      <syncfusion:GridDataStyleInfo Background="Crimson" Foreground="White"/>
    </syncfusion:GridDataConditionalFormat.Style>
    <syncfusion:GridDataConditionalFormat.Conditions>
      <syncfusion:GridDataCondition ColumnName="Customers.ContactTitle"
      ConditionType="Equals" Value="Sales Representative"
      PredicateType="Or"/>
    </syncfusion:GridDataConditionalFormat.Conditions>
  </syncfusion:GridDataConditionalFormat>
</syncfusion:GridDataControl.ConditionalFormats>
// Adding Summaries.
<syncfusion:GridDataControl.TableSummaryRows>
  <syncfusion:GridDataSummaryRow ShowSummaryInRow="True" Title="Total :
  {CountSummary} Items" TitleColumnCount="2">
```

```

<syncfusion:GridDataSummaryRow.SummaryColumns>
<syncfusion:GridDataSummaryColumn Name="CountSummary"
MappingName="Customers.CustomerID" SummaryType="CountAggregate"
Format="{Count:d}" />
</syncfusion:GridDataSummaryRow.SummaryColumns>
</syncfusion:GridDataSummaryRow>
</syncfusion:GridDataControl.TableSummaryRows>

```

Order ID	Customer ID	Customers.CompanyName	Customers.ContactTitle
10000	FRANS	Franchi S.p.A.	Sales Representative
10001	MEREP	Mère Paillarde	Marketing Assistant
10002	FOLKO	Folk och få HB	Owner
10003	SIMOB	Simons bistro	Owner
10004	VAFFE	Vaffeljernet	Sales Manager
10005	WARTH	Wartian Herkku	Accounting Manager
10006	FRANS	Franchi S.p.A.	Sales Representative
10007	MORGK	Morgenstern Gesundkost	Marketing Assistant
10008	FURIB	Furia Bacalhau e Frutos do Mar	Sales Manager
10009	SEVES	Seven Seas Imports	Sales Manager
Total : 1078 Items			

Notify Property changes

You can keep the grid notified of the underlying data source changes by setting a Boolean property called `NotifyPropertyChanges`. When it is set to true, the grid continues to listen to the data source changes and gets its field values updated accordingly. For this feature to take effect, the data source should implement the `INotifyPropertyChanged` interface.

The following code illustrates this property:

XML

```

<syncfusion:GridDataControl x:Name="grid" AutoPopulateColumns="True"
AutoPopulateRelations="False" NotifyPropertyChanges="True">

```

The following image corresponds to the output of the above given code:

Symbol	LastTrade	TradeTime	Change	PreviousClose
MSFT	39	6:17pm	-0.76	22.78
GOOG	39	6:17pm	-0.76	22.78
YAH0	39	6:17pm	-0.76	22.78
NBD	39	6:17pm	-0.76	22.78
YSD	39	6:17pm	-0.76	22.78
MSFT	39	6:17pm	-0.76	22.78
GOOG	39	6:17pm	-0.76	22.78
YAH0	39	6:17pm	-0.76	22.78
NBD	26	11:26am	+0.5	14.99
YSD	39	6:17pm	-0.76	22.78
MSFT	39	6:17pm	-0.76	22.78
GOOG	39	6:17pm	-0.76	22.78
YAH0	39	6:17pm	-0.76	22.78
NBD	39	6:17pm	-0.76	22.78

Data Error Validation

Essential Grid now provides support to validate the grid data and display error information. This is achieved by subscribing to the `IDataErrorInfo`, an interface that provides the functionality to display custom error information in any control.

To validate data errors, follow the steps below:

1. Ensure that your data source implements the `IDataErrorInfo` interface, in which two of the properties, `Error` (which we can be left empty optionally) and `Indexer` (where the validation code is placed) must be defined.
2. Then display the error information by setting the `ShowErrorToolTips` property of the `GridData` control to `true`.

C#

```
dataGrid.ShowErrorToolTips = true;
```

The following code example illustrates how the `GridData` control throws an error message when the `Freight` value becomes lesser than 10.

C#

```
partial class Orders : IDataErrorInfo
{
    public string Error
    {
        get
        {
            throw new NotImplementedException();
        }
    }
    public string this[string columnName]
    {
        get
        {
            if (columnName == "Freight" && Freight < 10)
            {
                return "Freight must be greater than 10";
            }
            return null;
        }
    }
}
```

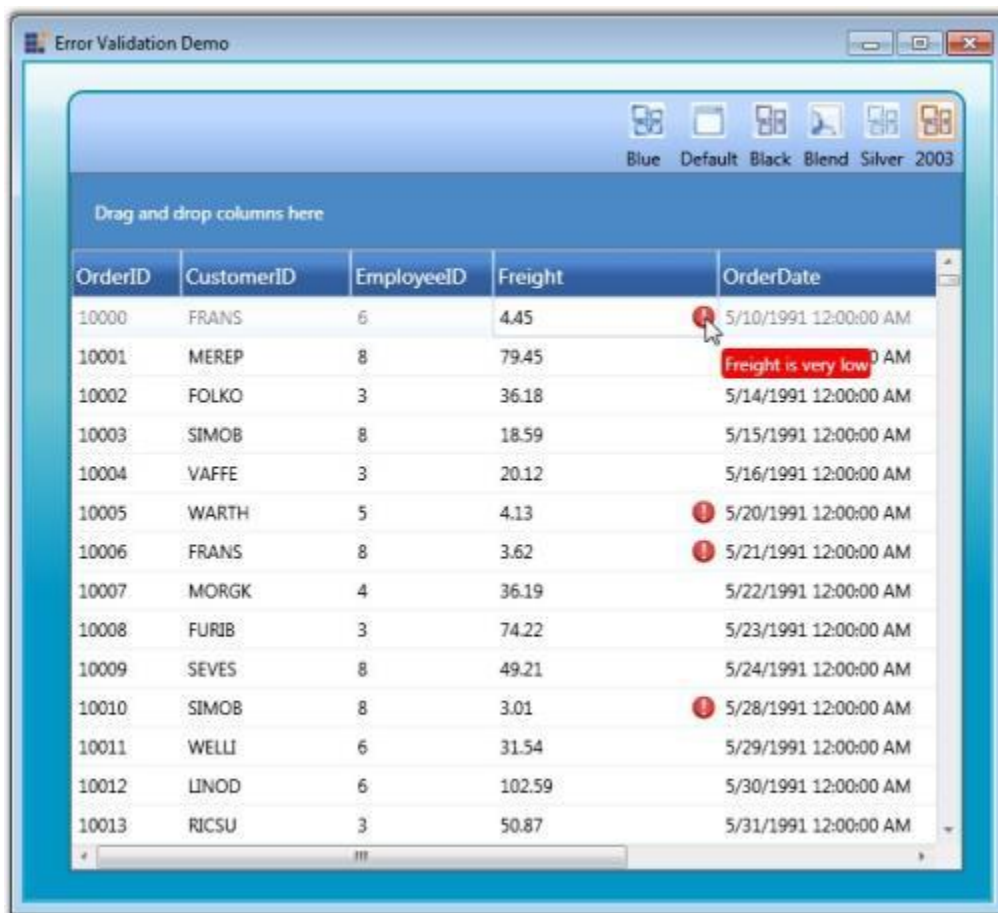


```

{
    get
    {
        var result = string.Empty;
        if (columnName == "Freight")
        {
            if (this.Freight.Value < 10)
            {
                result = "Freight is very low";
            }
        }
        return result;
    }
}

```

The following screenshot illustrates Data Error Validation in the GridData control.



Custom Data Error Validation

The GridData control supports two types of custom error validation methods. They are:

- Cell Level Validation
- Row Level Validation

Cell Level Validation

This validation takes place while the current cell editing completes. This type of validation is achieved by the CurrentCellValidating event.

Events

Events

Event	Description	Arguments	Type
CurrentCellValidating	It triggered while editing complete	CurrentCellValidatingEventArgsCancel: Boolean property to cancel the commit. NewValue: Values that we have edited. OldValue: Values before editing. Style: Style for the current editing cell.	CurrentCellValidatingEventHandler

Note: Once Cancel is set as true, the current cell doesn't leave from edit mode until Cancel is set to false.

Adding CurrentCellValidating Event to an Application

To add the CurrentCellValidating event to an application:

1. Add the GridData control to the application. Refer to the following section to add the GridData control to an application
2. In the following code example, we have set ProductList as ItemsSource. Hooking the CurrentCellValidating event follows.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid"
AutoPopulateColumns="False"
AutoPopulateRelations="False"
ItemsSource="{Binding ProductList}"
ShowErrorToolTips="True"
CurrentCellValidating="dataGrid_CurrentCellValidating"
syncfusion:GridTooltipService.ShowToolTips="True"
UpdateMode="PropertyChanged"
VisualStyle="Metro">
this.dataGrid.CurrentCellValidating += new CurrentCellValidatingEventHandler
(dataGrid_CurrentCellValidating);
```

3. The QueryCellInfo event is used to show the error icon in the cells while loading.

CSHARP

```
this.dataGrid.Model.QueryCellInfo += new GridQueryCellInfoEventHandler (Model
_QueryCellInfo);
void Model_QueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
```

```
if (e.Cell.RowIndex <= 0 || e.Cell.ColumnIndex <=0)
return;
IsValid(e.Style, e.Style.CellValue);
}
private bool IsValid(GridStyleInfo style, object CellValue)
{
    string column = this.grid.VisibleColumns[style.ColumnIndex].MappingName;
    bool isInError = false;
    style.ErrorInfo.ErrorContentAlignment = ImageContentAlignment.Left;
    style.ErrorInfo.ErrorType = ErrorType.ErrorMessage;
    if (column == "SupplierID")
    {
        if (CellValue is DBNull || CellValue == null || (int)CellValue <= 0)
        {
            style.ErrorInfo.ErrorMessage = "Supplier ID cannot be null or less than 0.";
            isInError = true;
        }
    }
    if (column == "CategoryID")
    {
        if (CellValue is DBNull || CellValue == null || (int)CellValue <= 0)
        {
            style.ErrorInfo.ErrorMessage = "Category ID cannot be null or 0.";
            isInError = true;
        }
    }
    return !isInError;
}
```

4. Data validation is performed in the CurrentCellValidating event as shown in the following code example.

CSHARP

```
void grid_CurrentCellValidating(object sender, CurrentCellValidatingEventArgs e)
{
    /// To validate the new value
    if(!this.IsValid(e.Style,e.NewValue))
    e.Cancel = true;
}
```

Output

Product ID	Supplier ID	Category ID	Product
1			1 Dharamsala Tea
2		1	0 Tibetan Barley Beer
3		0	2 Licorice Syrup
4	Supplier ID cannot be null or less than 0.		
5	2		2 Chef Anton's Gumbo Mix
6	3		2 Grandma's Boysenberry...
7	3		7 Uncle Bob's Organic Drie...
8	3		2 Northwoods Cranberry S...

Row Level Validation

This validation takes place while the focus moves from the current row to any other row. This type of validation is achieved by the RowValidation event.

Events

Events

Event	Description	Arguments	Type
RowValidating	Triggers while focus moves from current row to any other row.	GridDataRowValidatingEventArgs contains the following arguments: IsValid: Boolean property to validate the row values. NewValues: Values which are edited in current row. Record: Current row record. RowIndex: Index of current row.	GridDataRowValidatingEventHandler

Note: Once IsValid is set as false, the current cell doesn't leave from edit mode until IsValid is set as true.

Adding RowValidating Event to an Application

To add the RowValidating event to an application:

1. Add the GridData control to the application. Refer to the following section to add the GridData control to an application:

In the following code we have set ProductList as ItemsSource. Hooking the RowValidating event follows.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid"
AutoPopulateColumns="False"
AutoPopulateRelations="False"
ItemsSource="{Binding ProductList}"
ShowErrorTooltips="True"
RowValidating="dataGrid_RowValidating"
syncfusion:GridTooltipService.ShowTooltips="True"
UpdateMode="PropertyChanged">
```

CSHARP

```
this.dataGrid.RowValidating += new GridDataRowValidatingEventHandler(dataGrid_RowValidating);
```

2. The QueryCellInfo event is used to show the error icon in the cells while loading.

CSHARP

```
this.dataGrid.Model.QueryCellInfo += new GridQueryCellInfoEventHandler(Model_QueryCellInfo);
void Model_QueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    if (e.Style.RowIndex > 0 && e.Style.RowIndex != errorRowIndex)
    {
        var recordIdx = this.dataGrid.Model.ResolveIndexToRecordPosition(e.Style.RowIndex);
        var record = this.dataGrid.Model.View.Records[recordIdx] as GridDataRecord;
        Products currentRecord = (Products)record.Data;
        if (Validate(e.Style, currentRecord))
        {
            e.Style.ShowTooltip = true;
            e.Style.Tag = "Some fields in this record are having invalid data";
            if (e.Style.ColumnIndex > 0)
            e.Style.Background = new SolidColorBrush(GridUtil.GetXamlConvertedValue<Color>("#FFCDD3F1"));
            this.dataGrid.Model[e.Style.RowIndex, 0].ErrorInfo = new GridErrorStyleInfo
            {
                ErrorMessage = "Some fields in this record are having invalid data",
            };
        }
    }
}

public bool Validate(GridStyleInfo style, Products product)
{
    bool isInError = false;
    object checkValue1 = product.UnitPrice;
    if (checkValue1 is DBNull || checkValue1 == null || (double.Parse(checkValue1.ToString()) < 10))
    {
        ErrorMsg = "Unit price cannot be less than 10.";
        isInError = true;
    }
    object checkValue2 = product.UnitsOnOrder;
    if (checkValue2 is DBNull || checkValue2 == null || (short)checkValue2 > 100)
    {
        ErrorMsg = "Units in order cannot be more than 100.";
        isInError = true;
    }
    object checkValue3 = product.ReorderLevel;
    if (checkValue3 is DBNull || checkValue3 == null || (short)checkValue3 > 50)
    {
        ErrorMsg = "Reorder level cannot be more than 50.";
        isInError = true;
    }
}
```

```

}
return isInError;
}

```

3. Data validation is performed in the RowValidating event as shown in the following code example.

C#

```

void OnRowValidating(object sender, GridDataRowValidatingEventArgs args)
{
    if (this.Validate(null, args.Record as Products))
    {
        this.SetErrorInformation(args.RowIndex, ErrorMessageforRowValidation);
        //To Change the Background and set the error tooltip.
        this.dataGrid.Model.RowStyles[args.RowIndex].Background = new SolidColorBrush(GridUtil.GetXamlConvertedValue<Color>("#FFD0D0"));
        this.dataGrid.Model.RowStyles[args.RowIndex].Tag = ErrorMessageforRowValidation;
        this.dataGrid.Model.RowStyles[args.RowIndex].ShowTooltip = true;
        this.Grid.Model.InvalidateCell(GridRangeInfo.Row(args.RowIndex));
        errorRowIndex = args.RowIndex;
        args.IsValid = false;
    }
    else
    {
        errorRowIndex = -1;
        ErrorMsg = "";
        args.IsValid = true;
        //Need to revert the color once error cleared in the row.
        this.dataGrid.Model.RowStyles[args.RowIndex].ShowTooltip = false;
        this.dataGrid.Model.RowStyles[args.RowIndex].Background = Brushes.Transparent;
        this.dataGrid.Model.InvalidateCell(GridRangeInfo.Row(args.RowIndex));
    }
}

```

Output

	Product ID	Supplier ID	Category ID	Product
	1	1	1	Dharamsala Tea
!	2	1	1	Tibetan Barley Beer
!	3	1	2	Licorice Syrup
	4	2	2	Chef Anton's Cajun Seas...
	5	2	2	Chef Anton's Gumbo Mix
	6	3	2	Grandma's Boysenberry...
	7	3	7	Uncle Bob's Organic Drie...
	8	3	2	Northwoods Cranberry S...

Some fields in this record have invalid data

Synchronize Current Selection

The GridData control provides support to keep any Selector-derived control synchronized with its current selection. If you change the current record in the grid, then the current selection of the other

control also shifts to this new position and vice versa. It is obvious that the controls should be bound to the same data source.

The `IsSynchronizedWithCurrentItem` property does this work for you. It is a dependency property of Boolean type. You have to associate this property to the control that should be in sync. When this property is set to true, it keeps the controls synchronized and relieve this behavior, when set to false.

Technically, this synchronization is achieved through `CollectionView`, by binding its `CurrentItem` property to the current record position of the grid and hence, when another control binds to the same `CollectionView`, its current selection is bound indirectly to the grid's current record position. Therefore, when you change this control's current selection, it affects the `CollectionView.CurrentItem` property, which in turn affects the current record's position of the grid too.

Example

Following is the code example that synchronizes `GridData` control with a `ListView` control.

ListView Implementation with `IsSynchronizedWithCurrentItem` Set to True

XML

```
<ListView IsSynchronizedWithCurrentItem="True" ItemsSource="{Binding
Source={StaticResource ordersSource}}"
HorizontalContentAlignment="Stretch"
ScrollViewer.HorizontalScrollBarVisibility="Disabled"
Background="Transparent" Name="lstShelvedOrders" Height="300">
  <ListView.View>
    <GridView>
      <GridViewColumn DisplayMemberBinding="{Binding Path=OrderID}" Header="Shelve
ID" />
      <GridViewColumn DisplayMemberBinding="{Binding Path=CustomerID}"
Header="Customer" />
      <GridViewColumn DisplayMemberBinding="{Binding Path=EmployeeID}" Header="PO
Number" />
      <GridViewColumn DisplayMemberBinding="{Binding Path=OrderDate,
StringFormat=MMM dd\, yyyy}" Header="Date" />
      <GridViewColumn DisplayMemberBinding="{Binding Path=ShipCountry}"
Header="Ship Country" />
    </GridView>
  </ListView.View>
</ListView>
```

GridData Control Implementation

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" ShowAddNewRow="False"
ShowFilters="False" AutoPopulateColumns="False"
AutoPopulateRelations="False"
ItemsSource="{StaticResource ordersSource}" ShowGroupDropArea="True"
Height="300">
  <syncfusion:GridDataControl.VisibleColumns>
    <syncfusion:GridDataVisibleColumn MappingName="OrderID" IsReadOnly="True"
HeaderText="Order ID" Width="90" />
    <syncfusion:GridDataVisibleColumn MappingName="CustomerID"
HeaderText="Customer ID" Width="90" />
    <syncfusion:GridDataVisibleColumn MappingName="EmployeeID"
HeaderText="Employee ID" Width="90" />
  </syncfusion:GridDataControl.VisibleColumns>
</syncfusion:GridDataControl>
```

```

<syncfusion:GridDataVisibleColumn MappingName="OrderDate" HeaderText="Order
Date" Width="120"/>
<syncfusion:GridDataVisibleColumn MappingName="ShipCountry" HeaderText="Ship
Country" Width="100" />
</syncfusion:GridDataControl.VisibleColumns>
</syncfusion:GridDataControl>

```

Following are the sample outputs with `IsSynchronizedWithCurrentItem` set to true and false.

Shelve ID	Customer	PO Number	Date	Ship Country
10000	FRANS	6	May 10, 1991	Italy
10001	MEREP	8	May 13, 1991	Canada
10002	FOLKO	3	May 14, 1991	Sweden
10003	SIMOB	8	May 15, 1991	Denmark
10004	VAFFE	3	May 16, 1991	Denmark
10005	WARTH	5	May 20, 1991	Finland
10006	FRANS	8	May 21, 1991	Italy
10007	MORGK	4	May 22, 1991	Germany
10008	FURIB	3	May 23, 1991	Portugal
10009	SEVES	8	May 24, 1991	UK
10010	SIMOB	8	May 28, 1991	Denmark
10011	WELLI	6	May 29, 1991	Brazil
10012	LINOD	6	May 30, 1991	Venezuela

Order ID	Customer ID	Employee ID	Order Date	Ship Country
10000	FRANS	6	10-05-1991 00:00:00	Italy
10001	MEREP	8	13-05-1991 00:00:00	Canada
10002	FOLKO	3	14-05-1991 00:00:00	Sweden
10003	SIMOB	8	15-05-1991 00:00:00	Denmark
10004	VAFFE	3	16-05-1991 00:00:00	Denmark
10005	WARTH	5	20-05-1991 00:00:00	Finland
10006	FRANS	8	21-05-1991 00:00:00	Italy
10007	MORGK	4	22-05-1991 00:00:00	Germany
10008	FURIB	3	23-05-1991 00:00:00	Portugal
10009	SEVES	8	24-05-1991 00:00:00	UK

Shelve ID	Customer	PO Number	Date	Ship Country
10000	FRANS	6	May 10, 1991	Italy
10001	MEREP	8	May 13, 1991	Canada
10002	FOLKO	3	May 14, 1991	Sweden
10003	SIMOB	8	May 15, 1991	Denmark
10004	VAFFE	3	May 16, 1991	Denmark
10005	WARTH	5	May 20, 1991	Finland
10006	FRANS	8	May 21, 1991	Italy
10007	MORGK	4	May 22, 1991	Germany
10008	FURJB	3	May 23, 1991	Portugal
10009	SEVES	8	May 24, 1991	UK
10010	SIMOB	8	May 28, 1991	Denmark
10011	WELLJ	6	May 29, 1991	Brazil
10012	LINOD	6	May 30, 1991	Venezuela

Order ID	Customer ID	Employee ID	Order Date	Ship Country
10000	FRANS	6	10-05-1991 00:00:00	Italy
10001	MEREP	8	13-05-1991 00:00:00	Canada
10002	FOLKO	3	14-05-1991 00:00:00	Sweden
10003	SIMOB	8	15-05-1991 00:00:00	Denmark
10004	VAFFE	3	16-05-1991 00:00:00	Denmark
10005	WARTH	5	20-05-1991 00:00:00	Finland
10006	FRANS	8	21-05-1991 00:00:00	Italy
10007	MORGK	4	22-05-1991 00:00:00	Germany
10008	FURJB	3	23-05-1991 00:00:00	Portugal
10009	SEVES	8	24-05-1991 00:00:00	UK

Unbound Columns

Essential Grid supports addition of extra columns to the data source columns. Such additional columns are called unbound columns as they do not belong to the data source. These unbound fields can be used, when you want to add some additional or custom information for each record.

You can create an unbound column by instantiating the class `GridDataUnboundVisibleColumn`, which is a derivative of `GridDataVisibleColumn`. It contains a property named `Format`, which is used to specify a format for the unbound column. Given a visible column, it is possible to check whether it is an unbound column or not, using the `IsUnbound` property of that visible column.

Note: Sorting and filtering operations does not work with an unbound column unless the column is associated with the item source that is bound. So, it is good to turn off the filters wherever applicable.

The following example adds an unbound column and displays the record values in the format {Freight:c} for {ShipCity}.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid2" AutoPopulateColumns="True"
AutoPopulateRelations="False"
ItemsSource="{StaticResource ordersSource}"
ShowFilters="False"
ShowAddNewRow="False"
>
<syncfusion:GridDataControl.VisibleColumns>
<syncfusion:GridDataVisibleColumn MappingName="OrderID" IsReadOnly="True"
IsIdentity="True" HeaderText="Order ID" />
```

```

<syncfusion:GridDataVisibleColumn MappingName="ShipCountry" HeaderText="Ship
Country"/>
<syncfusion:GridDataVisibleColumn MappingName="ShipCity" HeaderText="Ship
City" />
<syncfusion:GridDataUnboundVisibleColumn HeaderText="UnboundCol1"
Format="'{Freight:c} for {ShipCity}'" />
</syncfusion:GridDataControl.VisibleColumns>
</syncfusion:GridDataControl>

```

The following screenshot shows the output of the above given code:



Order ID	Ship Country	Ship City	UnboundCol1
10000	Italy	Torino	\$4.45 for Torino
10001	Canada	Montréal	\$79.45 for Montréal
10002	Sweden	Bräcke	\$36.18 for Bräcke
10003	Denmark	København	\$18.59 for København
10004	Denmark	Århus	\$20.12 for Århus
10005	Finland	Oulu	\$4.13 for Oulu
10006	Italy	Torino	\$3.62 for Torino
10007	Germany	Leipzig	\$36.19 for Leipzig
10008	Portugal	Lisboa	\$74.22 for Lisboa
10009	UK	London	\$49.21 for London

GDC is added with the unbound column, "UnboundCol1".

Accessing Unbound Column Values

The Grid Table exposes an API named `GetUnboundValue` that can be used to retrieve the unbound value of a specific unbound cell. Given the record index and the respective Unbound Column object, it returns the unbound field value for the specified record. It is overloaded to accept row and column indices as parameters using which it fetches the corresponding unbound visible column and yields its value as output.

C#

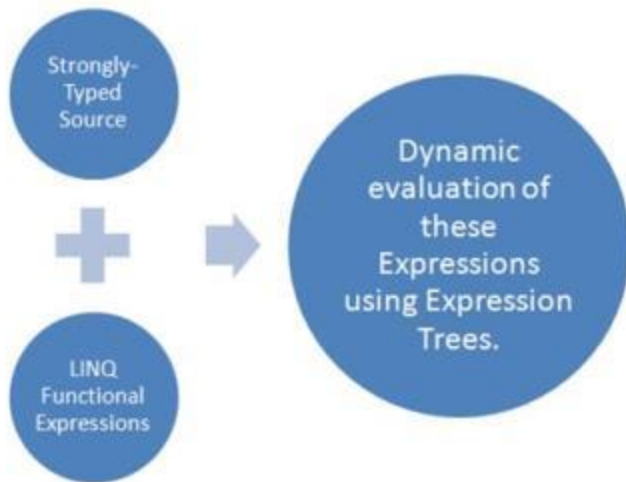
```

//Retrieve the unbound value using record index and the unbound column
instance.
object value = this.dataGrid.Model.Table.GetUnboundValue(recordIndex,
unboundColumn);
//Retrieve the unbound value using row and column indices.
object value = this.dataGrid.Model.Table.GetUnboundValue(5, 5);

```

Operations on Unbound Columns

Unbound columns allow association of related values that are bound to an expression in the unbound column or through handling the `QueryUnboundColumnValue` event. Operations like sorting, filtering, grouping, summaries and conditional formatting can now be applied on these dynamic values bound to the underlying source. It uses LINQ Functional Expressions to be dynamically evaluated at run time, and thus only strongly-typed source can be used with this feature.



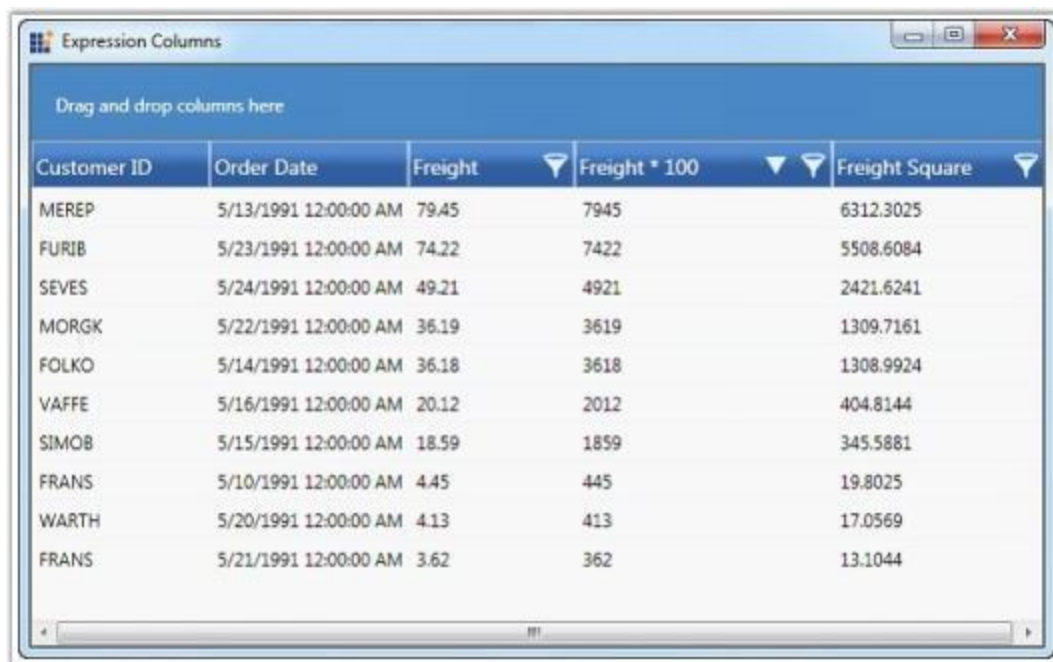
Sorting Operation

Sorting can be done interactively by clicking the header or can be declared by using XAML code as follows.

XML

```

<syncfusion:GridDataControl.SortColumns>
<syncfusion:GridDataSortColumn ColumnName="Multiply"
SortDirection="Descending" />
</syncfusion:GridDataControl.SortColumns>
<syncfusion:GridDataControl.VisibleColumns>
<syncfusion:GridDataVisibleColumn MappingName="CustomerID"
HeaderText="Customer ID" />
<syncfusion:GridDataVisibleColumn MappingName="OrderDate" HeaderText="Order
Date" />
<syncfusion:GridDataVisibleColumn MappingName="Freight" HeaderText="Freight"
AllowFilter="True">
<syncfusion:GridDataVisibleColumn.FilterPane>
<syncfusion:GridDataTextFilteringPane IsThemed="False" Foreground="Black"
PredicateType="And" />
</syncfusion:GridDataVisibleColumn.FilterPane>
</syncfusion:GridDataVisibleColumn>
<syncfusion:GridDataUnboundVisibleColumn MappingName="Multiply"
HeaderText="Freight * 100" Expression="Freight * 100"
AllowFilter="True" AllowDrag="True" AllowGroup="True">
</syncfusion:GridDataUnboundVisibleColumn>
<syncfusion:GridDataUnboundVisibleColumn MappingName="FreightSqr"
HeaderText="Freight Square" Expression="Freight ^ 2"
AllowFilter="True">
</syncfusion:GridDataUnboundVisibleColumn>
</syncfusion:GridDataControl.VisibleColumns>
  
```



Expression Columns

Drag and drop columns here

Customer ID	Order Date	Freight	Freight * 100	Freight Square
MEREP	5/13/1991 12:00:00 AM	79.45	7945	6312.3025
FURIB	5/23/1991 12:00:00 AM	74.22	7422	5508.6084
SEVES	5/24/1991 12:00:00 AM	49.21	4921	2421.6241
MORGK	5/22/1991 12:00:00 AM	36.19	3619	1309.7161
FOLKO	5/14/1991 12:00:00 AM	36.18	3618	1308.9924
VAFFE	5/16/1991 12:00:00 AM	20.12	2012	404.8144
SIMOB	5/15/1991 12:00:00 AM	18.59	1859	345.5881
FRANS	5/10/1991 12:00:00 AM	4.45	445	19.8025
WARTH	5/20/1991 12:00:00 AM	4.13	413	17.0569
FRANS	5/21/1991 12:00:00 AM	3.62	362	13.1044

Filtering Operation

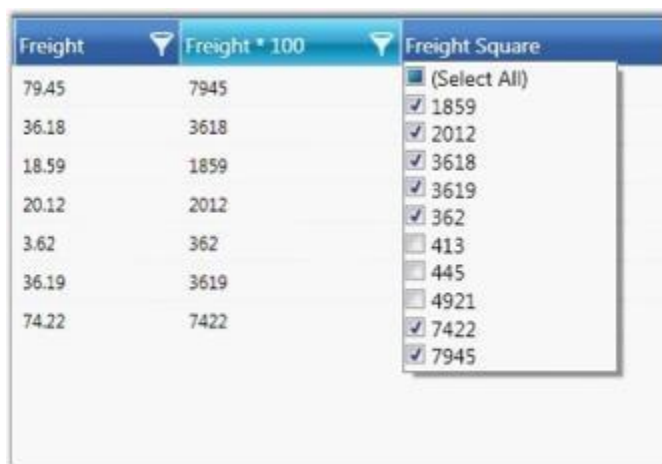
We have two modes of filtering in the WPF GridData control: Excel-like Filtering and Advanced Filtering. Both these modes are supported by unbound columns.

Excel-like Filtering Mode

Excel-like filtering can be enabled in unbound columns by setting AllowFilter property to true.

XML

```
<syncfusion:GridDataUnboundVisibleColumn MappingName="Multiply"
HeaderText="Freight * 100" Expression="Freight * 100" AllowFilter="True"
AllowDrag="True" AllowGroup="True">
</syncfusion:GridDataUnboundVisibleColumn>
```



Freight	Freight * 100	Freight Square
79.45	7945	(Select All)
36.18	3618	<input checked="" type="checkbox"/> 1859
18.59	1859	<input checked="" type="checkbox"/> 2012
20.12	2012	<input checked="" type="checkbox"/> 3618
3.62	362	<input checked="" type="checkbox"/> 3619
36.19	3619	<input checked="" type="checkbox"/> 362
74.22	7422	<input type="checkbox"/> 413
		<input type="checkbox"/> 445
		<input type="checkbox"/> 4921
		<input checked="" type="checkbox"/> 7422
		<input checked="" type="checkbox"/> 7945

Advanced Filtering Mode

Advanced Filtering mode can be enabled by adding the following code.

XML

```
<syncfusion:GridDataUnboundVisibleColumn MappingName="Multiply"
HeaderText="Freight * 100" Expression="Freight * 100" AllowFilter="True"
AllowDrag="True" AllowGroup="True">
<syncfusion:GridDataVisibleColumn.FilterPane>
<syncfusion:GridDataTextFilteringPane IsThemed="False" Foreground="Black"
PredicateType="And" />
</syncfusion:GridDataVisibleColumn.FilterPane>
</syncfusion:GridDataUnboundVisibleColumn>
```

Freight	Freight * 100	Freight Square
79.45	7945	
36.18	3618	
18.59	1859	
20.12	2012	
36.19	3619	
74.22	7422	
49.21	4921	2421.6241

Grouping Operation

Grouping can be performed interactively or declaratively on the Unbound Columns by using XAML code.

XML

```
<syncfusion:GridDataControl.GroupedColumns>
<syncfusion:GridDataGroupColumn ColumnName="Multiply" />
</syncfusion:GridDataControl.GroupedColumns>
```

Customer ID	Order Date	Freight	Freight * 100	Freight Square
Multiply : 362 - 1 Items				
FRANS	5/21/1991 12:00:00 AM	3.62	362	13.1044
Multiply : 413 - 1 Items				
WARTH	5/20/1991 12:00:00 AM	4.13	413	17.0569
Multiply : 445 - 1 Items				
Multiply : 1859 - 1 Items				
Multiply : 2012 - 1 Items				
Multiply : 3618 - 1 Items				
Multiply : 3619 - 1 Items				
Multiply : 4921 - 1 Items				
Multiply : 7422 - 1 Items				
Multiply : 7945 - 1 Items				

Summary Operations

The default summaries work much the same way as summaries with bound columns. Following is the list of different types of Summary declaration in XAML.

Table Summary

XML

```
<syncfusion:GridDataControl.TableSummaryRows>
<syncfusion:GridDataSummaryRow ShowSummaryInRow="False" Title="Total :
{FreightMultiplySummary}" TitleColumnCount="3">
<syncfusion:GridDataSummaryRow.SummaryColumns>
```

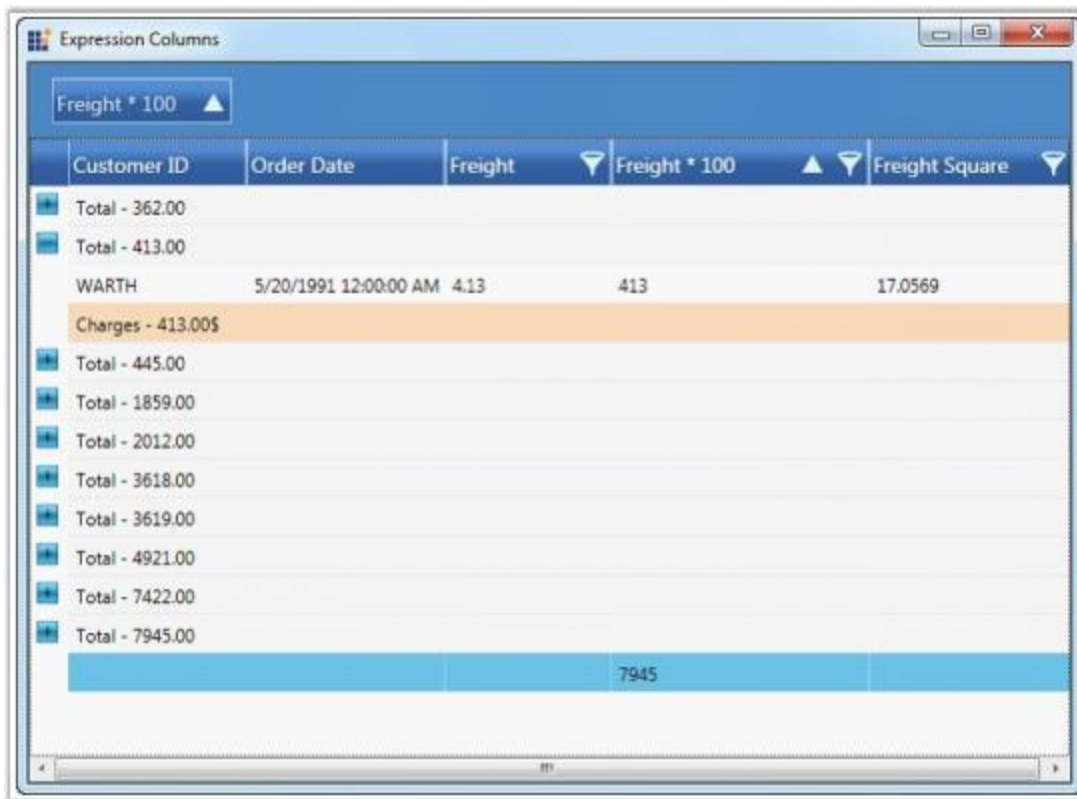
```
<syncfusion:GridDataSummaryColumn Name="FreightMultiplySummary"
MappingName="Multiply" SummaryType="Int32Aggregate"
Format="{Max:d}" />
</syncfusion:GridDataSummaryRow.SummaryColumns>
</syncfusion:GridDataSummaryRow>
</syncfusion:GridDataControl.TableSummaryRows>
```

*Caption Summary***XML**

```
<syncfusion:GridDataControl.CaptionSummaryRow>
<syncfusion:GridDataSummaryRow ShowSummaryInRow="False"
Title="{CountSummary}" TitleColumnCount="4">
<syncfusion:GridDataSummaryRow.SummaryColumns>
<syncfusion:GridDataSummaryColumn Name="CountSummary" MappingName="Multiply"
SummaryType="DoubleAggregate"
Format="Total - '{Sum:###.00}'" >
</syncfusion:GridDataSummaryColumn>
</syncfusion:GridDataSummaryRow.SummaryColumns>
</syncfusion:GridDataSummaryRow>
</syncfusion:GridDataControl.CaptionSummaryRow>
```

*Group Summary***XML**

```
<syncfusion:GridDataControl.CaptionSummaryRow>
<syncfusion:GridDataSummaryRow ShowSummaryInRow="False"
Title="{CountSummary}" TitleColumnCount="4">
<syncfusion:GridDataSummaryRow.SummaryColumns>
<syncfusion:GridDataSummaryColumn Name="CountSummary" MappingName="Multiply"
SummaryType="DoubleAggregate"
Format="Total - '{Sum:###.00}'" >
</syncfusion:GridDataSummaryColumn>
</syncfusion:GridDataSummaryRow.SummaryColumns>
</syncfusion:GridDataSummaryRow>
</syncfusion:GridDataControl.CaptionSummaryRow>
```



Custom Summary Operations

Summary calculation differs for unbound columns. It has an additional parameter to supply a dynamic lambda delegate for invoking the unbound values at run time.

C#

```
public interface ISummaryExpressionAggregate : ISummaryAggregate
{
    Action<IEnumerable, string, Expression<Func<string, object, object>>,
    PropertyDescriptor> CalculateAggregateExpressionFunc();
}
```

All the default Summary Aggregates now implement ISummaryExpressionAggregate interface. This typically informs the ICollectionViewAdv to expect an Expression to evaluate at run time.

Expression Trees Evaluation for Aggregate Methods

A return value is required for any LINQ Aggregate method to be implemented. In order to invoke an unbound column through the lambda delegate, you have an internal wrapper lambda that is generic.

C#

```
private static Expression
GetInvokeExpressionAggregateFunc<TResult>(ParameterExpression paramExp,
string propertyName, Expression<Func<string, object, object>>
expressionFunc)
{
    // Constructing a wrapper function that would return a generic value.
    Func<Expression<Func<string, object, object>>, string, object, TResult> fun
    = (func, prop, rec) =>
```



```

{
    var lambda = func.Compile();
    TResult val = (TResult)lambda.DynamicInvoke(new object[] { prop, rec });
    return val;
};
Expression<Func<Expression<Func<string, object, object>>, string, object, TResult>> eIFunc = (func, prop, rec) => fun(func, prop, rec);
var invokeExp = Expression.Invoke(Expression.Constant(fun), new Expression[]
{ Expression.Constant(expressionFunc),
  Expression.Constant(propertyName), paramExp });
return invokeExp;
}

```

The above method wraps the unbound column's lambda expression into an expression that would return typed value, thus enabling direct calls to `Sum<TResult>(Expression<Func<T,TResult>>`, where `TResult` can be any numeric data type.

Selected Items Collection

GridData control allows you to select the required records and retrieve selected record values. Once a record is selected, it is added to the `GridDataControl.SelectedItems` collection and `GridDataControl.SelectedItem` highlights the current record in selection.

Following is the code example that iterates through the `SelectedItems` collections and prints the values of those records that are in selection.

C#

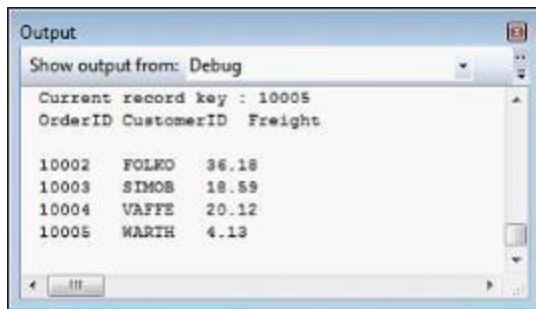
```

Console.WriteLine("Current record key : "+
  ((Orders) this.gdc.SelectedItem).OrderID);
Console.WriteLine("OrderID" + "\t" + "CustomerID" + "\t" + "Freight");
Console.WriteLine();
foreach (object obj in gdc.SelectedItems)
{
    Orders c = (Orders)obj;
    Console.WriteLine(c.OrderID + "\t" + c.CustomerID + "\t" + c.Freight);
}

```

Output

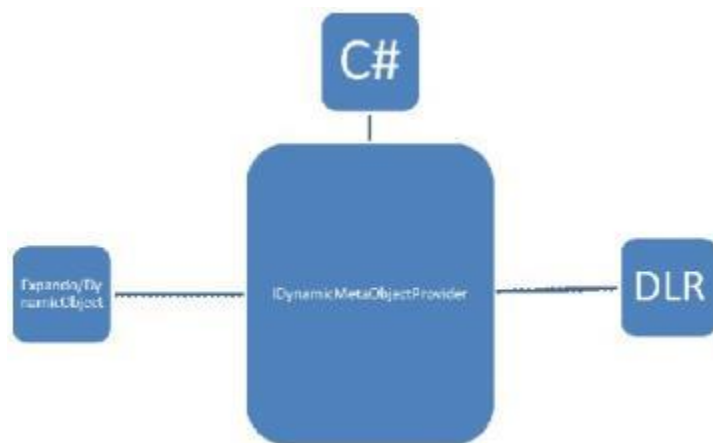
OrderID	CustomerID	Freight
10000	FRANS	4.45
10001	MEREP	79.45
10002	FOLKO	36.18
10003	SIMOB	18.59
10004	VAFFE	20.12
10005	WARTH	4.13
10006	FRANS	3.62
10007	MORGK	36.19
10008	FURIB	74.22
10009	SEVES	49.21
10010	CHACB	7.05



Dynamic Keyword Support

C# 4.0 introduces a new type of object: dynamic. More information can be found in MSDN. The dynamic object offers a way to get or set values dynamically using CallSites and binders from the C# language service. Using this service, we are able to generate dynamic compile-time execution and representing that acts as an object to GridDataControl. By enabling it to work with dynamic objects, the grid seamlessly allows working with any DLR-based language.

The IDynamicMetaObjectProvider interface defines the interface between the dynamic run-time types.



Usage Scenarios

Define a List<dynamic> object to be set as ItemsSource for the grid.

C#

```

var list = new List<dynamic>();
dynamic d = new ExpandoObject();
d.Name = "Robert";
d.Age = 24;
d.Occupation = "Software Engineer";
d.HasCar = true;
dynamic d1 = new ExpandoObject();
d1.Name = "James";
d1.Age = 25;
d1.Occupation = "Business";
d1.HasCar = false;
list.Add(d);
list.Add(d1);
this.dataGrid.ItemsSource = list;
  
```

Features that work with dynamic objects

- Sorting
- Filtering
- Grouping
- Summaries
- Conditional formatting

The internal view uses LINQ expression syntax to generate operations, this works well for including custom LINQ expressions inside operations such as sorting, filtering, grouping, etc.

Specific Dynamic Type Handling

Column Type

Since the collections are dynamic, we need to specify the proper type for the column to work properly in all scenarios. GridDataVisibleColumn provides a DataType property to specify the type for the column.

XML

```
<syncfusion:GridDataVisibleColumn MappingName="OrderDate"
    DataType="DateTime" />
```

Handling Relations

Relations have to be handled in XAML or C#. The AutoPopulateRelations property does not work on dynamic object types.

XML

```
<syncfusion:GridDataControl.Relations>
    <syncfusion:GridDataRelation RelationalColumn="OrderDetails" />
</syncfusion:GridDataControl.Relations>
```

UnboundRows

Essential Grid supports addition of extra rows in the view that does not affect the DataSource. Such additional rows are called UnboundRows as they do not belong to the data source. These unbound fields can be used, when you want to add some additional or custom information for GridDataControl

You can create an UnboundRow by just setting the UnboundRowCount. It contains a property named Format, which is used to specify a format for the UnboundRow. Given an UnboundRow we can check if this is an UnboundRow or not using IsInUnboundRow(int Rowindex) method in Grid Model.

C#

```
this.grid.Model.UnboundRowCount = 5;
this.grid.UnboundRowPosition = Position.Top;
```

Observable Collection Demo

Drag and drop columns here

	CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone
▶	Unbound...	Unbound Cell R 1 C 2	Unbound Cell R...	Unbound Cell R 1...	Unbound Cell R 1 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...
	Unbound...	Unbound Cell R 2 C 2	Unbound Cell R...	Unbound Cell R 2...	Unbound Cell R 2 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...
	Unbound...	Unbound Cell R 3 C 2	Unbound Cell R...	Unbound Cell R 3...	Unbound Cell R 3 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...
	Unbound...	Unbound Cell R 4 C 2	Unbound Cell R...	Unbound Cell R 4...	Unbound Cell R 4 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...
	Unbound...	Unbound Cell R 5 C 2	Unbound Cell R...	Unbound Cell R 5...	Unbound Cell R 5 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...
	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321
	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729
	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932
	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(71) 555-7788
	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65
	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460
	BLOPP	Blondel père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31
	BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82
	BONAP	Bon app'	Laurence Leblanc	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40
	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen BC		T2F 8M4	Canada	(604) 555-4729

Observable Collection Demo

Drag and drop columns here

	CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone
▶	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321
	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729
	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932
	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(71) 555-7788
	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65
	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460
	BLOPP	Blondel père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31
	BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82
	BONAP	Bon app'	Laurence Leblanc	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40
	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen BC		T2F 8M4	Canada	(604) 555-4729
	Unbound...	Unbound Cell R 11 C 2	Unbound Cell R...	Unbound Cell R 11...	Unbound Cell R 11 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...
	Unbound...	Unbound Cell R 12 C 2	Unbound Cell R...	Unbound Cell R 12...	Unbound Cell R 12 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...
	Unbound...	Unbound Cell R 13 C 2	Unbound Cell R...	Unbound Cell R 13...	Unbound Cell R 13 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...
	Unbound...	Unbound Cell R 14 C 2	Unbound Cell R...	Unbound Cell R 14...	Unbound Cell R 14 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...
	Unbound...	Unbound Cell R 15 C 2	Unbound Cell R...	Unbound Cell R 15...	Unbound Cell R 15 C 5	Unbound...	Unbo...	Unbound...	Unbou...	Unbound Cel...

Properties, Methods and Events tables

Properties

Properties

Properties	Description	Type Of Property	Acceptable Value	Reference Link
UnboundRowPosition	Gets or sets a value, which represents the position of the Unbound Rows.	Position	Top/Bottom	NA

Methods

Methods

Method	Description	Parameters	Type	Return Type
IsInUnboundRows(int RowIndex)	this method checks if the rows provided is an unbound row or not.this can be accessed from Grid Model as grid.Model.IsInUnboundRows(row);	(int RowIndex)	Integer	Boolean

Data-Presentation in WPF GridDataControl (Classic)

GridData control data can be presented in several ways.

This section illustrates those data presentation techniques in the following topics:

- Grouping—This topic discusses on grouping feature in the grid.
- Sorting—This topic discusses on sorting feature in the grid.
- Filters—This topic discusses on applying filters in the grid.
- Summaries—This topic describes on adding summaries in the grid.
- Column Drag and Drop—This topic discusses on column-level drag-and-drop support.
- Hierarchy—This topic discusses on nested table hierarchy.
- Stacked Headers—This topic discusses on the addition of unbound header rows called stacked headers.
- Expression Fields—This topic discusses how to create expression columns that hold calculated values based on other fields in the same record.
- ToolTips—This topic describes the addition of tooltips for the grid cells.

Grouping

This section elaborates the grouping feature of Essential Grid. By using this feature, you can wrap a set of records that belong to the same category into a separate block, called Group. Grid enables you to group data by one or more columns. When grouping is applied, the records having identical values for the grouped columns are combined together forming a hierarchical structure.

Each group is identified by its Group Caption Section that can be expanded to bring the underlying records into view. This Group Caption Section contains the information about a particular group such as group name, number of records in the group, and so on.

This section comprises the following:

Creating Groups

Grid groups can be created at design-time as well as run time. They are managed by the GroupedColumns collection which holds one entry for every grouped column. A group can be created programmatically by adding the desired column into this collection. The records are sorted in the ascending (default) or descending order of their Grouped Column values. The GroupedColumns collection can have more than one entry to form Nested Groups.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" ShowAddNewRow="False"
ShowFilters="False" AutoPopulateColumns="True"
AutoPopulateRelations="True" ItemsSource="{StaticResource ordersSource}"
ShowGroupDropArea="True">
<syncfusion:GridDataControl.GroupedColumns >
```

```
<syncfusion:GridDataGroupColumn  
ColumnName="ShipCountry"></syncfusion:GridDataGroupColumn>  
</syncfusion:GridDataControl.GroupedColumns>  
</syncfusion:GridDataControl>
```

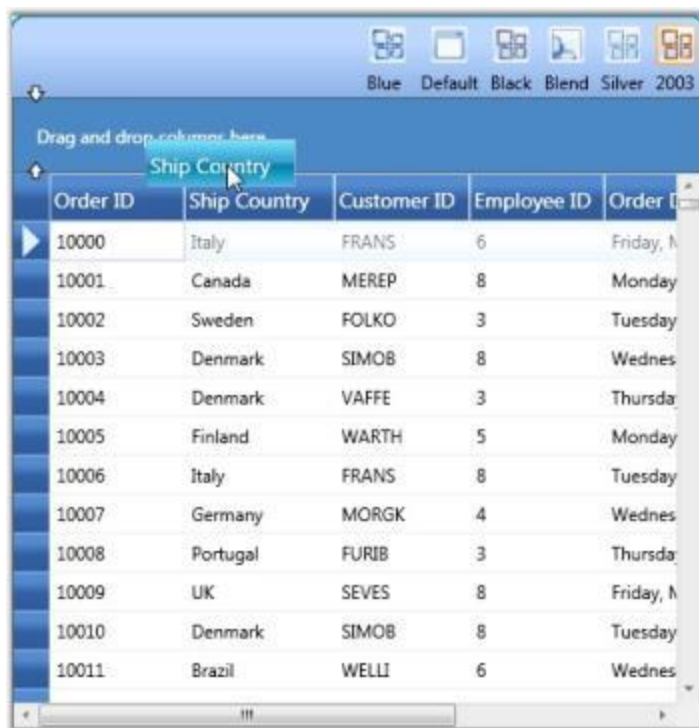
C#

```
GridDataGroupColumn groupedCol = new GridDataGroupColumn();  
groupedCol.ColumnName = "ShipCountry";  
dataGridControl.GroupedColumns.Add(groupedCol);
```



Run Time Grouping

Run time grouping is enabled by displaying the group drop area, a placeholder to store the current groups. Such groups can be created interactively through the drag-and-drop operation. For example, to group data against a particular column, drag the desired column header and drop it into the group drop area.



Nested Groups and Custom Groups

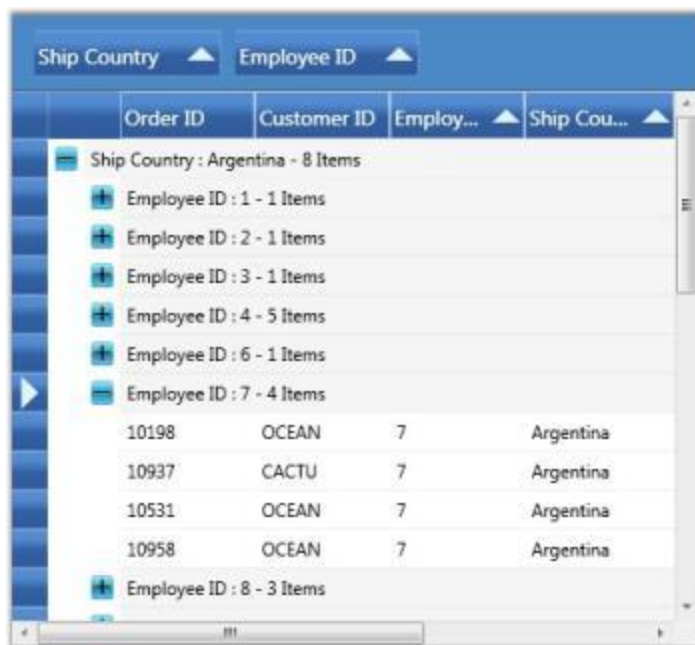
Nested Groups

When you have the grid data grouped against more than one column, the groups are nested in different levels forming a hierarchical, multilevel structure. You can expand or collapse the underlying groups and records from a parent group by clicking the PlusMinus button preceding its group caption.

MultiColumn (Nested) groups can be easily created by simply adding multiple columns into the GroupedColumns collection. You can also generate multilevel groups by just dragging multiple column headers into the group drop area.

C#

```
GridDataGroupColumn groupedCol1 = new GridDataGroupColumn();
groupedCol1.ColumnName = "ShipCountry";
dataGrid.GroupedColumns.Add(groupedCol1);
GridDataGroupColumn groupedCol2 = new GridDataGroupColumn();
groupedCol2.ColumnName = "EmployeeID";
dataGrid.GroupedColumns.Add(groupedCol2);
```



Creating Groups

Group Events

The following table lists the events that are associated with Grid Groups.

Events

Event	Description
GroupExpanding	This event is raised when a group is about to be expanded. This operation can be optionally canceled.
GroupExpanded	This event is raised when a group is expanded.
GroupCollapsing	This event is raised when a group is about to be collapsed. This operation can be optionally canceled.
GroupCollapsed	This event is raised when a group is collapsed.

GroupExpanding Event

The following code example illustrates how to handle the GroupExpanding event.

C#

```
// Subscribe to the event.
dataGrid.Model.Table.GroupExpanding+=new
GroupExpandingEventHandler(Table_GroupExpanding);
// Handle the event.
void Table_GroupExpanding(object sender, GroupExpandingEventArgs args)
{
    args.Cancel = true;
}
```

```
}
```

GroupExpanded Event

The following code example illustrates how to handle the GroupExpanded event.

C#

```
// Subscribe to the event.
dataGrid.Model.Table.GroupExpanded += new
GroupExpandedEventHandler(Table_GroupExpanded);
// Handle the event.
void Table_GroupExpanded(object sender, GroupExpandedEventArgs args)
{
    // Print the group caption text.
    Console.WriteLine("Expanded: " +
dataGrid.Model.Table.GroupModel.GetGroupCaptionText((GridDataGroupItem) args.
Group.Item));
}
```

GroupCollapsing Event

The following code example illustrates how to handle the GroupCollapsing event.

C#

```
// Subscribe to the event.
dataGrid.Model.Table.GroupCollapsing+=new
GroupCollapsingEventHandler(Table_GroupCollapsing);
// Handle the event.
void Table_GroupCollapsing(object sender, GroupCollapsingEventArgs args)
{
    args.Cancel = true;
}
```

GroupCollapsed Event

The following code example illustrates how to handle the GroupCollapsed event.

XML

```
// Subscribe to the event.
dataGrid.Model.Table.GroupCollapsed+=new
GroupCollapsedEventHandler(Table_GroupCollapsed);
// Handle the event.
void Table_GroupCollapsed(object sender, GroupCollapsedEventArgs args)
{
    // Print the group caption text.
    Console.WriteLine("Expanded: " +
dataGrid.Model.Table.GroupModel.GetGroupCaptionText((GridDataGroupItem) args.
Group.Item));
}
```

Sorting

Sorting arranges the records either in ascending or in descending order of the selected field values. Essential Grid allows you to sort the data against one or more columns. The number of columns on which the sorting can be applied is unlimited.

SortColumns Collection

The information about all the sorted columns for a grid is managed by Grid.SortColumns collection. It is an observable collection of type GridDataSortColumn, where each entry holds the following two properties:

- Name of the sorted column
- SortDirection-an object of type ListSortDirection

You can add, modify and remove any item of this collection to manage the sorted columns of a grid.

Apply Sorting

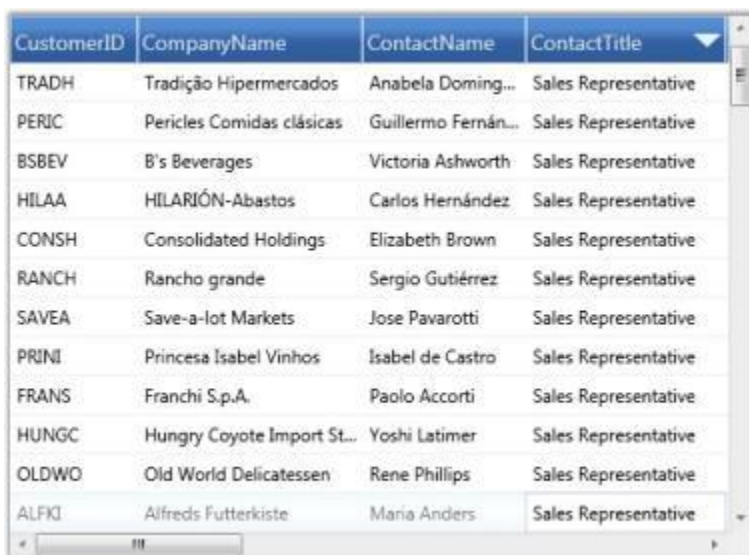
There are a couple of ways to apply sorting to the grid data. A simple one is just by clicking the column header by which the grid data needs to be sorted. Once the sorting is applied, the grid shows a sort icon in the respective column headers indicating the direction of sorting.

You can also perform sorting through the code. This requires you to define a number of GridDataSortColumn objects specifying the desired column names and sort directions and then add these objects into TableProperties.SortColumns collection. The following code illustrates this:

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource
customerSource}">
<syncfusion:GridDataControl.SortColumns>
<syncfusion:GridDataSortColumn ColumnName="ContactTitle"
SortDirection="Descending" />
</syncfusion:GridDataControl.SortColumns>
</syncfusion:GridDataControl>
```

The following screenshot shows a GDC enabled with sorting feature:



CustomerID	CompanyName	ContactName	ContactTitle
TRADH	Tradição Hipermercados	Anabela Doming...	Sales Representative
PERIC	Pericles Comidas clásicas	Guillermo Fernán...	Sales Representative
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative
HILAA	HILARIÓN-Abastos	Carlos Hernández	Sales Representative
CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative
RANCH	Rancho grande	Sergio Gutiérrez	Sales Representative
SAVEA	Save-a-lot Markets	Jose Pavarotti	Sales Representative
PRINI	Princesa Isabel Vinhos	Isabel de Castro	Sales Representative
FRANS	Franchi S.p.A.	Paolo Accorti	Sales Representative
HUNGC	Hungry Coyote Import St...	Yoshi Latimer	Sales Representative
OLDWO	Old World Delicatessen	Rene Phillips	Sales Representative
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative

Enable/Disable Sorting

It is possible to prevent sorting on specific grid columns or all the columns at once. Grid.AllowSort property is set to *true* by default. Set the Grid.AllowSort property to *false* to disable sorting on all the

columns. To disallow sorting on a particular column, set its `visibleColumn.AllowSort` property to *false*. The following code illustrates setting these properties:

Disable Sort on All Columns

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" ItemsSource="{StaticResource customerSource}" AllowSort="False" />
```

Disable Sort on Single Column (Example-CustomerID)

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource customerSource}">
<syncfusion:GridDataControl.VisibleColumns>
<syncfusion:GridDataVisibleColumn MappingName="CustomerID"
HeaderText="CustomerID" AllowSort="False" />
</syncfusion:GridDataControl.VisibleColumns>
</syncfusion:GridDataControl>
```

Multicolumn Sorting

To apply sorting on more than one column at run time, click the desired column headers by pressing the CTRL key.

Below is the code that sorts the grid by two columns:

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource customerSource}">
<syncfusion:GridDataControl.SortColumns>
<syncfusion:GridDataSortColumn ColumnName="CompanyName"
SortDirection="Ascending" />
<syncfusion:GridDataSortColumn ColumnName="ContactTitle"
SortDirection="Descending" />
</syncfusion:GridDataControl.SortColumns>
</syncfusion:GridDataControl>
```

Note: When the grid is sorted against multiple columns, the affected column headers get painted with a number that starts from 0, 1 ... representing the sort order.

The following screenshot shows a multicolumn sorting enabled GDC:

CustomerID	CompanyName	ContactName	ContactTitle
ALFKJ	Alfreds Futterkiste	Maria Anders	Sales Representative
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative
CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative
WANDK	Die Wandernde Kuh	Rita Müller	Sales Representative
FRANS	Franchi S.p.A.	Paolo Accorti	Sales Representative
HILAA	HILARIÓN-Abastos	Carlos Hernández	Sales Representative
HUNGC	Hungry Coyote Import St...	Yoshi Latimer	Sales Representative
LACOR	La corne d'abondance	Daniel Tonini	Sales Representative
LEHMS	Lehmanns Marktstand	Renate Messner	Sales Representative
OLDWO	Old World Delicatessen	Rene Phillips	Sales Representative

Custom Sorting

Custom sorting helps you to sort the records of the selected field values depending on your needs. It also allows you to sort the data against one or more columns in the GridDataControl.

The custom sorting sorts the values using custom sorting logic. To perform the custom sorting, you need to hook the `SortColumnChanging` or `SortColumnChangingCommand` event and pass the custom sorting logic to the `CustomComparer`.

To create a custom comparer to implement the custom sorting logic, you need to derive the logic from `IComparer<object>` and `ISortDirection`. The custom sorting is performed by comparing the values of particular columns. When grouping is applied, the custom sorting can be performed by comparing the Group keys.

Example Scenario

The following code examples illustrate how to perform the custom sorting for the names in the Company Name column according to the string length of the names.

To enable the custom sorting, hook the `SortColumnsChanging` event.

C#

```
this.dataGrid.Model.Table.SortColumnsChanging += new GridDataSortColumnsChangingEventHandler(Table_SortColumnsChanging);
```

Set the comparer for the column on which the data needs to be sorted using the custom sorting logic. Here, the comparer is assigned to `CompanyName`.

C#

```
//This method is hooked when clicking the header
void Table_SortColumnsChanging(object sender, GridDataSortColumnsChangingEventArgs args)
{
    if (args != null)
    {
        foreach (var item in args.AddedItems)
        {

```

```
//Choosing the column to be sorted
if (item.ColumnName.Equals("CompanyName"))
{
    //Passing the custom sort logic to CustomComparer
    item.CustomComparer = new CustomerInfo();
}
}
}
```

Check the direction of the sorting by using the SortDirection property of the ListSortDirection class. The Compare method of the IComparer interface uses two parameters to compare the length of the string.

C#

```
public class CustomerInfo : IComparer<Object>, ISortDirection
{
    //Implementation of ICompare method
    public int Compare(object x, object y)
    {
        int namX;
        int namY;
        //For Normal case
        if (x.GetType() == typeof(Customers))
        {
            //Calculating the length if the object is of Customers type
            namX = ((Customers)x).CompanyName.Length;
            namY = ((Customers)y).CompanyName.Length;
        }
        //While Grouping
        else if (x.GetType() == typeof(Group))
        {
            //Calculating the length if the object is of Group type
            namX = ((Group)x).Key.ToString().Length;
            namY = ((Group)y).Key.ToString().Length;
        }
        else
        {
            namX = x.ToString().Length;
            namY = y.ToString().Length;
        }
        //Object is passed through the Compare method and gets the SortDirection.
        if (namX.CompareTo(namY) > 0)
            return SortDirection == ListSortDirection.Ascending ? 1 : -1;
        else if (namX.CompareTo(namY) == -1)
            return SortDirection == ListSortDirection.Ascending ? -1 : 1;
        else
            return 0;
    }
    //gets or sets the SortDirection
    private ListSortDirection _SortDirection;
    public ListSortDirection SortDirection
    {
        get
        {
            return _SortDirection;
        }
    }
}
```

```

}
set
{
    SortDirection = value;
}
}
}

```

The following screenshot displays the Company Name column with the sorted names according to their length.

Company Name	Contact Name	Address	City	Region
Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	
Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	
Great Lakes Food Market	Howard Snyder	2732 Baker Blvd.	Eugene	OR
Drachenblut Delikatessen	Sven Ottlieb	Walsenweg 21	Aachen	
Lonesome Pine Restaurant	Fran Wilson	89 Chiaroscuro Rd.	Portland	OR
Bóldo Comidas preparadas	Martin Sommer	C/ Araquil, 67	Madrid	
Pericles Comidas clásicas	Guillermo Fernández	Calle Dr. Jorge Cash 321	México D.F.	
Vins et alcools Chevalier	Paul Henriot	59 rue de l'Abbaye	Reims	
Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	
Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.	
Hungry Coyote Import Store	Yoshi Latimer	City Center Plaza	Elgin	OR
Rattlesnake Canyon Grocery	Paula Wilson	2817 Milton Dr.	Albuquerque	NM
Hungry Owl All-Night Grocers	Patricia McKenna	8 Johnstown Road	Cork	Co. Co
Magazzini Alimentari Riuniti	Giovanni Rovelli	Via Ludovico il Moro 22	Bergamo	
Laughing Bacchus Wine Cellars	Yoshi Tannamuri	1900 Oak St.	Vancouver	BC
Furia Bacalhau e Frutos do Mar	Lino Rodriguez	Jardim das rosas n. 32	Lisboa	
Trail's Head Gourmet Provisioners	Helvetius Nagy	722 DaVinci Blvd.	Kirkland	WA
Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	
FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	C/ Moralzarzal, 86	Madrid	

Filters

Essential Grid allows you to restrict the display of records using a mechanism called Filters. A filter lets you to extract a subset of records that meet certain filter criteria. Filters can be applied to one or more columns.

Functions

- Apply Filters
- Enable/Disable Filters
- Filters Collection

Now, let us see these functionalities one-by-one in detail.

Apply Filters

When this feature is enabled, every column header displays a filter icon. Clicking this icon opens a drop-down list, which is actually a Checked List Box control that holds the possible values for the column clicked. To filter the data by a specific value, just select the check box that prefixes the desired value. This is then reload the grid with only those records, which has the selected value on the particular

column. The filter drop-down also provides a SelectAll option either to select or deselect all the values at once.

Enable/Disable Filters

It is possible to turn on and turn off the filters in specific column or in all the columns as a whole. To enable filters in all the columns, the Grid.ShowFilters must be set to true. To turn on the filters for specific column, simply set its AllowFilter property to true.

The following are the code snippets that illustrate this property:

Enable Filter for the Whole Grid

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource customerSource}"
ShowFilters="True" />
```

Enable Filter for Single Column

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="False"
AutoPopulateRelations="False" ItemsSource="{StaticResource
customerSource}">
<syncfusion:GridDataControl.VisibleColumns>
<syncfusion:GridDataVisibleColumn MappingName="CustomerID"
HeaderText="CustomerID"/>
<syncfusion:GridDataVisibleColumn MappingName="CompanyName"
HeaderText="CompanyName" AllowFilter="True" />
<syncfusion:GridDataVisibleColumn MappingName="ContactTitle"
HeaderText="ContactTitle" />
</syncfusion:GridDataControl.VisibleColumns>
</syncfusion:GridDataControl>
```

The following image shows a GridData control with filter feature enabled for "CompanyName" column:



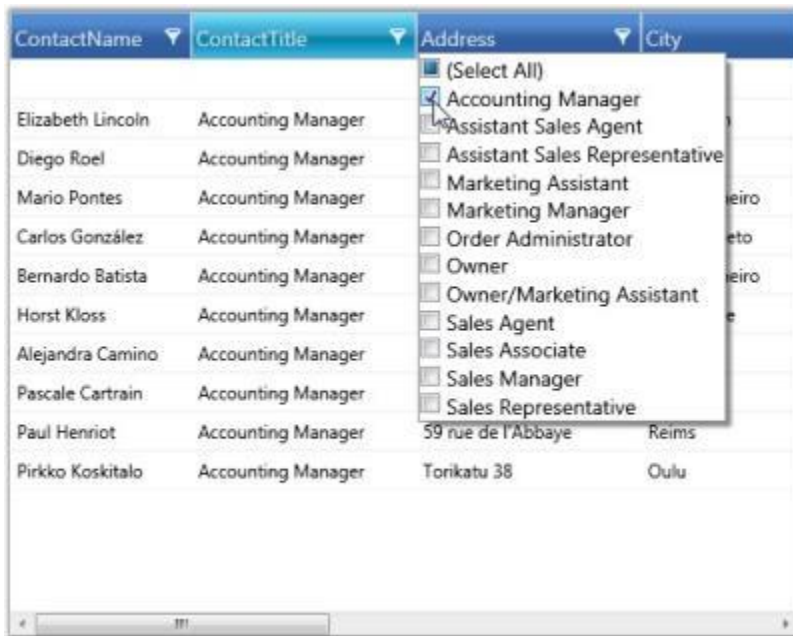
CompanyName in GridData control is now enabled with filtering feature.

Here is an example code that enables filter on all the columns:

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource customerSource}"
ShowFilters="True" />
```

The following image illustrates the columns, filtered against the criteria ContactTitle='Accounting Manager'.



Filters Collection

All the filters for a particular column are managed by VisibleColumn.Filters property. You can apply or clear any number of filters by adding or removing the appropriate entries from this collection.

The following code illustrates the addition of a filter to the column 'ContactTitle'.

XML

```
<syncfusion:GridDataVisibleColumn MappingName="ContactTitle"
HeaderText="ContactTitle">
<syncfusion:GridDataVisibleColumn.Filters>
<syncfusion:GridDataFilterPredicate FilterType="StartsWith"
FilterValue="Sales" IsCaseSensitive="False" />
</syncfusion:GridDataVisibleColumn.Filters>
</syncfusion:GridDataVisibleColumn>
```

Advanced Filtering

This feature provides advanced filtering options for the end-user. It overrides the default filter and displays an advanced filter drop-down that lists the available filter operators for the respective filtering column and provides a text box, where the user is allowed to type the filter string. It exhibits a dynamic filtering mechanism by applying filter as the characters are typed.

APIs Used

This advanced filter pane comes in following two forms.

- GridDataTextFilteringPane, which can be used with any column type
- GridDataInt32SliderFilteringPane, which works only with integer columns

Filter Operators

The filter combo is the place where you get the available list of filter operators supported for the corresponding column type. The following table provides the list of filter choices for each column type.

Filter Choices

Column Type	Filter Choices
String	StartsWithEndsWithContainsEqualsNotEquals
Int32 /DateTime	LessThanLessThanOrEqualGreaterThanGreaterThanOrEqualEqualNotEqual

The GridDataTextFilteringPane, which is common to all column types, associate appropriate filter choice list according to the type of the column. When GridDataInt32SliderFilteringPane is used, the filter combo is always loaded with Int32 filter choices.

Clearing Filter

To clear any filter, you can simply erase the characters from filter text box. If you are using SliderFilteringPane, select the “None” option from the filter combo in order to clear the filter.

Customization Options

The Filtering Pane classes provide different customization properties that assist you to customize the filtering pane. These properties are described below:

Properties

Customization Property	Description
Background	Brush for filter pane background
Foreground	Brush for filter pane foreground
PredicateType	Specifies the predicate type: OR (or) AND
CurrentFilterType	Specifies the filter operator to be used for the filter column
MatchCase	Boolean property; when <i>true</i> , performs case-sensitive filtering

The filtering panes are optional. It is possible to enable the filtering panes feature in few specific columns and allow other columns to use the default filter.

To enable this feature, you can simply attach an object of the desired filtering pane to the corresponding visible column. The columns that does not have this specification use the default filtering mechanism. It is obvious that the AllowFilter property of the visible column should be set to true in both the cases.

Example

Let us see some code examples that set up the two kinds of filtering panes.

XML

```

<syncfusion:GridDataControl x:Name="dataGrid" ShowAddNewRow="False"
AutoPopulateColumns="False" AutoPopulateRelations="False"
ItemsSource="{StaticResource orderSource}" ShowColumnOptions="True">
  <syncfusion:GridDataControl.VisibleColumns>
    <syncfusion:GridDataVisibleColumn MappingName="OrderID"/>
    <!--Frieght column using Slider Filtering Pane.-->
    <syncfusion:GridDataVisibleColumn MappingName="Freight" AllowFilter="True">
      <syncfusion:GridDataVisibleColumn.FilterPane>
        <syncfusion:GridDataInt32SliderFilteringPane />
      </syncfusion:GridDataVisibleColumn.FilterPane>
    </syncfusion:GridDataVisibleColumn>
    <!--ShipCountry column using Text Filtering Pane with the Foreground and
    PredicateType properties set.-->
    <syncfusion:GridDataVisibleColumn MappingName="ShipCountry"
    AllowFilter="True">
      <syncfusion:GridDataVisibleColumn.FilterPane>
        <syncfusion:GridDataTextFilteringPane IsThemed="True" Foreground="Black"
        PredicateType="And" />
      </syncfusion:GridDataVisibleColumn.FilterPane>
    </syncfusion:GridDataVisibleColumn>
    <!--ShipCountry column using Text Filtering Pane with the Background and
    CurrentFilterType properties set.-->
    <syncfusion:GridDataVisibleColumn MappingName="ShipCity" AllowFilter="True">
      <syncfusion:GridDataVisibleColumn.FilterPane>
        <syncfusion:GridDataTextFilteringPane IsThemed="False" Background="Orange"
        CurrentFilterType="GreaterThan" />
      </syncfusion:GridDataVisibleColumn.FilterPane>
    </syncfusion:GridDataVisibleColumn>
  </syncfusion:GridDataControl.VisibleColumns>
</syncfusion:GridDataControl>

```





Grid FilterBar

Essential Grid supports FilterBar, which filters the records with different expressions depending upon the Column type. The FilterBar is displayed at the top of the Grid below the Header Row by setting the “ShowFilterBar” property to true in GridDataControl class. The filtering tokens are tabulated in the Tokens to filter the value table.

Use Case Scenarios

FilterBar can be used for applications for which the user wants to filter the Grid at run time.

Adding FilterBar to an Application

This topic explains the implementation of the FilterBar in an application. The following steps explain the implementation of FilterBar support in an application.

1. Creating an application

Create a WPF application and add GridDataControl to it.

2. Setting the FilterBar Property

Set the FilterBar property to “true” for the GridDataControl object. The Filter status message can be viewed by enabling the property ShowFilterStatusMessage. The filtering mode can be set to Immediate or OnEnter by setting the Enum property GridDataFilterBarMode. The following code snippet explains the implementation of the FilterBar.

CSHARP

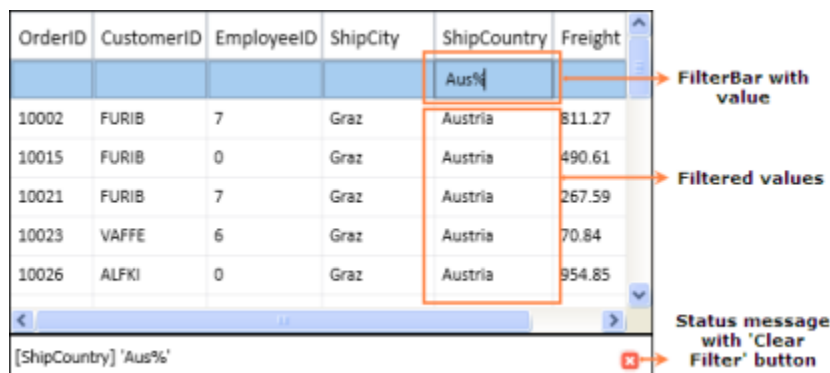
```
this.dataGrid.ShowFilterBar = true;
this.dataGrid.ShowFilterStatusMessage = true;
this.dataGrid.FilterBarMode = GridDataFilterBarMode.Immediate;
//this.dataGrid.FilterBarMode = GridDataFilterBarMode.OnEnter;
```

VB.NET

```
Me.dataGrid.ShowFilterBar = True
```

```
Me.dataGrid.ShowFilterStatusMessage = True
Me.dataGrid.FilterBarMode = GridDataFilterBarMode.Immediate
'Me.dataGrid.FilterBarMode = GridDataFilterBarMode.OnEnter
```

3. Run the application and use the filtering tokens in the FilterBar. The valid tokens are listed in Tokens to filter the value table. The following is a sample output of FilterBar implementation.



4. Clearing the Filter

The Current filter value with the column name is displayed at the bottom of the GridDataControl (just like status bar). It contains the button (red color) called "Clear Filter", which is used to clear the entire filter and show the default level records.

[Tables for Properties, Methods, and Events](#)

[Properties](#)

FilterBar Support Table

Property	Description	Data Type	Default value	Class Name
ShowFilterBar	Shows the FilterBar, if it is true.	Boolean	False	GridDataControl
ShowFilterStatusMessage	Shows the message at the bottom of the grid depending on the current Filter applied, if it is true.	Boolean	True	GridDataControl
FilterBarMode	Filter result is shown immediately if "Immediately" is set and is shown on pressing the Enter key if "OnEnter" is set	Enum	Immediate	GridDataControl
FilterBarMode	Filter result is shown immediately if "Immediately" is set and is shown on pressing the	Enum	Immediate	GridDataVisibleColumn

	Enter key if "OnEnter" is set			
--	-------------------------------	--	--	--

Tokens to Filter the Value

FilterBar Support Table

Filter Token	Examples(should be used as like below)	Description	Used at
%	value%	StartsWith	AlphaNumeric
	%value	EndsWith	AlphaNumeric
#	#value	Contains	AlphaNumeric
<	<value	LessThan	Numeric & DateTime
<=	<=value	LessThanOrEqual	Numeric & DateTime
>	>value	GreaterThan	Numeric & DateTime
>=	>=value	GreaterThanOrEqual	Numeric & DateTime
=	=value	Equals	Numeric & DateTime
!	!value	Not Equals	Numeric & DateTime
and	>value and <=value>value and <value>=value and <value>=value and <=value	between	Numeric & DateTime
or	>value or <=value>value or <value>=value or <value>=value or <=value	between	Numeric & DateTime
0	0	Equals	Boolean
1	1	Equals	Boolean

*values can be entered in any format (not case sensitive)

[Sample Link](#)

Refer to the samples in the shipped Sample Browser.

Go to Essential Studio WPF Sample Browser Grid GridDataControl-AdvancedFilterBarDemo.

[Dropdown FilterBar](#)

Dropdown FilterBar feature is similar to Text Box Filter instead of entering the key word the dropdown button can be used to filter the items.

To filter items using Dropdown Filter

The Dropdown button is used to filter the required items. We have to set the FilterBarStyle to change the FilterBar cell type.

XML

```

<syncfusion:GridDataVisibleColumn MappingName="EmployeeID" HeaderText="Employee ID">
  <syncfusion:GridDataVisibleColumn.ColumnStyle>
    <syncfusion:GridDataColumnStyle CellType="IntegerEdit"
      HorizontalAlignment="Right">
    </syncfusion:GridDataColumnStyle>
  </syncfusion:GridDataVisibleColumn.ColumnStyle>
  <syncfusion:GridDataVisibleColumn.FilterBarStyle>
    <syncfusion:GridDataFilterBarStyle CellType="ComboBox" />
  </syncfusion:GridDataVisibleColumn.FilterBarStyle>
</syncfusion:GridDataVisibleColumn>

```

Order ID	Employee ID	Freight	Ship Via	Order Date	Ship Name	Ship Address	Ship City
10000	(All)	\$4.45	3	5/10/1991	Franchi S.p.A.	Via Monte Bianco 34	Torino
10001	1	\$79.45	3	5/13/1991	Mère Paillard	43 rue St. Laurent	Montréal
10002	2	\$36.18	3	5/14/1991	Folk och få HB	Åkergatan 24	Bräcke
10003	3	\$18.59	1	5/15/1991	Simons bistro	Vinbæltet 34	København
10004	4	\$20.12	2	5/16/1991	Vaffeljernet	Smagsløget 45	Århus
10005	5	\$4.13	3	5/20/1991	Wartian Herkku	Tonikatu 38	Oulu
10006	6	\$3.62	1	5/21/1991	Franchi S.p.A.	Via Monte Bianco 34	Torino
10007	7	\$36.19	3	5/22/1991	Morgenstern Gesundkost	Heerstr. 22	Leipzig
10008	8	\$74.22	1	5/23/1991	Furia Bacalhau e Frutos d...	Jardim das rosas n. 32	Lisboa
10009	9	\$49.21	1	5/24/1991	Seven Seas Imports	90 Wadhurst Rd.	London
10010		\$3.01	1	5/28/1991	Simons bistro	Vinbæltet 34	København
10011		\$31.54	3	5/29/1991	Wellington Importadora	Rua do Mercado, 12	Resende
10012		\$102.59	1	5/30/1991	LINO-Delicateses	Ave. 5 de Mayo Porlamar	I. de Margarita
10013		\$50.87	3	5/31/1991	Richter Supermarkt	Starenweg 5	Genève
10014		\$17.67	3	6/3/1991	GROSELLA-Restaurante	5ª Ave. Los Palos Grandes	Caracas
10015		\$22.10	1	6/5/1991	Piccolo und mehr	Geislweg 14	Salzburg
10016		\$113.01	2	6/6/1991	Folies gourmandes	184, chaussée de Tournai	Lille
10017		\$111.81	1	6/7/1991	Blondel père et fils	24, place Kléber	Strasbourg
10018		\$65.46	1	6/10/1991	Rattlesnake Canyon Groc...	2817 Milton Dr.	Albuquerque
10019		\$2.42	3	6/11/1991	Magazzini Alimentari Ri...	Via Ludovico il Moro 22	Bergamo
10020		\$27.51	1	6/13/1991	Vins et alcools Chevalier	59 rue de l'Abbaye	Reims
10021		\$75.17	3	6/14/1991	Ernst Handel	Kirchgasse 6	Graz
10022		\$46.00	1	6/17/1991	La maison d'Asie	1 rue Alsace-Lorraine	Toulouse
10023		\$66.87	1	6/18/1991	Toms Spezialitäten	Luisenstr. 48	Münster
10024		\$5.19	1	6/19/1991	Rattlesnake Canyon Groc...	2817 Milton Dr.	Albuquerque
10025		\$3.32	3	6/21/1991	Rattlesnake Canyon Groc...	2817 Milton Dr.	Albuquerque

Properties, Methods and Events tables

Properties

Property	Description	Type	Data Type	Reference links
CellType	Used to select ComboBox or TextBox	Dependency	Enum	NA
ItemsSource	Used to bind the external item source	Dependency	Object	NA
DisplayMember	This decides which member should be displayed.	Dependency	String	NA
ValueMember	Based on the value the items are filtered.	Dependency	String	NA

FilterBarStyle	This property used to set the style of the filterbar for the corresponding visible column.	Dependency	GridDataFilterBarStyle	NA
----------------	--	------------	------------------------	----

To Edit items in Dropdown list



The behavior is similar to Auto-complete combo box.

Property

Method	Description	Parameters	Return Type
IsEditable	Combine an editable text field and provide users the additional option of typing an item and predict a word or phrase that the user wants to type in the associated text box without the user actually typing it completely.	Dependency	Boolean

Paging Support for GridDataControl

Paging support is used to return pages of data with entries where selection of the pages can be done using the numbered buttons. There are 3 types:

- OnDemandPaging

Current page item source adding by OnDemand basis. Using this we can fetch the data from the data source for the current page.

- ViewLevelPaging

ItemsSource for the page load on Grid load. In this type sorting, filtering and grouping is applicable for the Current view element. Excel-like filtering is not applicable here.

- SourceLevelPaging

ItemsSource for the page load while grid load. In this type sorting, filtering and grouping is applicable for the whole collection. Excel-like filtering is not applicable here.

Use Case Scenario

Paging Support is a very useful feature to load large amount of data. We can load millions of records in an efficient way.

OrderID	CustomerID	EmployeeID	ShipCity	ShipCountry	Freight	IsClosed
10125	VAFFE	2	Montréal	Canada	91.17	True
10126	WARTH	7	Buenos Aires	Argentina	799.08	True
10127	VAFFE	0	Graz	Austria	940.05	True
10128	MEREP	5	Buenos Aires	Argentina	681.74	True
10129	WELLI	5	Graz	Austria	540.4	True
10130	WELLI	0	Buenos Aires	Argentina	648.72	True
10131	ALFKI	3	Buenos Aires	Argentina	643.54	True
10132	VAFFE	2	Buenos Aires	Argentina	835.21	False
10133	SEVES	3	Graz	Austria	423.2	False
10134	SIMOB	2	Bruxelles	Belgium	681.42	False
10135	FRANS	2	Bruxelles	Belgium	336.71	True
10136	FOLKO	0	Buenos Aires	Argentina	205.89	False
10137	SIMOB	6	Bruxelles	Belgium	188.33	False
10138	SEVES	2	Graz	Austria	514.86	True
10139	MEREP	8	Campinas	Brazil	118.46	False
10140	FOLIG	4	Graz	Austria	500.76	True
10141	WARTH	1	Graz	Austria	312.07	False
10142	SEVES	4	Montréal	Canada	89.19	True
10143	FRANS	7	Bruxelles	Belgium	848.96	True
10144	FRANS	0	Buenos Aires	Argentina	430.84	True
10145	FOLKO	6	Graz	Austria	574.92	True
10146	ALFKI	1	Graz	Austria	793.97	True
10147	FURIB	0	Bruxelles	Belgium	273.72	True
10148	RISCU	5	Tsawassen	Canada	798.39	True
10149	FRANS	7	Graz	Austria	233.85	False

Properties, Methods and Events tables

Properties

Property	Description	Type	Data Type	Reference links
IsPagingOnDemand	Loads the page based on demand	NA	Boolean	NA
EnablePaging	When the property is set as true, it is loaded page wise	NA	Boolean	NA
IsViewLevelPaging	It differentiated paging as view level or source level	NA	Boolean	NA
PageCount	It sets the number of pages that can be viewed. This is valid only for OnDemandPage.	NA	Integer	NA
PageSize	It sets the number of items to be displayed on a page.	Dependency	Integer	NA

Events

Event	Description	Arguments	Type	Reference links
-------	-------------	-----------	------	-----------------

OnDemandDataSource Load	the event is triggered when it moves to the next page or when the page changes	PagedRowsMaximumRows	GridDataOnDemandPageLoadingEventArgs	NA
-------------------------	--	----------------------	--------------------------------------	----

Features of Paging Support

OnDemandPaging

The GridDataControl supports paging on demand by specifying queries in order to get the paged records from a database. The records are displayed only when it is required. This sample retrieves fifty records from a database and displays them. Using this type we can fetch the data from the data source for the current page. The entire data is not needed to be fetched from the datasource. We can get high performance even if there are millions of records.

XML

```
<syncfusion:DataPagerExt x:Name="dataPager" Grid.Row="1"
HorizontalAlignment="Right" VerticalAlignment="Top"
DisplayMode="PreviousNextNumeric" NumericButtonCount="10"
AutoEllipsis="True" PageSize="25"/>
```

C#

```
pager.PageCount = 400;
pager.IsPagingOnDemand = true;
pager.OnDemandDataSourceLoad += new GridDataOnDemandPageLoadingEventHandler(
pager_OnDemandDataSourceLoad);
void pager_OnDemandDataSourceLoad(object sender, GridDataOnDemandPageLoading
EventArgs e)
{
this.grid.ItemsSource = Get_Data(e.PagedRows, e.PagedRows + e.MaximumRows);
}
```

ViewLevelPaging

ViewLevel Sorting, grouping and filtering is provided. Allows to sort the view element. Sorting, filtering and grouping only for the Current page items. This type supports the Filter bar to filter the items. Advance filtering and Excel-like filtering is not applicable for this type.

XML

```
<syncfusion:DataPagerExt x:Name="dataPager" Grid.Row="1"
HorizontalAlignment="Right" VerticalAlignment="Top"
DisplayMode="PreviousNextNumeric" NumericButtonCount="10"
AutoEllipsis="True" PageSize="25"/>
```


C#

```
var item = new NorthwindOrders(1000);  
var itemlist = new PagedCollectionView(item);  
pager.Source = itemlist;  
grid.ItemsSource = itemlist;  
grid.EnablePaging = true;  
grid.IsViewLevelPaging = true;
```

SourceLevelPaging

Sorting, grouping and filtering are provided at the Source level. Allows to sort the view element. Sorting, filtering and grouping only for Whole collection. This type supports the Filter bar to filter the items. Advance filtering and Excel-like filtering is not applicable for this type.

XML

```
<syncfusion:DataPagerExt x:Name="dataPager" Grid.Row="1"  
HorizontalAlignment="Right" VerticalAlignment="Top"  
DisplayMode="PreviousNextNumeric" NumericButtonCount="10"  
AutoEllipsis="True" PageSize="25"/>
```

C#

```
var item = new NorthwindOrders(1000);  
var itemlist = new PagedCollectionView(item);  
pager.Source = itemlist;  
grid.ItemsSource = itemlist;  
grid.EnablePaging = true;  
grid.IsViewLevelPaging = false;
```

Details View

Essential Grid provides support to have a separate view for each record displayed in the grid called a details view. Every record bound to the Grid control can have some additional information about it. To display that information, you can define your own view and display it on demand whenever needed. The grid gets the view through DetailsViewTemplate and display it below every row. You can display or hide the details view of a row by using the expander available in the row header of the each row.

1. Instead of having all information in the grid initially, having abstract information initially and then displaying information on demand ensures fast performance and a compact UI look.
2. By listening to the events, you can dynamically change the template based on the record or row.
3. By listening to the events, you can set constraints and block, expand, or collapse the view of a record or row.
4. Can create a tabbed UI to display the hierarchy information in a compact way.

Use Case Scenarios

The details view can be used to create a compact UI to display a large amount of information. Information is displayed only on an on-demand basis. This separates the detailed information and abstract information, acting as a quick view and explorer for the required information.

*Tables for Properties, Methods, and Events**Properties*

Property	Description	Type	Data Type
DetailsViewTemplate	Get the user-defined UI of the details view in the form of DataTemplate. This is the core input to have the details view in the GridData control.	Dependency property	DataTemplate

Methods

Note: The following methods are available in GridDataControl.Model.Table.

Methods	Description	Parameters	Type	Return Type
ExpandAllDetailsView	Using this method, the DetailsView of all records can be expanded. When you want to expand all the details views at the same time you can use this method.			Void
ExpandDetailsViewAt	Using this method, the DetailsView of a particular record can be expanded.this method gets the record index as an input argument and expand the details view of the corresponding record.	Record Index	int	Void
CollapseAllDetailsView	Using this method, you can collapse all expanded details view at the same time.			Void
CollapseDetailsViewAt	Using this method, you can collapse the details view of a particular record. This method gets the record index as an input argument and collapse the details view of the corresponding record.	Record Index	int	Void

Events

Note: The following events are available in GridDataControl.Model.Table.

Events	Description	Arguments	Type
DetailsViewExpanding	this event triggers before the DetailsView is expanded.this event is a cancelable event. By canceling this event, the corresponding	GridDataDetailsViewExpandingEventArgs	Routed Event

	DetailsView remains collapsed.		
DetailsViewExpanded	this event is triggered after the DetailsView is expanded.	GridDataDetailsViewExpandedEventArgs	Routed Event
DetailsViewCollapsing	this event is triggered before the DetailsView is collapsed. this event is a cancelable event. By canceling this event, the corresponding DetailsView remains expanded.	GridDataValueCancelEventArgs<GridDataRecord>	
DetailsViewCollapsed	this event gets triggered after the DetailsView get collapsed.	GridDataValueEventArgs<GridDataRecord>	

[Sample Link](#)

To view samples:

1. Select Start > Programs > Syncfusion > Essential Studio x.x.xx > Dashboard.
2. Click Run Samples for WPF in the User Interface Edition panel.
3. Select GridDataControl.
4. Expand the Appearance Features item in the Sample Browser.
5. Choose the Details View Demo sample

[Adding Details View to an Application](#)

One can easily add the details view to the GridData control by defining the DetailsViewTemplate and binding it to the corresponding GridData control. In the following procedure we have bound the GridData control with a list of product information and created a details view template and bound it to the GridData control.

1. Bind an items source to the grid. Refer to the following link for more information about binding items source to the grid. [Data Binding](#)
2. Define a data template for the details view. You can bind the data by fetching it through Record.Data (data is the underlying object bound).

XML

```
<DataTemplate x:Key="EditableView">
<Border HorizontalAlignment="Left" BorderThickness="0.5" BorderBrush="Gray"
CornerRadius="2" ClipToBounds="True">
<Grid Margin="5" Width="300" HorizontalAlignment="Left">
```

```

<Grid.ColumnDefinitions>
<ColumnDefinition Width="0.5*" />
<ColumnDefinition Width="0.5*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Grid>
<Label Content="Supplier Info" Grid.Row="0" Grid.Column="0" Grid.ColumnSpan=
"2" HorizontalAlignment="Center" VerticalAlignment="Top" Margin="2" FontSize
="13" FontWeight="Bold" />
<TextBlock Text="Company" FontWeight="Bold" Margin="0,5,5,0" Grid.Row="1" Gr
id.Column="0" />
<TextBox Text="{Binding Record.Data.Suppliers.CompanyName}" Margin="0,5,0,0"
Grid.Row="1" Grid.Column="1" />
<TextBlock Text="Contact Person" FontWeight="Bold" Margin="0,5,5,0" Grid.Row
="2" Grid.Column="0" />
<TextBox Text="{Binding Record.Data.Suppliers.ContactName}" Margin="0,5,0,0"
Grid.Row="2" Grid.Column="1" />
<TextBlock Text="Title" FontWeight="Bold" Margin="0,5,5,0" Grid.Row="3" Grid
.Column="0" />
<TextBox Text="{Binding Record.Data.Suppliers.ContactTitle, Mode=OneTime}" M
argin="0,5,0,0" Grid.Row="3" Grid.Column="1" />
<TextBlock Text="Address" FontWeight="Bold" Margin="0,5,5,0" Grid.Row="4" Gr
id.Column="0" />
<TextBox Text="{Binding Record.Data.Suppliers.Address}" Margin="0,5,0,0" Gri
d.Row="4" Grid.Column="1" />
<TextBlock Text="City" FontWeight="Bold" Margin="0,5,5,0" Grid.Row="5" Grid.
Column="0" />
<TextBox Text="{Binding Record.Data.Suppliers.City}" Margin="0,5,0,0" Grid.R
ow="5" Grid.Column="1" />
<TextBlock Text="Postal Code" FontWeight="Bold" Margin="0,5,5,0" Grid.Row="6
" Grid.Column="0" />
<TextBox Text="{Binding Record.Data.Suppliers.PostalCode}" Margin="0,5,0,0"
Grid.Row="6" Grid.Column="1" />
<TextBlock Text="Country" FontWeight="Bold" Margin="0,5,5,0" Grid.Row="7" Gr
id.Column="0" />
<TextBox Text="{Binding Record.Data.Suppliers.Country}" Margin="0,5,0,0" Gri
d.Row="7" Grid.Column="1" />
<TextBlock Text="Phone" FontWeight="Bold" Margin="0,5,5,0" Grid.Row="8" Grid
.Column="0" />
<TextBox Text="{Binding Record.Data.Suppliers.Phone}" Margin="0,5,0,0" Grid.
Row="8" Grid.Column="1" />
</Grid>
</Border>
</DataTemplate>

```

3. Bind the defined data template to the GridData control.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid"
Grid.Row="0"
UpdateMode="PropertyChanged"
AutoPopulateColumns="True"
ColumnSizer="AutoOnLoad"
AutoPopulateRelations="False"
ShowGroupDropArea="True"
IsGroupsExpanded="True"
ItemsSource="{Binding ProductList}"
ShowAddNewRow="False"
ShowFilterBar="False"
UnboundRowPosition="Top"
ShowErrorTooltips="True"
DetailsViewTemplate="{StaticResource EditableView}"
VisualStyle="Office14Blue"/>
```

C#

```
/// Initializing ViewModel
ViewModel viewModel = new ViewModel();
/// Initializing GridDataControl
GridDataControl dataGrid = new GridDataControl();
/// Assigning ItemsSource
dataGrid.ItemsSource = viewModel.ProductList;
dataGrid.AutoPopulateColumns = true;
dataGrid.AutoPopulateRelations = false;
/// Assigning DetailsViewTemplate
dataGrid.DetailsViewTemplate = (DataTemplate) this.Resources["EditableView"];
```

4. Click on the expander cell in the grid to expand or collapse the details view of that particular record or row.

ProductID	SupplierID	CategoryID	ProductName	EnglishName
1	1	1	Chai	Dharamsala Tea

Supplier Info

Company: Exotic Liquids

Contact Person: Charlotte Cooper

Title: Purchasing Manager

Address: 49 Gilbert St.

City: London

Postal Code: EC1 4SD

Country: UK

Phone: (71) 555-2222

2	1	1	Chang	Tibetan Barley Beer
3	1	2	Aniseed Syrup	Licorice Syrup
4	2	2	Chef Anton's Cajun Seasoning	Chef Anton's Cajun Seasoning
5	2	2	Chef Anton's Gumbo Mix	Chef Anton's Gumbo Mix

Product ID	Supplier ID	Category ID	Product	Quantity Per Unit	Unit Price	Stock	Ordered Units	Reorder Level	Discontinued
1	1	1	Dharamsala Tea	10 boxes x 20 bags	18	39	0	10	<input type="checkbox"/>
2	1	1	Tibetan Barley Beer	24 - 12 oz bottles	19	17	40	25	<input type="checkbox"/>
3	1	2	Licorice Syrup	12 - 550 ml bottles	10	13	70	25	<input checked="" type="checkbox"/>

Product Info



Name: Licorice Syrup

Quantity Per Unit: 12 - 550 ml bottles

Unit Price: 10

Stock: 13

Orders Suppliers

OrderID	ProductID	UnitPrice	Quantity	Discount
10107	3	7	20	0.2
10126	3	7	28	0
10199	3	7	40	0
10289	3	8	30	0
10405	3	8	50	0
10485	3	8	20	0.1

4	2	2	Chef Anton's Cajun Seas...	48 - 6 oz jars	22	53	0	0	<input checked="" type="checkbox"/>
5	2	2	Chef Anton's Gumbo Mix	36 boxes	21.35	0	0	0	<input checked="" type="checkbox"/>
6	3	2	Grandma's Boysenberry...	12 - 8 oz jars	25	120	0	25	<input type="checkbox"/>

Column Drag and Drop

Grid allows you to rearrange the columns by a simple drag-and-drop. Once enabled, this feature lets you drag and drop any number of column headers from one position to another and thus lets you rearrange the grid columns.

You can turn on this feature at grid level for all the columns, by setting the `grid.AllowDragColumns` property to true. It is also possible to control dragging at column level. Setting the `AllowDrag` property of the respective visible columns to true enables column dragging on those columns. This is illustrated in the following code example.

The following code illustrates dragging at grid level:

XML

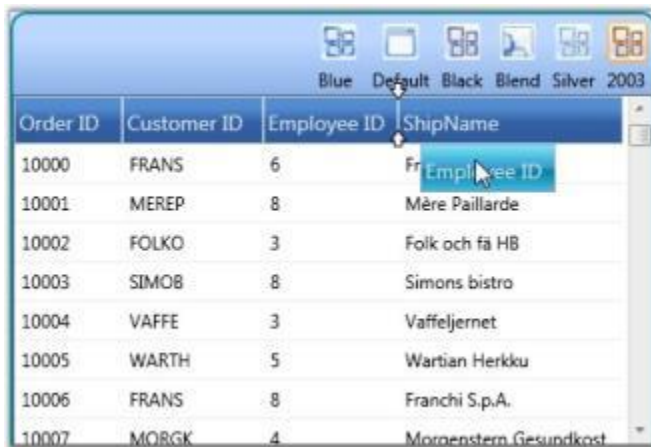
```
<syncfusion:GridDataControl x:Name="grid" AllowDragColumns="True"
ItemsSource="{StaticResource orderSource}" ShowColumnOptions="True" />
```

The following code illustrates dragging at column level:

XML

```
<syncfusion:GridDataControl x:Name="grid" AutoPopulateColumns="False"
AutoPopulateRelations="False" ItemsSource="{StaticResource orderSource}"
ShowColumnOptions="True" >
<syncfusion:GridDataControl.VisibleColumns>
<syncfusion:GridDataVisibleColumn MappingName="OrderID" HeaderText="Order
ID" AllowDrag="True" />
<syncfusion:GridDataVisibleColumn MappingName="CustomerID"
HeaderText="Customer ID" AllowDrag="False" />
<syncfusion:GridDataVisibleColumn MappingName="EmployeeID"
HeaderText="Employee ID" AllowDrag="True"/>
<syncfusion:GridDataVisibleColumn MappingName="ShipName"
HeaderText="ShipName" AllowDrag="True"/>
</syncfusion:GridDataControl.VisibleColumns>
</syncfusion:GridDataControl>
```

The following screenshot illustrates column level dragging:

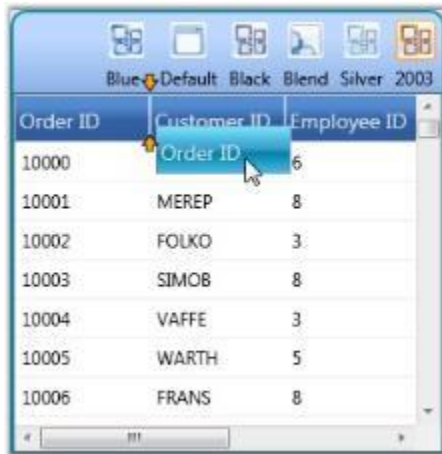
*Customizing the Drag Indicator*

Grid also provides properties that let you modify the color of the drag indicators.

The following code illustrates changing the color of drag indicator:

C#

```
dataGrid.Model.TableProperties.DragIndicatorInnerBrush = Brushes.Orange;
dataGrid.Model.TableProperties.DragIndicatorOuterBrush = Brushes.Black;
```



Column Options

The GridData control provides support to customize the individual columns in the grid by using the Column Options feature. The customization options are wrapped up into a panel that pops up on clicking the ColumnOptions icon in the column header, so that the users can work with them in an interactive manner. With this feature now available, you can toggle the customization options within seconds, without writing much code.

You can enable or disable this feature by using the ShowColumnOptions property of the GridData control.

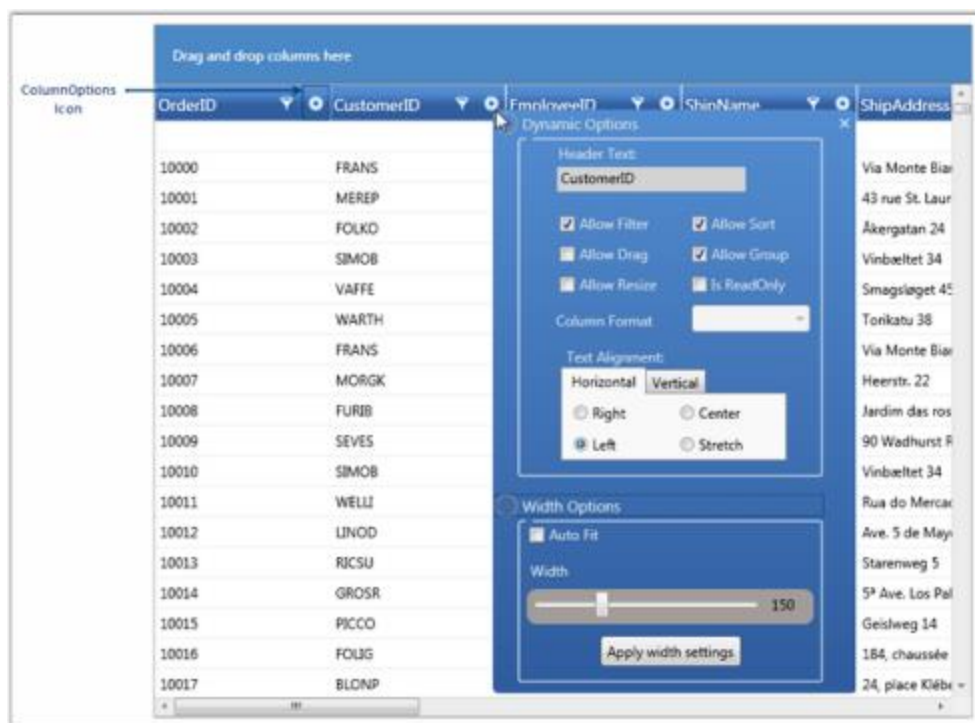
XML

```
<syncfusion:GridDataControl x:Name="dataGrid" ShowColumnOptions="True"
ItemsSource="{StaticResource orderSource}" >
</syncfusion:GridDataControl>
```

C#

```
this.dataGridControl.ShowColumnOptions = true;
```

The following screenshot illustrates how the ColumnOptions icon is displayed for individual columns.



Customization Options

The following are the column customization options provided by the GridData control.

Column Option

Column Option	Description
Allow Filter	Selecting this check box, enables filtering on the respective column. To enable "filtering" on all the columns in the grid at once, set the ShowFilters property to true.
Allow Sort	Selecting this check box, enables sorting when the user clicks on the column header of the respective column. To enable "sorting" on all the columns in the grid at once, set the AllowSort property to true.
Allow Drag	Selecting this check box, enables you to rearrange columns in the grid by dragging the desired column headers. Also you can group the columns by dragging the column headers onto the Group Drop Area. Note that the "Allow Group" option must be set to true for this to take effect. To enable "drag-and-drop" feature on all the columns in the grid at once, set the AllowDragColumns property to true.
Allow Group	Selecting this check box, enables you to create groups at run time by dragging column headers onto the Group Drop Area. Note that the "Allow Drag" option must be set to true for this to take effect. To enable "grouping" on all the columns in the grid at once, set the ShowGroupDropArea to true.
Allow Resize	Selecting this check box, enables you to resize columns in the grid, by pressing and dragging the "mouse-resize" pointer that is displayed over the column divider. To enable the "resizing" operation on all the columns in the grid at once, set the AllowResizeColumns property to true.

Is ReadOnly	Selecting this check box, makes the respective column read-only, allowing no edits to be performed on the column. To enable the "editing" operation on all the columns in the grid at once, set the AllowEdit property to <i>true</i> .
Column Format	This drop-down menu lists the available data formats for the respective columns types.
Text Alignment	This option enables to align text both horizontally and vertically.
Auto Fit	Selecting this check box, adjusts the width of the corresponding column to fit its content and clearing it resets the column width to its default value (i.e., DefaultColumnWidth value). Note that you must click the Apply width settings button for this option to take effect.
Width	You can select the appropriate width for the respective column by pressing and dragging the slider handle with the mouse and clicking the Apply width settings button.

ColumnBasedSizing

ColumnBasedSizing enables you to set width of visible columns based on available width. This feature is useful when column sizing is required for columns based on available width, cell content. Feature can be used when GridDataControl VisibleColumns are set individually with specific widths.

Applying Width Value

Columns are sized based on values users set.

Allowed width values are:

- SizeToCells
- SizeToHeader
- Auto

Note: You can also provide numeric values for , *where* can be preceded by any double value.

The following code illustrates how to set the Width property of GridDataVisibleColumn.

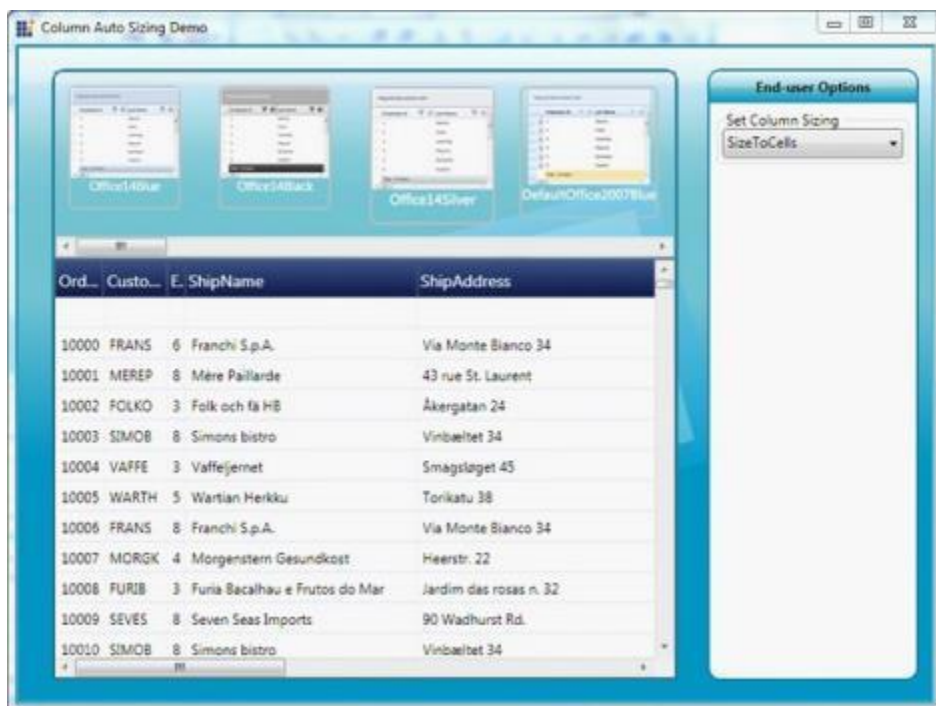
XML

```
Width Value set to Star:
<syncfusion:GridDataVisibleColumn MappingName="OrderID" Binding="{Binding OrderID}" Width="1.2*"/>
Width value set to "Double"
<syncfusion:GridDataVisibleColumn MappingName="CustomerID" Width="40" >
</syncfusion:GridDataVisibleColumn>
Width value set to "SizeToCells"
<syncfusion:GridDataVisibleColumn MappingName="EmployeeID"
Width="SizeToCells" Binding="{Binding EmployeeID}"/>
Width value set to "SizeToHeader"
<syncfusion:GridDataVisibleColumn MappingName="ShipName"
Width="SizeToHeader" Binding="{Binding ShipName}"/>
</syncfusion:GridDataControl.VisibleColumns>
```

Define the Width, by using the following code.

C#

```
Width = new GridDataControlLength(value, GridControlLengthUnitType.Star);
// (or)
Width = new GridDataControlLength(value);
```



Property Details

The following table consists of the property details.

Name of the Property	Description	Type of Property	Value It Accepts	Property Syntax
ActualWidth	GridDataVisibleColumn's actual width in double value. Users can only get the value.	Normal	Double	VisibleColumn.ActualWidth
Width	Holds the width value for GridDataVisibleColumn in units of GridDataControlLength	Dependency property	GridDataControlLength	visibleColumn.Width=new GridDataControlLength(value, GridControlLengthUnitType);
MaxLength	Holds the value for maximum records to be considered while calculating cell content	Normal	int	MaxLength

	size. Default value is 1000. Set as -1 to consider all values.			
--	---	--	--	--

Summaries

GridDataControl provides support to add additional rows at the bottom of the grid table. Such rows are fixed and are used to brief information about the grid data, called Summaries. For instance, you can display the record count or maximum value as summary.

The following are the built-in summary types supported by grid. They are otherwise called as basic summaries.

- CountAggregate
- Int32Aggregate
- DoubleAggregate
- CustomAggregate (used with custom summaries)

SummaryRows Collection

Grid provides three kinds of SummaryRows collections – SummaryRows, TableSummaryRows and CaptionSummaryRows, in order to separate the summary kinds. This collection stores all the summaries for a given grid, in which each entry corresponds to a summary row holding the various summary details such as summary title, summary style, its visibility, and more importantly the SummaryColumns collection.

SummaryColumns Collection

Every summary row contains a corresponding SummaryColumns collection. This collection stores the group of columns whose values are used for the summary calculation. As this is a collection of columns, you could infer that summaries can be calculated from more than one column. This collection explores the properties that are essential for generating summary information. The following are some of the properties that are used to generate the summary information.

- MappingName: mapping name of the column used
- Format: summary format, for example, "{SUM=##.00}"
- SummaryType: built-in summary type

This section comprises the following topics:

Creating Summaries

The following steps illustrate how to create a Summary.

1. Define the summary column that is used to calculate the summary.
2. Then define a summary row and associate the above summary column with this summary row.
3. Finally add the summary row into the corresponding SummaryRows collection. Make sure that the appropriate summary type is enabled.

These steps have been clearly explained for individual summary types.

Group Summaries

Table Summaries

Caption Summaries

Custom Summaries

Summary Types

Essential Grid supports the following summary types.

Group Summaries

As the name implies, the Group Summary is associated with every grid group. The `GridDataControl.SummaryRows` manages summaries of this type. It provides support to add multiple summaries, i.e., you can have more than one summary row for every group. These summaries are enabled by setting the `ShowGroupSummaries` property to *true*.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" ShowGroupDropArea="True"
AutoPopulateRelations="False" ShowAddNewRow="False"
ItemsSource="{StaticResource ordersSource}" ShowGroupSummaries="True">
  <syncfusion:GridDataControl.SummaryRows>
    <syncfusion:GridDataSummaryRow ShowSummaryInRow="True" Title="Total Freight:
{FreightSummary} For {CountSummary} Items" TitleColumnCount="2">
      <syncfusion:GridDataSummaryRow.SummaryColumns>
        <syncfusion:GridDataSummaryColumn Name="FreightSummary"
MappingName="Freight" SummaryType="Int32Aggregate"
Format="{Sum:c}" />
        <syncfusion:GridDataSummaryColumn Name="CountSummary"
MappingName="OrderDate" SummaryType="CountAggregate"
Format="{Count:d}" />
      </syncfusion:GridDataSummaryRow.SummaryColumns>
    </syncfusion:GridDataSummaryRow>
  </syncfusion:GridDataControl.SummaryRows>
</syncfusion:GridDataControl>
```

C#

```
this.dataGrid.SummaryRows.Add(new GridDataSummaryRow()
{
    ShowSummaryInRow = true,
    Title = "Total Freight: {FreightSummary} For {CountSummary} Items",
    TitleColumnCount = 2,
    SummaryColumns = new ObservableCollection<GridDataSummaryColumn>()
    {
        new GridDataSummaryColumn()
        {
            Name = "FreightSummary",
            MappingName="Freight",
            SummaryType= GridDataSummaryType.Int32Aggregate,
            Format="{Sum:c}"
        },
        new GridDataSummaryColumn()
    }
}
```

```

Name = "CountSummary",
MappingName="OrderDate",
SummaryType= GridDataSummaryType.CountAggregate,
Format="{Count:d}"
}
}
});

```

OrderID	CustomerID	Freight	ShipName
ShipName: Alfred's Futterkiste - 6 Items			
10702	ALFKI	23.94	Alfred's Futterkiste
10952	ALFKI	40.42	Alfred's Futterkiste
10692	ALFKI	61.02	Alfred's Futterkiste
10643	ALFKI	29.46	Alfred's Futterkiste
10835	ALFKI	69.53	Alfred's Futterkiste
10062	ALFKI	47.22	Alfred's Futterkiste
Total Freight: \$272.00 For 6 Items			
ShipName: Ana Trujillo Emparedados y helados - 4 Items			
10926	ANATR	39.92	Ana Trujillo Empa...
10308	ANATR	1.61	Ana Trujillo Empa...
10759	ANATR	11.99	Ana Trujillo Empa...
10625	ANATR	43.9	Ana Trujillo Empa...
Total Freight: \$97.00 For 4 Items			

Combining Summary Column Values in the GridDataControl

GridDataControl enables to combine summary values in summary columns and display them in a single summary row.

For example, let us consider you want to combine the summaries from all the summary columns and display them in a single row, instead of displaying them in the individual summary columns. In such a case, you must set the `ShowSummaryInRow` property to `true` and provide a proper summary title string specifying the combined summary value. The following code example illustrates this.

XML

```

<syncfusion:GridDataControl.SummaryRows>
<syncfusion:GridDataSummaryRow ShowSummaryInRow="True" Title="'Charges -
{FreightSummary}$ for {OrderCount} Items'">
<syncfusion:GridDataSummaryRow.RowStyle>
<syncfusion:GridDataStyleInfo Background="Yellow" />
</syncfusion:GridDataSummaryRow.RowStyle>
<syncfusion:GridDataSummaryRow.SummaryColumns>
<syncfusion:GridDataSummaryColumn Name="FreightSummary"
MappingName="Freight" SummaryType="Int32Aggregate"
Format="'{Sum:##.00}'" />
<syncfusion:GridDataSummaryColumn Name="OrderCount" MappingName="OrderDate"
SummaryType="CountAggregate"
Format="'{Count}'" />

```

```

</syncfusion:GridDataSummaryRow.SummaryColumns>
</syncfusion:GridDataSummaryRow>
</syncfusion:GridDataControl.SummaryRows>

```

Order ID	CustomerID	Freight	Order Date	Ship Name
10952	ALFKI	40.42	2/7/1994 12:00:00...	Alfred's Futterkiste
10702	ALFKI	23.94	9/6/1993 12:00:00...	Alfred's Futterkiste
10835	ALFKI	69.53	12/9/1993 12:00:00...	Alfred's Futterkiste
Charges - 273.00\$ for 7 Items				
CustomerID : ANATR - 4 Items				
10759	ANATR	11.99	10/22/1993 12:00:00...	Ana Trujillo Emparedado...
10625	ANATR	43.9	7/2/1993 12:00:00...	Ana Trujillo Emparedado...
10926	ANATR	39.92	1/26/1994 12:00:00...	Ana Trujillo Emparedado...
10308	ANATR	1.61	8/12/1992 12:00:00...	Ana Trujillo Emparedado...
Charges - 97.00\$ for 4 Items				

Order ID	CustomerID	Freight	Order Date	Ship Name
10692	ALFKI	61.02	8/27/1993 12:00:00 AM	Alfred's Futterkiste
10952	ALFKI	40.42	2/7/1994 12:00:00 AM	Alfred's Futterkiste
10702	ALFKI	23.94	9/6/1993 12:00:00 AM	Alfred's Futterkiste
10835	ALFKI	69.53	12/9/1993 12:00:00 AM	Alfred's Futterkiste
		273.00	7	
CustomerID : ANATR - 4 Items				
10759	ANATR	11.99	10/22/1993 12:00:00 AM	Ana Trujillo Empared...
10625	ANATR	43.9	7/2/1993 12:00:00 AM	Ana Trujillo Empared...
10926	ANATR	39.92	1/26/1994 12:00:00 AM	Ana Trujillo Empared...
10308	ANATR	1.61	8/12/1992 12:00:00 AM	Ana Trujillo Empared...
		97.00	4	

Table Summaries

Caption Summaries

Table Summaries

The Table Summary is associated with the whole grid table itself. The `GridDataControl.TableSummaryRows` collection manages summaries of this type. It provides support to add multiple summaries. These summaries are enabled by setting the `ShowTableSummaries` property to true.

XML

```

<syncfusion:GridDataControl x:Name="dataGrid" ShowGroupDropArea="True"
AutoPopulateRelations="False" ShowAddNewRow="False"
ItemsSource="{StaticResource ordersSource}" ShowTableSummaries="True">
<syncfusion:GridDataControl.TableSummaryRows>

```



```

<syncfusion:GridDataSummaryRow ShowSummaryInRow="True" Title="Total Freight:
{FreightSummary} For {CountSummary} Items"
TitleColumnCount="2">
<syncfusion:GridDataSummaryRow.SummaryColumns>
<syncfusion:GridDataSummaryColumn Name="FreightSummary"
MappingName="Freight" SummaryType="Int32Aggregate"
Format="{Sum:c}" />
<syncfusion:GridDataSummaryColumn Name="CountSummary"
MappingName="OrderDate" SummaryType="CountAggregate"
Format="{Count:d}" />
</syncfusion:GridDataSummaryRow.SummaryColumns>
</syncfusion:GridDataSummaryRow>
</syncfusion:GridDataControl.TableSummaryRows>
</syncfusion:GridDataControl>

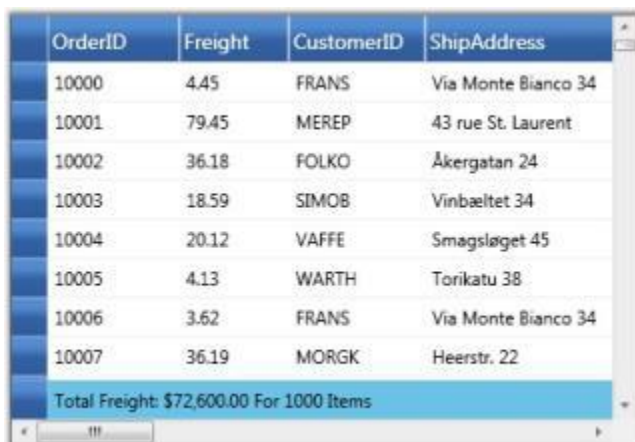
```

C#

```

this.dataGrid.TableSummaryRows.Add(new GridDataSummaryRow()
{
    ShowSummaryInRow = true,
    Title = "Total Freight: {FreightSummary} For {CountSummary} Items",
    TitleColumnCount = 2,
    SummaryColumns = new ObservableCollection<GridDataSummaryColumn>()
    {
        new GridDataSummaryColumn()
        {
            Name = "FreightSummary",
            MappingName="Freight",
            SummaryType= GridDataSummaryType.Int32Aggregate,
            Format="{Sum:c}"
        },
        new GridDataSummaryColumn()
        {
            Name = "CountSummary",
            MappingName="OrderDate",
            SummaryType= GridDataSummaryType.CountAggregate,
            Format="{Count:d}"
        }
    }
});

```



OrderID	Freight	CustomerID	ShipAddress
10000	4.45	FRANS	Via Monte Bianco 34
10001	79.45	MEREP	43 rue St. Laurent
10002	36.18	FOLKO	Åkergatan 24
10003	18.59	SIMOB	Vinbæltet 34
10004	20.12	VAFFE	Smagsløget 45
10005	4.13	WARTH	Torikatu 38
10006	3.62	FRANS	Via Monte Bianco 34
10007	36.19	MORGK	Heerstr. 22
Total Freight: \$72,600.00 For 1000 Items			

Group Summaries

Caption Summaries

Caption Summaries

Grid provides built-in support for caption summaries, where the summary values are displayed in the group caption cells. You can have only one caption summary row for a grid table. The `GridDataControl.CaptionSummaryRow` property is used to set up a caption summary. The caption summary is enabled by setting the `ShowGroupSummaryInCaption` property to true.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" ShowGroupDropArea="True"
AutoPopulateRelations="False" ShowAddNewRow="False"
ItemsSource="{StaticResource ordersSource}"
ShowGroupSummaryInCaption="True">
<syncfusion:GridDataControl.CaptionSummaryRow>
<syncfusion:GridDataSummaryRow ShowSummaryInRow="True" Title="Total Freight:
{FreightSummary} For {CountSummary} Items"
TitleColumnCount="2">
<syncfusion:GridDataSummaryRow.SummaryColumns>
<syncfusion:GridDataSummaryColumn Name="FreightSummary"
MappingName="Freight" SummaryType="Int32Aggregate"
Format="{Sum:c}" />
<syncfusion:GridDataSummaryColumn Name="CountSummary"
MappingName="OrderDate" SummaryType="CountAggregate"
Format="{Count:d}" />
</syncfusion:GridDataSummaryRow.SummaryColumns>
</syncfusion:GridDataSummaryRow>
</syncfusion:GridDataControl.CaptionSummaryRow>
</syncfusion:GridDataControl>
```

C#

```
this.dataGrid.ShowGroupSummaryInCaption = true;
this.dataGrid.CaptionSummaryRow = new GridDataSummaryRow()
{
    ShowSummaryInRow = true,
    Title = "Total Freight: {FreightSummary} For {CountSummary} Items",
    TitleColumnCount = 2,
    SummaryColumns = new ObservableCollection<GridDataSummaryColumn>()
    {
        new GridDataSummaryColumn()
        {
            Name = "FreightSummary",
            MappingName="Freight",
            SummaryType= GridDataSummaryType.Int32Aggregate,
            Format="{Sum:c}"
        },
        new GridDataSummaryColumn()
        {
            Name = "CountSummary",
            MappingName="OrderDate",
```

```

SummaryType= GridDataSummaryType.CountAggregate,
Format="{Count:d}"
}
}
};

```

OrderID	CustomerID	Freight	ShipName
Total Freight: \$272.00 For 6 Items			
10702	ALFKI	23.94	Alfred's Futter
10952	ALFKI	40.42	Alfred's Futter
10692	ALFKI	61.02	Alfred's Futter
10643	ALFKI	29.46	Alfred's Futter
10835	ALFKI	69.53	Alfred's Futter
10062	ALFKI	47.22	Alfred's Futter
Total Freight: \$97.00 For 4 Items			
10926	ANATR	39.92	Ana Trujillo En
10308	ANATR	1.61	Ana Trujillo En
10759	ANATR	11.99	Ana Trujillo En
10625	ANATR	43.9	Ana Trujillo En
Total Freight: \$521.00 For 13 Items			

Group Summaries

Table Summaries

Custom Summaries

You can also impose custom calculation logic for deriving summary information. This is done by implementing the `IGridDataSummaryAggregate` interface to define the custom logic, and associating this custom logic to the `SummaryColumn.CustomAggregate` property. The `IGridDataSummaryAggregate` interface aids in building user-defined logic for summary calculation.

The following steps illustrate the functioning of the `IGridDataSummaryAggregate` interface.

1. First, you need to define a custom property to get and set the summary value.
2. Then implement the `CalculateAggregateFunc` interface method and in-place your own logic of calculating the summary value. It returns a `System.Action` delegate for the aggregate, where T1 represents the source list of items for which the summary needs to be calculated, T2 specifies the property (summary column) and T3 is the Property Descriptor of the custom aggregate class itself. The `CalculateAggregateFunc` calculates the summary value using these parameters, and assigns the final summary value to the custom property defined in the first step.

The following code example illustrates the partial implementation of the built-in `Int32Aggregate` that implements the `IGridDataSummaryAggregate`. It calculates the Sum value for a specific column.

C#

```
class GridDataInt32Aggregate : IGridDataSummaryAggregate
{
    public GridDataInt32Aggregate()
    {
    }
    public int Sum
    {
        get;
        set;
    }
    public Action<IEnumerable, string, PropertyDescriptor>
    CalculateAggregateFunc()
    {
        return (items, property, pd) =>
        {
            if(pd.Name == "Sum")
            {
                this.Sum = Convert.ToInt32(items.AsQueryable().Sum(property));
            }
        };
    }
}
```

Example

This example uses the Stock Portfolio Database that has a column named "Change" that shows the rate of change of market value of the stocks. Let us consider, you need to display the Standard Deviation of the values of the "Change" column, industry-wise. This can be achieved by using a group summary as the grid is already grouped by Industry. Since the built-in summaries do not support this type of calculation, you have to create custom summaries and write custom code to calculate the standard deviation values. The following steps illustrate this.

1. Define the custom summary logic to calculate the standard deviation.

CSHARP

```
public class CustomAggregate : IGridDataSummaryAggregate
{
    public CustomAggregate()
    {
    }
    public double StdDev
    {
        get;
        set;
    }
    public Action<System.Collections.IEnumerable, string,
    System.ComponentModel.PropertyDescriptor> CalculateAggregateFunc()
    {
        return (items, property, pd) =>
        {
            var enumerableItems = items as IEnumerable<Quotes>;
            if (pd.Name == "StdDev")
            {
                this.StdDev = enumerableItems.StdDev<Quotes>(q => q.Change);
            }
        };
    }
}
```

```

    }
    };
    }
    }
    public static class LinqExtensions
    {
        public static double StdDev<T>(this IEnumerable<T> values, Func<T, double?>
selector)
        {
            double ret = 0;
            var count = values.Count();
            if (count > 0)
            {
                // Compute the Average
                double? avg = values.Average(selector);
                // Perform the Sum of (value-avg)^2
                double sum = values.Select(selector).Sum(d =>
                {
                    if (d.HasValue)
                    {
                        return Math.Pow(d.Value - avg.Value, 2);
                    }
                    return 0.0;
                });
                // Put it all together
                ret = Math.Sqrt((sum) / (count - 1));
            }
            return ret;
        }
    }
}

```

2. Bind the custom summary instance to the grid summary calculation.

XML

```

<syncfusion:LayoutControl.LeftContent>
<syncfusion:GridDataControl
ItemsSource="{Binding Source={StaticResource Quotes}}"
Name="gridDataControl1"
ShowGroupSummaries="True"
ShowGroupSummaryInCaption="False"
AutoPopulateColumns="True"
AutoPopulateRelations="False"
ShowAddNewRow="False"
AllowEdit="False"
AllowDelete="False">
<syncfusion:GridDataControl.GroupedColumns>
<syncfusion:GridDataGroupColumn ColumnName="Industry_IndustryID" />
</syncfusion:GridDataControl.GroupedColumns>
<syncfusion:GridDataControl.SummaryRows>
<syncfusion:GridDataSummaryRow Name="StdDev" ShowSummaryInRow="False"
Title="{Name} - {Count} Items" TitleColumnCount="2">
<syncfusion:GridDataSummaryRow.SummaryColumns>

```

```

<syncfusion:GridDataSummaryColumn Name="StdDevCol" MappingName="Change"
SummaryType="Custom" Format="'StdDev ({StdDev})'" CustomAggregate="{Binding
Source={StaticResource stdDevAggregate}}">
<syncfusion:GridDataSummaryColumn.ColumnStyle>
<syncfusion:GridDataStyleInfo Background="Pink" HorizontalAlignment="Center"
/>
</syncfusion:GridDataSummaryColumn.ColumnStyle>
</syncfusion:GridDataSummaryColumn>
</syncfusion:GridDataSummaryRow.SummaryColumns>
</syncfusion:GridDataSummaryRow>
</syncfusion:GridDataControl.SummaryRows>
</syncfusion:GridDataControl>

```

Symbol	CompanyName	Price	Change	Perce
Industry_IndustryID : 1 - 8 Items				
TNH	Terra Nitrogen Co...	93.16	-6.32	-6.35
EDEN	EDEN Bioscience...	1.06	0.14	15.22
AVD	American Vangua...	10.23	0.09	0.89
SYT	Syngenta AG	33.70	-0.65	-1.89
LSCO	LESCO Inc.	0.00	0	0
SMG	The Scotts Comp...	28.07	0.22	0.79
AGU	Agrium Inc.	31.01	-1.63	-4.99
BG	Bunge Limited	38.90	0.2	0.52
			StdDev (2.243644595614...	
Industry_IndustryID : 2 - 4 Items				
AL	Alcan Inc.	0.00	0	0
AA	Alcoa Inc.	9.67	-1.17	-10.75
CENX	Century Aluminu...	9.65	-2.47	-20.38
ACH	Aluminum Corp...	10.43	-0.32	-2.98
			StdDev (1.103298086043...	

Creating Summaries

Summary Style

You can customize the appearance of summary cells by applying the desired formatting settings to the `GridDataSummaryColumn.ColumnStyle` and `GridDataSummaryRow.RowStyle` properties. The following code example illustrates this.

XML

```

<syncfusion:GridDataSummaryRow>
<syncfusion:GridDataSummaryRow.SummaryColumns>
<syncfusion:GridDataSummaryColumn>
<syncfusion:GridDataSummaryColumn.ColumnStyle>
<syncfusion:GridDataStyleInfo Background="LightPink"
Foreground="MidnightBlue" />

```

```

</syncfusion:GridDataSummaryColumn.ColumnStyle>
</syncfusion:GridDataSummaryColumn>
</syncfusion:GridDataSummaryRow.SummaryColumns>
<syncfusion:GridDataSummaryRow.RowStyle>
<syncfusion:GridDataStyleInfo Background="LightGreen" />
</syncfusion:GridDataSummaryRow.RowStyle>
</syncfusion:GridDataSummaryRow>

```

C#

```

GridDataSummaryColumn summaryCol = new GridDataSummaryColumn();
summaryCol.ColumnStyle = new GridDataStyleInfo();
summaryCol.ColumnStyle.Background = new SolidColorBrush(Colors.LightPink);
summaryCol.ColumnStyle.Foreground = new
SolidColorBrush(Colors.MidnightBlue);
GridDataSummaryRow summaryRow = new GridDataSummaryRow();
summaryRow.SummaryColumns.Add(summaryCol);
summaryRow.RowStyle = new GridDataStyleInfo();
summaryRow.RowStyle.Background = new SolidColorBrush(Colors.LightGreen);

```

Symbol	CompanyName	Price	Change	Percent
Industry_IndustryID: 1 - 8 Items				
TNH	Terra Nitrogen...	93.16	-6.32	-6.35
EDEN	EDEN Bioscienc...	1.06	0.14	15.22
AVD	American Vang...	10.23	0.09	0.89
SYT	Syngenta AG	33.70	-0.65	-1.89
LSCO	LESCO Inc.	0.00	0	0
SMG	The Scotts Com...	28.07	0.22	0.79
AGU	Agrium Inc.	31.01	-1.63	-4.99
BG	Bunge Limited	38.90	0.2	0.52
			StdDev (2.243644595...	

Hierarchy

GridData control can display nested tables in a hierarchy using a master-detail configuration. In a hierarchical view, all the tables in the data source are inter-connected by means of relations. Generally a relation between any two tables can take one of the following forms:

- 1:1 (One parent record to one child record)
- 1:n (One parent record to n child records)
- n:1 (n parent records to one child record)
- n:n (n parent record to n child records)

where n is a number of records in a table

With the nested tables (one table nested inside another table), each record in the parent table has an associated set of records in the child table. Every record in the relation is provided with +/- button called RecordPlusMinus that can be expanded and collapsed to bring the underlying records in the child table for display. The number of tables that can be nested with relations using a GridData control is unlimited.

The Relations Collection

A relation can be created by defining a GridDataRelation and adding it into the Grid.Relations property. This property is an Observable collection of GridDataRelation objects. It manages the entire relations for a grid. The GridDataRelation instance defines a grid relation in the following properties.

Properties

Property	Description
RelationalColumn	Specifies the column that defines the relation.
RelationType	Specifies the type of relation. Currently, only MasterDetails type relation is supported.
TableProperties	Used to set the table properties of the child table.

Example

The following code snippet illustrates how to create a relational column:

C#

```
//Fetching parent table.
DataSet ds = new DataSet();
using (SqlCeConnection con = new SqlCeConnection(string.Format(@"Data Source
= {0}", LayoutControl.FindFile("Northwind.sdf"))))
{
    con.Open();
    SqlCeDataAdapter sda = new SqlCeDataAdapter("SELECT * FROM Employees", con);
    sda.Fill(ds, "Employees");
}
//Fetching child table.
using (SqlCeConnection con1 = new SqlCeConnection(string.Format(@"Data
Source = {0}", LayoutControl.FindFile("Northwind.sdf"))))
{
    con1.Open();
    SqlCeDataAdapter sda1 = new SqlCeDataAdapter("SELECT * FROM Orders", con1);
    sda1.Fill(ds, "Orders");
}
//Adding relation.
ds.Relations.Add(new DataRelation("Employee_Orders",
ds.Tables[0].Columns["Employee ID"], ds.Tables[1].Columns["Employee ID"]));
```

The following code illustrates binding of the above relation to the grid, and also customizing the child table:

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="False"
AutoPopulateRelations="False" ItemsSource="{StaticResource orderSource}">
    ShowAddNewRow="False">
    <syncfusion:GridDataControl.Relations >
    <syncfusion:GridDataRelation RelationalColumn="Employee_Orders"
    RelationType="MasterDetails" >
    <syncfusion:GridDataRelation.TableProperties>
```

```
<syncfusion:GridDataTableProperties AutoPopulateColumns="True"
AlternatingRowBackground="BlanchedAlmond" RowBackground="Beige" >
</syncfusion:GridDataTableProperties>
</syncfusion:GridDataRelation.TableProperties>
</syncfusion:GridDataRelation>
</syncfusion:GridDataControl.Relations>
</syncfusion:GridDataControl>
```

The following image shows the output of the above given code:

Employee Id	Last Name	First Name	Title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
5	Buchanan	Steven	Sales Manager

Order ID	Customer ID	Employee ID	Ship Name
10005	WARTH	5	Wartian Herkku
10026	MORGK	5	Morgenstern Gesundkost
10049	TOMSP	5	Toms Spezialitäten
10051	FOLKO	5	Folk och få HB
10062	ALFKI	5	Alfred's Futterkiste

The preceding screenshot shows a GDC bound with a nested table whose child table is set with a customized background.

Auto Generate Relations

The grid can automatically detect the data relations in a data set for display. By default, a relation is created for each such data relation found in the data set. Hence the data relations defined in a data set are sufficient enough for the grid.

To auto-generate the relations, set the `AutoPopulateRelations` property of the GridData control to *true*.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="True" ItemsSource="{StaticResource orderSource}">
ShowAddNewRow="False"/>
```

Stacked Headers

Essential Grid allows you to have additional unbound header rows, called Stacked Header Rows that span across visible grid columns. You can group one or more columns under each stacked header.

The StackedHeaderRows Collection

Stacked Header rows for a given grid are gathered under `Grid.StackedHeaderRow` collection. This collection contains the property definitions that control the behavior and appearance of the Stacked

Headers. A StackedHeaderRow collection can be viewed as a set of stacked header rows in which each header row contains a collection of stacked headers, which span across multiple columns.

Every stacked header row is defined by a GridDataStackedHeaderRow. This class contains a property named Columns that is a collection of GridDataStackedHeaderColumn objects and this collection contains an entry for each stacked header.

Following are the properties of the GridDataStackedHeaderColumn:

Property	Description
ColumnSpan	Specifies the number of columns that a particular stacked header should span.
ColumnStyle	Specifies the style for the stacked header column.
HeaderText	Specifies the header text for the stacked header.
VisibleColumns	Specifies the collection of visible columns under the stacked header.

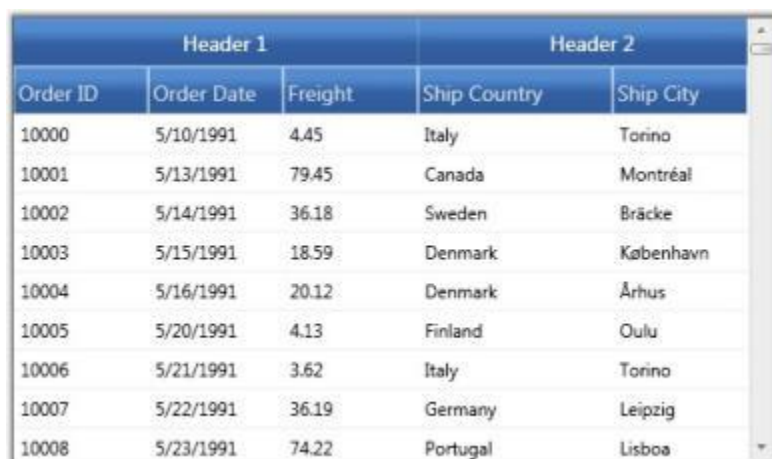
Example

The following code example illustrates the creation of two stacked headers:

XML

```
<syncfusion:GridDataControl x:Name="dataGrid2" AutoPopulateColumns="True"
AutoPopulateRelations="False"
ItemsSource="{StaticResource ordersSource}">
  <syncfusion:GridDataControl.StackedHeaderRows>
    <syncfusion:GridDataStackedHeaderRow Name="Row1">
      <syncfusion:GridDataStackedHeaderRow.Columns>
        <syncfusion:GridDataStackedHeaderColumn HeaderText="Header 1" Name="Header1"
ColumnSpan="3" />
        <syncfusion:GridDataStackedHeaderColumn HeaderText="Header 2" Name="Header2"
ColumnSpan="2" />
      </syncfusion:GridDataStackedHeaderRow.Columns>
    </syncfusion:GridDataStackedHeaderRow>
  </syncfusion:GridDataControl.StackedHeaderRows >
</syncfusion:GridDataControl>
```

Output of the above given code is the following image:



Header 1			Header 2	
Order ID	Order Date	Freight	Ship Country	Ship City
10000	5/10/1991	4.45	Italy	Torino
10001	5/13/1991	79.45	Canada	Montréal
10002	5/14/1991	36.18	Sweden	Bräcke
10003	5/15/1991	18.59	Denmark	København
10004	5/16/1991	20.12	Denmark	Århus
10005	5/20/1991	4.13	Finland	Oulu
10006	5/21/1991	3.62	Italy	Torino
10007	5/22/1991	36.19	Germany	Leipzig
10008	5/23/1991	74.22	Portugal	Lisboa

The preceding screenshot shows a GridData control with stacked headers.

Expression Fields

Expression Fields enable you to add a column that holds calculated values based on other fields in the same record. These expression columns are created in the same way as any unbound column, by using the GridDataUnboundVisibleColumn class. This contains the Expression property that needs to be set with a non-null value for an expression column. Expressions can include arithmetic, logical, relational, and few string operators that finally gets translated into LINQ expressions for evaluation.

The following table lists the supported operators and examples for each.

Supported Operators

Expression	Syntax	Description	Example Usage
Mod	%	Divides first argument by second argument and returns remainder.	[UnitPrice] % 10
Multiplication, Division	*,/	Multiplies/Divides first argument by second argument.	[QunatityPerUnit] * [UnitsInStock]
Addition, Subtraction	+,-	Adds first argument with second argument/Subtracts second argument from the first one.	[UnitsInStock]+[Quantity]
Or	OR	Returns 1 if either the first argument or the second one returns true.	[Val]=50 OR [Val]=100
And	AND	Returns 1 if both parameters return true.	[Val]< 50 AND [Val]>100
Less than	<	Returns true if first parameter is less than the second one.	[OrderID] < 2000
Greater than	>	Returns true if first parameter is greater than the second one.	[OrderID] > 2500

Less than Or Equal to	<=	Returns true if first parameter is less than or equal to the second one.	[OrderID] <= 2050
Greater than Or Equal to	>=	Returns true if first parameter is greater than or equal to the second one.	[OrderID] >= 2056
Equal	=	Returns true if both arguments have same value.	[CustomerID] = 90
Not Equal to	<>	Returns true if both arguments does not have same value.	[CustomerID] <> 95
StartsWith	StartsWith	Returns true if the value starts with the given string.	ProductName StartsWith Char
EndsWith	EndsWith	Returns true if the value ends with the specified string.	ProductName EndsWith i
Contains	Contains	Returns true if the value contains the specified string.	ProductName Contains ha

Example

1. Instantiate a GridDataControl and bind it to a data source.

CSHARP

```
<syncfusion:GridDataControl x:Name="dataGrid" ShowAddNewRow="False"
ShowFilters="False" AutoPopulateColumns="False"
AutoPopulateRelations="False" ItemsSource="{StaticResource productsSource}"
ShowGroupDropArea="True">
</syncfusion:GridDataControl>
```

2. Add an unbound visible column and set its Expression property to the desired formula expression. The unbound visible column also contains the CaseSensitive property that makes the column names specified in the expression, case sensitive, when set to true. If necessary, you can also customize the expression like any other visible column.

XML

```
<syncfusion:GridDataControl.VisibleColumns>
<syncfusion:GridDataUnboundVisibleColumn MappingName="100UnitPrice"
HeaderText="Price of 100 units" Expression="UnitPrice * 100">
<syncfusion:GridDataUnboundVisibleColumn.ColumnStyle>
<syncfusion:GridDataColumnStyle Background="PeachPuff"/>
</syncfusion:GridDataUnboundVisibleColumn.ColumnStyle>
</syncfusion:GridDataUnboundVisibleColumn>
</syncfusion:GridDataControl.VisibleColumns>
```

Product ID	Product Name	Unit Price	Units In Stock	Price of 100 units
1	Chai	18	39	1800
2	Chang	19	17	1900
3	Aniseed Syrup	10	13	1000
4	Chef Anton's Caj...	22	53	2200
5	Chef Anton's Gu...	21.35	0	2135
6	Grandma's Boys...	25	120	2500
7	Uncle Bob's Org...	30	15	3000
8	Northwoods Cra...	40	6	4000
9	Mishi Kobe Niku	97	29	9700

Accessing Expression Values

You can use the `GetUnboundValue` method of Grid Table to access the computed expression value of a particular unbound cell. This is an overloaded method with the following prototypes:

- `GetUnboundValue(RecordIndex, GridDataUnboundVisibleColumn)`
- `GetUnboundValue(RowIndex, ColumnIndex)`

The following code example illustrates how to use this method.

C#

```
// Retrieve the expression value by using row and column indices.
object value = this.dataGrid.Model.Table.GetUnboundValue(5, 5);
```

ToolTips

Essential Grid provides support to associate individual cells with ToolTips. Tooltip is a small pop-up box that appears when you move the mouse over a visual element. It is used to display additional information about the elements without increasing the window size. They are mainly used to display some text data. You can also place any style content such as a group of lines of text, an image, or any control into the tooltip host.

Grid exposes a style property named `TooltipTemplateKey` that is the name of the template to be used for generating tooltip. You can define this template in xaml and then simply assign its name to the `style.TooltipTemplateKey` property. This infers that any style content that is defined in this template could be used to display the tooltip. Once the template is defined, you need to enable its display by setting the `style.ShowTooltip` property to true.

Note: If `style.ShowTooltip` is not set, then the default template associated with the Grid is loaded, and the default template would try to show the `style.CellValue` in a tooltip.

Example

The example below displays a Chart control in the tooltip host. The grid is bound to the Customers table in which the second column is assigned with a tooltip template that holds a data bound chart for tooltip display. Follow the steps below:

1. Define a Template for Tooltip as shown in the following code:

XML

```

<DataTemplate x:Key="chartTemplate">
<syncfusion:Chart Grid.Row="1" Name="Chart1" Height="400" Width="400"
syncfusion:SkinStorage.VisualStudio="Office2007Blue">
<!--Chart Legend declaration-->
<syncfusion:Chart.Legends>
<syncfusion:ChartLegend syncfusion:ChartDockPanel.Dock="Bottom"/>
</syncfusion:Chart.Legends>
<!--Chart area to present chart segments.-->
<syncfusion:ChartArea FontWeight="Bold" FontSize="14" >
<!--X-axis declaration with required property settings.-->
<syncfusion:ChartArea.PrimaryAxis >
<!--Assigning text for the labels in the Primary Axis with the Product
name.-->
<syncfusion:ChartAxis Header="Category" PositionPath="CategoryID"
ContentPath="CategoryName" RangePadding="None"
LabelRotateAngle="270" LabelsSource="{Binding ItemsSource.Categories}"/>
</syncfusion:ChartArea.PrimaryAxis>
<!--Y-axis declaration with required property settings.-->
<syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartAxis Header="Product" RangePadding="None" />
</syncfusion:ChartArea.SecondaryAxis>
<!-- Binding data to the series from the database.-->
<syncfusion:ChartSeries Type="Column" Label="Categories List"
BindingPathsY="Count"
Interior="{StaticResource SeriesBInterior}" DataSource="{Binding
ItemsSource}" />
</syncfusion:ChartArea>
</syncfusion:Chart>
</DataTemplate>

```

2. Associating the above template with the Grid Cell as shown in the following code:

CSHARP

```

var style = model[1, 2];
// Enable tooltip.
style.ShowTooltip = true;
style.CellValue = customer.ContactName;
var cust = customer.Orders.Select(o => o.OrderDetails.Select(od =>
od.Products).Select(p => p.Categories)).ToList();
var finalList = cust.Select(c => new
{
Count = c.Count(),
Categories = c
}).ToList();
style.ItemsSource = finalList;
// Assign template.
style.TooltipTemplateKey = "chartTemplate";

```

Here is a sample screenshot.



Note: For the complete code, refer to the following browser sample.

...\My Documents\Syncfusion\EssentialStudio\<Version Number>\WPF\Grid.WPF\Samples\3.5\WindowsSamples\Product Showcase\Tooltips Demo.

Cell Comments

Essential Grid supports the association of individual cells with Cell Comments.

Comments are notes used to provide context to your data in grid cells. They are used to display information about a cell or range of cells. Text in the comments can be placed as rich text format to emphasize a comment for a cell. You can also place content such as a group of lines (text), an image, or a control into the comment host.

There are two properties that are used to set a Comment for the cells:

- Comment Property
- CommentTemplateKey Property

Comment Property

Comment is a string type property. You can set a comment using this property for cells. The default Data Template is displayed with the comment text.

Example

The following code example illustrates how to display a comment in the default comment Data Template.

C#

```
style.Comment = style.Text;
```

VB.NET

```
style.Comment = style.Text
```

ALFKI	Alfreds Futterkiste	
ANATR	Ana Trujillo Emparedados y helados	
ANTON	Antonio Moreno Taquería	Ana Trujillo Emparedados y helados
AROUT	Around the Horn	

CommentTemplateKey Property

Grid exposes a style property named CommentTemplateKey that is the name of the template to be used for generating Comment. You can define this template in XAML and assign its name to the style.CommentTemplateKey property. This ensures that any style content defined in the template can be used to display the comment.

Example

The following code example illustrates how to display a Chart control in the comment host. The grid is bound to the 'Customers' table in which the second column is assigned with a comment template that holds a data bound chart for comment display.

The following steps illustrate the same:

1. Define a Template for Comments.

XML

```
<DataTemplate x:Key="chartTemplate">
<syncfusion:Chart Grid.Row="1" Name="Chart1" Height="400" Width="400"
syncfusion:SkinStorage.VisualStudio="Office2007Blue">
<!--Chart Legend declaration.-->
<syncfusion:Chart.Legends>
<syncfusion:ChartLegend syncfusion:ChartDockPanel.Dock="Bottom"/>
</syncfusion:Chart.Legends>
<!--Chart Area to present Chart Segments.-->
<syncfusion:ChartArea FontWeight="Bold" FontSize="14" >
<!--X-axis declaration with required property settings.-->
<syncfusion:ChartArea.PrimaryAxis >
<!--Assigning text for the labels in the Primary Axis with the product
name.-->
```

```

<syncfusion:ChartAxis Header="Category" PositionPath="CategoryID"
ContentPath="CategoryName" RangePadding="None"
LabelRotateAngle="270" LabelsSource="{Binding ItemsSource.Categories}"/>
</syncfusion:ChartArea.PrimaryAxis>
<!--Y-axis declaration with required property settings.-->
<syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartAxis Header="Product" RangePadding="None" />
</syncfusion:ChartArea.SecondaryAxis>
<!-- Binding data to the series from the database.-->
<syncfusion:ChartSeries Type="Column" Label="Categories List"
BindingPathsY="Count"
Interior="{StaticResource SeriesBInterior}" DataSource="{Binding
ItemsSource}" />
</syncfusion:ChartArea>
</syncfusion:Chart>
</DataTemplate>

```

2. Associate the above template with the Grid Cell.

CSHARP

```

var style = model[1, 2];
style.CellValue = customer.ContactName;
var cust = customer.Orders.Select(o => o.OrderDetails.Select(od =>
od.Products).Select(p => p.Categories)).ToList();
var finalList = cust.Select(c => new
{
    Count = c.Count(),
    Categories = c
}).ToList();
style.ItemsSource = finalList;
// Assign template.
style.CommentTemplateKey = "chartTemplate";
Dim style = model(1, 2)
style.CellValue = customer.ContactName
Dim cust = customer.Orders.Select(Function(o)
o.OrderDetails.Select(Function(od) od.Products).Select(Function(p)
p.Categories)).ToList()
Dim finalList = cust.Select(Function(c) New With {Key .Count = c.Count(),
Key .Categories = c}).ToList()
style.ItemsSource = finalList
' Assign template.
style.CommentTemplateKey = "chartTemplate"

```




Comment Alignment

By using the `CommentAlignment` property, you can make the cell comments appear at the top-left, top-right, bottom-left or bottom-right corners.

C#

```
style.CommentAlignment = CommentAlignment.Bottom - Left;
```

VB.NET

```
Style.CommentAlignment = CommentAlignment.Bottom - Left
```

ALFKI	Alfreds Futterkiste
ANATR	Ana Trujillo Emparedados y helados
ANTON	Antonio Moreno Taquería
AROUT	Around the Horn
BERGS	Berglunds snabbköp
BLAUS	Blauer See Delikatessen

Column Auto Sizing

This feature allows the grid columns to resize themselves automatically to fit the column content. This resize action is performed based on the size of cells, size of header or size of parent control. According

to this criterion, the column resize options are defined below in the GridControlLengthUnitType enumeration.

Resize Options

1. GridControlLengthUnitType.Auto

In Auto type, column widths of the Grid control/GridData control are adjusted with respect to the cell and header content, i.e., each column's header length and cell content length is taken into account.

CSHARP

```
this.grid.Model.Options.ColumnSizer = GridControlLengthUnitType.Auto;
```

VB.NET

```
Me.grid.Model.Options.ColumnSizer = GridControlLengthUnitType.Auto
```



Order ID	Customer ID
10000	FRANS
10001	MEREP
10002	FOLKO
10003	SIMOB
10004	VAFFE

2. GridControlLengthUnitType.AutoWithLastColumnFill

In AutoWithLastColumnFill type, column width of Grid Control/GridData Control is adjusted with respect to cell and header content. The last column's width fills the unoccupied space in the parent Framework element.

CSHARP

```
id.Model.Options.ColumnSizer =  
GridControlLengthUnitType.AutoWithLastColumnFill;
```

VB.NET

```
Me.grid.Model.Options.ColumnSizer =  
GridControlLengthUnitType.AutoWithLastColumnFill
```



Order ID	Customer ID
10000	FRANS
10001	MEREP
10002	FOLKO
10003	SIMOB
10004	VAFFE

3. GridControlLengthUnitType.SizeToCells

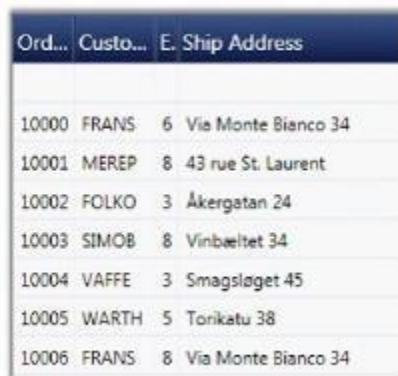
In SizeToCells type, column width of Grid Control/GridData Control is adjusted with respect to cell content only.

CSHARP

```
this.grid.Model.Options.ColumnSizer = GridControlLengthUnitType.SizeToCells;
```

VB.NET

```
Me.grid.Model.Options.ColumnSizer = GridControlLengthUnitType.SizeToCells
```



Ord...	Custo...	E. Ship Address
10000	FRANS	6 Via Monte Bianco 34
10001	MEREP	8 43 rue St. Laurent
10002	FOLKO	3 Åkergatan 24
10003	SIMOB	8 Vinbæltet 34
10004	VAFFE	3 Smagsløget 45
10005	WARTH	5 Torikatu 38
10006	FRANS	8 Via Monte Bianco 34

4. GridControlLengthUnitType.SizeToHeader

In SizeToHeader type, column widths of Grid Control/GridData Control are adjusted with respect to header content only.

CSHARP

```
this.grid.Model.Options.ColumnSizer =  
GridControlLengthUnitType.SizeToHeader;
```

VB.NET

```
Me.grid.Model.Options.ColumnSizer = GridControlLengthUnitType.SizeToHeader
```

Order ID	Customer ID
10000	FRANS
10001	MEREP
10002	FOLKO
10003	SIMOB
10004	VAFFE

5. GridControlLengthUnitType.Star

In Star type, column widths are equal and the control and the content occupies total space in the Parent cell. The user need not specify the width for every grid column. They can opt one of these options instead.

CSHARP

```
this.grid.Model.Options.ColumnSizer = GridControlLengthUnitType.Star;
```

VB.NET

```
Me.grid.Model.Options.ColumnSizer = GridControlLengthUnitType.Star
```

Ord...	Cus...	Emp...	Ship...	Sh...	Shi...
10000	FRANS	6	Via...	Torino	
10001	MER...	8	43 ru...	Mon...	Qué...
10002	FOL...	3	Åker...	Bräcke	
10003	SIM...	8	Vinb...	Kebe...	
10004	VAFFE	3	Sma...	Århus	
10005	WAR...	5	Torik...	Oulu	

6. MaxLength

The user can specify the number of rows that should be considered for calculating column widths using MaxLength property. The following code snippet allows the grid to consider only the data of first 12 rows for calculating column widths.

CSHARP

```
grid.Model.Options.MaxLength = 12;
```

VB.NET

```
grid.Model.Options.MaxLength = 12
```

Look and Feel

This section discusses different ways of enhancing the grid appearance such as,

- Row styles are used for row-wise formatting
- Conditional Formats apply formatting only when the given criteria is met
- Table Options discuss various options available to customize the table
- Grid Skins lists the in-built themes supported by the grid

Row Styles

There are two ways to format the grid rows. They are,

- Using properties
- By handling QueryCellInfo event

Using Properties

You can change the background of the grid rows by setting a color for the RowBackground property. To override the color of the alternative rows in the same grid use the AlternatingRowBackground property.

The following code illustrates the properties settings.

C#

```
grid.AlternatingRowBackground = new SolidColorBrush(Colors.Orchid);  
grid.RowBackground = new SolidColorBrush(Colors.Tan);
```

The following image corresponds to the output of the above given code:



Order ID	Customer ID	Employee ID	ShipName
10000	FRANS	6	Franchi S.p.A.
10001	MEREP	8	Mère Paillarde
10002	FOLKO	3	Folk och få HB
10003	SIMOB	8	Simons bistro
10004	VAFFE	3	Vaffeljernet
10005	WARTH	5	Wartian Herkku
10006	FRANS	8	Franchi S.p.A.
10007	MORGK	4	Morgenstern Gesund
10008	FURIB	3	Furia Bacalhau e Frut

The row styles of the GDC are customized using background properties.

Using QueryCellInfo Event

QueryCellInfo event is handled whenever a grid cell needs to be redrawn or repainted. In the GDC, you can use Model.QueryCellInfo event to format the rows by checking the row and column indices on the event arguments.

The following code illustrates this:

C#

```

grid.Model.QueryCellInfo += new
GridQueryCellInfoEventHandler(Model_QueryCellInfo);
void Model_QueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    if (e.Cell.RowIndex > 0)
    {
        if (e.Cell.RowIndex % 2 == 0)
            e.Style.Background = Brushes.BlanchedAlmond;
        else
            e.Style.Background = Brushes.LightCyan;
    }
}

```

The following image corresponds to the output of the above given code:



Order ID	Customer ID	Employee ID	ShipName
10000	FRANS	6	Franchi S.p.A.
10001	MEREP	8	Mère Paillarde
10002	FOLKO	3	Folk och få HB
10003	SIMOB	8	Simons bistro
10004	VAFFE	3	Vaffeljernet
10005	WARTH	5	Wartian Herkku

The row styles of the GDC are customized by handling the QueryCellInfo event.

Conditional Formatting

The GridData control has in-built support for conditional formatting. This feature allows you to format grid cells based on a certain condition. This can be achieved by defining a GridDataConditionalFormat for the grid. Using this class, you can specify the filter criteria for the cells and the style to be applied for the filtered cells. Once these specifications are defined, the given styles are applied to only those cells that satisfy the condition specified.

Conditional formatting can be specified through the GridDataControl.ConditionalFormats property. This is an observable collection, into which you can add required number of formatters of type GridDataConditionalFormat. The filter criteria are specified by the GridDataConditionalFormat.Conditions property that is a collection of GridDataCondition objects. The following table describes the important properties involved:

Property

Property	Description
GridDataConditionalFormat	
Conditions	A collection of filter conditions.
Style	A GridDataStyleInfo that should be applied when the given conditions are met.
ApplyStyleToColumn	The name of the column to apply the specified style. By default, the style is applied to the entire record row. Once this property is set, the cell

	corresponding to this column is applied with the style.
GridDataCondition	
ColumnName	Name of the column whose values should be checked.
ConditionType	This field corresponds to one of the following conditional operators: EqualsNot EqualsLess ThanLess Than Or EqualGreater ThanGreater Than Or Equal
Value	The value to compare.
PredicateType	Specifies the PredicateType that is used to combine more than one condition. There are two types- AND, OR.

Example

Now, let us consider an example for conditional formatting. The first conditional formatter in the following example specifies the filter criteria, “{Freight} > 200 OR {Freight} < 500” that applies Yellow background to the cells satisfying this condition.

The second conditional formatter indicates the criteria–“{ShipCountry} Equals USA OR {ShipCountry} Equals UK”. If this condition is satisfied, then the given style, Crimson background and White foreground is applied only to the corresponding record’s ShipCountry field, instead of being applied to the entire row.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource ordersSource}"
ShowGroupDropArea="True" UpdateMode="PropertyChanged">
  <syncfusion:GridDataControl.ConditionalFormats>
    <syncfusion:GridDataConditionalFormat Name="C1">
      <syncfusion:GridDataConditionalFormat.Style>
        <syncfusion:GridDataStyleInfo Background="Yellow" />
      </syncfusion:GridDataConditionalFormat.Style>
      <syncfusion:GridDataConditionalFormat.Conditions>
        <syncfusion:GridDataCondition ColumnName="Freight"
          ConditionType="GreaterThan" Value="200" PredicateType="Or"/>
        <syncfusion:GridDataCondition ColumnName="Freight" ConditionType="LessThan"
          Value="500" PredicateType="And"/>
      </syncfusion:GridDataConditionalFormat.Conditions>
    </syncfusion:GridDataConditionalFormat>
    <syncfusion:GridDataConditionalFormat Name="C2"
      ApplyStyleToColumn="ShipCountry" >
      <syncfusion:GridDataConditionalFormat.Style>
        <syncfusion:GridDataStyleInfo Background="Crimson" Foreground="White" />
      </syncfusion:GridDataConditionalFormat.Style>
      <syncfusion:GridDataConditionalFormat.Conditions>
        <syncfusion:GridDataCondition ColumnName="ShipCountry"
          ConditionType="Equals" Value="USA" PredicateType="Or"/>
      </syncfusion:GridDataConditionalFormat.Conditions>
    </syncfusion:GridDataConditionalFormat>
  </syncfusion:GridDataControl.ConditionalFormats>
</syncfusion:GridDataControl>
```

```
<syncfusion:GridDataCondition ColumnName="ShipCountry"
ConditionType="Equals" Value="UK" PredicateType="Or"/>
</syncfusion:GridDataConditionalFormat.Conditions>
</syncfusion:GridDataConditionalFormat>
</syncfusion:GridDataControl.ConditionalFormats>
</syncfusion:GridDataControl>
```

The following image shows the output of the above given code:



Order ID	Order Date	Freight	Ship Country	Ship City
10087	10/3/1991 12:00:00 AM	79.65	Canada	Montréal
10088	10/4/1991 12:00:00 AM	4.79	Venezuela	I. de Margarita
10089	10/7/1991 12:00:00 AM	17.4	Spain	Barcelona
10090	10/9/1991 12:00:00 AM	79.64	UK	London
10091	10/10/1991 12:00:00 AM	995.33	Brazil	Rio de Janeiro
10092	10/11/1991 12:00:00 AM	16.46	Austria	Graz
10093	10/14/1991 12:00:00 AM	263.66	Germany	Cunewalde
10094	10/15/1991 12:00:00 AM	135.23	USA	Albuquerque
10095	10/17/1991 12:00:00 AM	2.58	USA	Walla Walla
10096	10/18/1991 12:00:00 AM	468.02	Switzerland	Genève
10097	10/21/1991 12:00:00 AM	390.23	UK	London
10098	10/22/1991 12:00:00 AM	176.7	USA	Boise
10099	10/23/1991 12:00:00 AM	77.2	Italy	Bergamo

The preceding screenshot shows a GDC applied with conditional formatting.

Table Options

There are numerous options to customize the appearance and behavior of the GridData control. These options are exposed in the following Grid.Model.TableProperties.

Property

Property	Description
AllowNewRowPosition	Indicates the position of the new row, to be added:TopBottom
AllowDelete	Indicates whether the data can be deleted.
AllowDragColumns	Allows the user to re-arrange columns, by dragging the headers.
AllowEdit	Indicates whether the data can be edited.
AllowGroup	Indicates whether the data can be grouped.
AllowResizeColumns	Indicates whether the columns can be resized.
AllowResizeRows	Indicates whether the rows can be resized.
AllowSort	Allows user to sort the columns by clicking the column header.
AlternatingRowBackground	Specifies a background brush for alternate rows.

AlternatingRowCount	Indicates the row count to apply alternate row styles. Default value is 1.
AutoPopulateColumns	If true, the grid is automatically populated with the data columns.
AutoPopulateRelations	If true, the grid is automatically loaded with dataset relations.
CaptionSummaryRow	Defines the caption summary.
ConditionalFormats	Defines the conditional formats for the grid.
DefaultColumnWidth	Default width for columns.
DragIndicatorInnerBrush	Specifies the inner brush for drag indicator.
DragIndicatorOuterBrush	Specifies the outline brush for drag indicator.
GroupCaptionText	Specifies the text to display as group caption.
GroupedColumns	Defines the grouped columns for the grid.
IsSynchronizedWithCurrentItem	If true, keeps any selector-driven control's current selection synchronized with the selection of the grid.
ItemsSource	Specifies the item template to populate the grid.
NotifyPropertyChanges	When true, keeps the grid notified of data source changes.
NullFilterText	Sets a text to indicate null filter.
PrimaryKeyColumns	Holds the primary key columns.
Relations	Defines data relations for the grid.
RowBackground	Indicates a background brush for entire grid row.
ShowAddNewRow	Indicates whether the add new row should be visible.
ShowColumnOptions	Indicates whether the column options UI should be allowed.
ShowFilters	Indicates whether the filters should be allowed.
ShowGroupCaptionPlusMinus	Indicates whether a PlusMinus cell should appear next to group captions.
ShowGroupDropArea	Indicates whether the group drop area should be visible.
ShowGroupSummaries	Indicates whether the group summaries should be visible.
ShowGroupSummaryInCaption	Indicates whether the group summary should be displayed in group caption.
ShowRecordPlusMinus	Indicates whether a PlusMinus cell should appear next to the records. It is applicable only for nested tables.
ShowRowHeader	Indicates whether the row header column should be visible.

ShowTableSummaries	Indicates whether the table summaries should be visible.
SortColumns	Defines the sorted columns for the grid.
StackedHeaders	Defines the stacked headers for the grid.
SummaryRows	Defines the group summary rows for the grid.
TableSummaryRows	Defines the table summary rows for the grid.
VisibleColumns	Defines the visible columns of the grid.
VisualStyle	Specifies the skin for the grid.

Font Settings

The GridData control lets you to set the grid font properties from the root grid. By setting the font related properties in the root GridDataControl instance, you can change the font settings for entire grid cells at once.

Skins

GridDataControl implements visual styles that set up a common appearance to all the components in the grid. The term appearance refers not only to the way the grid elements appear but also the manner in which they behave in response to the user interactions like hovering mouse over them, clicking, and so on. Grid has in-built support for the following skins:

- Default
- Blend
- Office2007Blue
- Office2007Silver
- Office2007Black
- BureauBlue
- GlassyGreen
- ShinyBlue
- ShinyRed
- SunBlack
- TwilightBlue
- Office14Blue
- Office14Silver
- Office14Black
- VS2010

Use Case Scenarios

Skin provides better look and feel for an application.

Sample Link

Blend Styling Demo sample in the sample browser is purely customized in XAML through GridDataStyleManager class. To access Blend Styling Demo:

1. Open Syncfusion Dashboard.
2. Select User Interface.

3. Select WPF drop-down list and select Run Locally Installed Samples.
4. Select GridDataControl.
5. In Visual styles menu, select Grid Visual style Demo.

Adding Skin to an Application

Visual styles can be set for a grid by using the VisualStyle property. The following code illustrates this:

XML

```
<syncfusion:GridDataControl x:Name="dataGrid2" AutoPopulateColumns="False"
AutoPopulateRelations="False" ItemsSource="{StaticResource ordersSource}"
syncfusion:SkinStorage.VisualStudio="Office2007Blue" />
```

C#

```
SkinStorage.SetVisualStyle(dataGrid2, "Office2007Blue");
```

The following images show different visual styles applied to the grid.









Order				
Order ID	Customer ID	Order Date	Freight	S
Ship City : Aachen - 9 Items				
10112	DRACD	11/14/1991	\$1.48	Drac
10192	DRACD	3/18/1992	\$26.69	Drac
10232	DRACD	5/7/1992	\$12.48	Drac
10363	DRACD	10/20/1992	\$30.54	Drac
10391	DRACD	11/16/1992	\$5.45	Drac
10797	DRACD	11/18/1993	\$33.35	Drac
10825	DRACD	12/3/1993	\$79.25	Drac
11036	DRACD	3/14/1994	\$149.47	Drac
11067	DRACD	3/28/1994	\$7.98	Drac

Order				
Order ID	Customer ID	Order Date	Freight	S
Ship City : Aachen - 9 Items				
10112	DRACD	11/14/1991	\$1.48	Drac
10192	DRACD	3/18/1992	\$26.69	Drac
10232	DRACD	5/7/1992	\$12.48	Drac
10363	DRACD	10/20/1992	\$30.54	Drac
10391	DRACD	11/16/1992	\$5.45	Drac
10797	DRACD	11/18/1993	\$33.35	Drac
10825	DRACD	12/3/1993	\$79.25	Drac
11036	DRACD	3/14/1994	\$149.47	Drac
11067	DRACD	3/28/1994	\$7.98	Drac

Ship City ▲					
Order					
Order ID	Customer ID	Order Date	Freight	S	
Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48	Drac	
10192	DRACD	3/18/1992	\$26.69	Drac	
10232	DRACD	5/7/1992	\$12.48	Drac	
10363	DRACD	10/20/1992	\$30.54	Drac	
10391	DRACD	11/16/1992	\$5.45	Drac	
10797	DRACD	11/18/1993	\$33.35	Drac	
10825	DRACD	12/3/1993	\$79.25	Drac	
11036	DRACD	3/14/1994	\$149.47	Drac	
11067	DRACD	3/28/1994	\$7.98	Drac	

Ship City ▲					
Order					
Order ID	Customer ID	Order Date	Freight	S	
Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48	Drac	
10192	DRACD	3/18/1992	\$26.69	Drac	
10232	DRACD	5/7/1992	\$12.48	Drac	
10363	DRACD	10/20/1992	\$30.54	Drac	
10391	DRACD	11/16/1992	\$5.45	Drac	
10797	DRACD	11/18/1993	\$33.35	Drac	
10825	DRACD	12/3/1993	\$79.25	Drac	
11036	DRACD	3/14/1994	\$149.47	Drac	
11067	DRACD	3/28/1994	\$7.98	Drac	

Ship City ▲

Order								
Order ID	 	Customer ID	 	Order Date	 	Freight	 	S
Ship City : Aachen - 9 Items								
10112		DRACD		11/14/1991		\$1.48		Drac
10192		DRACD		3/18/1992		\$26.69		Drac
10232		DRACD		5/7/1992		\$12.48		Drac
10363		DRACD		10/20/1992		\$30.54		Drac
10391		DRACD		11/16/1992		\$5.45		Drac
10797		DRACD		11/18/1993		\$33.35		Drac
10825		DRACD		12/3/1993		\$79.25		Drac
11036		DRACD		3/14/1994		\$149.47		Drac
11067		DRACD		3/28/1994		\$7.98		Drac

Ship City ▲					
Order					
Order ID	Customer ID	Order Date	Freight		S
Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48		Drac
10192	DRACD	3/18/1992	\$26.69		Drac
10232	DRACD	5/7/1992	\$12.48		Drac
10363	DRACD	10/20/1992	\$30.54		Drac
10391	DRACD	11/16/1992	\$5.45		Drac
10797	DRACD	11/18/1993	\$33.35		Drac
10825	DRACD	12/3/1993	\$79.25		Drac
11036	DRACD	3/14/1994	\$149.47		Drac
11067	DRACD	3/28/1994	\$7.98		Drac

Ship City ▲

Order					
Order ID	Customer ID	Order Date	Freight		S
Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48		Drac
10192	DRACD	3/18/1992	\$26.69		Drac
10232	DRACD	5/7/1992	\$12.48		Drac
10363	DRACD	10/20/1992	\$30.54		Drac
10391	DRACD	11/16/1992	\$5.45		Drac
10797	DRACD	11/18/1993	\$33.35		Drac
10825	DRACD	12/3/1993	\$79.25		Drac
11036	DRACD	3/14/1994	\$149.47		Drac
11067	DRACD	3/28/1994	\$7.98		Drac

Ship City ▲

Order					
Order ID	Customer ID	Order Date	Freight		S
Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48	Drac	
10192	DRACD	3/18/1992	\$26.69	Drac	
10232	DRACD	5/7/1992	\$12.48	Drac	
10363	DRACD	10/20/1992	\$30.54	Drac	
10391	DRACD	11/16/1992	\$5.45	Drac	
10797	DRACD	11/18/1993	\$33.35	Drac	
10825	DRACD	12/3/1993	\$79.25	Drac	
11036	DRACD	3/14/1994	\$149.47	Drac	
11067	DRACD	3/28/1994	\$7.98	Drac	

Ship City ▲

Order				
Order ID	Customer ID	Order Date	Freight	
Ship City : Aachen - 9 Items				
10112	DRACD	11/14/1991	\$1.48	Drac
10192	DRACD	3/18/1992	\$26.69	Drac
10232	DRACD	5/7/1992	\$12.48	Drac
10363	DRACD	10/20/1992	\$30.54	Drac
10391	DRACD	11/16/1992	\$5.45	Drac
10797	DRACD	11/18/1993	\$33.35	Drac
10825	DRACD	12/3/1993	\$79.25	Drac
11036	DRACD	3/14/1994	\$149.47	Drac
11067	DRACD	3/28/1994	\$7.98	Drac

Ship City ▲

Order					
Order ID	Customer ID	Order Date	Freight		S
Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48	Drac	
10192	DRACD	3/18/1992	\$26.69	Drac	
10232	DRACD	5/7/1992	\$12.48	Drac	
10363	DRACD	10/20/1992	\$30.54	Drac	
10391	DRACD	11/16/1992	\$5.45	Drac	
10797	DRACD	11/18/1993	\$33.35	Drac	
10825	DRACD	12/3/1993	\$79.25	Drac	
11036	DRACD	3/14/1994	\$149.47	Drac	
11067	DRACD	3/28/1994	\$7.98	Drac	

Ship City ▲

Order				
Order ID	Customer ID	Order Date	Freight	
Ship City : Aachen - 9 Items				
10112	DRACD	11/14/1991	\$1.48	Drac
10192	DRACD	3/18/1992	\$26.69	Drac
10232	DRACD	5/7/1992	\$12.48	Drac
10363	DRACD	10/20/1992	\$30.54	Drac
10391	DRACD	11/16/1992	\$5.45	Drac
10797	DRACD	11/18/1993	\$33.35	Drac
10825	DRACD	12/3/1993	\$79.25	Drac
11036	DRACD	3/14/1994	\$149.47	Drac
11067	DRACD	3/28/1994	\$7.98	Drac

Ship City ▲

Order					
Order ID	Customer ID	Order Date	Freight		S
▲ Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48	Drac	
10192	DRACD	3/18/1992	\$26.69	Drac	
10232	DRACD	5/7/1992	\$12.48	Drac	
10363	DRACD	10/20/1992	\$30.54	Drac	
10391	DRACD	11/16/1992	\$5.45	Drac	
10797	DRACD	11/18/1993	\$33.35	Drac	
10825	DRACD	12/3/1993	\$79.25	Drac	
11036	DRACD	3/14/1994	\$149.47	Drac	
11067	DRACD	3/28/1994	\$7.98	Drac	

Ship City ▲					
Order					
Order ID	Customer ID	Order Date	Freight	S	
▲ Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48	Drac	
10192	DRACD	3/18/1992	\$26.69	Drac	
10232	DRACD	5/7/1992	\$12.48	Drac	
10363	DRACD	10/20/1992	\$30.54	Drac	
10391	DRACD	11/16/1992	\$5.45	Drac	
10797	DRACD	11/18/1993	\$33.35	Drac	
10825	DRACD	12/3/1993	\$79.25	Drac	
11036	DRACD	3/14/1994	\$149.47	Drac	
11067	DRACD	3/28/1994	\$7.98	Drac	

Ship City ▲

Order					
Order ID	Customer ID	Order Date	Freight		S
Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48	Drac	
10192	DRACD	3/18/1992	\$26.69	Drac	
10232	DRACD	5/7/1992	\$12.48	Drac	
10363	DRACD	10/20/1992	\$30.54	Drac	
10391	DRACD	11/16/1992	\$5.45	Drac	
10797	DRACD	11/18/1993	\$33.35	Drac	
10825	DRACD	12/3/1993	\$79.25	Drac	
11036	DRACD	3/14/1994	\$149.47	Drac	
11067	DRACD	3/28/1994	\$7.98	Drac	

Ship City ▲

Order				
Order ID	Customer ID	Order Date	Freight	
Ship City : Aachen - 9 Items				
10112	DRACD	11/14/1991	\$1.48	Drac
10192	DRACD	3/18/1992	\$26.69	Drac
10232	DRACD	5/7/1992	\$12.48	Drac
10363	DRACD	10/20/1992	\$30.54	Drac
10391	DRACD	11/16/1992	\$5.45	Drac
10797	DRACD	11/18/1993	\$33.35	Drac
10825	DRACD	12/3/1993	\$79.25	Drac
11036	DRACD	3/14/1994	\$149.47	Drac
11067	DRACD	3/28/1994	\$7.98	Drac

Ship City ▲

Order					
Order ID	Customer ID	Order Date	Freight		S
Ship City : Aachen - 9 Items					
10112	DRACD	11/14/1991	\$1.48	Drac	
10192	DRACD	3/18/1992	\$26.69	Drac	
10232	DRACD	5/7/1992	\$12.48	Drac	
10363	DRACD	10/20/1992	\$30.54	Drac	
10391	DRACD	11/16/1992	\$5.45	Drac	
10797	DRACD	11/18/1993	\$33.35	Drac	
10825	DRACD	12/3/1993	\$79.25	Drac	
11036	DRACD	3/14/1994	\$149.47	Drac	
11067	DRACD	3/28/1994	\$7.98	Drac	

Ship City ▲

Order						
Order ID	Customer ID	Order Date	Freight	Ship Name	Ship Address	
Ship City : Aachen - 9 Items						
10112	DRACD	11/14/1991	\$1.48	Drachenblut Delikatessen	Walserweg 21	
10192	DRACD	3/18/1992	\$26.69	Drachenblut Delikatessen	Walserweg 21	
10232	DRACD	5/7/1992	\$12.48	Drachenblut Delikatessen	Walserweg 21	
10363	DRACD	10/20/1992	\$30.54	Drachenblut Delikatessen	Walserweg 21	
10391	DRACD	11/16/1992	\$5.45	Drachenblut Delikatessen	Walserweg 21	
10797	DRACD	11/18/1993	\$33.35	Drachenblut Delikatessen	Walserweg 21	
10825	DRACD	12/3/1993	\$79.25	Drachenblut Delikatessen	Walserweg 21	
11036	DRACD	3/14/1994	\$149.47	Drachenblut Delikatessen	Walserweg 21	
11067	DRACD	3/28/1994	\$7.98	Drachenblut Delikatessen	Walserweg 21	
Ship City : Albuquerque - 25 Items						
10018	RATTC	6/10/1991	\$65.46	Rattlesnake Canyon Groc...	2817 Milton Dr.	
10024	RATTC	6/19/1991	\$5.19	Rattlesnake Canyon Groc...	2817 Milton Dr.	
10025	RATTC	6/21/1991	\$3.32	Rattlesnake Canyon Groc...	2817 Milton Dr.	
10094	RATTC	10/15/1991	\$135.23	Rattlesnake Canyon Groc...	2817 Milton Dr.	
10111	RATTC	11/13/1991	\$7.10	Rattlesnake Canyon Groc...	2817 Milton Dr.	
10169	RATTC	2/18/1992	\$111.06	Rattlesnake Canyon Groc...	2817 Milton Dr.	
10216	RATTC	4/17/1992	\$28.83	Rattlesnake Canyon Groc...	2817 Milton Dr.	

Custom Skin

It is possible to define your own visual style for the GridData control. As a first step, you need to define a custom style by deriving from the `IGridDataVisualStyle` interface, and by defining custom brushes for various grid elements.

You should direct the grid to use this custom style by specifying Custom option in its `VisualStyle` property. This can be done by using the following code:

C#

```
this.dataGrid.CustomVisualStyle = new GridDataGlassyGreenStyle();
this.dataGrid.VisualStyle = VisualStyle.Custom;
```

The following screenshot shows the custom visual style set for the grid using the above given code:

Drag and drop columns here			
Employee ID	Last Name	First Name	
*			
1	Davolio	Nancy	
2	Fuller	Andrew	
3	Leverling	Janet	
4	Peacock	Margaret	
5	Buchanan	Steven	
6	Suyama	Michael	

Custom Visual Style can be defined for nested tables too. The following code illustrates this:

XML

```
<Window.Resources>
<ObjectDataProvider x:Key="CustomerTable"
Method="GetDataTable" ObjectType="{x:Type local:Data}" />
<CollectionViewSource x:Key="orderSource" Source="{StaticResource
CustomerTable}" >
</CollectionViewSource>
<local:GridDataVisualStyleConverter x:Key="styleConverter" />
<ObjectDataProvider x:Key="GreenStyle" ObjectType="{x:Type
local:GridDataGlassyGreenStyle}" />
</Window.Resources>
<syncfusion:GridDataControl x:Name="dataGrid" ShowRowHeader="True"
ShowColumnOptions="True"
ShowGroupDropArea="True" ShowFilters="True" AutoPopulateColumns="True"
AutoPopulateRelations="False" ItemsSource="{StaticResource orderSource}">
<syncfusion:GridDataControl.Relations >
<syncfusion:GridDataRelation RelationalColumn="Employee_Orders"
RelationType="MasterDetails" >
<syncfusion:GridDataRelation.TableProperties>
<syncfusion:GridDataTableProperties CustomVisualStyle="{Binding
Source={StaticResource GreenStyle}, Converter={StaticResource
styleConverter}}" VisualStyle="Custom" AutoPopulateColumns="True">
</syncfusion:GridDataTableProperties>
</syncfusion:GridDataRelation.TableProperties>
</syncfusion:GridDataRelation>
</syncfusion:GridDataControl.Relations>
</syncfusion:GridDataControl>
```

Employee ID	Last Name	First Name	Title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative

Order ID	Customer ID	Employee ID	Ship Name
10002	FOLKO	3	Folk och få HB
10004	VAFFE	3	Vaffeljernet
10008	FURIB	3	Furia Bacalhau e Frutos d...
10013	RICSU	3	Richter Supermarkt
10016	FOLIG	3	Folies gourmandes
10023	TOMSP	3	Toms Spezialitäten
10029	CAATC	3	Chateaufort

Note: *IGridDataVisualStyle* is deprecated and this information is provided only for legacy reasons. The recommended approach for customizing the GridDataControl is using [GridDataStyleManager](#) class through Microsoft Expression Blend.

Backward Compatibility

There are substantial differences in the implementation of the new Skins in version 9.2 (and later). To maintain backward compatibility with the Skins included in earlier versions, EnableLegacyStyle property needs to be set to true.

Breaking Change in 9.2

Previously, AlphaBlend was set in DrawSelectionOptions by default. But currently it is turned off. If you need AlphaBlend color to be applied for selection, we need to set DrawSelectionOptions property in application as mentioned in the following code snippet:

C#

```
this.grid.DrawSelectionOptions = GridDrawSelectionOptions.AlphaBlend |  
GridDrawSelectionOptions.ReplaceBackground  
| GridDrawSelectionOptions.ReplaceTextColor;
```

Blend Support

Essential GridDataControl supports the setting of its styles by XAML and supports editing styles using Microsoft Expression Blend.

Use Case Scenarios

This feature helps in customizing GridDataControl through Microsoft Expression Blend 3 or 4®.

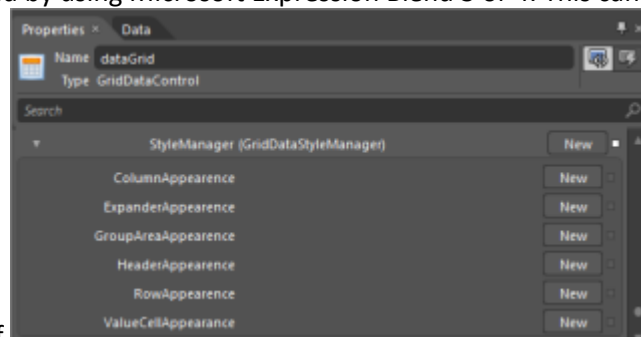
Sample Link

Blend Styling Demo sample in the sample browser is purely customized in XAML through GridDataStyleManager class. To access Blend Styling Demo:

1. Open Syncfusion Dashboard.
2. Select User Interface.
3. Select WPF drop-down list and select Run Locally Installed Samples.
4. Select GridDataControl.
5. In Visual styles menu select Blend styling Demo.

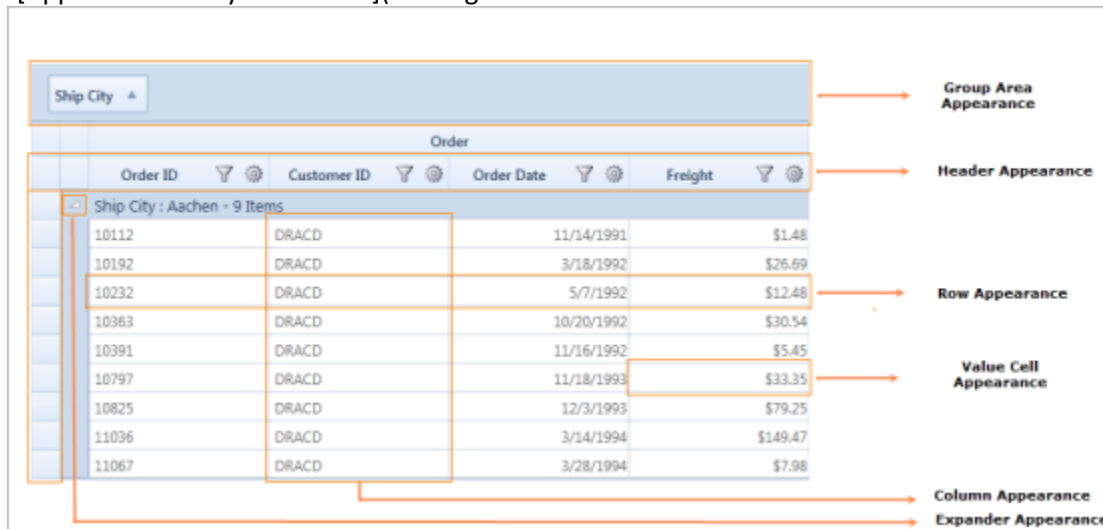
Adding Styles to an Application

GridDataControl can be customized by using Microsoft Expression Blend 3 or 4. This can be achieved



through StyleManager property of GridDataStyleManager type. The properties used to customize the appearance are defined in the GridDataStyleManager class.

![[Applied visual styles to Blend]](Getting-Started



images/Getting-Startedimg114.png)

GridDataStyleManager properties are organized under the following seven groups, each representing a specific area of the GridDataControl:

- Column Appearance
- Expander Appearance
- Group Area Appearance
- Header Appearance
- Row Appearance
- Value Cell Appearance
- Nested Grid Appearance

![[Appearance of WPF GridDataControl]](Getting-Startedimages/Getting-Startedimg115.png)

Note: Previously, the appearance of the GridDataControl could be customized through the [IGridDataVisualStyle](#) interface; even if a visual style was set for the GridDataControl, the values set in the GridDataStyleManager would override it

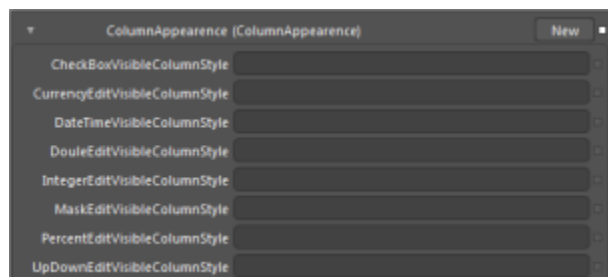
Customizing the Column Appearance

In the Column group, all the properties are of *Style* type. These properties affect the appearance of a whole column. The following table lists each property and its corresponding target *type*.

Property

Property	Description
CheckBoxVisibleColumnStyle	GridDataCheckBoxVisibleColumnControl
CurrencyEditVisibleColumnStyle	GridDataCurrencyEditVisibleColumnControl
DatetimeVisibleColumnStyle	GridDataDateTimeVisibleColumnControl
DoubleEditVisibleColumnStyle	GridDataDoubleEditVisibleColumnControl
IntegerEditVisibleColumnStyle	GridDataIntegerEditVisibleColumnControl

PercentEditVisibleColumnStyle	GridDataPercentEditVisibleColumnControl
UpDownEditVisibleColumnStyle	GridDataUpDownEditVisibleColumnControl



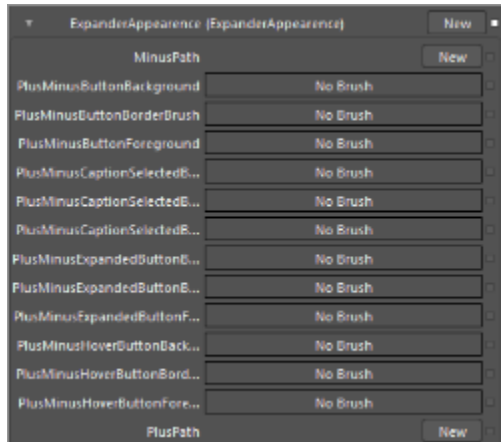
Customizing the Expander Appearance

The Expander group properties are used to customize the Expand and Collapse buttons.

Property

Property	Description
PlusPath	Gets or sets the vector path of Collapsed Parent and the Group Caption rows.
MinusPath	Gets or sets the vector path of Expanded Parent and the Group Caption rows.
PlusMinusButtonBackground	Gets or sets the background color for the default Expander icon (that is, Expander icon in Collapsed state).
PlusMinusButtonBorderBrush	Gets or sets the border color for the default Expander icon.
PlusMinusButtonForeground	Gets or sets the foreground color for the default Expander icon.
PlusMinusExpandedButtonBackground	Gets or sets the background color for the default expanded Expander icon.
PlusMinusExpandedButtonBorderBrush	Gets or sets the border color for the default expanded Expander icon.
PlusMinusExpandedButtonForeground	Gets or sets the foreground color for the default expanded Expander icon.
PlusMinusHoverButtonBackground	Gets or sets the background color to be displayed for the default Expander icon when the pointer is moved over it.
PlusMinusHoverButtonBorderBrush	Gets or sets the border color to be displayed for the default Expander icon when the pointer is moved over it.

PlusMinusHoverButtonForeground	Gets or sets the foreground color to be displayed for the default Expander icon when the pointer is moved over it.
PlusMinusCaptionSelectedButtonBackground	Obsolete.
PlusMinusCaptionSelectedButtonBorderBrush	Obsolete.
PlusMinusCaptionSelectedButtonForeground	Obsolete.



The following code example illustrates the properties in the Expander group.

XML

```
<Path x:Key="plusPath" Width="8.666" Height="11.42"
HorizontalAlignment="Left"
VerticalAlignment="Top"
Data="M1.5,4.4500742 C2.3284271,4.4500742 3,5.1216469 3,5.9500742 C3,6.77850
15 2.3284271,7.4500742 1.5,7.4500742 C0.67157292,7.4500742 0,6.7785015 0,5.9
500742 C0,5.1216469 0.67157292,4.4500742 1.5,4.4500742 z M2.8990631,0.000367
17192 C3.0453069,0.005159697 3.1827641,0.056897067 3.284497,0.15714154 L8.51
83868,5.3089428 C8.6242905,5.4130292 8.6701775,5.5520487 8.6654148,5.6966238
L8.6658916,5.6966238 C8.6662092,5.7011499 8.6654148,5.7055159 8.6656513,5.7
10042 C8.6654148,5.714488 8.6662092,5.718854 8.6658916,5.7233 L8.6654148,5.7
233 C8.6701775,5.867878 8.6242905,6.0069752 8.5183868,6.1111407 L3.284497,11
.262862 C3.0519648,11.491915 2.6327903,11.467778 2.3486581,11.208635 C2.3486
581,11.208635 2.2061534,11.101613 1.9369454,10.833023 C1.6678151,10.56459 1.7
572079,10.396356 1.9367857,10.137055 C2.1164434,9.8779926 4.7354136,6.097723
4.7354136,6.097723 C4.9301553,5.7890387 4.933569,5.630013 4.7354136,5.32228
14 C4.7354136,5.3222814 2.1164434,1.5420915 1.9367857,1.2827913 C1.7572079,1
.0237284 1.667151,0.85541332 1.9369454,0.58698106 C2.2061534,0.31847125 2.34
86581,0.21136875 2.3486581,0.21136881 C2.5084825,0.065645903 2.711035,-
0.0057947943 2.8990631,0.00036717192 z" Fill="#FFF5913F"
Stretch="Fill" />
<Path x:Key="minusPath" Width="8.666" Height="11.42"
HorizontalAlignment="Left"
VerticalAlignment="Top"
Data="M1.5,4.4500742 C2.3284271,4.4500742 3,5.1216469 3,5.9500742 C3,6.77850
15 2.3284271,7.4500742 1.5,7.4500742 C0.67157292,7.4500742 0,6.7785015 0,5.9
500742 C0,5.1216469 0.67157292,4.4500742 1.5,4.4500742 z M2.8990631,0.000367
```

```

17192 C3.0453069,0.005159697 3.1827641,0.056897067 3.284497,0.15714154 L8.51
83868,5.3089428 C8.6242905,5.4130292 8.6701775,5.5520487 8.6654148,5.6966238
L8.6658916,5.6966238 C8.6662092,5.7011499 8.6654148,5.7055159 8.6656513,5.7
10042 C8.6654148,5.714488 8.6662092,5.718854 8.6658916,5.7233 L8.6654148,5.7
233 C8.6701775,5.867878 8.6242905,6.0069752 8.5183868,6.1111407 L3.284497,11
.262862 C3.0519648,11.491915 2.6327903,11.467778 2.3486581,11.208635 C2.3486
581,11.208635 2.2061534,11.101613 1.9369454,10.833023 C1.6678151,10.56459 1.
7572079,10.396356 1.9367857,10.137055 C2.1164434,9.8779926 4.7354136,6.09772
3 4.7354136,6.097723 C4.9301553,5.7890387 4.933569,5.630013 4.7354136,5.3222
814 C4.7354136,5.3222814 2.1164434,1.5420915 1.9367857,1.2827913 C1.7572079,
1.0237284 1.6678151,0.85541332 1.9369454,0.58698106 C2.2061534,0.31847125 2.
3486581,0.21136875 2.3486581,0.21136881 C2.5084825,0.065645903 2.711035,-
.0057947943 2.8990631,0.00036717192 z" Fill="#FFF5913F"
RenderTransformOrigin="0.5,0.5" Stretch="Fill">
<Path.RenderTransform>
<RotateTransform Angle="90" />
</Path.RenderTransform>
</Path>
<syncfusion:GridDataStyleManager>
<syncfusion:GridDataStyleManager.ExpanderAppearance>
<syncfusion:ExpanderAppearance MinusPath="{StaticResource minusPath}"
PlusMinusButtonBackground="#FFA21A1E"
PlusMinusButtonBorderBrush="#FFA21A1E"
PlusMinusButtonForeground="#FFAAAA1E"
PlusMinusCaptionSelectedButtonBackground="White"
PlusMinusCaptionSelectedButtonBorderBrush="White"
PlusMinusCaptionSelectedButtonForeground="White"
PlusMinusExpandedButtonBackground="White"
PlusMinusExpandedButtonBorderBrush="#FFA21A1E"
PlusMinusExpandedButtonForeground="Gray"
PlusPath="{StaticResource plusPath}" />
</syncfusion:GridDataStyleManager.ExpanderAppearance>
</syncfusion:GridDataStyleManager>

```

EmployeeID	LastName	FirstName	Title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative

OrderID	CustomerID	EmployeeID	ShipName
10007	MORGK	4	Morgenstern Gesundkost
10014	GROSR	4	GROSELLA-Restaurante

OrderID	ProductID	UnitPrice	Quantity	Discount
10014	18	43	4	0
10014	41	6.7	3	0

10017	BLONP	4	Blondel père et fils
10018	RATTC	4	Rattlesnake Canyon Grocery
10024	RATTC	4	Rattlesnake Canyon Grocery

Customizing the Group Area

Group Area group properties are used to customize the Group Area.

Customizing Group Area Appearance through the GridDataStyleManager

The following properties illustrate how to customize the appearance of the Group Area through the GridDataStyleManager.

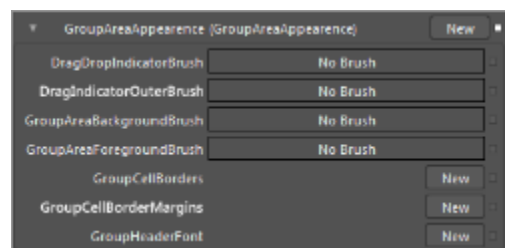
Property

Property	Description
GroupAreaBackgroundBrush	Gets or sets the background color for the Group Drop Area.
GroupAreaForegroundBrush	Gets or sets the foreground color for the text of the Group Drop Area.
GroupHeaderFont	Gets or sets the font information for the grouped header cell in the Group Drop Area.
GroupCellBorders	Gets or sets the cell border information for the grouped header cell in the Group Drop Area.
GroupCellBorderMargins	Gets or sets the cell margin information for the grouped header cell in the Group Drop Area.
DragDropIndicatorBrush	Gets or sets the background color for the Drag Drop Indicator.
DragDropIndicatorOuterBrush	Gets or sets the outer border color for the Drag Drop Indicator.

The following code example illustrates the properties defined in this group.

XML

```
<Pen x:Key="GroupCellBorder" Brush="#FFC2C2C2" Thickness="1" />
<syncfusion:GridDataStyleManager.GroupAreaAppearance>
  <syncfusion:GroupAreaAppearance GroupAreaBackgroundBrush="#FFE7CD9B"
  GroupAreaForegroundBrush="White" DragDropIndicatorBrush="Peru"
  DragDropIndicatorOuterBrush="Gray">
    <syncfusion:GroupAreaAppearance.GroupHeaderFont>
      <syncfusion:GridFontInfo FontFamily="RockWell" FontSize="12" />
    </syncfusion:GroupAreaAppearance.GroupHeaderFont>
    <syncfusion:GroupAreaAppearance.GroupCellBorders>
      <syncfusion:CellBordersInfo
        Bottom="{StaticResource GroupCellBorder}"
        Left="{StaticResource GroupCellBorder}"
        Right="{StaticResource GroupCellBorder}"
        Top="{StaticResource GroupCellBorder}" />
    </syncfusion:GroupAreaAppearance.GroupCellBorders>
  </syncfusion:GroupAreaAppearance>
</syncfusion:GridDataStyleManager.GroupAreaAppearance>
```



Customizing Group Area Appearance through the GridDataControl

The following properties illustrate how to customize the appearance of the Group Area through the GridDataControl.

Property

Property	Description
GroupDropAreaHeight	Specifies the height of Group Drop Area.
GroupDropAreaText	Specifies user-defined text to be displayed in the Group Drop Area.
ShowGroupDropArea	Shows or hides the Group Drop Area.
DragIndicatorInnerBrush	Gets or sets the inner background color for the Drag Indicator.
DragIndicatorOuterBrush	Gets or sets the outer border color for the Drag Indicator.

The following code example illustrates the properties defined in this group.

XML

```
<syncfusion:GridDataControl x:Name="grid" ShowGroupDropArea="True"
GroupDropAreaHeight="50" GroupDropAreaText="Customized Group Area"
DragIndicatorInnerBrush="Gray" DragIndicatorOuterBrush="Brown"/>
```

EmployeeID	LastName	FirstName	Title	BirthDate
Title : Advertising Specialist - 1 Items				
BirthDate : 12/9/1965 12:00:00 AM - 1 Items				
15	Pereira	Laurent	Advertising Specialist	12/9/1965 12:00:00 AM
Title : Business Manager - 1 Items				
BirthDate : 3/13/1960 12:00:00 AM - 1 Items				
10	Helistern	Albert	Business Manager	3/13/1960 12:00:00 AM
Title : Inside Sales Coordinator - 1 Items				
Title : Mail Clerk - 1 Items				
Title : Marketing Associate - 1 Items				
Title : Marketing Director - 1 Items				

Customizing the Header Appearance

The Header Appearance group properties are used to customize the header.

Customizing Header Appearance through the GridDataStyleManager

The following properties illustrate how to customize the Header appearance through the GridDataStyleManager.

Property

Property	Description
HeaderBackgroundBrush	Gets or sets the background color for the Header cell.
HeaderForegroundBrush	Gets or sets the foreground color for the Header cell.

HeaderHoverBackgroundBrush	Gets or sets the background color to be displayed for the Header cell when the pointer is moved over it.
HeaderHoverForegroundBrush	Gets or sets the foreground color to be displayed for the Header cell when the pointer is moved over it.
HeaderCellBorders	Gets or sets the cell border information for the Header cell area.
HeaderInnerBorder	Gets the inner border color of the Header cell.
HeaderInnerBorderThickness	Gets the inner border thickness of the Header cell.
HeaderFont	Gets or sets the text font of the Header cell.
HeaderTextMargins	Gets or sets the text margin of the Header cell.
SortWidgetBrush	Gets or sets the color of the Sort icon.
HeaderOptionsHoverBackground	Gets or sets the background color for the Header cell options such as Sort icon, Filter icon and Column Options icon.
HeaderOptionsBorderBrush	Gets or sets the border color of the Header cell options such as Sort icon, Filter icon and Column Options icon.
HeaderOptionsCheckedBackground	Gets or sets the background color for Header cell options such as Sort icon, Filter icon and Column Options icon when it is checked.
ColumnOptionsPopupBackground	Gets or sets the background color for the Column Options pop up.
ColumnOptionsPopupForeground	Gets or sets the foreground color for the Column Options pop up.
ColumnOptionsButtonBackground	Gets or sets the background color for the Column Options icon.
ColumnOptionsButtonBorderBrush	Gets or sets the border color for the Column Options icon.
FilterButtonInnerBrush	Gets or sets the background color for the Filter icon.
FilterButtonOuterBrush	Gets or sets the border color for the Filter icon.
FilterButtonHoverInnerBrush	Gets or sets the background color to be displayed for the Filter icon when the pointer is moved over it.
FilterButtonHoverOuterBrush	Gets or sets the border color to be displayed for the Filter icon when the pointer is moved over it.
FilterButtonAppliedBrush	Gets or sets the background color to be displayed for the Filter icon when the filter is applied.

The following code example illustrates the properties defined in this group.

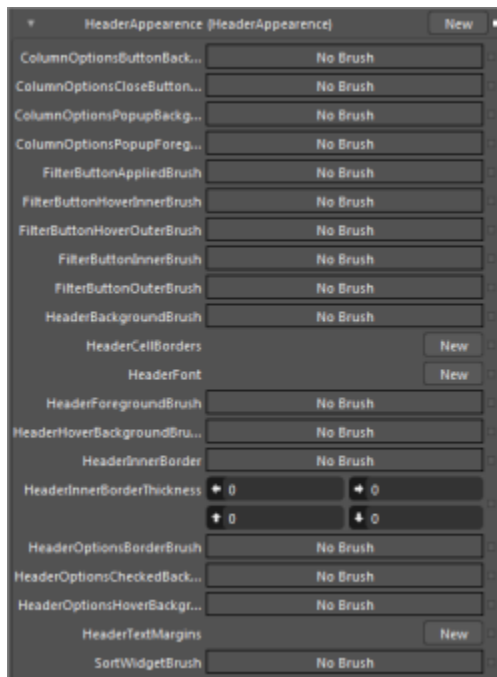
XML

```
<syncfusion:GridDataStyleManager.HeaderAppearance>
<syncfusion:HeaderAppearance ColumnOptionsPopupBackground="#FFF2E2C4"
HeaderBackgroundBrush="#FFFFFF" HeaderForegroundBrush="#FF412641"
HeaderHoverBackgroundBrush="White" HeaderHoverForegroundBrush="Black"
```

```

HeaderInnerBorder="#FF674B67" HeaderInnerBorderThickness="0.18"
SortWidgetBrush="#FFF68909">
<syncfusion:HeaderAppearance.HeaderFont>
<syncfusion:GridFontInfo FontFamily="Segoe UI" FontSize="14.73"
FontWeight="SemiBold" />
</syncfusion:HeaderAppearance.HeaderFont>
<syncfusion:HeaderAppearance.HeaderTextMargins>
<syncfusion:CellMarginsInfo Left="12" />
</syncfusion:HeaderAppearance.HeaderTextMargins>
<syncfusion:HeaderAppearance.HeaderCellBorders>
<syncfusion:CellBordersInfo
Bottom="{StaticResource ValueCellBorder}"
Left="{StaticResource ValueCellBorder}"
Right="{StaticResource ValueCellBorder}"
Top="{StaticResource ValueCellBorder}" />
</syncfusion:HeaderAppearance.HeaderCellBorders>
</syncfusion:HeaderAppearance>
</syncfusion:GridDataStyleManager.HeaderAppearance>

```



Customizing Header Appearance through the GridDataControl

The following properties illustrate how to customize the Header appearance through the GridDataControl.

Property

Property	Description
HeaderCellTemplate	Gets or sets the template for the Header cell.
ShowFilters	Shows or hides the filters in the Header cell.
AllowSort	Enables or disables sorting in the Header cell.

DefaultHeaderRowHeight	Gets or sets the row height of the Header cell.
HeaderStyle	Gets or sets the style for the Header cell.

The following code example illustrates the properties defined in this group.

XML

```
<!--HeaderTemplate-->
<DataTemplate x:Name="HeaderTemplate">
  <TextBox Height="30" Width="120"/>
</DataTemplate>
<syncfusion:GridDataControl x:Name="grid" ShowFilters="True"
AllowSort="True"
DefaultHeaderRowHeight="50"
HeaderCellTemplate="{StaticResource HeaderTemplate}"
HeaderStyle="{StaticResource GridDataHeaderCellControlStyle}"/>
<!-- HeaderStyle in Visible Column -->
<syncfusion:GridDataVisibleColumn MappingName="OrderId">
  <syncfusion:GridDataVisibleColumn.HeaderStyle>
    <syncfusion:GridDataColumnStyle Background="White"
    Foreground="Black"/>
  </syncfusion:GridDataVisibleColumn.HeaderStyle>
</syncfusion:GridDataVisibleColumn>
```

Order ID	Product ID	Unit Price	Quantity	Discount
10000	12	23	5	10.00 %
10000	14	59	10	2.00 %

Customizing the Row Appearance

The Row Appearance group properties enable to customize the appearance of rows in the GridDataControl.

Customizing Row Appearance through the GridDataStyleManager

The following properties illustrate how to customize row appearance through the GridDataStyleManager.

Property

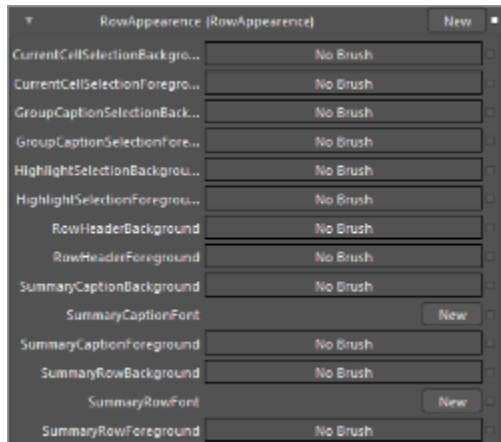
Property	Description
SummaryCaptionBackground	Gets or sets the background color for the summary caption row.
SummaryCaptionForeground	Gets or sets the foreground color for the summary caption row.
SummaryCaptionFont	Gets or sets the text font for the summary caption row.

SummaryRowBackground	Gets or sets the background color for the summary row.
SummaryRowForeground	Gets or sets the foreground color for the summary row.
SummaryRowFont	Gets or sets the text font for the summary row.
GroupCaptionSelectionBackground	Gets or sets the background color for the selected group caption row.
GroupCaptionSelectionForeground	Gets or sets the foreground color for the selected group caption row.
CurrentCellSelectionForeground	Gets or sets the foreground color for the selected cell in a row.
CurrentCellSelectionBackground	Gets or sets the background color for the selected cell in a row.
RowHeaderIconPath	Gets or sets the icon path for the row header.

The following code example illustrates the properties defined in this group.

XML

```
<syncfusion:GridDataStyleManager.RowAppearance>
<syncfusion:RowAppearance CurrentCellSelectionBackground="#FFB31B20"
CurrentCellSelectionForeground="#FFFFFF"
GroupCaptionSelectionBackground="#FFB71B21"
GroupCaptionSelectionForeground="#FFFFFF"
HighlightSelectionBackground="#FFCDBEA3"
HighlightSelectionForeground="#FFB71B21" RowHeaderBackground="White"
RowHeaderForeground="#FFFFFF" SummaryCaptionBackground="#FFF2E2C4"
SummaryCaptionForeground="#FF730202" SummaryRowBackground="#FFCBB284"
SummaryRowForeground="#FFFFFF">
<syncfusion:RowAppearance.SummaryCaptionFont>
<syncfusion:GridFontInfo FontFamily="Rockwell" FontSize="13.333"
FontWeight="Normal" />
</syncfusion:RowAppearance.SummaryCaptionFont>
<syncfusion:RowAppearance.SummaryRowFont>
<syncfusion:GridFontInfo FontFamily="Rockwell"
FontSize="13.333" />
</syncfusion:RowAppearance.SummaryRowFont>
</syncfusion:RowAppearance>
</syncfusion:GridDataStyleManager.RowAppearance>
```



Customizing Row Appearance through the GridDataControl

The following properties illustrate how to customize row appearance through the GridDataControl.

Properties

Property	Description
ShowGroupSummaries	Shows or hides group summaries in Grid.
ShowGroupSummaryInCaption	Shows or hides group summaries in Caption.
ShowTableSummaries	Shows or hides table summaries.
TableSummaryPosition	Gets or sets the position of table summary in Grid.
GroupCaptionText	Gets or sets the text in the Group Caption Area.
RowBackground	Gets or sets the background color of the row.
RowForeground	Gets or sets the foreground color of the row.
ShowRowHeader	Shows or hides the row header.
ShowRowHeaderArrow	Shows or hides the icon in the row header.
RowStyle	Gets or sets the styles for the row.
AlternateRowStyle	Gets or sets the styles for alternative rows.
AlternatingRowBackground	Gets or sets the background color for alternative rows.
AlternatingRowForeground	Gets or sets the foreground color for alternative rows.
UnboundRowPosition	Gets or sets the position of the unbound row in Grid.
AddNewRowPosition	Gets or sets the position of the new row to be added in Grid.

The following code example illustrates this.

XML

```
<syncfusion:GridDataControl x:Name="grid" ShowGroupSummaries="True"
ShowGroupSummaryInCaption="True" ShowTableSummaries="True"
```

```

TableSummaryPosition="Bottom"
GroupCaptionText="Customized Group Caption Text" RowBackground="White"
RowForeground="Black" ShowRowHeader="True" ShowRowHeaderArrow="True"
RowStyle="{StaticResource RowStyle}"
AlternateRowStyle="{StaticResource alternativeRowStyle}"
AlternatingRowBackground="White" AlternatingRowBackground="Wheat"
UnboundRowPosition="Bottom" AddNewRowPosition="Top"/>

```

Customizing the Value Cell Appearance

The Value Cell group enables to customize cells by changing their margins, borders, and so on.

Customizing the Value Cell Appearance through the GridDataStyleManager

The following properties illustrate how to customize the Value Cell appearance through the GridDataStyleManager.

Properties

Property	Description
ValueCellBorders	Gets or sets the cell border information of a value cell.
ValueFont	Gets or sets the font information of a value cell.
ValueTextMargins	Gets or sets the text margin of a value cell.
ValueBackgroundBrush	Gets or sets the background color of a value cell.
ValueForegroundBrush	Gets or sets the foreground color of a value cell.
CurrentCellBorderWidth	Gets or sets the border thickness for the selected cell.
CurrentCellBorderBrush	Gets or sets the border color for the selected cell.
HighlightSelectionBackground	Gets or sets the background color for the selected row.
HighlightSelectionForeground	Gets or sets the foreground color for the selected row.
HoveringRecordCellBackground	Gets or sets the background color to be displayed for the row when the pointer is moved over it.
HoveringRecordCellForeground	Gets or sets the foreground color to be displayed for the row when the pointer is moved over it.

The following code example illustrates the properties defined in this group.

XML

```

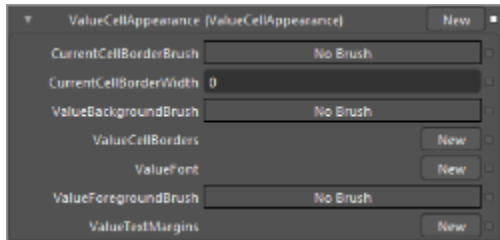
<syncfusion:GridDataStyleManager.ValueCellAppearance>
<syncfusion:ValueCellAppearance CurrentCellBorderBrush="#FFD6BE92"
ValueBackgroundBrush="#FFFDFBFC" ValueForegroundBrush="#FF333333">
<syncfusion:ValueCellAppearance.ValueCellBorders>
<syncfusion:CellBordersInfo
Bottom="{StaticResource ValueCellBorder}"
Left="{StaticResource ValueCellBorder}"
Right="{StaticResource ValueCellBorder}"
Top="{StaticResource ValueCellBorder}" />

```

```

</syncfusion:ValueCellAppearance.ValueCellBorders>
<syncfusion:ValueCellAppearance.ValueFont>
<syncfusion:GridFontInfo FontFamily="Rockwell" FontSize="10"
FontWeight="Normal" />
</syncfusion:ValueCellAppearance.ValueFont>
<syncfusion:ValueCellAppearance.ValueTextMargins>
<syncfusion:CellMarginsInfo Bottom="2" Left="2" Right="2" Top="2" />
</syncfusion:ValueCellAppearance.ValueTextMargins>
</syncfusion:ValueCellAppearance>
</syncfusion:GridDataStyleManager.ValueCellAppearance>

```



Customizing the Value Cell Appearance through the GridDataControl

The following properties illustrate how to customize the Value Cell appearance through the GridDataControl.

Properties

Property	Description
ExcelLikeSelectionFrame	Enables or disables excel-like selection frame in the cells.
Allow Edit	Enables or disables editing in the cells.
HighlightSelectionAlphaBlend	Gets or sets the background color of the selected row. GridDrawSelectionOptions must be set to AlphaBlend.
HighlightSelectionBackground	Gets or sets the background color of the selected row. GridDrawSelectionOptions must be set to ReplaceBackground.
HighlightSelectionBorder	Gets or sets the border color of the selected row.
HighlightSelectionBorderWidth	Gets or sets the border width of the selected row.
HighlightSelectionForeground	Gets or sets foreground color of the selected row.

The following code example illustrates this.

XML

```

<syncfusion:GridDataControl x:Name="dataGrid1"
ExcelLikeSelectionFrame="True"
AllowEdit="True" HighlightSelectionAlphaBlend="WhiteSmoke"
HighlightSelectionBackground="BlanchedAlmond"
HighlightSelectionBorder="Azure" HighlightSelectionBorderWidth="0.5"
HighlightSelectionForeground="Black"/>

```


Customizing the Nested Grid Appearance

The Nested Grid Group enables to customize the Nested cell borders by modifying the margin, borders, and so on.

Note: For Nested grid, Style Manager can be set by the Child Model. By default, Child inherits the styles from Parent.

The following code example illustrates this.

C#

```
void Table_RecordExpanded(object sender, GridDataValueEventArgs<GridDataRecord>e)
{
    e.Value.ChildModels[0].TableProperties.StyleManager = new GridDataStyleManager();
}
```

The following properties illustrate how to customize the Nested Grid appearance through the GridDataStyleManager.

Properties

Property	Description
TopLeftCellHeaderCellBorder	Gets or sets the cell border information for the top-left header cell of the Nested grid.
NestedHeaderCellBorder	Gets or sets the cell border information for the header cells excluding the top-left header cell of the Nested grid.
FirstHeaderColumnBorder	Gets or sets the border information for the first header column of value cells of the Nested grid.
LastHeaderColumnBorder	Gets or sets the border information for the last header column of value cells of the Nested grid.

The following code example illustrates the properties defined in this group.

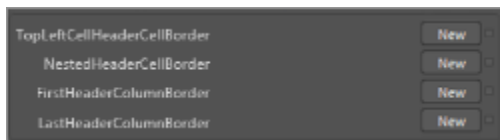
XML

```
<Pen x:Key="ValueCellBorder" Brush="#FFD6BE92" Thickness="0.25" />
<syncfusion:GridDataStyleManager.NestedGridAppearance>
  <syncfusion:NestedGridAppearance>
    <syncfusion:NestedGridAppearance.FirstHeaderColumnBorder>
      <syncfusion:CellBordersInfo
        Bottom="{StaticResource ValueCellBorder}"
        Left="{StaticResource ValueCellBorder}"
        Right="{StaticResource ValueCellBorder}"
        Top="{StaticResource ValueCellBorder}" />
    </syncfusion:NestedGridAppearance.FirstHeaderColumnBorder>
    <syncfusion:NestedGridAppearance.LastHeaderColumnBorder>
      <syncfusion:CellBordersInfo
        Bottom="{StaticResource ValueCellBorder}"
        Left="{StaticResource ValueCellBorder}"
        Right="{StaticResource ValueCellBorder}"
```

```

Top="{StaticResource ValueCellBorder}" />
</syncfusion:NestedGridAppearance.LastHeaderColumnBorder>
<syncfusion:NestedGridAppearance.NestedHeaderCellBorder>
<syncfusion:CellBordersInfo
Bottom="{StaticResource ValueCellBorder}"
Left="{StaticResource ValueCellBorder}"
Right="{StaticResource ValueCellBorder}"
Top="{StaticResource ValueCellBorder}" />
</syncfusion:NestedGridAppearance.NestedHeaderCellBorder>
<syncfusion:NestedGridAppearance.TopLeftCellHeaderCellBorder>
<syncfusion:CellBordersInfo
Bottom="{StaticResource ValueCellBorder}"
Left="{StaticResource ValueCellBorder}"
Right="{StaticResource ValueCellBorder}"
Top="{StaticResource ValueCellBorder}" />
</syncfusion:NestedGridAppearance.TopLeftCellHeaderCellBorder>
</syncfusion:NestedGridAppearance>
</syncfusion:GridDataStyleManager.NestedGridAppearance>

```



ContactTitle

Customer ID	Company Name	Contact Title	City	Country	Phone
ContactTitle : Accounting Manager - 10 Items					
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Tsawassen	Canada	(604) 555-4729
FISSA	FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	Madrid	Spain	(91) 555 94 44
HANAR	Hanari Carnes	Mario Pontes	Rio de Janeiro	Brazil	(21) 555-0091
Order ID	Customer ID	Employee ID	Freight	Ship Name	Ship Address
10180	HANAR	8	\$18.19	Hanari Carnes	Rua do Paço, 67
10250	HANAR	4	\$65.83	Hanari Carnes	Rua do Paço, 67
10253	HANAR	3	\$58.17	Hanari Carnes	Rua do Paço, 67
10541	HANAR	2	\$68.65	Hanari Carnes	Rua do Paço, 67
10645	HANAR	4	\$12.41	Hanari Carnes	Rua do Paço, 67

Editing Additional Template and Styles

Apart from the properties mentioned above, template and style for ScrollViewer, ContextMenu, GridDataHeaderCellControl, GridDataColumnOptionPane can be customized using the following properties.

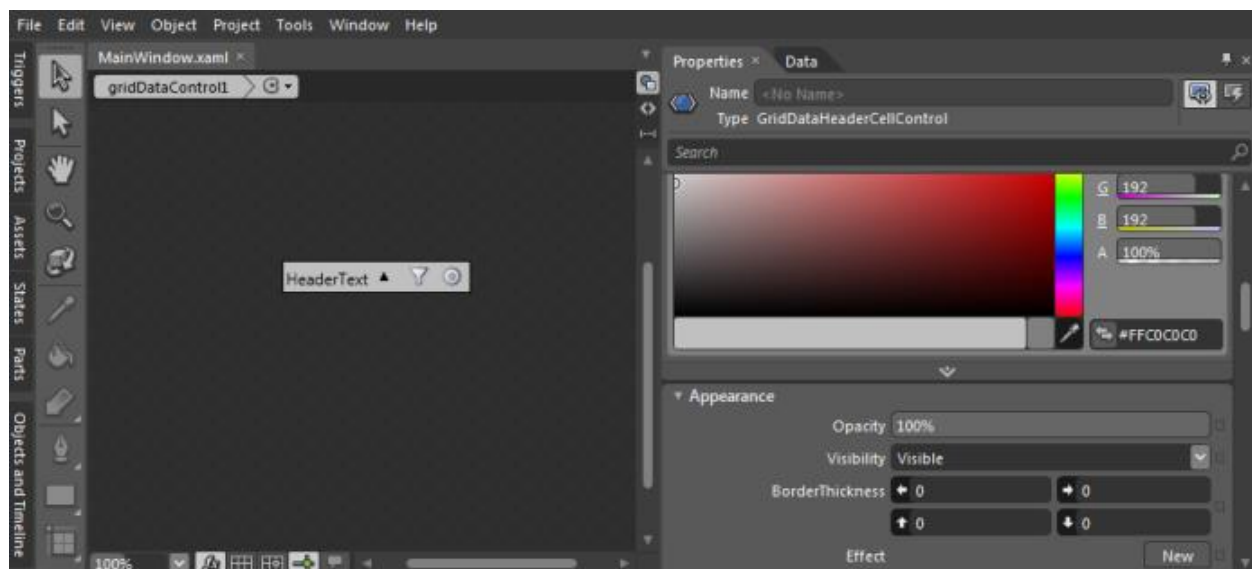
Properties

Property	TargetType
ContextMenuStyle	ContextMenu

ScrollViewerStyle	ScrollViewer
ColumnOptionPaneStyle	GridDataColumnOptionsPane
HeaderStyle	GridDataHeaderCellControl
RowStyle	GridDataRowControl
AlternateRowStyle	GridDataRowControl

Step by step instructions for Editing Templates

1. Open the application in Microsoft Expression Blend
2. Right-click on the GridDataControl and from Object menu, select Edit Additional Templates -> Edit \$PartName\$Style (e.g. HeaderStyle).



3. Blend creates a copy of the Template and opens the XAML. All the “Parts” that make up the selected template can be edited to make the required styling changes.

RowStyle and AlternateRowStyle

RowStyle and AlternateRowStyle are FrameworkElement Styles specifies the style for alternate rows in GridDataControl. These styles support all the basic properties of controls namely Background, Foreground, FontFamily, FontSize and FontFamily.

RowStyle

RowStyle can be set using XAML by overriding the style for GridDataRowControl class. This overrides the styles for each row. RowStyle supports all the basic control styles. The following code example explains the implementation of the RowStyle.

XML

```
<Style x:Key="RowStyle1" TargetType="{x:Type local:GridDataRowControl}">
  <Setter Property="Background" Value="Corn silk"/>
</Style>
```

```

<Setter Property="Foreground" Value="Fuchsia" />
<Setter Property="FontFamily" Value="Monotype Corsiva"/>
<Setter Property="FontSize" Value="13" />
<Setter Property="FontStretch" Value="Expanded" />
<Setter Property="FontStyle" Value="Normal" />
<Setter Property="FontWeight" Value="Normal" />
<Setter Property="Cursor" Value="Pen" />
<Setter Property="FlowDirection" Value="RightToLeft" />
<Setter Property="VerticalAlignment" Value="Center" />
<Setter Property="HorizontalAlignment" Value="Center" />
</Style>
<!--Assigning the style resource to the GridDataControl-->
RowStyle="{StaticResource RowStyle1}"

```

AlternateRowStyle

AlternateRowStyle defines the style for the GridDataRowControl element for alternate rows. If this is defined in XAML then the RowStyle and AlternateRowStyle applies for alternate elements.

AlternateRowStyle supports all the basic control styles. The following code example explains the implementation of the AlternateRowStyle.

XML

```

<Style x:Key="AlternateRowStyle1" TargetType="{x:Type
local:GridDataRowControl}">
<Setter Property="Background" Value="Cornsilk"/>
<Setter Property="Foreground" Value="Fuchsia" />
<Setter Property="FontFamily" Value="Monotype Corsiva"/>
<Setter Property="FontSize" Value="13" />
<Setter Property="FontStretch" Value="Expanded" />
<Setter Property="FontStyle" Value="Normal" />
<Setter Property="FontWeight" Value="Normal" />
<Setter Property="Cursor" Value="Pen" />
<Setter Property="FlowDirection" Value="RightToLeft" />
<Setter Property="VerticalAlignment" Value="Center" />
<Setter Property="HorizontalAlignment" Value="Center" />
</Style>
<!--Assigning the style resource to the GridDataControl-->
AlternateRowStyle="{StaticResource AlternateRowStyle1}"

```

Enhancement of GridDataCommandManager

Command is an input mechanism that provides input handling at a more semantic level. Commands in WPF are created by implementing the ICommand interface. ICommand exposes two methods, Execute, and CanExecute and an event CanExecuteChanged.

- Execute performs the actions that are associated with the command
- CanExecute determines whether the command can execute on the current command target
- CanExecuteChanged is raised if the command manager that centralizes the commanding operations detects a change in the command source that might invalidate a command that has been raised but not yet executed by the command binding.

GridDataControl provides Commands for the following events. This helps write the application in pure MVVM model.

- QueryCellInfoCommand
- SortColumnChangingCommand

QueryCellInfoCommand

QueryCellInfoCommand is widely used to allow customization of each and every cell in the required format. QueryCellInfo accepts an argument of type GridQueryCellInfoEventArgs. The GridQueryCellInfoEventArgs contains the following customization properties. Now you can make use of this

Properties

Property	Description
Cell	Gives the cell co-ordinates
Style	Specifies the style for the cell represented by the above Cell property

SortColumnChangingCommand

SortColumnChangingCommand allows customizing the sort columns at run time and this command get hooked before the column is sorted. SortColumnsChanging event accepts an argument of type GridDataSortColumnsChangingEventArgs. The GridDataSortColumnsChangingEventArgs contains the following customization properties. Now you can make use of this

Properties

Property	Description
AddedItems	Gives the Collection of GridDataSortColumns that are newly added to the GridDataControl.
RemovedItems	Gives the Collection of GridDataSortColumns that are removed from the GridDataControl.
NotifyCollectionChangedAction	Gives the Action that are going to perform in the SortColumns Collection in GridDataControl.

Properties

Property	Description	Type	Data Type
QueryCellInfoCommand	Gets or sets the command to invoke when QueryCellInfo event is triggered.	Dependency Property	ICommand
SortColumnChangingCommand	Gets or sets the command to invoke when SortColumnChanging event is triggered.	Dependency Property	ICommand

GridDataControl Column Chooser

The GridData control's column chooser allows you to add and remove columns dynamically from the current grid view via drag-and-drop operations. Dragging a column's header and dropping it onto the column chooser removes the column. To show the column, drag its header from the column chooser back to the view.

Features

- Add hidden columns in the Column Chooser window.
- Display a customized Column Chooser window.

Add Hidden Columns in the Column Chooser Window

By default, the column chooser doesn't display headers of hidden columns. A column is hidden if its `IsHidden` property is set to `false`. If you want to display a hidden column within the column chooser, set the column chooser's `CanAddHiddenColumns` property to `true`.

Display a Customized Column Chooser Window

The GridData control's column chooser enables you to customize the column chooser from the application side. This can be done by passing the Action method as a parameter of the `ShowColumnChooser()` method.

Use Case Scenarios

If an item source contains 15 columns and the user defines only three visible columns, activating the column chooser shows a window containing the header cells of the remaining columns and also enables the user to add and remove columns from the view by dragging and dropping them.

Tables for Properties, Methods, Events, and Commands

Properties

Property	Description	Type	Data Type
<code>CanAddHiddenColumns</code>	By setting this property to <code>true</code> , you can display the hidden <code>GridDataVisibleColumns</code> in the column chooser window.	<code>boolean</code>	<code>boolean</code>

Methods

Method	Description	Parameters	Type	Return Type
<code>ShowColumnChooser()</code>	By invoking this method, the GridData control's ColumnChooser window pops up.	<code>Null</code>	<code>----</code> <code>--</code>	<code>Void</code>
<code>ShowColumnChooser(Action<GridDataColumnChooserWindow>)</code>	Passing the Action method with a customized window	<code>Action<GridDataColumnChooserWindow></code>	<code>----</code> <code>--</code>	<code>Void</code>

	pops up tde customized GridDataColumnChoo serWindow.			
--	--	--	--	--

Commands

Command	Description
ColumnChooserCommand	This command is used to display the column chooser window.

Sample Link

To view sample:

1. Select Start > Programs > Syncfusion > Essential Studio x.x.xx > Dashboard.
2. Click Run Samples for WPF in the User Interface Edition panel.
3. Select GridDataControl.
4. Expand the Export Features item in the Sample Browser.

Choose the Export to PDF sample to launch.

Adding Column Chooser to an Application

1. Bind an ItemsSource to GridDataControl. Refer to the following link for more information about binding an ItemsSource to GridDataControl:

Data Binding.

XML

```
<syncfusion:GridDataControl x:Name="grid"
AutoPopulateColumns="False"
AutoPopulateRelations="False"
ColumnSizer="Star"
ItemsSource="{Binding OrdersDetail}"
ShowAddNewRow="False"
ShowGroupDropArea="True"
VisualStyle="Office14Blue">
<syncfusion:GridDataControl.VisibleColumns>
<!-- Set IsReadyOnly for columns -->
<syncfusion:GridDataVisibleColumn AllowSort="True"
HeaderText="Order ID"
MappingName="OrderID" />
<syncfusion:GridDataVisibleColumn AllowSort="True"
HeaderText="Customer ID"
MappingName="CustomerID" />
<syncfusion:GridDataDateTimeVisibleColumn AllowSort="True"
HeaderText="Order Date"
MappingName="OrderDate">
<syncfusion:GridDataDateTimeVisibleColumn.ColumnStyle>
<syncfusion:GridDataColumnStyle HorizontalAlignment="Right" />
</syncfusion:GridDataDateTimeVisibleColumn.ColumnStyle>
```

```
</syncfusion:GridDataDateTimeVisibleColumn>  
</syncfusion:GridDataControl.VisibleColumns>  
</syncfusion:GridDataControl>
```

Bind a button command to the ColumnChooserCommand command

The following code shows how to show the Column Chooser window using the built-in command of the GridData control.

XML

```
<Button Name="ColumnChooserBtn"  
Command="{Binding ColumnChooserCommand,ElementName=gridDataControl}"  
Content="Show Column Chooser" />
```

This shows the Column Chooser window whenever the user clicks the button.

Displaying Default Column Chooser by Invoking Method

The following code demonstrates how to show the Column Chooser window by hooking the button's click event.

XML

```
<Button Name="ColumnChooserBtn"  
Content="Show Column Chooser"  
Click="ColumnChooserBtn_Click"/>
```

C#

```
private void ColumnChooserBtn_Click(object sender, RoutedEventArgs e)  
{  
    this.gridDataControl.ShowColumnChooser();  
}
```

By using this, the user can get the default Column Chooser without any customization.

Customizing the Column Chooser by Invoking Parameterized Method

The following code shows how to customize the Column Chooser window from the application side. This is achieved by passing a method as a parameter for the ShowColumnChooser method.

XML

```
<Button Name="ColumnChooserBtn"  
Content="Show Column Chooser"  
Click="ColumnChooserBtn_Click"/>
```

C#

```
private void ColumnChooserBtn_Click(object sender, RoutedEventArgs e)  
{  
    gridDatacontrol.ShowColumnChooser((Chooser) =>  
    {  
        Chooser.Title = "Customized Column Chooser";  
        Chooser.CanAddHiddenColumns =
```



```
(gridDatacontrol.DataContext as ViewModel).CanAddHiddenColumns;
});
}
```

In the result you can get the column chooser window in your application as shown below.

Order ID	Customer ID	Order Date
10000	FRANS	5/10/1991
10001	MEREP	5/13/1991
10002	FOLKO	5/14/1991
10003	SIMOB	5/15/1991
10004	VAFFE	5/16/1991
10005	WARTH	
10006	FRANS	
10007	MORGK	
10008	FURIB	
10009	SEVES	
10010	SIMOB	
10011	WELLI	
10012	LINOD	
10013	RICSU	
10014	GROSR	
10015	PICCO	
10016	FOLIG	

Column Chooser Window	
EmployeeID	
ShipName	
ShipAddress	
ShipCity	
ShipRegion	
ShipPostalCode	
ShipCountry	
ShipVia	
RequiredDate	
ShippedDate	
Freight	

Exporting and Persistence in WPF GridDataControl (Classic)

This section covers the Exporting and persistence:

Exporting GDC to Excel

The GridExcelConverter class provides support for exporting data from a GridDataControl to an Excel spreadsheet for verification and/or computation. This control automatically copies the GridDataControl's styles and formats to Excel. The GridExcelConverter control is derived from GridExcelConverterBase. The XlsIO libraries are used to support the conversion of the GridDataControl contents to Excel. The following dll files should be added, along with the default dll files in the reference folder:

- Syncfusion.XlsIO.Base
- Syncfusion.XlsIO.WPF
- Syncfusion.GridConverter.Wpf

Features

Entire Content

You can convert the entire content of a GridDataControl 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 GridDataControl contents to an Excel Spreadsheet:

C#

```
gridDataControl.ExportToExcel("Sample.xlsx", ExcelVersion.Excel2007 );
(or)
gridDataControl.ExportToExcel("Sample.xls", ExcelVersion.Excel97to2003 );
```

Order ID	Customer ID	Employee ID	Order Date	Ship Via
10000	FRANS	6	10-05-1991 0...	3
10001	MEREP	8	13-05-1991 0...	3
10002	FOLKO	3	14-05-1991 0...	3
10003	SIMOB	8	15-05-1991 0...	1
10004	VAFFE	3	16-05-1991 0...	2
10005	WARTH	5	20-05-1991 0...	3
10006	FRANS	8	21-05-1991 0...	1
10007	MORGK	4	22-05-1991 0...	3
10008	FURIB	3	23-05-1991 0...	1
10009	SEVES	8	24-05-1991 0...	1
10010	SIMOB	8	28-05-1991 0...	1
10011	WELLI	6	29-05-1991 0...	3
10012	LINOD	6	30-05-1991 0...	1

Order ID	Customer ID	Employee ID	Order Date	Ship Via
10000	FRANS	6	05-10-1991	3
10001	MEREP	8	05-13-1991	3
10002	FOLKO	3	05-14-1991	3
10003	SIMOB	8	05-15-1991	1
10004	VAFFE	3	05-16-1991	2
10005	WARTH	5	05-20-1991	3
10006	FRANS	8	05-21-1991	1
10007	MORGK	4	05-22-1991	3
10008	FURIB	3	05-23-1991	1
10009	SEVES	8	05-24-1991	1

The above images shows how the entire content of the GridDataControl is exported to an Excel Spreadsheet.

[Selected Rows](#)

You can also avail the choice of converting the selected rows of GridDataControl to an Excel Spreadsheet.

The following code illustrates the conversion of selected rows of GridDataControl to an Excel Spreadsheet:

C#

```
grid.ExportToExcel(grid.Model.SelectedRanges.ActiveRange, "sample.xlsx",
ExcelVersion.Excel2007);
```

GridDataControl with Nested Child

You can convert the content of a GridDataControl, with Nested Child to an Excel Spreadsheet. Parent and visible child content are exported to Excel Spreadsheet.

The following code illustrates the conversion of GridDataControl with Nested Child to an Excel Spreadsheet:

C#

```
gridDataControl.ExportToExcel("Sample.xlsx", ExcelVersion.Excel2007 );
(or)
gridDataControl.ExportToExcel("Sample.xls", ExcelVersion.Excel97to2003 );
```

Note: Only the visible child's contents are exported.

Order ID	Customer ID	Employee ID	Order Date	Ship Via	Freight	Ship Name																				
10000	FRANS	6	10-05-1991 0...	3	4.45	Franchi S.p.A.																				
<table><tr><th>Order ID</th><th>Product ID</th><th>Unit Price</th><th>Quantity</th><th>Discount</th></tr><tr><td>10000</td><td>17</td><td>27</td><td>4</td><td>0</td></tr></table>							Order ID	Product ID	Unit Price	Quantity	Discount	10000	17	27	4	0										
Order ID	Product ID	Unit Price	Quantity	Discount																						
10000	17	27	4	0																						
10001	MEREP	8	13-05-1991 0...	3	79.45	Mère Paillarde																				
10002	FOLKO	3	14-05-1991 0...	3	36.18	Folk och få HB																				
<table><tr><th>Order ID</th><th>Product ID</th><th>Unit Price</th><th>Quantity</th><th>Discount</th></tr><tr><td>10002</td><td>31</td><td>8</td><td>35</td><td>0</td></tr><tr><td>10002</td><td>39</td><td>12.6</td><td>18</td><td>0</td></tr><tr><td>10002</td><td>71</td><td>15</td><td>15</td><td>0</td></tr></table>							Order ID	Product ID	Unit Price	Quantity	Discount	10002	31	8	35	0	10002	39	12.6	18	0	10002	71	15	15	0
Order ID	Product ID	Unit Price	Quantity	Discount																						
10002	31	8	35	0																						
10002	39	12.6	18	0																						
10002	71	15	15	0																						
10003	SIMOB	8	15-05-1991 0...	1	18.59	Simons bistro																				
10004	VAFFE	3	16-05-1991 0...	2	20.12	Vaffeljernet																				
10005	WARTH	5	20-05-1991 0...	3	4.13	Wartian Herkku																				
10006	FRANS	8	21-05-1991 0...	1	3.62	Franchi S.p.A.																				

Order ID	Customer ID	Employee ID	Order Date	Ship Via	Freight	Ship Name
10000	FRANS	6	05-10-1991	3	4.45	Franchi S.p.A.
10000	Order ID Product ID Unit Price Quantity Discount					
	17	27	4	0		
10001	MEREP	8	05-13-1991	3	79.45	Mère Paillarde
10002	FOLKO	3	05-14-1991	3	36.18	Folk och få HB
10002	Order ID Product ID Unit Price Quantity Discount					
	31	8	35	0		
10002		39	12.6	18	0	
10002		71	15	15	0	
10003	SIMOB	8	05-15-1991	1	18.59	Simons bistro
10004	VAFFE	3	05-16-1991	2	20.12	Vaffeljernet
10005	WARTH	5	05-20-1991	3	4.13	Wartian Herkkö

The above images shows how the GridControl, with Nested Child is exported to an Excel Spreadsheet.

GridDataControl with Grouping

You can convert the content of a GridDataControl, with Grouping to an Excel Spreadsheet. The following code illustrates this feature:

C#

```
gridDataControl.ExportToExcel("Sample.xlsx", ExcelVersion.Excel2007 );
(or)
gridDataControl.ExportToExcel("Sample.xls", ExcelVersion.Excel97to2003 );
```

Note: Only the visible grouping contents are exported.

Order ID	CustomerID	Freight	Order Date
7 Items		Total - 272.800	
11011	ALFKI	1.21	03-03-1994 00:00
10643	ALFKI	29.46	19-07-1993 00:00
10062	ALFKI	47.22	22-08-1991 00:00
10692	ALFKI	61.02	27-08-1993 00:00
10952	ALFKI	40.42	07-02-1994 00:00
10702	ALFKI	23.94	06-09-1993 00:00
10835	ALFKI	69.53	09-12-1993 00:00
Charges - 273.00\$ for 7 Items			
4 Items		Total - 97.420	
13 Items		Total - 521.350	
14 Items		Total - 503.320	
23 Items		Total - 1664.770	

	Order ID	CustomerID	Freight	Order Date
1	7 Items		Total - 272.800	
2	11011	ALFKJ	1.21	03-03-1994
3	10643	ALFKJ	29.46	07-19-1993
4	10062	ALFKJ	47.22	08-22-1991
5	10692	ALFKJ	61.02	08-27-1993
6	10952	ALFKJ	40.42	02-07-1994
7	10702	ALFKJ	23.94	09-06-1993
8	10835	ALFKJ	69.53	12-09-1993
9	Charges - 273.00\$ for 7 Items			
10	4 Items		Total - 97.420	
11	13 Items		Total - 521.350	
12	14 Items		Total - 503.320	
13	23 Items		Total - 1664.270	

The above images shows how the GridControl, with Grouping is exported to an Excel Spreadsheet.

[GridDataControl Export to CSV](#)

The ExportToCSV method of the GridModelExportExtensions class enables GridDataControl to easily be 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

Converting GridDataControl to CSV format

You can convert the entire content of a grid control to a CSV file by using the following code:

C#

```
this.gdc.Model.ExportToCSV("Sample.csv")
```

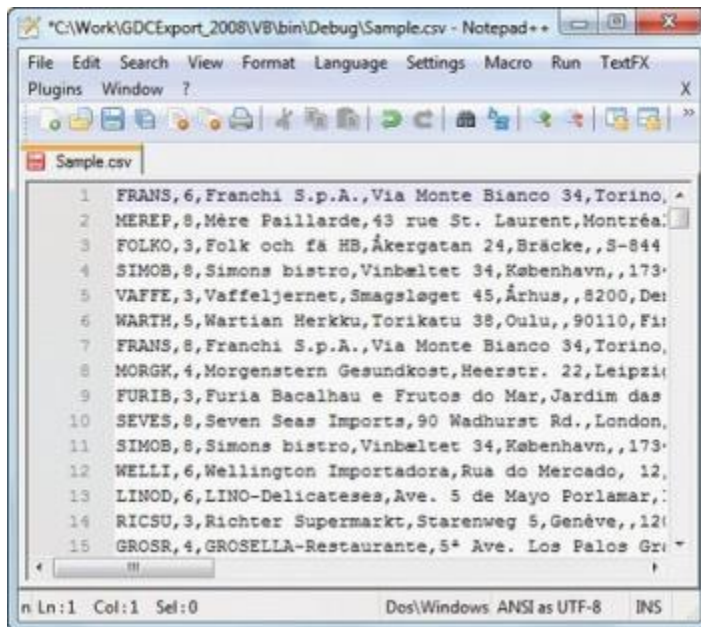
VB.NET

```
Me.gdc.Model.ExportToCSV("Sample.csv")
```

When the code runs, the following output displays.



When you are ready to export the entire grid, click Export to CSV; the grid content can then be converted to CSV format.



Export to PDF

Essential GridData control enables you to export the content of the GridData control into a pdf file. This feature allows you to maintain the records as a pdf file. The pdf libraries are used to support the conversion of the GridData control's content to pdf. The following dll files should be added along the default dll in the reference folder:

- Syncfusion.Pdf.Base.
- Syncfusion.GridConverter.Wpf

The pdf export can be performed in the following two ways:

- Export by PdfGrid
- Export by PdfLightTable

PdfGrid: In the PdfGrid, the formatting can be done to all levels of the PdfGrid. The features like row and column spanning are also supported by the PdfGrid. It offers, full control over the appearance of the PdfGrid table and is recommended to draw complex table structures.

PdfLightTable: It allows you to perform simple formatting, using the events. The PdfLightTable allows minimal customization options. Rendering the table using PdfLightTable is faster than PdfGrid and drawing a simple table is recommended.

Features

The export to pdf comprises the following features:

- Export entire content.
- Export selected range.
- Export GridData control with grouping.
- Export with styles and formatted cell value (This works by default).

Export Entire Content

Essential GridData control allows you to export the GridData control's entire content as a PDF file.

Use Case Scenario

A large data can be maintained as PDF file and the entire content of the GridData control can be exported iPDF a pdf file.

The following XAML code example shows, how the GridData control is defined in an application.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid"
AutoPopulateColumns="False"
AutoPopulateRelationships="False"
ColumnSizer="Star"
ItemsSource="{Binding OrderDetails}"
ShowAddNewRow="False"
VisualStyle="SyncfusionTheme">
<!--
- code for Visible Columns syncfusion:GridDataControl.VisibleColumns>
syncfusion:GridDataVisibleColumn HeaderText="Order ID" MappingName="OrderID">
syncfusion:GridDataVisibleColumn.HeaderStyle>
syncfusion:GridDataColumnStyle HorizontalAlignment="Center" />
syncfusion:GridDataVisibleColumn.HeaderStyle>
syncfusion:GridDataVisibleColumn.ColumnStyle>
syncfusion:GridDataColumnStyle HorizontalAlignment="Right" />
syncfusion:GridDataVisibleColumn.ColumnStyle>
syncfusion:GridDataVisibleColumn>
syncfusion:GridDataVisibleColumn AllowSort="False"
HeaderText="Customer ID"
MappingName="CustomerId">
```



```

yncfusionyncfusion:GridDataVisibleColumn.HeaderStyle>
yncfusionyncfusion:GridDataColumnStyle HorizontalAlignment="Center" />
yncfusionyncfusion:GridDataVisibleColumn.HeaderStyle>
yncfusionyncfusion:GridDataVisibleColumn>
yncfusionyncfusion:GridDataVisibleColumn.HeaderText="Ship Name"
MapBindingName="ShipName">
yncfusionyncfusion:GridDataVisibleColumn.HeaderStyle>
yncfusionyncfusion:GridDataColumnStyle HorizontalAlignment="Center" />
yncfusionyncfusion:GridDataVisibleColumn.HeaderStyle>
yncfusionyncfusion:GridDataVisibleColumn>
yncfusionyncfusion:GridDataVisibleColumn.HeaderText="Ship Address"
MapBindingName="ShipAddress" />yncfusionyncfusion:GridDataControl.VisibleColumns
yncfusionyncfusion:GridDataControl>

```

Order ID	Customer ID	Ship Name	Ship Address
10000	FRANS	Franchi S.p.A.	Via Monte Bianco 34
10001	MEREP	Mère Paillarde	43 rue St. Laurent
10002	FOLKO	Folk och få HB	Åkergatan 24
10003	SIMOB	Simons bistro	Vinbæltet 34
10004	VAFFE	Vaffeljernet	Smagsløget 45
10005	WARTH	Wartian Herkku	Torikatu 38
10006	FRANS	Franchi S.p.A.	Via Monte Bianco 34
10007	MORGK	Morgenstern Gesundkost	Heerstr. 22
10008	FURIB	Furia Bacalhau e Frutos...	Jardim das rosas n. 32
10009	SEVES	Seven Seas Imports	90 Wadhurst Rd.
10010	SIMOB	Simons bistro	Vinbæltet 34
10011	WELLI	Wellington Importadora	Rua do Mercado, 12
10012	LINOD	LINO-Delicateses	Ave. 5 de Mayo Porlamar
10013	RICSU	Richter Supermarkt	Starenweg 5

Exporting to PdfGrid

The following code example illustrates how to export the entire content of the GridData control into a PdfGrid.

C#

```

// Dialog to save the newly created pdf document.
SaveFileDialog sfd = new SaveFileDialog
{
    DefaultExt = ".pdf",
    Filter = "Adobe PDF Files (*.pdf) | *.pdf",
    FilterIndex = 1
};

// Newly created pdf document object.
PdfDocument document = new PdfDocument();
if (sfd.ShowDialog() == true)
{
    using (Stream stream = sfd.OpenFile())
    {
        //Method calling to export the grid content into pdf.
        document = grid.Model.ExportToPdfGridDocument(GridRangeInfo.Table());
        document.Save(stream);
        Process.Start(sfd.FileName);
    }
}

```



```
}
}
```

The following screenshot shows the exported pdf document:



Order ID	Customer ID	Ship Name	Ship Address
10000FRANS		Franchi S.p.A.	Via Monte Bianco 34
10001MEREP		Mère Paillarde	43 rue St. Laurent
10002FOLKO		Folk och få HB	Åkergatan 24
10003SIMOB		Simons bistro	Vinbæillet 34
10004VAFFE		Vaffeljernet	Smagøløget 45
10005WARTH		Wartian Herkkku	Torikatu 38
10006FRANS		Franchi S.p.A.	Via Monte Bianco 34
10007MORGK		Morgenstern Gesundkost	Heerstr. 22
10008FURIB		Furia Bacalhau e Frutos	Jardim das rosas n. 32
10009SEVES		Seven Seas Imports	90 Wadhurst Rd.
10010SIMOB		Simons bistro	Vinbæillet 34
10011WELLI		Wellington Importadora	Rua do Mercado, 12
10012LINOD		LINO-Delicateses	Ave. 5 de Mayo Porlamar
10013RICSU		Richter Supermarkt	Starenweg 5
10014GROSR		GROSELLA-Restaurante	5ª Ave. Los Palos
10015PICCO		Piccolo und mehr	Geistweg 14
10016FOLIG		Folies gourmandes	184, chaussée de Tournai
10017BLONP		Blondel père et fils	24, place Kléber
10018RATTC		Rattlesnake Canyon	2817 Milton Dr.
10019MAGAA		Magazzini Alimentari	Via Ludovico il Moro 22
10020VINET		Vins et alcools Chevalier	59 rue de l'Abbaye
10021ERNSH		Ernst Handel	Kirchgasse 6
10022LAMAI		La maison d'Asie	1 rue Alsace-Lorraine
10023TOMSP		Toms Spezialitäten	Luisenstr. 48
10024RATTC		Rattlesnake Canyon	2817 Milton Dr.
10025RATTC		Rattlesnake Canyon	2817 Milton Dr.
10026MORGK		Morgenstern Gesundkost	Heerstr. 22
10027ERNSH		Ernst Handel	Kirchgasse 6
10028ANTON		Antonio Moreno Taqueria	Mataderos 2312
10029SANTO		Santa Gourmet	Erling Skjelleve, rute 78

Exporting to PdfLightTable Document

The following code example illustrates how to export the entire content of the GridData control into a PdfLightTable document file.

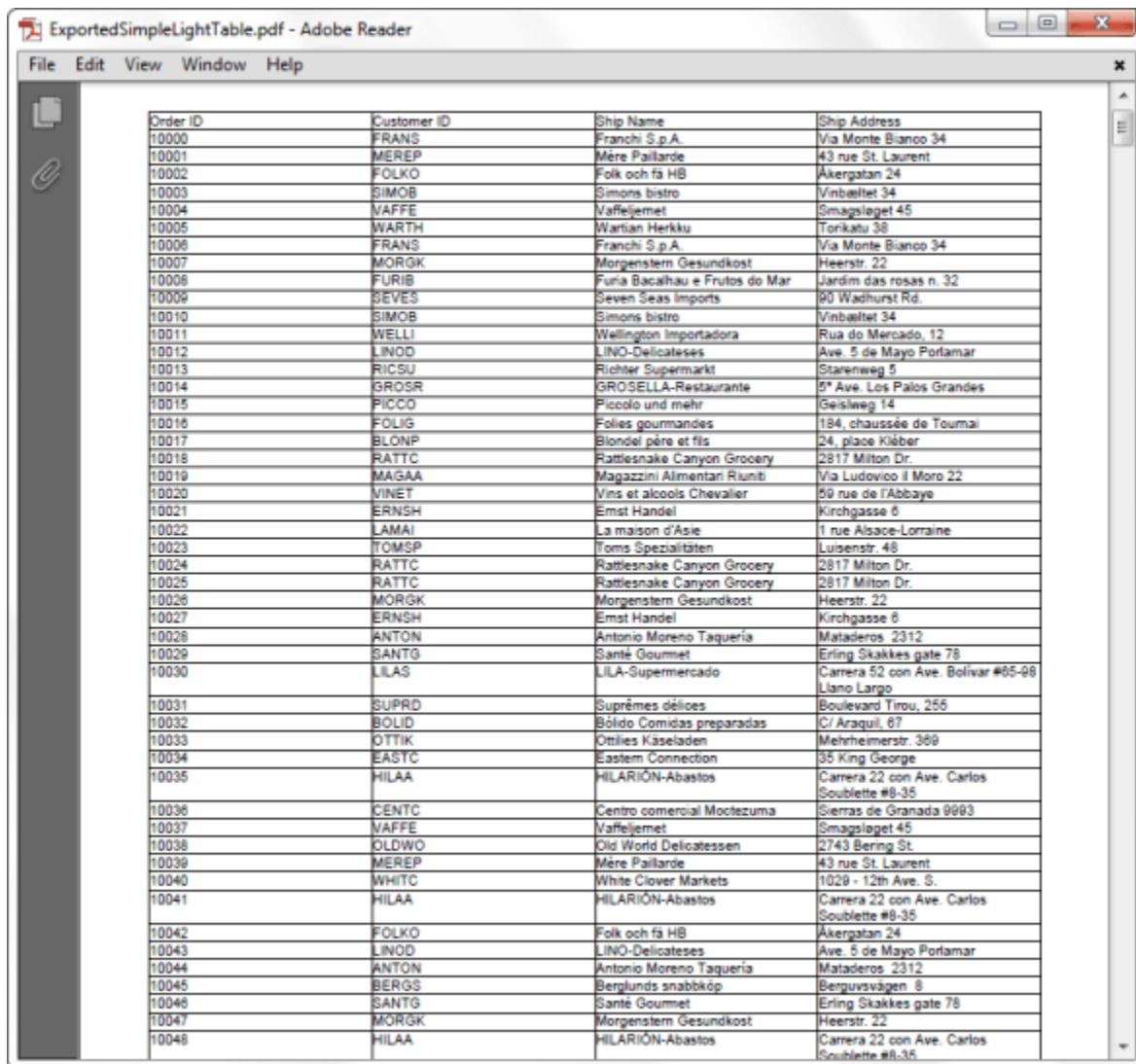
C#

```
// Dialog to save the newly created pdf document.
SaveFileDialog sfd = new SaveFileDialog
{
    DefaultExt = ".pdf",
    Filter = "Adobe PDF Files (*.pdf) | *.pdf",
    FilterIndex = 1
};
// Newly created pdf document object.
```

```

PdfDocument document = new PdfDocument();
if (sfd.ShowDialog() == true)
{
    using (Stream stream = sfd.OpenFile())
    {
        //Method calling to export the grid content into pdf.
        document = grid.Model.ExportToPdfLightTableDocument(GridRangeInfo.Table());
        document.Save(stream);
        Process.Start(sfd.FileName);
    }
}

```



Order ID	Customer ID	Ship Name	Ship Address
10000	FRANS	Franchi S.p.A.	Via Monte Bianco 34
10001	MEREP	Mère Paillardé	43 rue St. Laurent
10002	FOLKO	Folk och få HB	Åkergatan 24
10003	SIMOB	Simons bistro	Vinbæltet 34
10004	VAFFE	Vaffeljernet	Smagslaget 45
10005	WARTH	Wartian Herkku	Torikatu 38
10006	FRANS	Franchi S.p.A.	Via Monte Bianco 34
10007	MORGK	Morgenstern Gesundkost	Heerstr. 22
10008	FURIB	Furia Bacalhau e Frutos do Mar	Jardim das rosas n. 32
10009	SEVES	Seven Seas Imports	90 Wadhurst Rd.
10010	SIMOB	Simons bistro	Vinbæltet 34
10011	WELLI	Wellington Importadora	Rua do Mercado, 12
10012	LINOD	LINO-Delicatesses	Ave. 5 de Mayo Portamar
10013	RICSU	Richter Supermarkt	Starenweg 5
10014	GROSR	GROSELLA-Restaurante	5* Ave. Los Palos Grandes
10015	PICCO	Piccolo und mehr	Geislweg 14
10016	FOLIG	Folies gourmandes	184, chaussée de Toulmai
10017	BLONP	Blondel père et fils	24, place Kléber
10018	RATTC	Rattlesnake Canyon Grocery	2817 Milton Dr.
10019	MAGAA	Magazzini Alimentari Riuniti	Via Ludovico il Moro 22
10020	VINET	Vins et alcools Chevalier	59 rue de l'Abbaye
10021	ERNSH	Ernst Handel	Kirchgasse 6
10022	LAMAI	La maison d'Asie	1 rue Alsace-Lorraine
10023	TOMSP	Toms Spezialitäten	Luisenstr. 48
10024	RATTC	Rattlesnake Canyon Grocery	2817 Milton Dr.
10025	RATTC	Rattlesnake Canyon Grocery	2817 Milton Dr.
10026	MORGK	Morgenstern Gesundkost	Heerstr. 22
10027	ERNSH	Ernst Handel	Kirchgasse 6
10028	ANTON	Antonio Moreno Taquería	Mataderos. 2312
10029	SANTG	Santé Gourmet	Erling Skakkes gate 78
10030	LILAS	LILA-Supermercado	Carrera 52 con Ave. Bolívar #65-98 Llano Largo
10031	SUPRD	Suprêmes délices	Boulevard Tirou, 255
10032	BOLID	Bólido Comidas preparadas	C/ Araquil, 67
10033	OTTIK	Ottiles Käseladen	Mehrheimerstr. 369
10034	EASTC	Eastern Connection	35 King George
10035	HILAA	HILARIÓN-Abastos	Carrera 22 con Ave. Carlos Soublette #8-35
10036	CENTC	Centro comercial Moctezuma	Sierras de Granada 9993
10037	VAFFE	Vaffeljernet	Smagslaget 45
10038	OLDWO	Old World Delicatessen	2743 Bering St.
10039	MEREP	Mère Paillardé	43 rue St. Laurent
10040	WHITC	White Clover Markets	1029 - 12th Ave. S.
10041	HILAA	HILARIÓN-Abastos	Carrera 22 con Ave. Carlos Soublette #8-35
10042	FOLKO	Folk och få HB	Åkergatan 24
10043	LINOD	LINO-Delicatesses	Ave. 5 de Mayo Portamar
10044	ANTON	Antonio Moreno Taquería	Mataderos. 2312
10045	BERGS	Berglunds snabbköp	Berguvsvägen 8
10046	SANTG	Santé Gourmet	Erling Skakkes gate 78
10047	MORGK	Morgenstern Gesundkost	Heerstr. 22
10048	HILAA	HILARIÓN-Abastos	Carrera 22 con Ave. Carlos Soublette #8-35

Export Selected Range

You can convert the selected range of the GridData control into a pdf file.

Order ID	Customer ID	Ship Name	Ship Address
10000	FRANS	Franchi S.p.A.	Via Monte Bianco 34
10001	MEREP	Mère Paillarde	43 rue St. Laurent
10002	FOLKO	Folk och få HB	Åkergatan 24
10003	SIMOB	Simons bistro	Vinbæltet 34
10004	VAFFE	Vaffeljernet	Smagsløget 45
10005	WARTH	Wartian Herkku	Torikatu 38
10006	FRANS	Franchi S.p.A.	Via Monte Bianco 34
10007	MORGK	Morgenstern Gesundkost	Heerstr. 22
10008	FURIB	Furia Bacalhau e Frutos do M...	Jardim das rosas n. 32
10009	SEVES	Seven Seas Imports	90 Wadhurst Rd.
10010	SIMOB	Simons bistro	Vinbæltet 34
10011	WELLI	Wellington Importadora	Rua do Mercado, 12
10012	LINOD	LINO-Delicateses	Ave. 5 de Mayo Porlamar
10013	RICSU	Richter Supermarkt	Starenweg 5
10014	GROSR	GROSELLA-Restaurante	5ª Ave. Los Palos Grandes

Exporting to PdfGrid

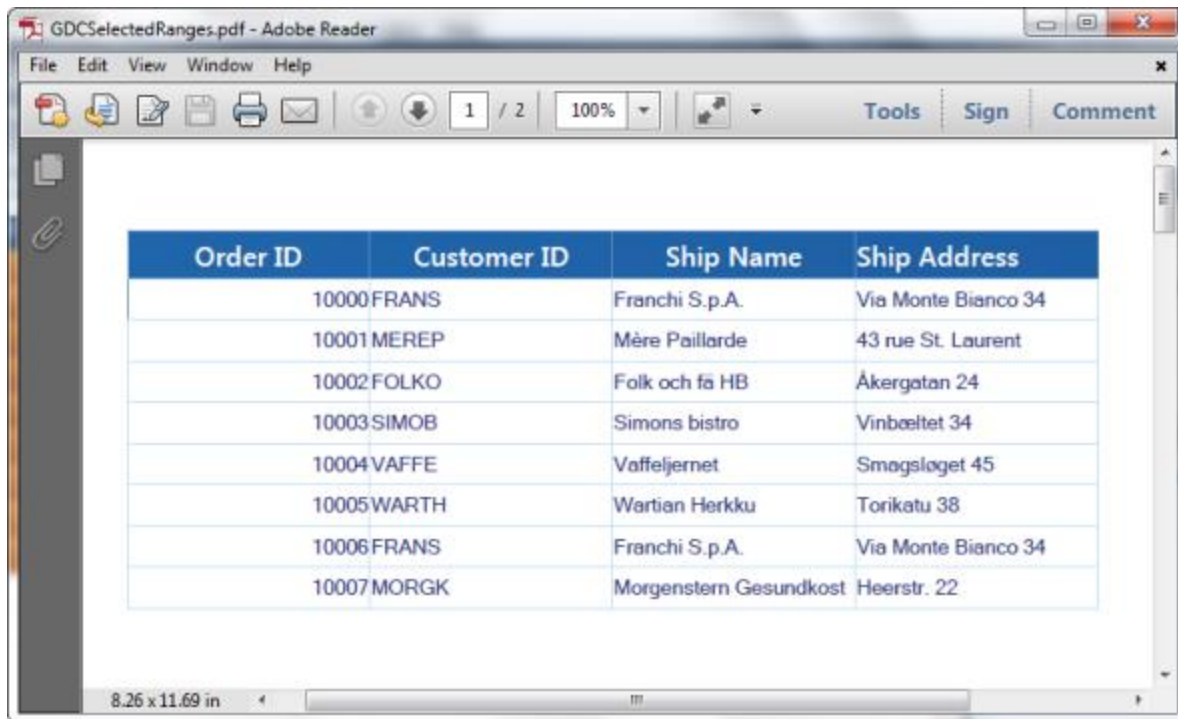
The following code illustrates the conversion of a selected range of the GridData control to a PdfGrid.

C#

```
// Dialog to save the newly created pdf document.
SaveFileDialog sfd = new SaveFileDialog
{
    DefaultExt = ".pdf",
    Filter = "Adobe PDF Files (*.pdf) | *.pdf",
    FilterIndex = 1
};

// Newly created pdf document object.
PdfDocument document = new PdfDocument();
if (sfd.ShowDialog() == true)
{
    using (Stream stream = sfd.OpenFile())
    {
        //Method calling to export the grid content into pdf.
        document = grid.Model.ExportToPdfGridDocument(
            grid.Model.SelectedRanges.ActiveRange);
        document.Save(stream);
        Process.Start(sfd.FileName);
    }
}
```

The following screenshot shows the exported pdf document of a selected range of the GridData control:



Exporting to PdfLightTable Document

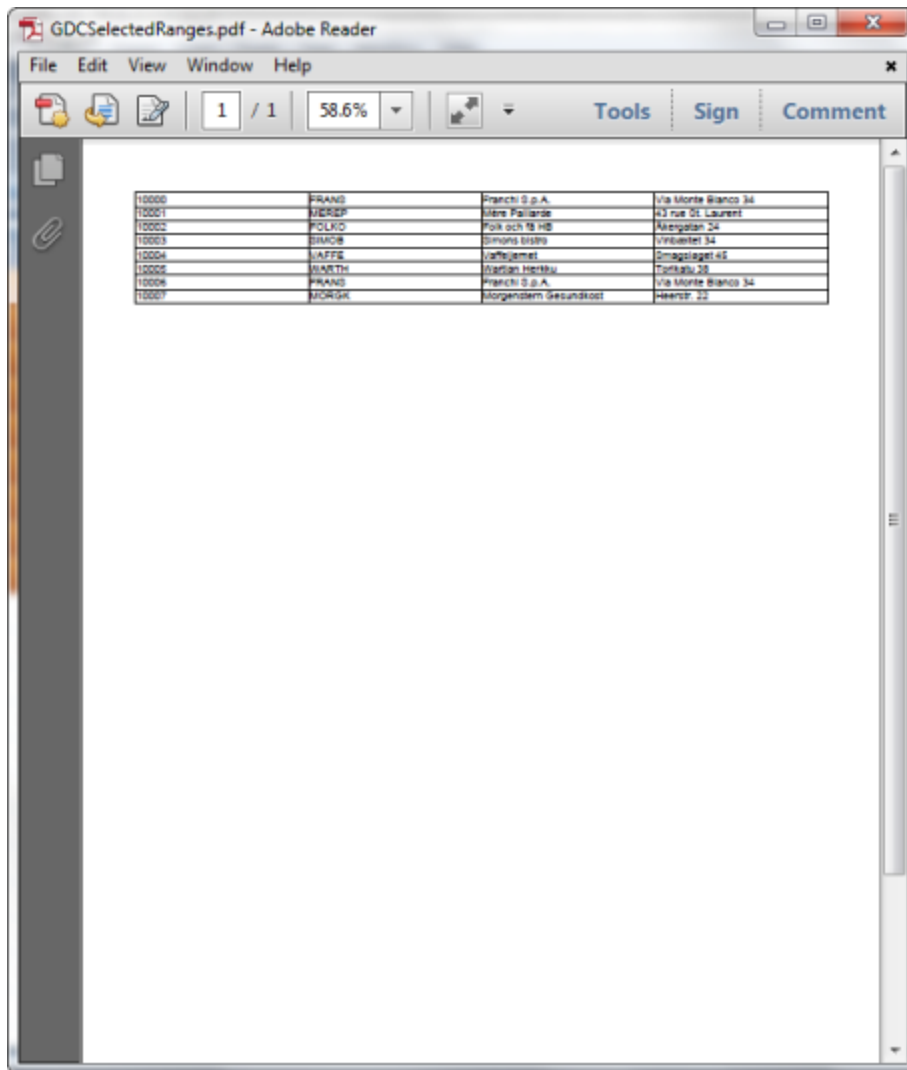
The following code illustrates the conversion of a selected range of the GridData control to a PdfLightTable document.

C#

```
// Dialog to save the newly created pdf document.
SaveFileDialog sfd = new SaveFileDialog
{
    DefaultExt = ".pdf",
    Filter = "Adobe PDF Files (*.pdf) | *.pdf",
    FilterIndex = 1
};

// Newly created pdf document object.
PdfDocument document = new PdfDocument();
if (sfd.ShowDialog() == true)
{
    using (Stream stream = sfd.OpenFile())
    {
        //Method calling to export the grid content into pdf.
        document = grid.Model.ExportToPdfLightTableDocument(grid.Model.SelectedRanges.ActiveRange);
        document.Save(stream);
        Process.Start(sfd.FileName);
    }
}
```

The following screenshot shows the exported PdfLightTable document of the selected range of the GridData control.



[Export GridDataControl with Grouping](#)

The GridData control converts the content of the GridData control to a pdf document with grouping.

The following screenshot illustrates how the GridData control appears as a pdf file after grouping the data.

	Order ID	Customer ID	Ship Name	Ship Address	Freight
▲ 11 Items					
	10000	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$4.45
	10006	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$3.62
	10086	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$178.17
	10147	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$63.69
	10239	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$45.17
	10422	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$3.02
	10710	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$4.98
	10753	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$7.70
	10807	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$1.36
	11026	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$47.09
	11060	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$10.98
				370.00	
	Total - 11 Items				
▲ 20 Items					
	10001	MEREP	Mère Paillarde	43 rue St. Laurent	\$79.45
	10039	MEREP	Mère Paillarde	43 rue St. Laurent	\$7.37
	10087	MEREP	Mère Paillarde	43 rue St. Laurent	\$79.65
	10121	MEREP	Mère Paillarde	43 rue St. Laurent	\$112.83
	10131	MEREP	Mère Paillarde	43 rue St. Laurent	\$114.02

Order ID	Customer ID	Ship Name	Ship Address	Freight
- 11 Items				
10000	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$4.45
10006	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$3.62
10086	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$178.17
10147	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$63.69
10239	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$45.17
10422	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$3.02
10710	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$4.98
10753	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$7.70
10807	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$1.36
11026	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$47.09
11060	FRANS	Franchi S.p.A.	Via Monte Bianco 34	\$10.98
				370.00
Total - 11 Items				
- 20 Items				
10001	MEREP	Mère Paillarde	43 rue St. Laurent	\$79.45
10039	MEREP	Mère Paillarde	43 rue St. Laurent	\$7.37
10087	MEREP	Mère Paillarde	43 rue St. Laurent	\$79.65
10121	MEREP	Mère Paillarde	43 rue St. Laurent	\$112.83
10131	MEREP	Mère Paillarde	43 rue St. Laurent	\$214.92
10174	MEREP	Mère Paillarde	43 rue St. Laurent	\$45.44
10214	MEREP	Mère Paillarde	43 rue St. Laurent	\$78.77
10332	MEREP	Mère Paillarde	43 rue St. Laurent	\$52.84
10339	MEREP	Mère Paillarde	43 rue St. Laurent	\$15.66
10376	MEREP	Mère Paillarde	43 rue St. Laurent	\$20.39
10424	MEREP	Mère Paillarde	43 rue St. Laurent	\$370.61

The following screenshot illustrates how the GridData control appears as a PdfLightTable document after grouping the data.

ExportedGDCLightTablewithgrouping.pdf - Adobe Reader

File Edit View Window Help

1 / 23 78.4%

Tools Sign Comment

	Order ID	Customer ID	Ship Name	Ship Address	Freight
True	11 Items				
	10000	FRANS	Franchi S.p.A.	Via Monte Bianco 34	4.45
	10006	FRANS	Franchi S.p.A.	Via Monte Bianco 34	3.62
	10086	FRANS	Franchi S.p.A.	Via Monte Bianco 34	178.17
	10147	FRANS	Franchi S.p.A.	Via Monte Bianco 34	63.69
	10239	FRANS	Franchi S.p.A.	Via Monte Bianco 34	45.17
	10422	FRANS	Franchi S.p.A.	Via Monte Bianco 34	3.02
	10710	FRANS	Franchi S.p.A.	Via Monte Bianco 34	4.98
	10753	FRANS	Franchi S.p.A.	Via Monte Bianco 34	7.7
	10807	FRANS	Franchi S.p.A.	Via Monte Bianco 34	1.36
	11026	FRANS	Franchi S.p.A.	Via Monte Bianco 34	47.09
	11060	FRANS	Franchi S.p.A.	Via Monte Bianco 34	10.98
					370.00
	Total - 11 Items				
True	20 Items				
	10001	MEREP	Mère Pailarde	43 rue St. Laurent	79.45
	10039	MEREP	Mère Pailarde	43 rue St. Laurent	7.37
	10087	MEREP	Mère Pailarde	43 rue St. Laurent	79.85
	10121	MEREP	Mère Pailarde	43 rue St. Laurent	112.83
	10131	MEREP	Mère Pailarde	43 rue St. Laurent	214.92
	10174	MEREP	Mère Pailarde	43 rue St. Laurent	45.44
	10214	MEREP	Mère Pailarde	43 rue St. Laurent	78.77
	10332	MEREP	Mère Pailarde	43 rue St. Laurent	52.84
	10339	MEREP	Mère Pailarde	43 rue St. Laurent	15.66
	10376	MEREP	Mère Pailarde	43 rue St. Laurent	20.39
	10424	MEREP	Mère Pailarde	43 rue St. Laurent	370.61
	10439	MEREP	Mère Pailarde	43 rue St. Laurent	4.07
	10505	MEREP	Mère Pailarde	43 rue St. Laurent	7.13
	10565	MEREP	Mère Pailarde	43 rue St. Laurent	7.15
	10570	MEREP	Mère Pailarde	43 rue St. Laurent	188.99
	10590	MEREP	Mère Pailarde	43 rue St. Laurent	44.77
	10605	MEREP	Mère Pailarde	43 rue St. Laurent	379.13
	10618	MEREP	Mère Pailarde	43 rue St. Laurent	154.68
	10619	MEREP	Mère Pailarde	43 rue St. Laurent	91.05
	10724	MEREP	Mère Pailarde	43 rue St. Laurent	57.75
					2013.00
	Total - 20 Items				
True	26 Items				
	10002	FOLKO	Folk och få HB	Åkergatan 24	36.18
	10042	FOLKO	Folk och få HB	Åkergatan 24	106.03
	10051	FOLKO	Folk och få HB	Åkergatan 24	21.73
	10072	FOLKO	Folk och få HB	Åkergatan 24	177.24
	10073	FOLKO	Folk och få HB	Åkergatan 24	97.25
	10103	FOLKO	Folk och få HB	Åkergatan 24	88.17
	10210	FOLKO	Folk och få HB	Åkergatan 24	9.49
	10264	FOLKO	Folk och få HB	Åkergatan 24	3.67
	10327	FOLKO	Folk och få HB	Åkergatan 24	63.28

Exporting Customized GridData Control

Use the following code to customize the GridData control with blend styling.

XML

```
<syncfusion:GridDataControl x:Name="grid"
Grid.Row="0"
Margin="10"
AllowEdit="False"
AutoPopulateColumns="False"
AutoPopulateRelations="False"
ColumnSizer="Star"
ContextMenuOptions="CustomWithDefault"
ContextMenuStyle="{StaticResource RContextMenuStyle}"
EnableBlendStyling="True"
HeaderStyle="{StaticResource
GridDataHeaderCellStyle2}"
HideColumnsWhenGrouped="True"
IsGroupsExpanded="True"
ItemsSource="{Binding Path=MovieDetails}"
```



```

ListBoxSelectionMode="MultiExtended"
PersistGroupsExpandState="True"
ShowAddNewRow="False"
ShowFilters="True"
ShowGroupDropArea="True"
ShowHoveringBackground="false"
ShowTableSummaries="True"
ShowTooltips="True"
StyleManager="{StaticResource CustomGridDataStyleManager}">
<!-- Table Summary rows created here -->
<syncfusion:GridDataControl.TableSummaryRows>
<syncfusion:GridDataSummaryRow Title="Total : {CountSummary} Items"
ShowSummaryInRow="True"
TitleColumnCount="2">
<syncfusion:GridDataSummaryRow.SummaryColumns>
<syncfusion:GridDataSummaryColumn Name="CountSummary"
Format="{Count:d}"
MappingName="OrderId"
SummaryType="CountAggregate" />
</syncfusion:GridDataSummaryRow.SummaryColumns>
</syncfusion:GridDataSummaryRow>
</syncfusion:GridDataControl.TableSummaryRows>
<!-- Grouped Column Created here -->
<syncfusion:GridDataControl.GroupedColumns>
<syncfusion:GridDataGroupColumn ColumnName="Movie" />
</syncfusion:GridDataControl.GroupedColumns>
<!-- Visible Column Created here -->
<syncfusion:GridDataControl.VisibleColumns>
<syncfusion:GridDataVisibleColumn MappingName="Movie">
<syncfusion:GridDataVisibleColumn.FilterPane>
<syncfusion:GridDataTextFilteringPane Foreground="Black"
IsThemed="False"
PredicateType="And" />
</syncfusion:GridDataVisibleColumn.FilterPane>
</syncfusion:GridDataVisibleColumn>
<syncfusion:GridDataVisibleColumn ColumnStyle="{StaticResource GridDataColumnStyle}"
MappingName="OrderId">
<syncfusion:GridDataVisibleColumn.FilterPane>
<syncfusion:GridDataTextFilteringPane Foreground="Black"
IsThemed="False"
PredicateType="And" />
</syncfusion:GridDataVisibleColumn.FilterPane>
</syncfusion:GridDataVisibleColumn>
<syncfusion:GridDataVisibleColumn ColumnStyle="{StaticResource GridDataColumnStyle}"
MappingName="Name">
<syncfusion:GridDataVisibleColumn.FilterPane>
<syncfusion:GridDataTextFilteringPane Foreground="Black"
IsThemed="False"
PredicateType="And" />
</syncfusion:GridDataVisibleColumn.FilterPane>
</syncfusion:GridDataVisibleColumn>
<syncfusion:GridDataVisibleColumn ColumnStyle="{StaticResource GridDataColumnStyle}"
MappingName="SeatNo">
<syncfusion:GridDataVisibleColumn.FilterPane>

```

```

<syncfusion:GridDataTextFilteringPane Foreground="Black"
IsThemed="False"
PredicateType="And" />
</syncfusion:GridDataVisibleColumn.FilterPane>
</syncfusion:GridDataVisibleColumn>
<syncfusion:GridDataVisibleColumn ColumnStyle="{StaticResource GridDataColumnStyle}"
ppingName="City">
<syncfusion:GridDataVisibleColumn.FilterPane>
<syncfusion:GridDataTextFilteringPane Foreground="Black"
IsThemed="False"
PredicateType="And" />
</syncfusion:GridDataVisibleColumn.FilterPane>
</syncfusion:GridDataVisibleColumn>
<syncfusion:GridDataVisibleColumn ColumnStyle="{StaticResource GridDataColumnStyle}"
ngName="Theatre" />
</syncfusion:GridDataControl.VisibleColumns>
</syncfusion:GridDataControl>

```

Use the following code to export a customized GridData control:

Button Code to Export:

XML

```

<Button Name="Exportbtn" Content="Export To Pdf" Click="Exportbtn_Click"/>

```

Button Click Event Code:

C#

```

private void Exportbtn_Click(object sender, RoutedEventArgs e)
{
    // Dialog to save the exported document.
    SaveFileDialog sfd = new SaveFileDialog
    {
        DefaultExt = ".pdf",
        Filter = "Adobe PDF Files (*.pdf)|*.pdf",
        FilterIndex = 1
    };
    // Pdf document object to save the data as a pdf file.
    PdfDocument document = new PdfDocument();
    if (sfd.ShowDialog() == true)
    {
        using (Stream stream = sfd.OpenFile())
        {
            document = dataGrid.Model.ExportToPdfGridDocument(GridRangeInfo.Table());
            document.Save(stream);
            Process.Start(sfd.FileName);
        }
    }
}

```

GridDataControl with Blend Styling

The below screenshot shows the customized blend styling of the GridData control.

Movie ▲

OrderId	Name	SeatNo	City	Theatre
Movie : Avatar - 3 Items				
10765	William	F23,24,25,26	UK	Lodo
10645	Smith	A-12	Canada	Modern
10667	Johnson	B-16, 17	Canada	Greek
Movie : Ice Age3 - 3 Items				
10767	Johnson	B-16, 17	Canada	Greek
10765	William	F23,24,25,26	UK	Lodo
10978	William	F23,24,25,26	UK	Lodo
Movie : Kung Fu Panda 2 - 2 Items				
10062	Smith	A-12	Canada	Modern
10978	William	F23,24,25,26	UK	Lodo
Movie : Shrek - 4 Items				
10643	Johnson	B-16, 17	Canada	Greek
10942	William	F23,24,25,26	UK	Lodo
10942	William	F23,24,25,26	UK	Lodo
10667	Johnson	B-16, 17	Canada	Greek
Movie : The Hangover Part II - 4 Items				
10065	Smith	A-12	Canada	Modern
10065	Smith	A-12	Canada	Modern
10767	Johnson	B-16, 17	Canada	Greek
Total : 27 Items				

[Exported PDF Document](#)

The screenshot below shows a PDF document of the blend styling GridData control.

OrderId	Name	SeatNo	City	Theatre
- Movie : Avatar - 3 Items				
10765	William	F23,24,25,26	UK	Lodo
10645	Smith	A-12	Canada	Modern
10567	Johnson	B-16, 17	Canada	Greek
- Movie : Ice Age3 - 3 Items				
10767	Johnson	B-16, 17	Canada	Greek
10765	William	F23,24,25,26	UK	Lodo
10978	William	F23,24,25,26	UK	Lodo
- Movie : Kung Fu Panda 2 - 2 Items				
10062	Smith	A-12	Canada	Modern
10978	William	F23,24,25,26	UK	Lodo
- Movie : Shrek - 4 Items				
10643	Johnson	B-16, 17	Canada	Greek
10942	William	F23,24,25,26	UK	Lodo
10942	William	F23,24,25,26	UK	Lodo
10567	Johnson	B-16, 17	Canada	Greek
- Movie : The Hangover Part II - 4 Items				
10865	Smith	A-12	Canada	Modern
10865	Smith	A-12	Canada	Modern
10767	Johnson	B-16, 17	Canada	Greek
10978	William	F23,24,25,26	UK	Lodo
- Movie : The Priest - 4 Items				
10645	Smith	A-12	Canada	Modern
10643	Johnson	B-16, 17	Canada	Greek
10767	Johnson	B-16, 17	Canada	Greek
10765	William	F23,24,25,26	UK	Lodo
- Movie : Toy Story 3 - 4 Items				
10942	William	F23,24,25,26	UK	Lodo
10062	Smith	A-12	Canada	Modern
10062	Smith	A-12	Canada	Modern

Serialization in GridDataControl

GridDataControl state can be serialized in XML format.

All the properties that are exposed in GridDataTableProperties can be serialized.

Serializing

There are two methods to serialize forms:

- XML string
- XML file

API Usage

Serializing as XML String

The following code illustrates how to serialize as an XML string.

C#

```
string result = this.dataGrid.Model.SerializeAsString();
```

Serializing as an XML File

The following code illustrates how to serialize as an XML file.

C#

```
this.dataGrid.Model.Serialize("sample.xml");
```

De-serializing

There are two methods to serialize forms:

- XML string
- XML file

API Usage

De-serialize from XML String

The following code illustrates how to de-serialize from an XML string.

C#

```
this.dataGrid.Model.DeserializeFromString(content);  
// the content should be an XML string saved during the serialization  
process.
```

De-serialize from XML File

The following code illustrates how to de-serialize from an XML file.

C#

```
this.dataGrid.Model.Deserialize("sample.xml");  
// sample.xml file should be the XML file saved during the serialization  
process.
```

Events in WPF GridDataControl (Classic)

The GridData control declares a number of events that it can handle, in response to the activities either by the end user or by the system. An event is a message that is handled, to notify an object or a class of the occurrence of an action. When an event is handled, all the event handlers are notified.

The GridData control offers technical benefits by declaring all its events as Routed Events. Hence, being the high level visual element in the visual tree, it need not hook the same event on all of its descendants (e.g. rows, columns and cells), such as MouseMove. Instead, it hooks the event on itself and hence, when the mouse moves over one of its descendants, the grid can be notified appropriately, whenever the event is handled without expecting its descendants to notify it.

Subscribing to Events

In order to get event notifications, the grid needs to be wired up with the required events. This process is called as subscribing to the events. It can be done using either XAML or C# code.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"  
ItemsSource="{StaticResource ordersSource}"  
MouseMove="dataGrid_MouseMove">
```

C#

```
this.dataGrid.MouseMove+=new MouseEventHandler(dataGrid_MouseMove);
```

For either of the above code languages, you should have the following C# code to handle the MouseMove event.

C#

```
private void dataGrid_MouseMove(object sender, MouseEventArgs e)
{
    Console.WriteLine("Mouse cursor Postion:" + e.GetPosition(sender as
    IInputElement));
}
```

Note: These Grid WPF mouse events are more advantageous than using any other default mouse events because the default mouse events are controlled by Mouse Controller that makes it very hard to access the underlying data; whereas in case of Grid mouse events, it is directly possible to access the underlying data easily.

[Unsubscribe the events](#)

If you do not want the grid to listen to the event, you can unwire the event from the grid as follows.

C#

```
this.dataGrid.MouseMove-=new MouseEventHandler(dataGrid_MouseMove);
```

GridData Control Events

Event	Description
CurrentCellActivating	This event is handled before the grid activates the specified cell as current cell.
CurrentCellActivated	This event is handled after the grid activates the specified cell as current cell.
CurrentCellActivateFailed	This event is handled after the grid fails to activate a specific cell as current cell.
CurrentCellDeactivating	This event is handled before the grid deactivates the current cell.
CurrentCellDeactivated	This event is handled after the grid deactivates current cell.
CurrentCellDeactivateFailed	This event is handled after the grid fails to deactivate the current cell.
CurrentCellConfirmChangesFailed	This event is handled when the grid fails to save the changes made to the active current cell.
CurrentCellAcceptedChanges	This event is handled when the grid accepts the changes made to the active current cell.

CurrentCellChanging	This event is handled when the user wants to modify contents of the current cell.
CurrentCellStartEditing	This event is handled before the current cell switches to the editing mode.
CurrentCellEditingComplete	This event is handled when the grid completes editing the active current cell.
CurrentCellRejectedChanges	This event is handled when the grid rejects the changes made to the active current cell.
CurrentCellChanged	This event is handled when the user changes the contents of the current cell.
CurrentCellMoved	This event is handled when the current cell is successfully moved to a new position.
CurrentCellMoveFailed	This event is handled when the current cell fails to move to a new position.
CurrentCellMoving	This event is handled when the current cell is about to be moved to a new position.
CurrentCellValidating	This event is handled when the grid is validating the contents of the active current cell.
CurrentCellValidated	This event is handled when the grid has completed validating the contents of the active current cell.
CellButtonClick	This event is handled when the button in the grid cell is clicked.
DropDownSelectionChanged	This event is handled when the selected item in the grid drop-down is changed.
CellClick	This event is handled when the grid cell is clicked.
CellCursor	This event is handled when the cell has a cursor.
CellMouseHoverEnter	This event is handled when the mouse hits the cell and signals that the cell wants to handle the mouse events. This event is handled before the CellMouseHover event.
CellMouseHover	This event is handled when the mouse is moved over a grid cell.
CellMouseHoverLeave	This event is handled when the mouse leaves a cell.
CellMouseDown	This event is handled when the mouse button is pressed.
CellMouseMove	This event is handled when the mouse is moved over the cell.
CellMouseUp	This event is handled when the mouse button is released.
CellCancelMode	This event is handled when any current user interaction is canceled, example-Cancel select cells when Escape key is pressed.

CellRestoreMode	This event is handled when a cell is trying to restore the canceled changes.
ResizingColumns	This event is handled when the user is about to resize a column or is in the process of re-sizing a column.
QueryAllowDragColumn	This event is handled when the user hovers over a column header or drags a column header using mouse.
ResizingRows	This event is handled when the user is about to resize a row or is in the process of re-sizing a row.
RowValueCommitting	This event triggers before the RowValue commit. This event works only when the UpdateMode is RowCacheMode.
RowValueCommitted	This event triggers after the RowValue commit. This event works only when the UpdateMode is RowCacheMode.
RowValueCommittingCancelled	If the RowValue committing is canceled in the RowValueCommitting event, then the RowValueCommittingCancelled event is fired. This event works only when the UpdateMode is RowCacheMode.
RowValidating	This event triggers when the focus moves from the current row to any other row.

Performance in WPF GridDataControl (Classic)

Essential Grid has a high performance standard, where you can make the grid to work with large amounts of data with few property settings, without a performance hit. It provides complete support for Virtual Mode, where the data is loaded only on demand and thus saves the memory consumption and provides quick response. It also handles very high frequency updates and refresh scenarios. The following topic discusses this:

High Frequency Updates

This section illustrates an example that let you know how to handle high frequency updates using grid while keeping the CPU usage at a minimum level. It uses a simple data source with few double-valued columns. It changes random cell values and also does record insertions and removals, using a timer event.

The following code illustrates changing of random cell values:

C#

```
private void timer_Tick(object sender, EventArgs e)
{
    .....
    for (int i = 0; i < numberOfChangesEachTimer; i++)
    {
        int recNum = rand.Next(table.Rows.Count - 1);
        int col = rand.Next(table.Columns.Count - 1) + 1;
        DataRow drow = table.Rows[recNum];
        if (drow.RowState != DataRowState.Deleted && !(drow[col] is DBNull))
        {

```



```
double value = (int) (Convert.ToDouble(drow[col]) * (rand.Next(50) / 100.0f + 0.8));
//Console.WriteLine("{0}, {1}: {2}", recNum, col, value);
drow[col] = value;
}
}
}
```

The following code illustrates record insertions and removals:

C#

```
private void timer_Tick(object sender, EventArgs e)
{
    .....
    if (toggleInsertRemove > 0 && (timerCount % insertRemoveModulus) == 0)
    {
        icount = ++icount % (toggleInsertRemove * 2);
        shouldInsert = icount < toggleInsertRemove;
        if (shouldInsert)
        {
            for (int ri = 0; ri < insertRemoveCount; ri++)
            {
                int recNum = rand.Next(Math.Min(30, table.Rows.Count));
                double next = rand.Next(100);
                object[] values = new object[table.Columns.Count];
                values[0] = "H" + ti.ToString("00000");
                for (int n = 1; n < table.Columns.Count; n++)
                {
                    values[n] = next + n;
                }
                DataRow drow = table.NewRow();
                drow.ItemArray = values;
                table.Rows.InsertAt(drow, recNum);
                ti++;
            }
        }
        else
        {
            for (int ri = 0; ri < insertRemoveCount; ri++)
            {
                int recNum = 5; //rand.Next(m_set.Count - 1);
                int rowNum = recNum + 1;
                // Underlying data structure (this could be a data table or whatever
                // structure you use behind a virtual grid).
                if (table.Rows.Count > 10)
                {
                    table.Rows.RemoveAt(recNum);
                }
            }
        }
    }
}
```

Note: For complete code of this example, refer the following browser sample: ...\\My Documents\\Syncfusion\\EssentialStudio\\<Version Number>\\WPF\\Grid.WPF\\Samples\\3.5\\WindowsSamples\\Grid Data Control – Advanced\\Trader Grid Test Demo

PLINQ Support in GridDataControl

[PLINQ](#) is the parallel implementation of the standard LINQ. GridDataControl uses a QueryableCollectionView that works on top of LINQ expressions for performing major operations such as Sorting, Filtering, Grouping and Summaries calculation. Since PLINQ works on top of LINQ expression trees, QueryableCollectionView now has a property UsePLINQ = true / false (this class implements *IParallelizable View* interface) that would add a AsParallel() to change it into a Parallel Query. Sorting, Grouping and Summary operations would be automatically done in parallel when this property is set.

XML

```
<syncfusion:GridDataControl x:Name="grid"
AutoPopulateColumns="True"
AutoPopulateRelations="False"
UsePLINQ="True"
VisualStyle="Officel4Silver">
</syncfusion:GridDataControl>
```

C#

```
this.gridDataControl1.UsePLINQ = true;
```

Note: This only works for strongly-typed collections and not for legacy object models like DataTable.

Features-that-work-in-PLINQ in WPF GridDataControl (Classic)

The following features work by default when UsePLINQ is set to true:

- Sorting
- Grouping and
- Summaries

Note: For custom summaries, PLINQ code has to be implemented at the client implementation.

Real-Time-Application in WPF GridDataControl (Classic)

Essential Grid finds a wide range of applications in real time. It is completely optimized to deal with large amount of real time data. Its virtual mode support and high frequency updates behavior make the grid to take part in stock portfolio applications.

Portfolio Grid

Essential Grid is the proper choice to use in portfolio applications, as it deals with both huge and real time data without a performance hit. This section illustrates how to employ the grid in portfolio applications.

This example displays three GridData controls (GDCs) and three Chart controls hosted in a Grid control using its CellTypes feature. The three GDCs display stock details in three different views – Overview, Sector Industry View and Stock Exchange View. Among the charts, one illustrates the performance of large portfolio account, and the other two illustrates the individual country contributions to specific portfolio accounts. You can click any portfolio account in the PortfolioAccounts chart to drill down to its contributions.

The GDCs are extremely customized with appropriate groups and summaries in order to provide the desired view. It also highlights the change of stock values. Value increase is indicated by green foreground and decrease in values is indicated by red foreground.

The following image shows the Essential Grid being used in a portfolio application:



Note: For complete code of this example, refer to the following browser sample:

...\\My Documents\\Syncfusion\\EssentialStudio\\<Version Number>\\WPF\\Grid.WPF\\Samples\\3.5\\WindowsSamples\\Product Showcase\\Stock Dashboard Demo

VS2010-Designer-support in WPF GridDataControl (Classic)

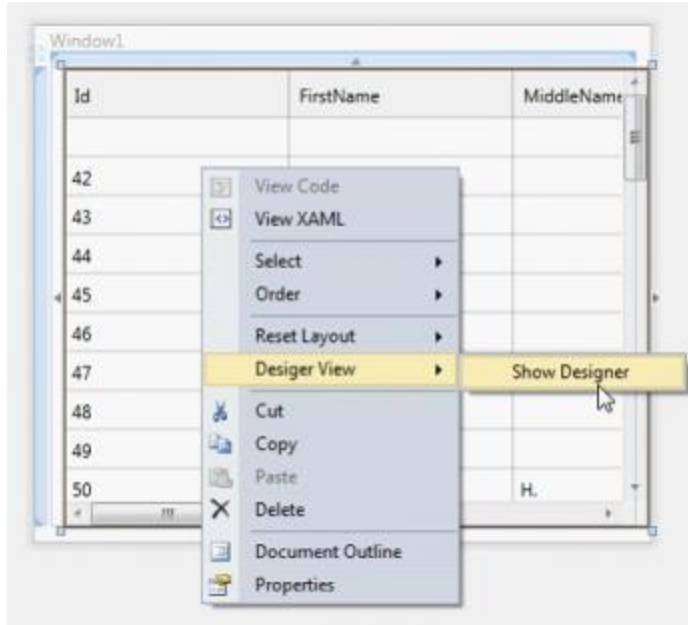
GridDataControl provides rich design time experience by associating a designer. This allows the users to modify the various grid settings to change the look and feel of the grid.

The grid designer is populated with numerous options when ItemsSource assigned to the grid. This enables the users to edit the basic grid properties and the properties of individual column. Changes in any of these properties in the designer have an immediate impact on its XAML code and hence the designer makes the grid more user-friendly.

Activating Designer

1. Open Design window.
2. Right-click on the grid.

Note: An edit menu opens.



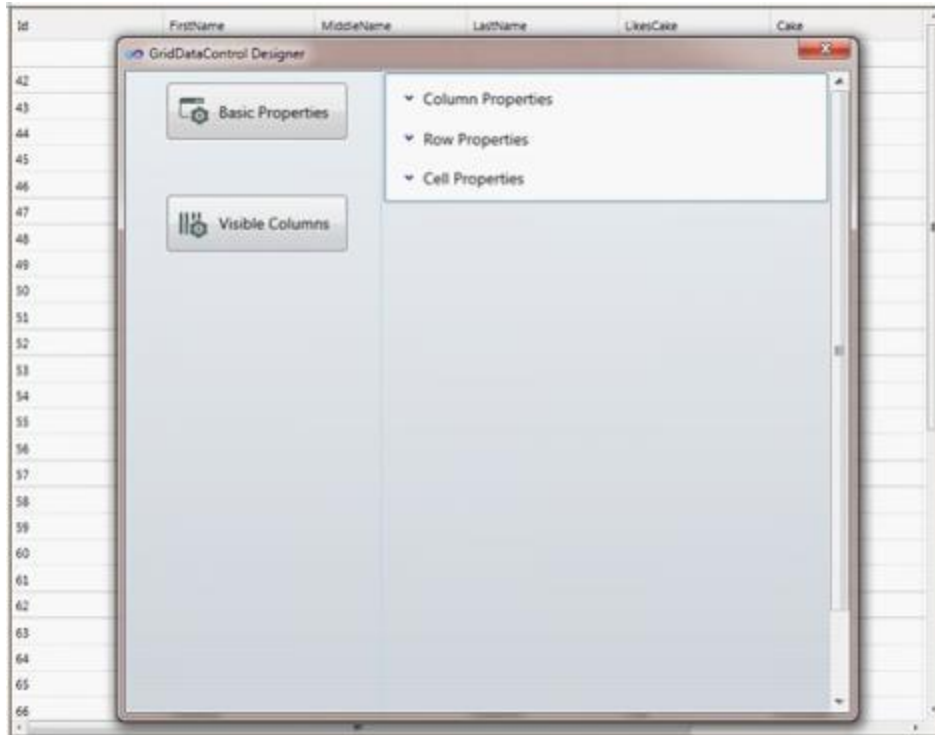
3. Select Designer View -> Show Designer.

Note: Ensure that the grid is assigned with an ItemsSource.

4. Designer Window is displayed.

Designer Window has two options:

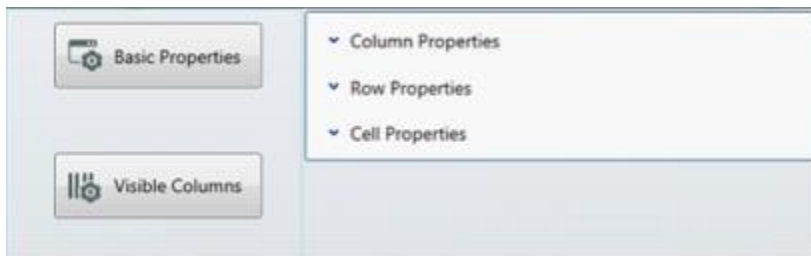
- Basic Properties - Modify the overall settings of the grid.
- Visible Columns - Automatically generates a property list for each visible column in the grid.



Basic Properties

This option enables the users to modify the overall settings of the grid. The properties are categorized into three types:

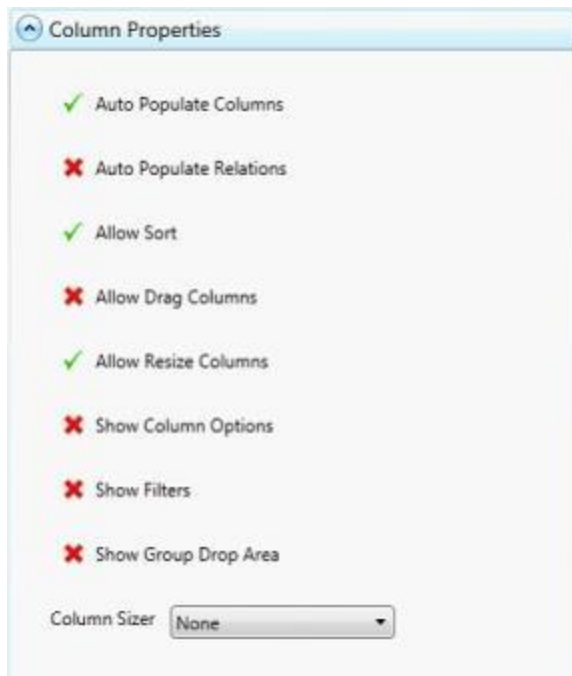
- Column Properties
- Row Properties
- Cell Properties



Column Properties

This section explores the various column options such as Auto Populate Columns, Auto Populate Relations, Allow Sort, Allow Drag Columns, Allow Resize Columns, Show Column Options, Show Filters, Show Group Drop Area and Column Sizer combo box.

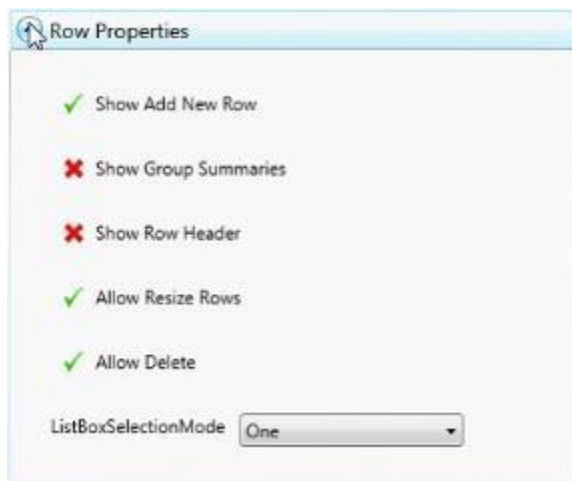
1. Select as you require in this list.



Row Properties

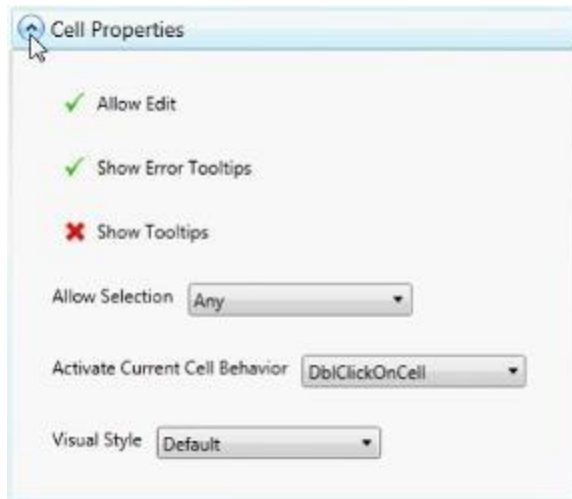
This section explores the row-related properties such as Show Add New, Show Group Summaries, Show Row Header, Allow Resize Rows, Allow Delete and List Box Selection Modes combo box.

Select as you require in this list.



Cell Properties

This section explores cell level properties such as Allow Edit, Show Error Tooltips, Show Tooltips, Allow Selection combo box, Activate Current Cell Behavior combo box and Visual Style combo box.



Select as you require in this list.

Visible Columns

This section automatically generates a property list for each visible column in the grid. Each list includes the column level properties such as Allow Filter, Allow Sort, Allow Drag, Allow Group, Allow Resize, Is Read Only, Auto fit, Width, Header Text field, Column Format combo box and Cell Type combo box.

- Click Visible Columns.
- Two options are displayed

Generate Columns

Clear Columns



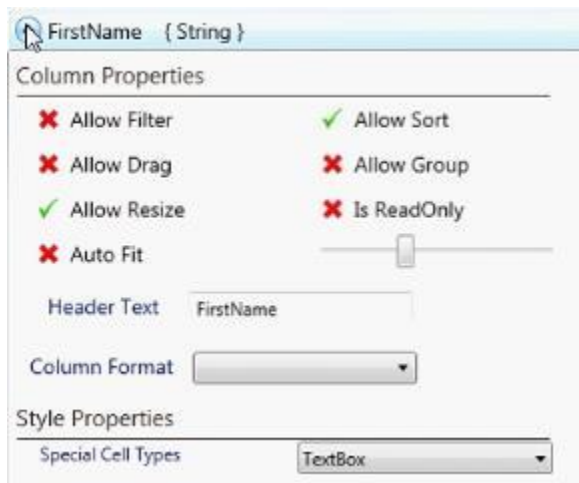
Generate Columns

1. Click Generate Columns to populate the properties for each visible column.

Note: Property list for visible column in the grid displays.

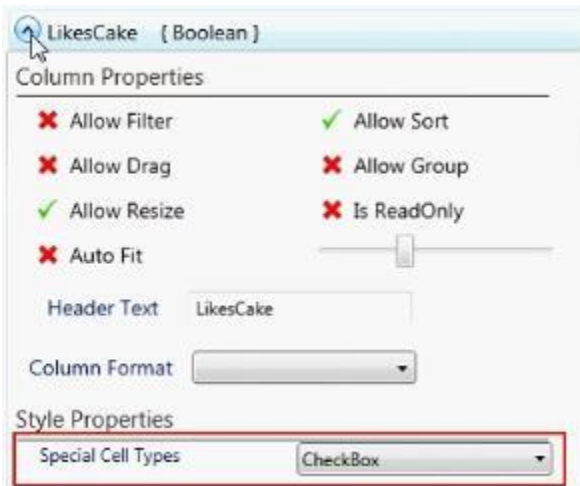


2. Select as you require.



Special Cell Types

This combo box lists the various possible cell types applicable to the column. It also automatically deduces the column type of the grid columns and sets the CellType of it. For example, if the column is of Boolean type, it automatically has a CheckBox.



Clear Columns

Click Clear Columns to clear the visible column settings.

Freezable Support in WPF GridDataControl (Classic)

GridDataControl now supports inheritance context using Freezable. Developers can use this feature to bind values to elements not present in the Visual Tree. Usually, the ElementName and DataContext bindings are resolved based on the target dependency object's position within the element tree (or the name scope to which the target dependency object belongs). But in case, if the target dependency object is not in the tree such as HeaderText in GridDataVisibleColumn, then having an inheritance context for DependencyObjects external to an element tree is the solution. We can achieve this by using Freezable.

The reason the Freezable trick works is because a Freezable object has its own notion of "inheritance context". When the property engine sets the effective value of a dependency property, it looks at that new value to determine whether it is a dependency object that would like to be part of a special inheritance tree. A Freezable is one such object that always wants to be in the inheritance tree when not frozen.

XML

```
<TextBox Text="Test" x:Name="txtBlock" />
A Separate TextBlock Text can be bound to GridDataVisibleColumn using the
following code.
<syncfusion:GridDataControl AutoPopulateColumns="False" x:Name="dataGrid"
ItemsSource="{Binding}">
  <syncfusion:GridDataControl.VisibleColumns >
    <syncfusion:GridDataVisibleColumn MappingName="Status" Width="200"
HeaderText="{Binding Text, ElementName=txtBlock}" />
  </syncfusion:GridDataControl.VisibleColumns>
</syncfusion:GridDataControl>
```

Grid-Localization-Support in WPF GridDataControl (Classic)

Localization is the process of making your application multi-lingual, by formatting content according to cultures. This involves configuring the application for a specific language. Culture is the combination of language and the location (e.g. En-US is the culture for English spoken in United States; En-GB is the culture for English spoken in Great Britain). Syncfusion Grid allows you to set custom resource through

the Resx file. The user can simply give the string values in the resource file for a specific culture and set the culture in the application. The given string values are set to the Grid that does not affect the Code Block of the Grid

Adding Localization to an Application

The following steps explain the implementation of Localization support in applications.

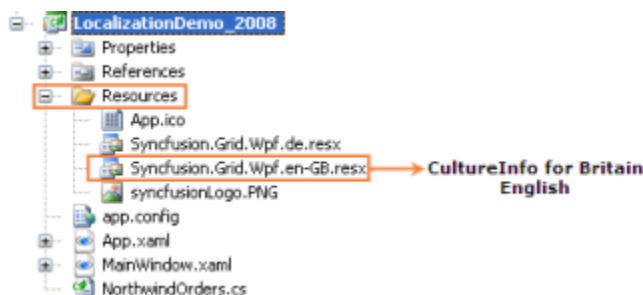
Creating an Application

Create a WPF application and add GridDataControl to it.

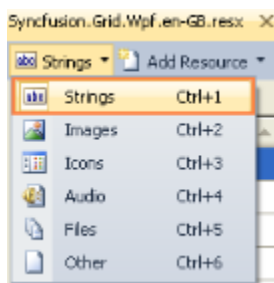
Creating a Resource File

To create a Resource file:

1. Create a folder named “Resources” in the application.
2. Create a resource file (Resx file) and name it “Syncfusion.Grid.Wpf..resx” E.g. Syncfusion.Grid.Wpf.en-GB.resx.
3. Use the prescribed naming convention as it is mandatory.
4. The following screenshot explains the addition of a Resource File to the application.

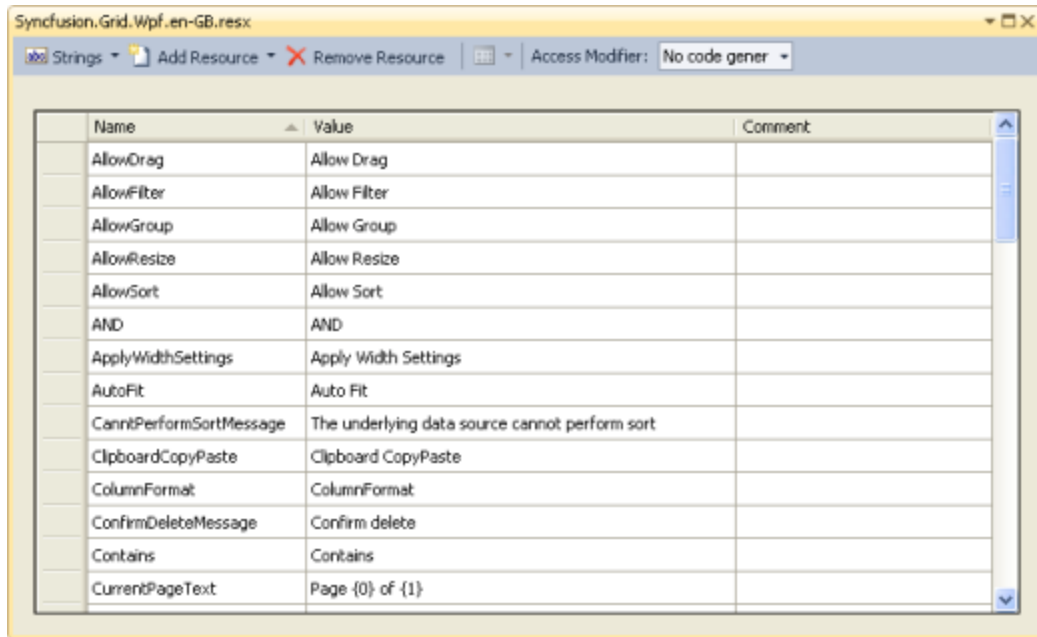


5. Select the String option in the Resource file. This is explained in the following screenshot.



6. Enter the “Name” and “Value” in the Resource file.

The String Property names used in the Grid are given in the Property table. This is explained in the following screenshot.



Setting the Culture Information in the Application

The culture information should be set in the application before the `InitializeComponent()` method is called. Now, the application is set to Britain English Culture info. The following code snippet explains the implementation of this.

C#

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

Properties

Localization Property Table

Property	Description	Type	Data Type
AllowDrag	Sets the string for AllowDrag property	static	string
AllowDrag	Sets the string for AllowDrag	static	string
AllowGroup	Sets the string for AllowGroup	static	string
AllowResize	Sets the string for AllowResize	static	string
AllowSort	Sets the string for AllowSort	static	string
AND	Sets the string for AND	static	string
ApplyWidthSettings	Sets the string for ApplyWidthSettings	static	string
AutoFit	Sets the string for AutoFit	static	string

CanntPerformSortMessage	Sets the string for CanntPerformSortMessage	static	string
ClipboardCopyPaste	Sets the string for ClipboardCopyPaste	static	string
ColumnFormat	Sets the string for ColumnFormat	static	string
ConfirmDeleteMessage	Sets the string for ConfirmDeleteMessage	static	string
Contains	Sets the string for Contains	static	string
CurrentPageText	Sets the string for CurrentPageText	static	string
DeleteMessage	Sets the string for DeleteMessage	static	string
DragDropText	Sets the string for DragDropText	static	string
DynamicOptions	Sets the string for DynamicOptions	static	string
EndsWith	Sets the string for EndsWith	static	string
EnterFilterValue	Sets the string for EnterFilterValue	static	string
Equals	Sets the string for Equals	static	string
First	Sets the string for First	static	string
GreaterThan	Sets the string for GreaterThan	static	string
GreaterThanOrEqual	Sets the string for GreaterThanOrEqual	static	string
HCenter	Sets the string for HCenter	static	string
HeaderText	Sets the string for HeaderText	static	string
HLeft	Sets the string for HLeft	static	string
Horizontal	Sets the string for Horizontal	static	string
HRight	Sets the string for HRight	static	string
HStretch	Sets the string for HStretch	static	string
InvalidColumn	Sets the string for InvalidColumn	static	string
InvalidDateTime	Sets the string for InvalidDateTime	static	string
InvalidDataToFilter	Sets the string for InvalidDataToFilter	static	string
IsReadOnly	Sets the string for IsReadOnly	static	string
Last	Sets the string for Last	static	string
LessThan	Sets the string for LessThan	static	string
LessThanOrEqual	Sets the string for LessThanOrEqual	static	string
MatchCase	Sets the string for MatchCase	static	string

Next	Sets the string for Next	static	string
NoMoreItemRemoveMessage	Sets the string for NoMoreItemRemoveMessage	static	string
None	Sets the string for None	static	string
NoRecordsfound	Sets the string for NoRecordsfound	static	string
NotEnoughSpaceMessage	Sets the string for NotEnoughSpaceMessage	static	string
NotEquals	Sets the string for NotEquals	static	string
NotSupportDeletingItemMessage	Sets the string for NotSupportDeletingItemMessage	static	string
OR	Sets the string for OR	static	string
PageSizes	Sets the string for PageSizes	static	string
Previous	Sets the string for Previous	static	string
PrintOutputColor	Sets the string for PrintOutputColor	static	string
PrintText	Sets the string for PrintText	static	string
PrintZoom	Sets the string for PrintZoom	static	string
SelectAllFilter	Sets the string for SelectAllFilter	static	string
StartsWith	Sets the string for StartsWith	static	string
TextAlignment	Sets the string for TextAlignment	static	string
VBottom	Sets the string for VBottom	static	string
VCenter	Sets the string for VCenter	static	string
Vertical	Sets the string for Vertical	static	string
VStretch	Sets the string for VStretch	static	string
VTop	Sets the string for VTop	static	string
Width	Sets the string for Width	static	string
WidthOptions	Sets the string for WidthOptions	static	string

[Sample Link](#)

Refer to the sample in the shipped Sample Browser:

Essential Studio - WPF Sample Browser - Grid - LocalizationSupport - LocalizationDemo.

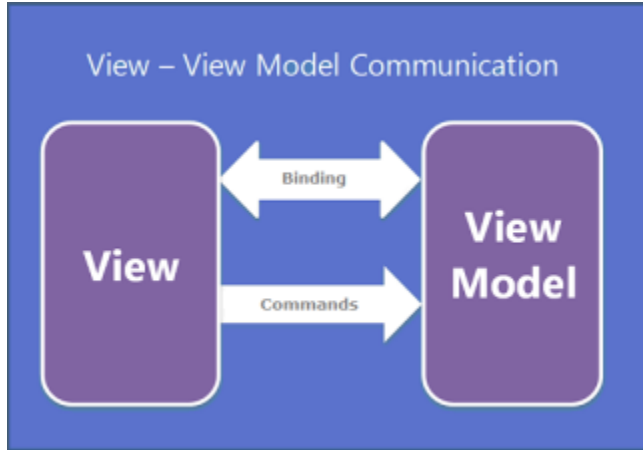
MVVM-Enhancements in WPF GridDataControl (Classic)

View – View Model Communication

In MVVM, commands are used to communicate between the View and View Model when a particular action takes place in the View. The GridData control contains events for all actions. In some cases, you

may have to use events to meet requirements that do not adhere to the MVVM policy; you can overcome this with the EventToCommand approach.

To support the EventToCommand approach, commands have been added for all events in the GridData control and tables to provide complete MVVM support.



GridData control has two commands for each event. The first one passes the event argument to the command as parameter, and the second one does not pass any parameter to the command. This apart, the command parameter can also be changed at Sample level.

Adding Commands to a GridData Control

You can add commands to a GridData control in the following three ways:

- By using Command with actual event arguments
- By using Command with custom parameter
- By overriding existing command behavior

By using a Command with Actual Event Arguments

This section explains how to add GridDataControlRecordsSelectionChangedCommandWithEventArgs command to the GridData control. The actual event arguments are passed to the Command method as parameters.

The following code example illustrates how to define the GridDataControlRecordsSelectionChangedCommandWithEventArgs command in XAML.

XML

```
<syncfusion:GridDataControl x:Name="dataGrid" AutoPopulateColumns="True"
AutoPopulateRelations="False"
ItemsSource="{Binding GDCSource}" ShowAddNewRow="False"
syncmvvm:GridDataControlRecordsSelectionChangedCommandWithEventArgs.Command=
"{Binding SelectedItemChanged}">
```

The following code example illustrates binding GridDataControlRecordsSelectionChangedCommandWithEventArgs defined in the View.

C#

```
private BaseCommand selectedItemChanged;
```

```

public BaseCommand SelectedItemChanged
{
    get
    {
        if (selectedItemChanged == null)
            selectedItemChanged = new BaseCommand(SelectedItemChangedMethod);
        return selectedItemChanged;
    }
}

void SelectedItemChangedMethod(object parameter)
{
    var item = parameter as GridDataRecordsSelectionChangedEventArgs;
    if (item != null)
    {
        var data = item.AddedItems[0] as Data;
        this.SelectedCustomerID = "Customer ID : " + data.CustomerID;
    }
}

```

When you select a record while running your application, the SelectedItemChanged command is triggered with the actual GridDataRecordsSelectionChangedEventArgs event argument.



Sample Location

A sample application can be downloaded from the following location:

<http://www.syncfusion.com/downloads/Support/DirectTrac/95643/MVVMWithActualEventArgs1100780673.zip>

By using a Command with a Custom Parameter

This section illustrates how to add the GridDataControlRecordsSelectionChangedCommand command to the GridData control and pass the GridData control as customer parameter.

The following code example can be used to define GridDataControlRecordsSelectionChangedCommand in XAML.

XML

```

<syncfusion:GridDataControl x:Name="dataGrid" Grid.Row="0"
AutoPopulateColumns="False" ItemsSource="{Binding GDCSource}"
syncmvvm:GridDataControlRecordsSelectionChangedCommand.Command="{Binding SelectedItemChanged}"
syncmvvm:GridDataControlRecordsSelectionChangedCommand.CommandParameter="{
Binding ElementName=dataGrid}">

```

The following code example is used for binding GridDataControlRecordsSelectionChangedCommand defined in the view.

C#

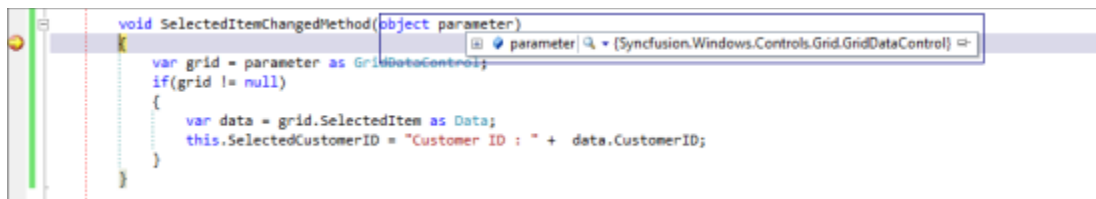
```

private BaseCommand selectedItemChanged;
public BaseCommand SelectedItemChanged
{
    get
    {
        if (selectedItemChanged == null)
            selectedItemChanged = new BaseCommand(SelectedItemChangedMethod);
        return selectedItemChanged;
    }
}

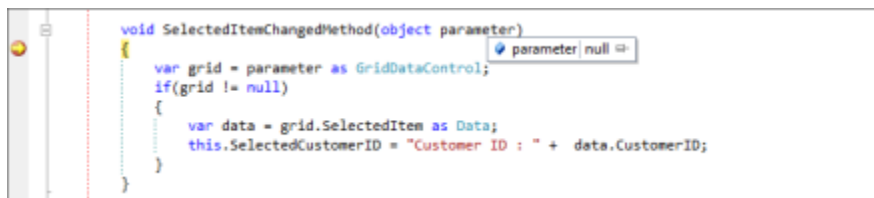
void SelectedItemChangedMethod(object parameter)
{
    var grid = parameter as GridDataControl;
    if (grid != null)
    {
        var data = grid.SelectedItem as Data;
        this.SelectedCustomerID = "Customer ID : " + data.CustomerID;
    }
}

```

When you select a record while running your application, the SelectedItemChanged command gets triggered with the custom GridDataControl parameter.



If there is no parameter set in the View, then the parameter is passed in the method call.

**Sample Location**

A sample application can be downloaded from the following location:

<http://www.syncfusion.com/downloads/Support/DirectTrac/95643/MVVMWithCustomParameter1792996222.zip>

By Overriding the Command Behavior

Another approach is to override a command's behavior with a custom parameter. This section explains how to override the GridDataControlCellMouseMoveCommandBehavior and return the record (i.e. return the record on which the pointer rests).

First, you need to create a class and override it from the GridDataControlCellMouseMoveCommandBehavior as shown in the following code example.

C#


```

public class MyGridDataControlMouseMoveBehavior : GridDataControlCellMouseMoveCommandBehavior<Data>
{
    public MyGridDataControlMouseMoveBehavior(): base((o, e) =>
    {
        var grid = o as GridDataControl;
        RowColumnIndex rowColumnIndex =
        grid.Model.Grid.PointToCellRowIndexOutsideCells(Mouse.GetPosition(grid
        .Model.Grid), true);
        Debug.WriteLine("Index is {0}", rowColumnIndex.RowIndex);
        var data = grid.ItemsSource as ObservableCollection<Data>;
        int index = rowColumnIndex.RowIndex - grid.Model.HeaderRows - 2;
        if (index >= 0)
        {
            var record = data[index];
            return record;
        }
        else
            return null;
    })
}

public class MyGridDataControlMouseMoveCommand : GridDataControlCellMouseMoveCommand<Data, MyGridDataControlMouseMoveBehavior>
{
}

```

Now, bind the behavior to the GridData control. The following code example illustrates this.

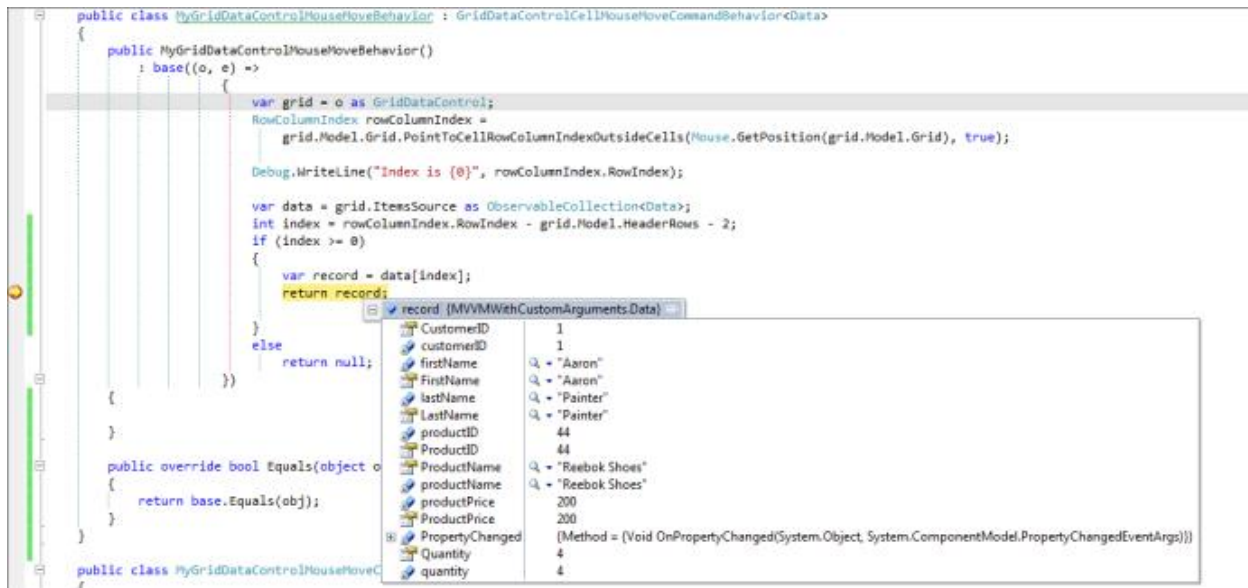
XML

```

<syncfusion:GridDataControl x:Name="dataGrid" Grid.Row="0"
AutoPopulateColumns="True" AutoPopulateRelations="False"
ItemsSource="{Binding GDCSource}"
Utils:MyGridDataControlMouseMoveCommand.Command="{Binding MouseMoveCommand}"
VisualStyle="Officel4Blue" />

```

When you hover the mouse over a row while running your application, the overridden behavior class triggers and returns the current record.



Sample Location

A sample application can be downloaded from the following location:

<http://www.syncfusion.com/downloads/Support/DirectTrac/95643/MVVMWithCustomArguments1965076929.zip>

GridTreeControl (Classic)

WPF GridTreeControl (Classic) 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.

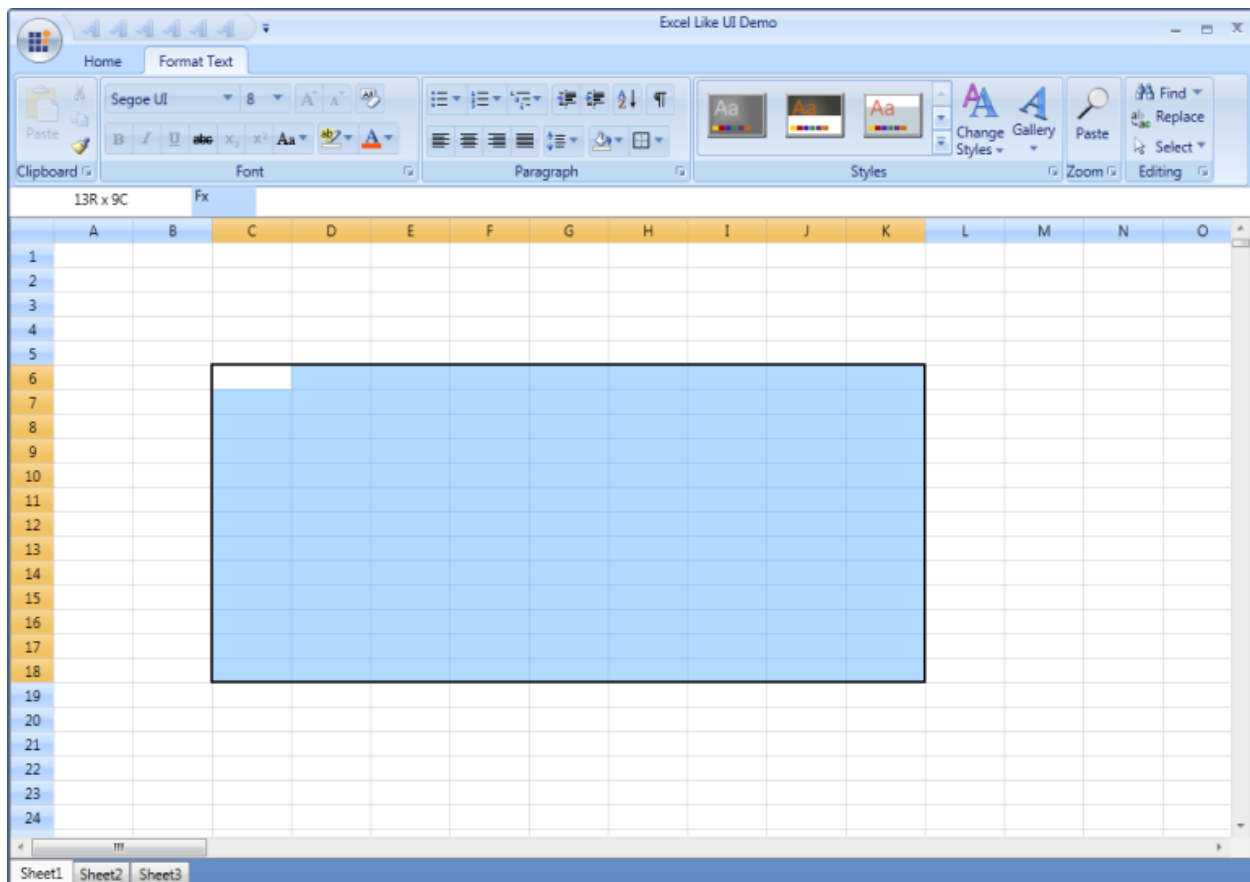
The display system also hosts a powerful and complete styles architecture. Settings can be specified at the cell level or higher levels using parent styles that are referred to as base styles. Base styles can affect groups of cells. Cell level settings override any higher-level settings and enable easy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

Use Case Scenarios

EssentialGrid for WPF can be applied to a variety of industries such as finance, banking, software, etc. Some of its important features are:

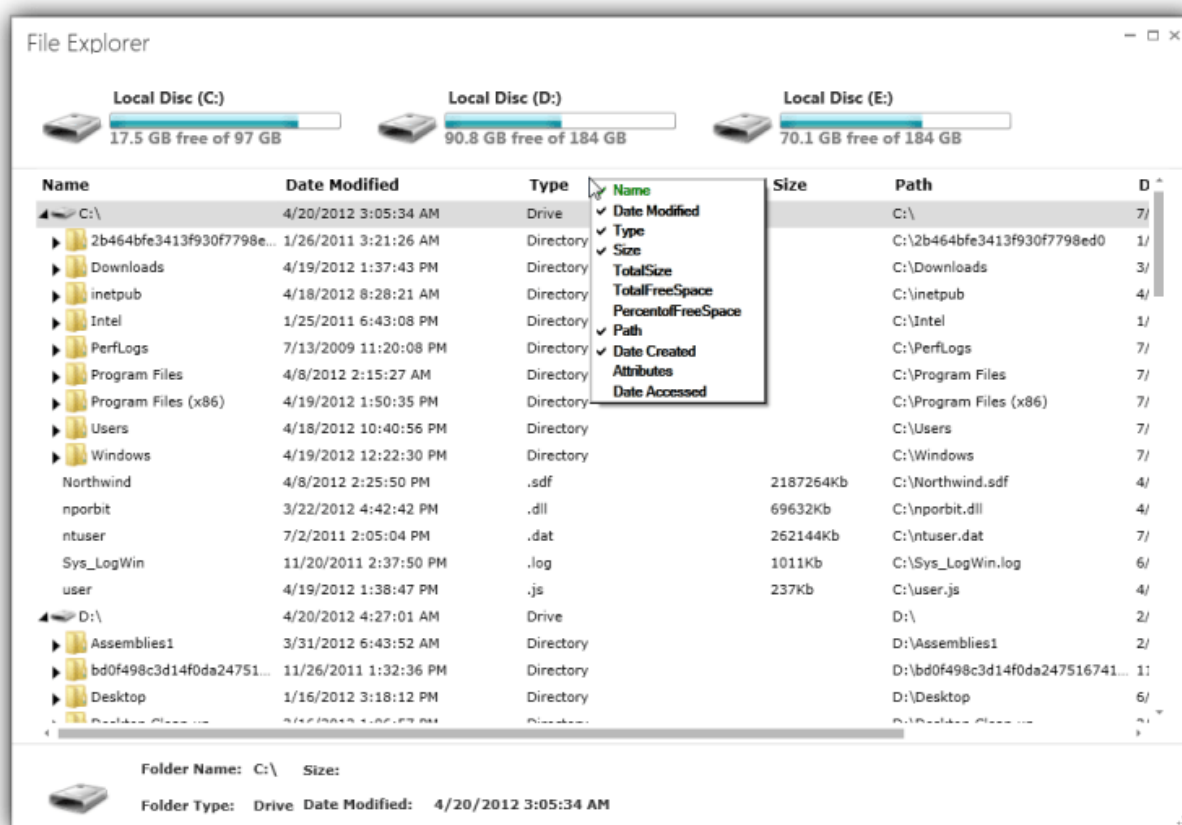
Excel-Like UI: Essential Grid's rich feature set allows you to build Excel-like UI applications.



High Performance - EssentialGrid is a great asset to high-performance applications, as it can display large amounts of real-time data that tends to periodic changes without any performance hits. Below is an illustration of a stock portfolio application using the GridData control.

Quotes Symbol ▲		Account ID ▲					
Holding ID	Quotes Symbol ▲	Quantity	Price Paid	Purchase Date	Account ID ▲	Stock Exchange ID ▲	
▲ Total Quantity		Qty - (421)	Price - (3.550)				
▲ Total Quantity		Qty - (1)	Price - (.830)				
6463	AAME	1	Rs. 0.83	23-12-2008	1	2	
▲ Total Quantity		Qty - (193)	Price - (.950)				
7293	AAME	193	Rs. 0.95	23-12-2008	5	4	
▲ Total Quantity		Qty - (227)	Price - (1.770)				
7415	AAME	110	Rs. 0.80	23-12-2008	6	7	
7450	AAME	117	Rs. 0.97	23-12-2008	6	6	
▲ Total Quantity		Qty - (397)	Price - (66.240)				
▲ Total Quantity		Qty - (397)	Price - (66.240)				
7656	AAON	34	Rs. 16.56	09-01-2009	7	7	
7657	AAON	121	Rs. 16.56	11-02-2009	7	7	
7658	AAON	121	Rs. 16.56	11-02-2009	7	7	
7659	AAON	121	Rs. 16.56	11-02-2009	7	7	
▶ Total Quantity		Qty - (186)	Price - (185.090)				
▶ Total Quantity		Qty - (234)	Price - (19.520)				
▶ Total Quantity		Qty - (279)	Price - (3391.460)				
▶ Total Quantity		Qty - (211)	Price - (86.070)				
▶ Total Quantity		Qty - (242)	Price - (28.000)				

File Explorer - Applications that deal with hierarchical data can make use of Essential Grid's file explorer feature, which allows child items to be displayed on-demand by using the GridTree control.



Key Features

You will find the following features of EssentialGrid for WPF:

- 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.

User Guide Organization

EssentialGrid for WPF comes with numerous samples as well as extensive documentation for your reference. This user guide provides detailed information on features and functionalities. It is organized into the following sections:

- Overview – This section provides a brief introduction to Essential Grid and its key features.
- Getting Started – This section guides you on getting started with a WPF application and WPF controls.
- Concepts and Features – Under this section, the features of individual controls are illustrated with use-case scenarios, code examples, and screen shots.
- Frequently Asked Questions – This section contains answers to frequently asked questions about Essential Grid.

Document Conventions

The conventions below will help you quickly identify important sections of information when using this user guide:

Document Conventions

Convention	Description of the Icon
Note	Represents important information to be noted.
Example	Represents an example.
Tip	Represents useful hints that will help you use the controls and features.
Additional information	Represents additional information on the corresponding topic.

Feature Summary

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

[Overview in WPF GridTreeControl \(Classic\)Control](#)

The GridTree control is a dynamic data-bound control used to present hierarchical data.

[Data Binding](#)

The GridTree control supports all popular data sources including observable collections, data tables, and binding lists.

Data sources can be bound to the GridTree control by either directly binding `ItemsSource` with the relational information or by retrieving node elements dynamically with the `RequestTreeItems` event. `RequestTreeItems` will load the data in an on-demand basis.

For instance, a database may have 100,000 records in it; in the first approach (binding `ItemSource`) you have to set `ItemsSource` of `GridTreeControl`, and it will take care of populating root and child nodes. But in the second approach, you have to dynamically pass the source of the root and child by handling the `RequestTreeItems` event. The child element will load only when the parent node expands.

Data Presentation: Cell Types

Several built-in cell types can be used to display and edit any underlying data type. The following cell types are available:

- Static-text cells.
- Check-box cells.
- Button cells.
- Image cells.
- Combo-box cells.
- Drop-down list cells.
- Currency cells.
- Date-time cells.
- Double edit cells.
- Integer edit cells.
- Mask edit cells.
- Percent edit cells.
- Rich-text-box cells.
- Up-down edit cells.

Title	First Name	Employee ID	Salary	DOJ	Rating	Recent Hike %	Weight(Kg)
Management							
Vice President	Andrew	1001	\$1,200,000.00	1/9/1937	5.00	1,000.00 %	79.45
GM	Janet	1002	\$1,000,000.00			800.00 %	75.25
Manager	Steven	1003	\$900,000.00			700.00 %	80.34
Accounts							
Accounts Manager	Nancy	1004	\$850,000.00			720.00 %	57.23
Accountant	Margaret	1008	\$700,000.00			690.00 %	68.27
Accountant	Michael	1009	\$700,000.00			690.00 %	82.88
Accountant	Robert	1010	\$650,000.00			570.00 %	76.19
Sales							
Sales Manager	Laura	1005	\$900,000.00			820.00 %	68.23
Sales Representative	Anne	1011	\$800,000.00	2/10/1945	4.00	800.00 %	75
Sales Representative	Albert	1012	\$750,000.00	2/10/1945	5.00	690.00 %	87.7
Sales Representative	Tim	1013	\$700,000.00	3/11/1945	5.00	523.00 %	72.35
Sales Representative	Justin	1014	\$700,000.00	3/11/1945	5.00	600.00 %	77.23

Expander Cell TextBox IntegerEdit CurrencyEdit DateTimeEdit UpDownEdit PercentEdit DoubleEdit

Interactive Features

The GridTree control provides an effective option for sorting. Also, its appearance can easily be customized through an API that can be used to set different types of glyphs such as triangles, plus and minus signs, and tree lines in various brush styles. By handling the GlyphDrawing event on the cell renderer, you can draw custom expand-and-collapse glyphs. The GridTree control also supports placing images next to expand-and-collapse glyphs.

Supports Node Image		Different types of Glyph		MultiColumn Sorting	
Name		Date Modified	Type	Path	Date Created
▶ C:\		4/24/2012 4:52:39 AM	Drive	C:\	7/13/2009 10:38:56 PM
▶ 2b464bfe3413f930f7798ed0		1/26/2011 3:21:26 AM	Directory	C:\2b464bfe3413f930f7798ed0	1/26/2011 3:21:26 AM
▶ Downloads		4/23/2012 2:37:05 AM	Directory	C:\Downloads	3/5/2012 12:11:26 PM
▶ inetpub		4/18/2012 8:28:21 AM	Directory	C:\inetpub	4/18/2012 8:28:16 AM
▶ Intel		1/25/2011 6:43:08 PM	Directory	C:\Intel	1/25/2011 6:40:01 PM
▶ Northwind		4/24/2012 2:31:54 AM	.sdf	C:\Northwind.sdf	4/8/2012 12:16:56 PM
▶ nporbit		3/22/2012 4:42:42 PM	.dll	C:\nporbit.dll	4/5/2012 12:35:18 PM
▶ ntuser		7/2/2011 2:05:04 PM	.dat	C:\ntuser.dat	7/2/2011 2:05:03 PM
▶ PerfLogs		7/13/2009 11:20:08 PM	Directory	C:\PerfLogs	7/13/2009 11:20:08 PM
▶ Program Files		4/8/2012 2:15:27 AM	Directory	C:\Program Files	7/13/2009 11:20:08 PM
▶ Program Files (x86)		4/24/2012 4:46:20 AM	Directory	C:\Program Files (x86)	7/13/2009 11:20:08 PM

Serialization

The GridTree control supports XML serialization to preserve and restore a grid's schema and style settings. All styles and properties that reflect the state of the Grid can be serialized.

Getting Started with WPF GridTreeControl (Classic)

This section is designed to help you understand and quickly get started using Essential Grid in your WPF application. Control appearance and structure are defined and the Essential Grid's relevant classes are depicted.

The following sections comprise the Getting Started section:

Appearance and Structure of the Grid

EssentialGrid for WPF is 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 EssentialGrid package is comprised of following three types of grid controls:

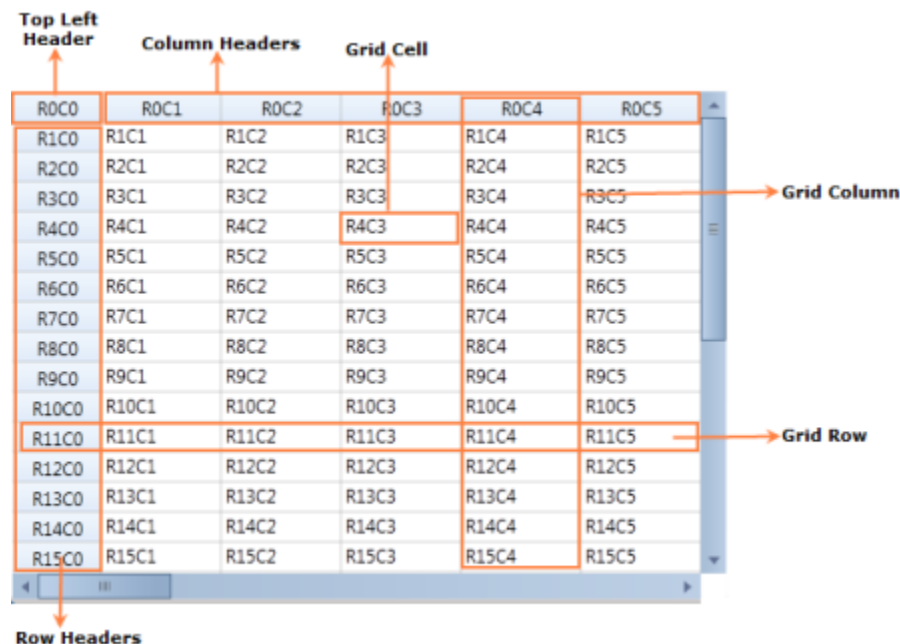
- Grid Control
- GridData Control
- GridTree Control

Now, take closer look at the characteristics of each of these controls.

Grid Control

This is a general-purpose grid that can be used in any form, either holding its own data or virtually bound to an external data source. It acts as a base grid for the other two types of grids (the GridData and GridTree controls). Most features are shared among the three grid types.

In the Grid control, each cell acts as a single entity, which is suitable for applications such as Excel simulator, where the data in the grid cells are not interrelated and need to be maintained in the specific cells themselves. You can also operate this control in virtual mode, where data is not stored in the grid's internal data structure but comes from an external source like a data table, for example. In virtual mode, data will be dynamically loaded into the grid on demand or when users need to view the data.



GridData Control

The GridData control is designed to be bound with a data source. In the GridData control, each column behaves as a single entity. This grid is more column-centric and can be used to display interrelated

tabular data. Unlike the base grid, this grid does not store data values in its data structures; instead, it is connected to an external data source.

For more detailed information about data source connections, refer to the Data Binding section.

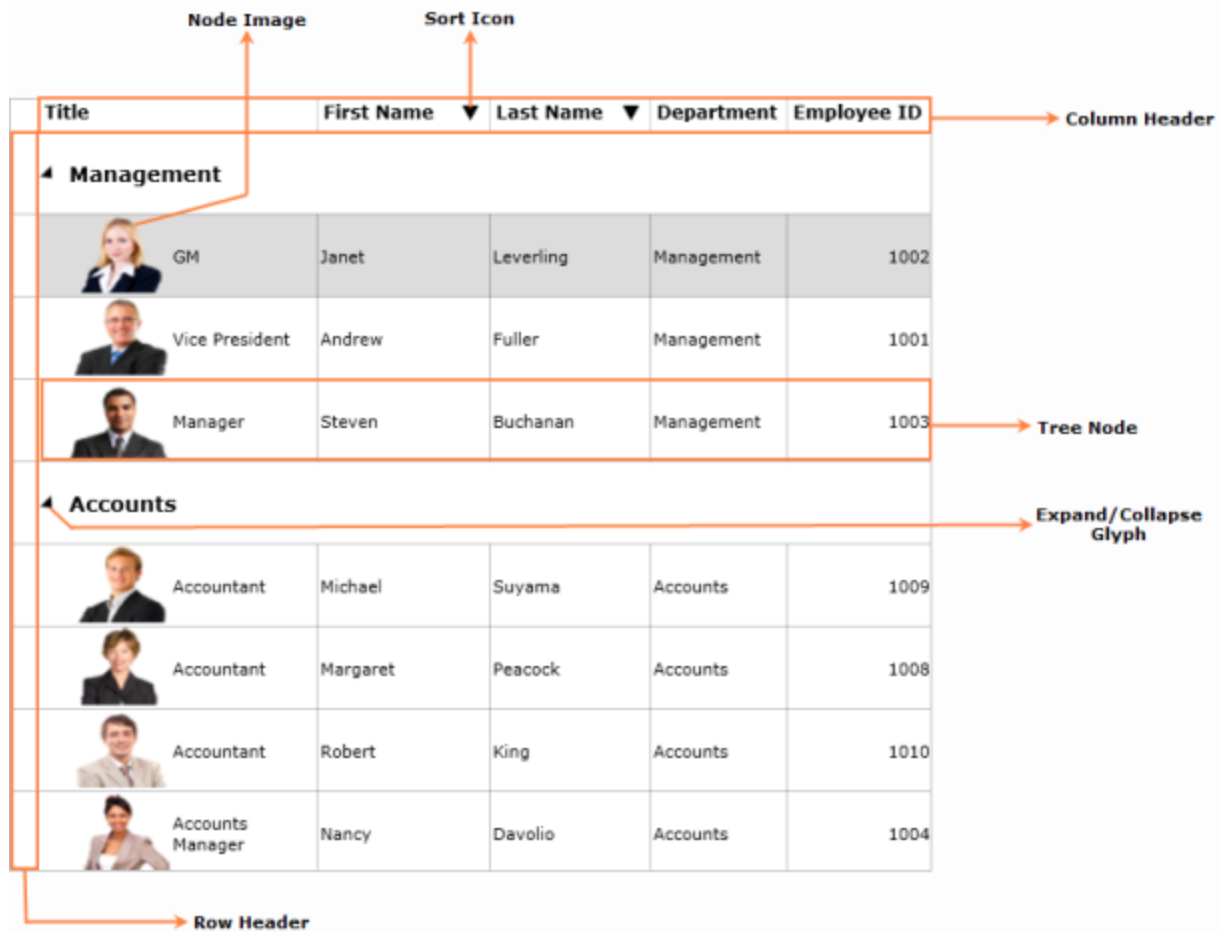
Annotations:

- Grouped by header
- Group Drop Area
- Column Options Icon
- Sort Icon
- Filter Icon
- Group Caption
- Record Row
- Summary Row
- Expander Cell

Order ID	Customer ID	Ship Name	Freight	Order Date
7 Items				
10835	ALFKI	Alfred's Futterkiste	Rs. 69.53	09-12-1993
10952	ALFKI	Alfred's Futterkiste	Rs. 40.42	07-02-1994
11011	ALFKI	Alfred's Futterkiste	Rs. 1.21	03-03-1994
10702	ALFKI	Alfred's Futterkiste	Rs. 23.94	06-09-1993
10062	ALFKI	Alfred's Futterkiste	Rs. 47.22	22-08-1991
10643	ALFKI	Alfred's Futterkiste	Rs. 29.46	19-07-1993
10692	ALFKI	Alfred's Futterkiste	Rs. 61.02	27-08-1993
Total - 7 Items			273.00	
4 Items				
10759	ANATR	Ana Trujillo Emparedados y...	Rs. 11.99	22-10-1993
10926	ANATR	Ana Trujillo Emparedados y...	Rs. 39.92	26-01-1994
10308	ANATR	Ana Trujillo Emparedados y...	Rs. 1.61	12-08-1992
10625	ANATR	Ana Trujillo Emparedados y...	Rs. 43.90	02-07-1993
Total - 4 Items			97.00	
13 Items				
10573	ANTON	Antonio Moreno Taqueria	Rs. 84.84	13-05-1993
10535	ANTON	Antonio Moreno Taqueria	Rs. 15.64	06-04-1993

GridTree Control

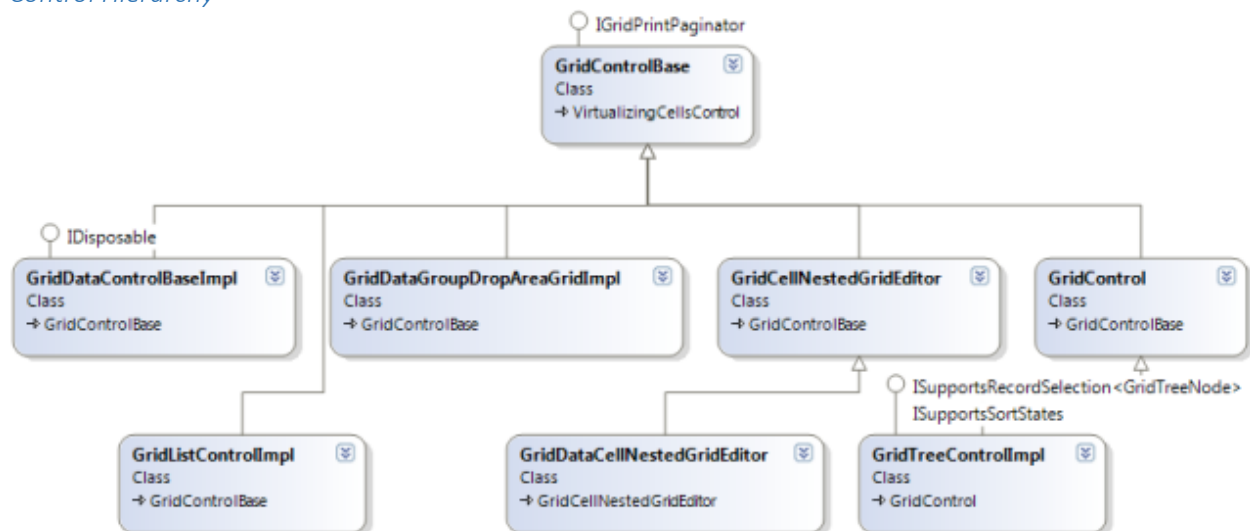
The GridTree control serves as a multicolumn tree control that is optimized to display thousands of items. This control uses a load-on-demand architecture to quickly generate a tree view. You can toggle the view of the underlying nodes by clicking the plus-minus glyphs of a root node. This control provides complete customization options such as custom level styles, glyphs, node images, and more.

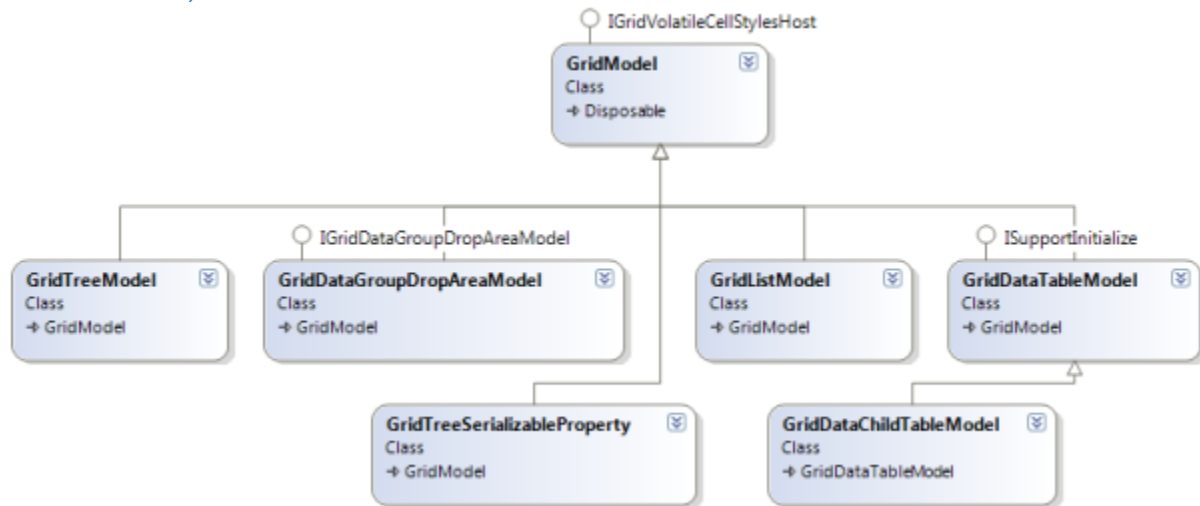


Class Diagram

The following illustration depicts the Class Diagram for Essential Grid for WPF.

Control Hierarchy



Model Hierarchy*Adding Essential Grid to an Application*

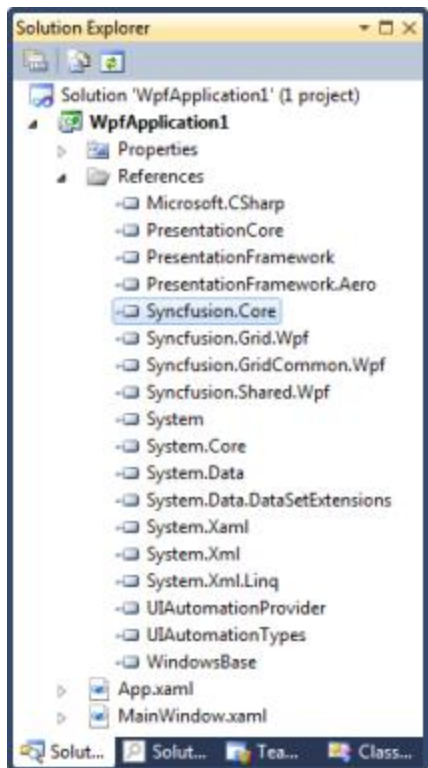
This section serves as a guide on how to deploy EssentialGrid in an application.

Adding the GridTree Control to a WPF Application

This section demonstrates how to add a GridTree control to a WPF application and how to load the grid with a data source. The GridTree control can be added to an application through programmatically.

Programmatically Adding GridTree Control

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

```
<Grid Name="layoutRoot"/>
```

4. Create a new GridTreeControl in code and add it as a child of layoutRoot (Grid). Now GridTreeControl will be added to the view.

CSHARP

```
//GridTreeControl defined here  
GridTreeControl treeGrid = new GridTreeControl();  
//To bring the GridTreeControl to the view, GridTreeControl should be  
//added to the children of layoutRoot.  
layoutRoot.Children.Add(treeGrid);
```

Data Population in the GridTree Control

The previous section explained how to add the GridTree control to an application. This section explains how to populate data in the GridTree control. There are three approaches to populating data:

- With the RequestTreeItems event.
- By self-relational collection binding.
- Using data-view binding.

The RequestTreeItems Event

The GridTree control can populate data on demand by handling the RequestTreeItems event. GridTreeControl will receive the source of root and child nodes through this event handler. This event is triggered when initially loading and expanding nodes.

To populate data using this event, follow these steps:

1. Create a collection of objects to bind with the GridTree control. In this example, a collection of objects containing employee information has been created.

CSHARP

```
public class EmployeesCollection:List<Employee>
{
    public EmployeesCollection()
    {
        this.Add(new Employee() { Title = "Management", ReportsTo = -1, ID = 2 });
        this.Add(new Employee() { Title = "Accounts", ReportsTo = -1, ID = 3 });
        this.Add(new Employee() { Title = "Sales", ReportsTo = -1, ID = 4 });
        //Management
        this.Add(new Employee() { FirstName = "Andrew", LastName = "Fuller", Department = "Management", EmpID = 1001, ID = 9, Salary = 1200000, ReportsTo = 2, Title = "Vice President" });
        this.Add(new Employee() { FirstName = "Janet", LastName = "Leverling", Department = "Management", EmpID = 1002, ID = 10, Salary = 1000000, ReportsTo = 2, Title = "GM" });
        //Accounts
        this.Add(new Employee() { FirstName = "Nancy", LastName = "Davolio", Department = "Accounts", EmpID = 1004, ID = 12, Salary = 850000, ReportsTo = 3, Title = "Accounts Manager" });
        this.Add(new Employee() { FirstName = "Margaret", LastName = "Peacock", Department = "Accounts", EmpID = 1008, ID = 13, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
        //Sales
        this.Add(new Employee() { FirstName = "Laura", LastName = "Callahan", Department = "Sales", EmpID = 1005, ID = 16, Salary = 900000, ReportsTo = 4, Title = "Sales Manager" });
        this.Add(new Employee() { FirstName = "Anne", LastName = "Dodsworth", Department = "Sales", EmpID = 1011, ID = 17, Salary = 800000, ReportsTo = 4, Title = "Sales Representative" });
    }
}

public class Employee
{
    int id;
    public int ID
    {
        get { return id; }
        set { id = value; }
    }
    int? empId;
    public int? EmpID
    {
        get { return empId; }
        set { empId = value; }
    }
}
```

```
string firstName;
public string FirstName
{
    get { return firstName; }
    set { firstName = value; }
}
string lastName;
public string LastName
{
    get { return lastName; }
    set { lastName = value; }
}
string department;
public string Department
{
    get { return department; }
    set { department = value; }
}
private string title;
public string Title
{
    get { return title; }
    set { title = value; }
}
double? salary;
public double? Salary
{
    get { return salary; }
    set { salary = value; }
}
int reportsTo;
public int ReportsTo
{
    get { return reportsTo; }
    set { reportsTo = value; }
}
}
```

2. The RequestTreeItems event can hook in either XAML or code.

Hooking RequestTreeItems Event in XAML

XML

```
<syncfusion:GridTreeControl Name="treeGrid"
RequestTreeItems="treeGrid_RequestTreeItems"
EnableNodeSelection="False"
AutoPopulateColumns="True"
PercentSizingBehavior="SizeUntouchedColumns" >
```

CSHARP

```
this.treeGrid.RequestTreeItems+=new GridTreeRequestTreeItemsHandler(treeGrid
_RequestTreeItems);
```

3. Handle the RequestTreeItems event to pass the source to the root and child nodes dynamically.

CSHARP

```

EmployeesCollection employees;
public MainWindow()
{
    InitializeComponent();
    this.gridTreeControl1.RequestTreeItems += new Syncfusion.Windows.Controls.GridTreeRequestTreeItemsHandler(treeGrid_RequestTreeItems);
    employees = new EmployeesCollection();
}

private void treeGrid_RequestTreeItems(object sender, GridTreeRequestTreeItemsEventArgs args)
{
    //When ParentItem is null, you need to set args.ChildList to be the root items
    if (args.ParentItem == null)
    {
        //Get the root list-get all employees who have no boss
        //Get all employees whose boss' id is -1 (no boss)
        args.ChildList = employees.Where(x => x.ReportsTo == -1);
    }
    else //If ParentItem not null, then set args.ChildList to the child items for the given ParentItem.
    {
        //Get the children of the parent object
        Employee emp = args.ParentItem as Employee;
        if (emp != null)
        {
            //Get all employees that report to the parent employee
            args.ChildList = employees.Where(x => x.ReportsTo == emp.ID);
        }
    }
}

```

When the application runs, the following output will be generated.

ID	EmpID	FirstName	LastName	Department	Title	Salary
2					Management	
9	1001	Andrew	Fuller	Management	Vice President	1200000
10	1002	Janet	Leverling	Management	GM	1000000
3					Accounts	
12	1004	Nancy	Davolio	Accounts	Accounts Ma...	850000
13	1008	Margaret	Peacock	Accounts	Accountant	700000
4					Sales	
16	1005	Laura	Callahan	Sales	Sales Manager	900000
17	1011	Anne	Dodsworth	Sales	Sales Represe...	800000

Samples

To view samples:

1. Select Start > Programs > Syncfusion > Essential Studio x.x.xx > Dashboard.
2. Click Run Samples for WPF under the User Interface Edition panel.
3. Select GridTreeControl.
4. Expand the Data Population Features item in the Sample Browser.
5. Select On-Demand Loading Demo to launch the sample.

Binding a Self-Relational Collection to the GridTree Control

A self-relational collection is a collection of objects in which each object has a hierarchy within. Each object will act as a parent and hold its children in an attribute. Each child acts as the next-level parent and holds children in an attribute, and so on. In this example, both child and parent will be of the same type (data type/object type). Specifying the child attribute name in ChildPropertyName of GridTreeControl will automatically fetch the hierarchy and populate it.

1. Create a self-relational collection of objects to bind with the GridTree control. In this example, we have created a collection of objects containing employee information.

CSHARP

```
//This code is used to create a list collection of hierarchical data
public class EmployeeDetails : List<Employee>
{
    public EmployeeDetails()
    {
        //Management
        //The child list is the ChildCollection of the node
        List<Employee> childList = new List<Employee>();
        childList.Add(new Employee() { FirstName = "Andrew", LastName = "Fuller", Department = "Management", EmpID = 1001, ID = 9, Salary = 1200000, ReportsTo = 2, Title = "Vice President" });
        childList.Add(new Employee() { FirstName = "Janet", LastName = "Leverling", Department = "Management", EmpID = 1002, ID = 10, Salary = 1000000, ReportsTo = 2, Title = "GM" });
        childList.Add(new Employee() { FirstName = "Steven", LastName = "Buchanan", Department = "Management", EmpID = 1003, ID = 11, Salary = 900000, ReportsTo = 2, Title = "Manager" });
        this.Add(new Employee() { Title = "Management", ReportsTo = 1, ID = 2, Child = childList });
        //Accounts
        childList = new List<Employee>();
        childList.Add(new Employee() { FirstName = "Nancy", LastName = "Davolio", Department = "Accounts", EmpID = 1004, ID = 12, Salary = 850000, ReportsTo = 3, Title = "Accounts Manager" });
        childList.Add(new Employee() { FirstName = "Margaret", LastName = "Peacock", Department = "Accounts", EmpID = 1008, ID = 13, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
        childList.Add(new Employee() { FirstName = "Michael", LastName = "Suyama", Department = "Accounts", EmpID = 1009, ID = 14, Salary = 700000, ReportsTo = 3, Title = "Accountant" });
    }
}
```



```
childList.Add(new Employee() { FirstName = "Robert", LastName = "King", Department = "Accounts", EmpID = 1010, ID = 15, Salary = 650000, ReportsTo = 3, Title = "Accountant" });
this.Add(new Employee() { Title = "Accounts", ReportsTo = 1, ID = 3, Child=childList });
}
}

public class Employee
{
    int id;
    public int ID
    {
        get { return id; }
        set { id = value; }
    }
    int? empId;
    public int? EmpID
    {
        get { return empId; }
        set { empId = value; }
    }
    string firstName;
    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }
    string lastName;
    public string LastName
    {
        get { return lastName; }
        set { lastName = value; }
    }
    string department;
    public string Department
    {
        get { return department; }
        set { department = value; }
    }
    private string title;
    public string Title
    {
        get { return title; }
        set { title = value; }
    }
    double? salary;
    public double? Salary
    {
        get { return salary; }
        set { salary = value; }
    }
    int reportsTo;
    public int ReportsTo
    {
        get { return reportsTo; }
        set { reportsTo = value; }
    }
}
```

```
public List<Employee> Child
{
    get;
    set;
}
```

2. Bind ItemsSource of GridTreeControl and assign ChildPropertyName—these can be set in either XAML or code.

Binding in XAML

XML

```
<syncfusion:GridTreeControl
Name="treeGrid"
AutoPopulateColumns="True"
ExpandStateAtStartup="AllNodesExpanded"
ChildPropertyName="Child"
ItemsSource="{Binding GTCSource}"/>
```

CSHARP

```
public MainWindow()
{
    InitializeComponent();
    this.DataContext = this;
    _gtcSource = new EmployeeDetails();
}

//This property is set as ItemsSource of GridTreeControl
private EmployeeDetails _gtcSource;
public EmployeeDetails GTCSource
{
    get
    {
        return _gtcSource;
    }
    set
    {
        _gtcSource = value;
    }
}
```

Assigning Items' Source Code

CSHARP

```
public MainWindow()
{
    InitializeComponent();
    //ItemsSource set to GridTreeControl
    this.treeGrid.ItemsSource = new EmployeeDetails();
}
```

When the application runs, the following output will be generated.

ID	EmpID	FirstName	LastName	Department	Title
2					Management
9	1001	Andrew	Fuller	Management	Vice President
10	1002	Janet	Leverling	Management	GM
11	1003	Steven	Buchanan	Management	Manager
3					Accounts
12	1004	Nancy	Davolio	Accounts	Accounts Ma...
13	1008	Margaret	Peacock	Accounts	Accountant
14	1009	Michael	Suyama	Accounts	Accountant
15	1010	Robert	King	Accounts	Accountant

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 GridTreeControl.
4. Expand the Data Population Features item in the Sample Browser.
5. Select Self-Relational Data Binding Demo to launch the sample.

Binding a Data View to the GridTree Control

The following steps explain how to bind a data view from a database to the GridTree control.

1. Connect a database to the current application. The database can be connected several ways, such as ADO.NET, LINQ to SQL, classes, and so on. In this example, a connection has been directly established to a simple northwind.sdf database.
2. Once the connection is established, fetch the required data table from the database. In this example, the data table has been fetched from the northwind.sdf by using SQL DataAdapter.

Note: Before using this procedure, check that System.data.SqlServerCe.dll has been added to your project.

C# SHARP

```
// Connect to a data table
public DataTable GetDataTable()
{
    DataSet ds = new DataSet();
    if (!LayoutControl.IsInDesignMode)
    {
        using (SqlCeConnection con = new SqlCeConnection(connectionString))
        {
            con.Open();
            SqlCeDataAdapter sda = new SqlCeDataAdapter("SELECT * FROM Employees", con);
            sda.Fill(ds, "Employee");
        }
    }
    //The following line is used to create the hierarchical relations
}
```

```
ds.Relations.Add(new DataRelation("Employee_Relation", ds.Tables["Employee"]
.Columns["Employee ID"], ds.Tables["Employee"].Columns["Reports To"], false))
;
}
if (ds.Tables.Count > 0)
return ds.Tables[0];
else
return null;
}
```

3. Now bind the data table as an ItemsSource of GridTreeControl in either XAML or code.

Binding in XAML

XML

```
<syncfusion:GridTreeControl Name="treeGrid"
AutoPopulateColumns="True"
ItemsSource="{Binding GTCSource}"
ExpandStateAtStartup="AllNodesExpanded"
ChildPropertyName="Employee_Relation" />
```

CSHARP

```
public MainWindow()
{
InitializeComponent();
this.DataContext = this;
dataTable = GetDataTable();
}
DataTable dataTable;
public DataView GTCSource
{
get
{
if (dataTable == null)
return null;
DataView dataView = new DataView(dataTable);
//RowFilter is applied to form the hierarchy
dataView.RowFilter = "[Reports To] Is NULL";
return dataView;
}
}
```

Assigning the Items' Source in Code

CSHARP

```
public MainWindow()
{
InitializeComponent();
//Relation name set as ChildPropertyName
this.treeGrid.ChildPropertyName = "Employee_Relation";
dataTable = GetDataTable();
}
```

```
//ItemsSource set to GridTreeControl
this.treeGrid.ItemsSource = GTCSource;
}
DataTable dataTable;
public DataView GTCSource
{
    get
    {
        if (dataTable == null)
            return null;
        DataView dataView = new DataView(dataTable);
        dataView.RowFilter = "[Reports To] Is NULL";
        return dataView;
    }
}
```

When the application runs, the following output will be generated.

Employee ID	Last Name	First Name	Title	Birth Date	Hire Date
2	Fuller	Andrew	Vice Preside...	19-02-1942 0...	12-07-1991 0...
1	Davolio	Nancy	Sales Repres...	08-12-1948 0...	29-03-1991 0...
3	Leverling	Janet	Sales Repres...	30-08-1963 0...	27-02-1991 0...
11	Smith	Tim	Mail Clerk	06-06-1973 0...	15-01-1993 0...
12	Patterson	Caroline	Receptionist	11-09-1972 0...	15-05-1993 0...
4	Peacock	Margaret	Sales Repres...	19-09-1937 0...	30-03-1992 0...
5	Buchanan	Steven	Sales Manag...	04-03-1955 0...	13-09-1992 0...
6	Suyama	Michael	Sales Repres...	02-07-1963 0...	13-09-1992 0...
7	King	Robert	Sales Repres...	29-05-1960 0...	29-11-1992 0...
9	Dodsworth	Anne	Sales Repres...	27-01-1966 0...	12-10-1993 0...
8	Callahan	Laura	Inside Sales...	09-01-1958 0...	30-01-1993 0...
10	Hellstern	Albert	Business Ma...	13-03-1960 0...	01-03-1993 0...

Sample

To view samples

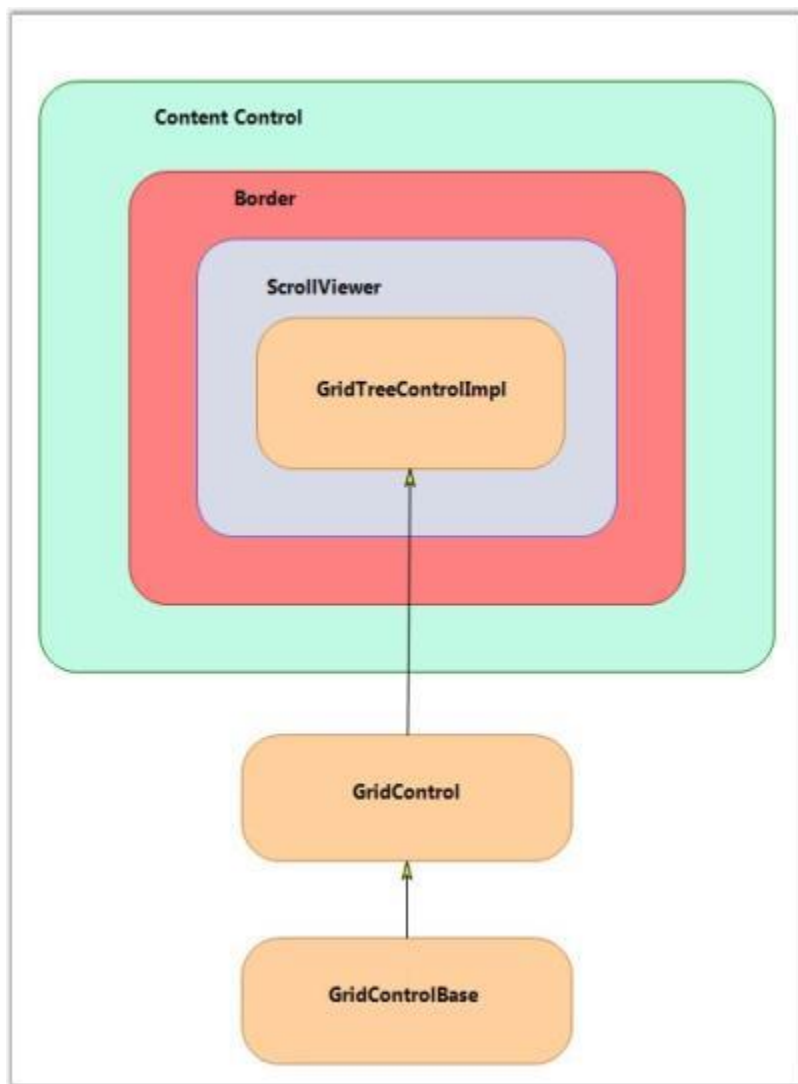
1. Select Start > Programs > Syncfusion > Essential Studio x.x.xx > Dashboard.
2. Click Run Samples for WPF under User Interface Edition panel.
3. Select GridTreeControl.
4. Expand the Data Population Features item in the Sample Browser.
5. Select Data View Binding Demo to launch the sample.

Architecture in WPF GridTreeControl (Classic)

The GridTree control derives from the WPF ContentControl, which allows it to support a ControlTemplate to define its content. By default, its content includes a Border object, which contains a ScrollViewer object that contains a GridControlImpl object. GridControlImpl is a GridControlBase derived class that provides the 'multi-column grid' functionality to the GridTree control.

For each node in the tree, there is a GridTreeNode object that holds the information of the node such as the underlying data item, whether the node is expanded, etc. The GridTreeNodes collection can be accessed by the GridTreeControl.InternalGrid.Nodes collection. InternalGrid is the GridTreeControlImpl object associated with the GridTree control.

The following screen shot illustrates the GridTree control architecture.



Accessing the Underlying Grid control

The GridTree control is a ContentControl derived class. To get its grid-like behavior, the GridTree control has a Grid control derived property named InternalGrid. InternalGrid is a GridTreeControlImpl class, which is derived from the Grid control. The GridTreeControlImpl is a virtual Grid control, which uses the virtual events to bind to the GridTreeControl.Nodes collection. So, to access the underlying Grid control associated with the GridTree control, you can use the GridTreeControl.InternalGrid property.

All the properties exposed in GridTree control (with the exception of Internal Grid) are mirrored in GridTreeControlImpl. There are methods and properties exposed on the Internal Grid that are not exposed on the GridTree control itself. In particular, to control the look of the Expand cell, you need to use the Internal Grid as discussed below. The Internal Grid has many protected methods that provide access to the tree-like functionality. So, deriving GridTreeControlImpl gives you access to this functionality if you need to use it for any reason.

GridTree Control Properties in WPF GridTreeControl (Classic)

The GridTree control has the following properties that allow you to control much of the behavior available within the control. Here is a list of these properties along with some discussion of each.

GridTree control Property	Description	Type of Property	Value It Accepts	Property Syntax
AllowAutoSizingNodeColumn	Controls whether the width of the ExpandCell is automatically adjusted as more levels are exposed. This property setting is ignored if PercentageSizing is enabled and you have the ExpandCell column marked to participate in the percentage sizing calculations.	DependencyProperty	bool	treeGrid.AllowAutoSizingNodeColumn
AllowDragColumns	Controls whether your user can rearrange the columns by moving the mouse down on the header cell and drag it to another location in the control.	Dependency property	bool	treeGrid.AllowDragColumns
AllowSort	Controls whether your user can sort columns by clicking the column header. When enabled, holding down the Ctrl key and clicking the columns will allow sorting on multiple columns. All sorting is done within the child lists.	Dependency property	bool	treeGrid.AllowSort
ColumnHeaderStyle	A GridStyleInfo object that controls the appearance of the column headers.	Dependency property	GridStyleInfo	treeGrid.ColumnHeaderStyle = new GridStyleInfo() { };
DefaultColumnWidth	Provides the default settings for the column width used in the GridTree control.	Dependency property	Double	treeGrid.DefaultColumnWidth
EnableHotRowMarker	Controls whether the row under the mouse is overdrawn with a special background brush so that the user can easily see the row under the mouse. The brush can be specified using	Dependency property	bool	treeGrid.EnableHotRowMarker

	the GridTreeControl.InternalGrid.markRowBrush property.			
EnableSelections	Controls whether selections are supported. The GridTree control support two types of selections, node selections and cell selections. Use the EnableNodeSelection property to determine which selection type is active.	Dependency property	bool	treeGrid.EnableSelections
EnableNodeSelection	Controls whether all the rows will be selected when a single row is clicked. When enabled, you can select multiple rows by holding the Shift key and dragging the mouse. When not enabled, the cell ranges will be selected.	Dependency property	bool	treeGrid.EnableNodeSelection
HideEmptyChildGlyphs	Controls whether you can possibly see the expand glyph on a node with no children. The default value is true that indicates glyphs on empty nodes will be hidden when initially displayed. In order to support this behavior, the GridTree control must request the child nodes at the time the parent node is expanded (instead of at the time the child node is clicked to be expanded).	Dependency property	bool	treeGrid.HideEmptyChildGlyphs
InternalGrid	Gets a reference to the GridControlImpl object associated with this GridTree control. This property will be null until the ModelLoaded event is raised by the GridTree control. ModelLoaded event is triggered once the grid model is loaded.	Normal	GridTreeControl	treeGrid.InternalGrid

LevelStyles	A List object that holds GridStyleInfo objects that will control the look of cells for a given tree level. The 0th GridStyleInfo will be applied to all root node rows, the 1st GridStyleInfo will be applied to level 1 rows, etc.	Dependency property	List	treeGrid.LevelStyles
Model	Gives access to the GridTreeModel object associated with this GridTree control. The Model.Options property gives access to the properties that control the behavior of the InternalGrid object associated with this GridTree control.	Normal	GridTreeModel	treeGrid.Model
ReadOnly	Determines whether the cells in the GridTree control can be edited. This is a GridTree control-wide setting. You can use the LevelStyles[].StyleInfo.ReadOnly to edit the particular levels. Additionally, you can use Columns[].StyleInfo.ReadOnly to control the edit operation, column-by-column. Finally, you can use the Model.QueryCellInfo event to set ReadOnly properties cell-by-cell in an on-demand manner.	Dependency property	bool	treeGrid.ReadOnly
RowHeaderWidth	Specifies the width of the row header column if it is visible. To control the visibility, use the ShowRowHeader property.	Dependency property	Double	treeGrid.RowHeaderWidth
SelectedNodes	A collection of nodes selected in the GridTree control. The items in the collection are GridTreeNode objects.	Normal	GridSelectedTreeNodes	treeGrid.SelectedNodes

ShowColumnHeaders	Determines whether the column header row is visible.	Dependency property	bool	treeGrid.ShowColumnHeaders
ShowExpandColumnBorders	Determines whether the GridTree control display cell borders for the Expand column.	Dependency property	bool	treeGrid.ShowExpandColumnBorders
ShowRowHeader	Determines whether the row header column is visible.	Dependency property	bool	treeGrid.ShowRowHeader
SupportRowSizing	Determines whether the user can change the row heights using the mouse.	Dependency property	bool	treeGrid.SupportRowSizing
SupportsVisualStyles	Determines whether the GridTree control will use the VisualStyle property to control the appearance of the GridTree control. If this property is <i>true</i> , you can directly set the VisualStyle property for the grid. You also have the option of applying the changes through the SkinManager found in the Syncfusion.Shared.WPF library. The GridTree control is bound to the SkinManager values through its Template.	Dependency property	bool	treeGrid.SupportsVisualStyle
VisualStyle	Determines the VisualStyle that is applied to the GridTree control when SupportsVisualStyles property is set to <i>true</i> .	Dependency property	VisualStyle	treeGrid.VisualStyle

GridTreeNode Objects in WPF GridTreeControl (Classic)

The GridTreeControl.InternalGrid.Nodes collection holds the GridTreeNodes that represents the visible nodes in the GridTree control. The GridTreeNode object has the following properties:

GridTreeNode Property	Description	Type of Property	Value It Accepts	Property Syntax
ChildNodes	A collection of GridTreeNodes that represent the child nodes for this node.	Normal	List	treeGrid.ParentNode.ChildNodes

Expanded	Controls the expand state for the node. Set this property to true, to expand this node, or set it to false, and to collapse this node.	Normal	bool	gridTreeNode.Expanded
HasChildNodes	Determines whether this node has child nodes or not.	Normal	bool	gridTreeNode.HasChildNodes
IsSelected	Controls whether this node is selected.	Normal	bool	gridTreeNode.IsSelected
Item	The data item (an object from the underlying bound data) associated with this node.	Normal	Object	gridTreeNode.Item
Level	Indicates tree level for this node.	Normal	int	gridTreeNode.Level
NodeHeight	The height of the grid row associated with this node.	Normal	Double	gridTreeNode.NodeHeight
ParentItem	The data item (an object from the underlying bound data) associated with the parent node of this node.	Normal	Object	gridTreeNode.ParentItem
ParentNode	The GridTreeNode that is the parent node of this node.	Normal	GridTreeNode	gridTreeNode.ParentNode
SelectedColumns	Holds the column names that are selected for this node. This property is only valid when GridTreeControl.EnableNodeSelections is <i>false</i> and GridTreeControl.EnableSelections is <i>true</i> .	Normal	List	gridTreeNoe.SelectedColumns

GridTree Control Events in WPF GridTreeControl (Classic)

Here is the list of events exposed in GridTree control. Additionally, you have access to all the GridControl events exposed on the GridTreeControl.InternalGrid.

GridTree control Event	Description
ModelLoaded	Handled once the InternalGrid is created and after the template is applied to the GridTree control. This is the correct place to set the properties and subscribe to events on the InternalGrid.
RequestTreelItems	Required event that is used to provide the underlying data to the GridTree control on demand. This event is discussed earlier in this document.
RequestNodeImage	By handling this event, you can provide an image to be used in the expand cell. This event is discussed earlier in this document.

ExpandStateChanging	By handling this event , you can cancel the node being expanded by the user.
ExpandStateChanged	This is a notification event that is handled once a node has been expanded by the user.

Note: Additionally, you have access to all the Grid control events exposed on the GridTreeControl.InternalGrid.

Data Population in WPF GridTreeControl (Classic)

The primary requirement of the data displayed in the GridTree control is that the parent node and the child node share the same schema (i.e. have the same columns to display). To populate the GridTree control, you need to handle the RequestTreeItems event (This event gets fired for every tree node that is being populated in order to retrieve its child nodes). In the event arguments, the ParentItem property indicates the item whose children are being requested. The list of children can be set using this property. If this property is empty, it implies that the event is requesting the root nodes. The following code example illustrates a minimal RequestTreeItems handler.

C#

```
private void treeGrid_RequestTreeItems(object sender,
GridTreeRequestTreeItemsEventArgs args)
{
    // When the ParentItem is null, you need to set args.ChildList to be the
    // root items.
    if (args.ParentItem == null)
    {
        // Get the root list - get all employees who have no boss.
        args.ChildList = employees.GetReportees(-1); //get all whose boss's id is -1
        (no boss)
    }
    // If ParentItem not null, set args.ChildList to the child items for the
    // given Parent.
    else
    {
        // Get the children of the parent object.
        Employee emp = args.ParentItem as Employee;
        if (emp != null)
        {
            // Get all employees that report to the parent employee.
            args.ChildList = employees.GetReportees(emp.ID);
        }
    }
}
```

The only requirement on ChildList returned by the RequestTreeItems is that it should be an IEnumerable list of typed objects that share a single type.

Order and Visibility of Columns in the GridTree Control

The default behavior of the GridTree control is to display all simple, public properties exposed in the items provided through the RequestTreeItems event (Here, properties indicate Data Source properties, which are represented as columns in the Grid control. For eg, in Customers table, CustomerID, CompanyName, etc. are termed as properties). To control the visibility and the order of the columns

that appear in the Grid Tree, you can use the GridTreeControl.Columns collection of GridTreeColumn object. The order of the GridTreeColumn objects in the Columns collection determines the display order of the columns in the Grid Tree.

- Each GridTreeColumn contains the following five properties: MappingName-The name of the property from the tree item that is bound with this column
- HeaderText-The HeaderText property sets the text for header cell of this column. If HeaderText is empty, then the MappingName will be used as the column header text.
- Width-Specifies width of this column in pixels.

The following code example illustrates how to set these properties for the GridTreeColumns by using XAML code.

XML

```
<sf:GridTreeControl Name="treeGrid"
RequestTreeItems="treeGrid_RequestTreeItems" >
<sf:GridTreeControl.Columns>
<sf:GridTreeColumn MappingName="FirstName" HeaderText="First Name"/>
<sf:GridTreeColumn MappingName="LastName" HeaderText="Last Name"/>
<sf:GridTreeColumn MappingName="Salary" />
<sf:GridTreeColumn MappingName="Rating" />
</sf:GridTreeControl.Columns>
</sf:GridTreeControl>
```

- PercentWidth-This is another width property that plays a role in column sizing, if you had set GridTreeControl.PercentSizingBehavior(The PercentSizingBehavior determines how the grid columns should be sized when the grid is resized) to some value other than 'None'. If you set a value to the PercentWidth property, this column will be resized according to the sizing algorithms. For more details refer the PercentSizingBehavior section.
- StyleInfo-StyleInfo is a GridStyleInfo object that controls the appearance of the cells in this column. This object enables you to set various column properties, including the Font, and Background and Foreground color. All properties exposed in the GridStyleInfo objects are customizable by using this StyleInfo property.

Columns in WPF GridTreeControl (Classic)

The GridTree control's columns are divided into two types. They are:

- Bound Column
- Unbound Column

Bound Column

A bound column displays information from a bound data source, which is specified by GridTreeControl.ItemSource, or by handling the RequestTreeItems event. You can typically add one bound column to the grid for every data column you want to display.

Unbound Column

An unbound column does not obtain data from the ItemSource or from the RequestTreeItems event. Instead, you can provide data for unbound columns by creating expressions using the

GridTreeUnboundColumn.Expression property, or by creating formats using the GridTreeUnboundColumn.Format property, or by handling the QueryUnboundColumnValue event.

Note: In Unbound columns, you can have the features that are available in the bound columns such as cell type, sorting, and customization.

Bound Columns

A bound column displays information from a data source which is specified by the GridTreeControl.ItemSource or by handling the RequestTreeItems event. You can typically add one bound column to the grid for every data column you want to display.

Tables for Properties, Methods, and Events

Properties

Property	Description	Type	Data Type
AutoPopulateColumns	By setting this property to <i>true</i> , the GridTree control populates the properties that are present in the underlying collection without defining them in the XAML or in C#.	Dependency Property	Boolean
AutoGenerateColumnsInfo	Assigns the cell type based on the data bound to this column.	Dependency Property	Boolean

AutoPopulateColumns

In the GridTree control, you can populate the columns automatically by setting the AutoPopulateColumns property to True.

XML

```
<syncfusion:GridTreeControl Name="treeGrid" AutoPopulateColumns="True">
```

C#

```
treeGrid.AutoPopulateColumns = true;
```

When we set this property to true, the properties that are available in underlying collection populates in the GridTree control as GridTreeColumn.

AutoGenerateColumnsInfo

By setting the AutoGenerateColumnsInfo property to *true*, the GridTree control assigns the appropriate cell type for each column automatically. The following common cell types loaded for the corresponding data.

Value Type	Cell type
String	TextBox
Boolean	Check Box
Double	DoubleEdit

DateTime	DateTimeEdit
Enums	Combo Box
TimeSpan	TimeSpanEdit
Uri	Hyperlink

XML

```
<syncfusion:GridTreeControl Name="treeGrid"
AutoGenerateColumnsInfo="True">
```

C#

```
treeGrid.AutoGenerateColumnsInfo = true;
```

Events

Event	Arguments	Description
QueryVisibleColumnInfo	(GridTreeColumn VisibleColumn)	This event is used to customize a GridTree column when the column is created. This event argument contains the GridTreeColumn that is created. By handling this event, you can perform any operation on the column.

The following code shows a simple customization on the GridTreeColumn by handling this event.

In the below code snippet, VisibleColumn properties are customized based on the MappingName property.

C#

```
treeGrid.QueryVisibleColumnInfo += (treeGrid_QueryVisibleColumnInfo);
void treeGrid_QueryVisibleColumnInfo(object sender, GridTreeQueryVisibleColumnInfoEventArgs Args)
{
    if (Args.VisibleColumn.MappingName == "EmpID")
    {
        Args.VisibleColumn.AllowSort = false;
        Args.VisibleColumn.HeaderText = "Customer ID";
        Args.VisibleColumn.StyleInfo = new GridStyleInfo() { CellType = "IntegerEdit",
        HorizontalAlignment = HorizontalAlignment.Right };
        Args.VisibleColumn.Width = 80;
    }
}
```

VisibleColumn Properties

The following are the list of properties that are available for all visible columns.

Property	Description
AllowSort	By applying true/false, you can enable/disable the Sorting operation for the particular column.
StyleInfo	Sets a new style for a particular column.
Width	Sets the width for a particular column.
PercentWidth	Fills the columns within the client area. The width of the columns is calculated with the client area.

The following are the list of common properties for all cell types and available in StyleInfo class.

Property	Description
ReadOnly	Applying true to this property makes the column as read only.
ToolTip	You can set tool tip for a particular column by applying value to this property
Background, Foreground, Font	You can set the background, foreground and font for a particular column by this property.
FlowDirection	You can set the flow direction of the cell value by using this property either by left-to-right or right-to-left.
MaxLength	By using this property you can set the Maximum length of the column.
Padding	You can apply padding to the column by using this property.
ShowTooltip	You can enable / disable the tooltip for a specific column by using this property.
TextMargins	You can apply margins for the text by using this property.
ShowDataValidationTooltip	Property enables/disables the data validation tooltip.

To know more about the GridStyleInfo and StyleInfo properties please refer the following link.

<http://help.syncfusion.com/wpf/grid/appearance#cell-styles>

Cell Types

In GridTree control you can define the cell type for each column by using the property “CellType”. When you leave a column without defining the cell type then that column’s cell type is considered as “Static” cell type. The list of cell types that are available in the GridTree control are as follows.

Static TextBlock TextBox RichText ExpanderCell Data Template ComboBox DropDownList CheckBox Currency edit Double edit DateTime edit Integer edit Mask edit Percent edit Up Down edit TimeSpan edit Image Hyperlink Button

>
It is not possible to change the cell type for the first column of the GridTree control.

Static

The Static cell type allows you to only to display the values in the cells and not to edit the cells. When you want to display a read only content then you can choose the static cell types.

The following is an example of how to define such a column.

XML

```
<syncfusion:GridTreeColumn Width="150" MappingName="Address">
<syncfusion:GridTreeColumn.StyleInfo>
<syncfusion:GridStyleInfo CellType="Static"/>
</syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("Address")
{
    StyleInfo = new GridStyleInfo() { CellType = "Static" },
});
```

First Name	Title	Last Name	ID	Address	City
Andrew	Vice President, Sales	Fuller		2 908 W. Capital Way	Tacoma
Nancy	Sales Representative	Davolio		1 507 - 20th Ave. E.	Seattle
Janet	Sales Representative	Leverling		3 722 Moss Bay Blvd.	Kirkland
Tim	Mail Clerk	Smith		11 30301 - 166th Ave. N.E.	Kent
Caroline	Receptionist	Patterson		12 16 Maple Lane	Auburn
Margaret	Sales Representative	Peacock		4 4110 Old Redmond Rd.	Redmond
Steven	Sales Manager	Buchanan		5 14 Garrett Hill	London
Michael	Sales Representative	Suyama		6 Coventry House	London
Robert	Sales Representative	King		7 Edgeham Hollow	London
Anne	Sales Representative	Dodsworth		9 7 Houndstooth Rd.	London
Laura	Inside Sales Coordinator	Callahan		8 4726 - 11th Ave. N.E.	Seattle
Albert	Business Manager	Hellstern		10 13920 S.E. 40th Street	Bellevue
Justin	Marketing Director	Brid		13 2 impasse du Soleil	Haguenau
Xavier	Marketing Associate	Martin		14 9 place de la Liberté	Schiltigheim
Laurent	Advertising Specialist	Pereira		15 7 rue Nationale	Strasbourg

GridTree Static CellType

TextBlock

The TextBlock cell type allows you to display the values in the cells and also to edit the cells. When you need to display a read only content then you can choose TextBlock cell type.

Note: The difference between the Static and TextBlock cell type is that the Static cell type draws the cell but the TextBlock cell type loads the cell with TextBlock.

The following is an example of how to define such a column.

XML

```
<syncfusion:GridTreeColumn Width="130" MappingName="FirstName">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridDataStyleInfo CellType="TextBlock" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
this.treeGrid.Columns.Add(new GridTreeColumn()
{
    MappingName = "FirstName",
    StyleInfo=new GridStyleInfo()
    {
        CellType="TextBlock",
    }
});
```

The following screenshot shows a simple demo of this cell type.

ID	FirstName	Department	Title
174	BIGBOSS1	department3	title1
413	first413_0	department2	title0
459	first459_0	department1	title2
26	first26_0	department2	title1
464	first464_0	department1	title4
130	first130_0	department1	title1
171	first171_0	department3	title3
8	first8_0	department0	title0
392	first392_0	department1	title0
492	first492_0	department3	title3
405	first405_0	department0	title4
305	first305_0	department2	title3
180	first180_0	department1	title0
445	first445_0	department0	title2
355	first355_0	department2	title1

GridTree TextBlock Cell type*TextBox*

The TextBox cell type allows you to display the cell values in text boxes. Each cell loads a text box and its Text property is bound with the cell value.

The following code snippet explains how to define such a column.

XML

```
<syncfusion:GridTreeColumn Width="130" MappingName="FirstName">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridDataStyleInfo CellType="TextBox" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
this.treeGrid.Columns.Add(new GridTreeColumn()
```

```
{
    MappingName = "FirstName",
    StyleInfo = new GridStyleInfo()
    {
        CellType = "TextBox",
    }
});
```

The following screenshot shows a simple demo of TextBox cell type.

ID	FirstName	Department	Title
174	BIGBOSS1	department3	title1
413	first413_0	department2	title0
459	first459_0	department1	title2
426	first26_0	department2	title1
464	first464_0	department1	title4
130	first130_0	department1	title1
171	first171_0	department3	title3
8	first8_0	department0	title0
392	first392_0	department1	title0
492	first492_0	department3	title3
405	first405_0	department0	title4
305	first305_0	department2	title3
180	first180_0	department1	title0
445	first445_0	department0	title2
355	first355_0	department2	title1
316	first316_0	department3	title1

GridTree TextBox Cell Type

RichText

The RichText cell types provide more advanced formatting features than the TextBox and TextBlock cell types. You can apply character and paragraph formatting to the text in the RichText cell type. A RichText cell type allows you to display string in a paragraph or in other formats and also the cell value should be a FlowDocument object.

The following is an example of how to define such a column.

XML

```
<syncfusion:GridDataVisibleColumn Width="32"
    HeaderText=" "
    MappingName="IsRead">
    <syncfusion:GridDataVisibleColumn.ColumnStyle>
    <syncfusion:GridDataColumnStyle HorizontalAlignment="Center"
    CellType="RichText" />
    </syncfusion:GridDataVisibleColumn.ColumnStyle>
</syncfusion:GridDataVisibleColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("Document")
{
    StyleInfo = new GridStyleInfo()
```

```

{
    CellType = "RichText",
}
}
);
Paragraph p = new Paragraph();
Run r1 = new Run(emp.ID.ToString());
r1.FontWeight = FontWeights.ExtraBold;
p.Inlines.Add(r1);
p.Inlines.Add(new LineBreak());
Run r2 = new Run(emp.FirstName);
p.Inlines.Add(r2);
p.Inlines.Add(new LineBreak());
Run r3 = new Run(emp.Salary.ToString());
p.Inlines.Add(r3);
p.Inlines.Add(new LineBreak());
Run r4 = new Run(emp.Department);
p.Inlines.Add(r4);
p.Inlines.Add(new LineBreak());
FlowDocument doc = new FlowDocument(p);
emp.Document = doc;

```

ID	FirstName	Details	ReportsTo
34	BIGBOSS1	34 first34_0	-1
68	first68_0	68 first68_0	34
75	first75_0	75 first75_0	68
55	first55_0	55 first55_0	75
88	first88_0	88 first88_0	75
73	first73_0	73 first73_0	68
20	first20_0	20 first20_0	73
67	first67_0	67 first67_0	73
83	first83_0	83 first83_0	73

GridTree RichText Cell Type

[ExpanderCell](#)

In the GridTree control by default, the ExpanderCell type is present at the first column. This column helps you to expand the parent node and navigate into the child nodes. In general, ExpanderCell contains a glyph to expand/collapse the parent nodes.

The following are the list of glyphs in the GridTree control.

Glyph Types

The glyph present in the ExpanderCell can be changed and customized. By default, it has the following types of glyph.

- PlusMinus
- PlusMinusLines

- Themed
- Triangle
- Custom

PlusMinus—contains a plus/minus symbol. The plus symbol appears when the node is collapsed and a minus symbol appears when the node is expanded.

PlusMinusLine—contains a plus/minus symbol with lines. When the plus symbol is visible then a line is drawn from the parent node to its child node and each child node has the line with its parent node.

Themed—loads the glyph based on the theme applied for the GridTree control. The patch applied in the theme for the glyph loads as a glyph for the all nodes.

Triangle—loads the triangle symbol for each node that contains the child nodes. The collapsed node's triangle symbol appears horizontal and the expanded node's triangle turns vertical.

Custom—a custom type glyph is user based and you can load any kind of path into it.

Node image support

In the GridTree control, you can display image in each ExpanderCell and it can be achieved by handling the RequestNodeImage event and applying *true* to the SupportNodeImages property

The following code snippet shows how to load the image in ExpanderCell.

C#

```
private void treeGrid_RequestNodeImage(object sender,
GridTreeRequestNodeImageEventArgs args)
{
    args.NodeImage = GetItemBitmap(args.Item as EmployeeInfo);
}
```

FirstName	LastName	Salary
▲ BIGBOSS1	last174	\$200,000.00
▲ first413_0	last413	\$110,000.00
▲ first459_0	last459	\$100,000.00
▲ first26_0	last26	\$100,000.00
first464_0	last464	\$40,000.00
first130_0	last130	\$40,000.00
▲ first171_0	last171	\$80,000.00
first8_0	last8	\$40,000.00
first392_0	last392	\$50,000.00
first492_0	last492	\$80,000.00
first405_0	last405	\$80,000.00
first305_0	last305	\$70,000.00
first180_0	last180	\$60,000.00
▲ first445_0	last445	\$40,000.00
first355_0	last355	\$110,000.00
first316_0	last316	\$90,000.00

GridTree ExpanderCell with Image

DataTemplate

In the GridTree control, DataTemplate cell type allows you to replace the visual appearance of a cell. You can load any controls into a cell and display the CellBoundValue or custom values. This provides an extensible way to display the cells. The DataContext of a DataTemplate is same as that of the GridTree control.

The following are the list of properties specific for this cell type.

Properties

Property	Description
CellItemTemplate	This a DateTemplate type property, which is used to set the Item DataTemplate for the cell.
CellEditTemplate	This a DateTemplate type property, which is used to set the Edit DataTemplate for the cell.
CellItemTemplateKey	This is s String type property, which is to get the DataTemplate key form the user and it assign to the CellItemTemplate for the cell.
CellEditTemplateKey	This is s String type property, which is to get the DataTemplate key form the user and it assign to the CellEditTemplate for the cell.
HasCellItemTemplate	This is a Boolean property, which returns true when there is a CellItemTemplate.
HasCellEditTemplate	This is a Boolean property, which returns true when there is a CellEditTemplate.
HasCellItemTemplateKey	This is a Boolean property, which returns true when there is a CellItemTemplateKey.
HasCellEditTemplateKey	This is a Boolean property, which returns true when there is a CellEditTemplateKey.

You can load any property value from the current record by using the following format in the DataTemplate binding.

Record.Data (data will be the underlying object bound).

Example: Record.Data.Suppliers.CompanyName

You can get any property value of the current record by passing the appropriate property name.

The following example code shows a simple DataTemplate.










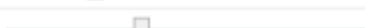
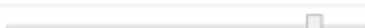

XML

```
<Window.Resources>
<DataTemplate x:Key="sliderTemplate">
<Slider Value="{Binding Path=CellBoundValue, Mode=TwoWay}"
Minimum="-10" Maximum="10000"
Height="30" Width="150" />
</DataTemplate>
</Window.Resources>
```

```
<syncfusion:GridTreeColumn MappingName="Weight">
<syncfusion:GridTreeColumn.StyleInfo>
<syncfusion:GridDataStyleInfo
CellType="DataBoundTemplate"
CellEditTemplate="{StaticResource sliderTemplate}"
CellItemTemplate="{StaticResource sliderTemplate}"/>
</syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("Weight")
{
    StyleInfo = new GridStyleInfo()
    {
        CellType = "DataBoundTemplate",
        CellItemTemplateKey = "sliderTemplate",
        CellEditTemplateKey = "sliderTemplate",
    }
});
```

Name	Weight	AngularValue
▲LName0		81
▲Name917		4
▲Name1009		66
▲Name1916		84
Name1983		13
Name1145		52
Name1303		50
Name1644		72
▲Name927		48
Name972		85
Name1813		54
▲Name1		58

GridTree DataTemplate CellType*ComboBox*

The ComboBox cell type allows you to select an item either by typing the text into the combo box, or by selecting it from the list. A ComboBox cell type is appropriate when there is a list of suggested choices, and a list box is appropriate when you want to limit the input to what is on the list.

The table below lists the various properties that can affect the combo box cells.

GridStyleInfo	Description
CellType	Sets to "ComboBox" for a ComboBox control.

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.
ShowButton	Boolean value indicating whether the Dropdown button should appear or not.

The table below lists the events available in the ComboBox cell type.

Events

Events	Description	Arguments
DropDownSelectionChanged	This event is raised when the SelectedItem of the combo box is changed.	CellRowColumnIndexâ€”contains the RowColumn index of the cell where the combo boxâ€™s value gets changed.SelectedItemâ€”this is the ComboBox item which is currently selected.

The following is an example of how to define such a column.

XML

```
<syncfusion:GridTreeColumn HeaderText="Department"
MappingName="Department">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridDataStyleInfo CellType="ComboBox"
DisplayMember="Shipcountry"
DropDownStyle="Editable"
ItemsSource="{StaticResource ShipDetails}"
ValueMember="Shipcity" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
GridTreeColumn coll = treeGrid.Columns[3];
coll.StyleInfo.CellType = "ComboBox";
coll.StyleInfo.ItemsSource = ships;
coll.StyleInfo.DisplayMember = "Shipcountry";
coll.StyleInfo.ValueMember = "Shipcity";
coll.StyleInfo.DropDownStyle = GridDropDownStyle.Editable;
```

We can change the combo box drop-down list style by using the GridDropDownStyle property. This property features the following options:

- Editable—combines an editable text field and provides users the option of typing an item that may or may not be in the drop-down list. The text entered in the combo box is not case-sensitive.
- Autocomplete—predicts the word or phrase that the user types; so that the user need not type it completely.
- Exclusive—non-editable combo box where the user is allowed to select only the options that are available in the drop-down list.

ID	FirstName	Department
174	BIGBOSS1	Japan
413	first413_0	India
459	first459_0	China
26	first26_0	China
464	first464_0	China
130	first130_0	Australia
171	first171_0	Australia
8	first8_0	Japan
392	first392_0	China
492	first492_0	Japan
405	first405_0	China
305	first305_0	China
180	first180_0	Australia
445	first445_0	China

Combo Box Cell Type

[DropDownList](#)

This cell type serves the same purpose as the ComboBox control. The difference is that it associates 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 following are the list of properties that are applicable for this cell type.

[Properties](#)

GridStyleInfo	Description
CellType	Set to "ComboBox" for a Combo box control
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.
ShowButton	Boolean value indicating whether the drop-down button should appear or not.
StaysOpenOnEdit	Allows the drop-down list to open while editing.

The following are the list of events that are available for the DropDownList cell type.

Events

Events	Description	Arguments
DropDownSelectionChanged	This event will be raised when the SelectedItem of the combo box gets changed.	CellRowColumnIndex- This contains the RowColumn index of the cell where the combo box's value gets changed. SelectedItem- This is the Combo Box item which is currently selected.

We can change the drop-down list style by using the GridDropDownStyle property. This property features the following options:

- Editable—combines an editable text field and provides an option to type an item that may or may not be in the drop-down list. The text entered in the drop-down list is not case-sensitive.
- Autocomplete—predicts the word or phrase the user types so that the user need not type it completely.
- Exclusive—non-editable combo box where the user is allowed to select only the options that are available in the drop-down list.

The following is an example of how to define such a column.

XML

```
<syncfusion:GridTreeColumn Width="130"
HeaderText="Department"
MappingName="Department">
<syncfusion:GridTreeColumn.StyleInfo>
<syncfusion:GridDataStyleInfo CellType="DropDownList"
DisplayMember="Shipcity"
DropDownStyle="Editable"
ItemsSource="{StaticResource ShipDetails}"
ValueMember="Shipcity">
<syncfusion:GridDataStyleInfo.DropDownEdit>
<syncfusion:GridDropDownEditStyleInfo ShowButton="False"/>
</syncfusion:GridDataStyleInfo.DropDownEdit>
</syncfusion:GridDataStyleInfo>
</syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
this.treeGrid.Columns.Add(new GridTreeColumn()
{
MappingName = "Department",
StyleInfo = new GridStyleInfo()
{
CellType = "DropDownList",
ValueMember = "shipcity",
DisplayMember = "shipcity",
```

```

ItemsSource = ShipDetails,
DropDownStyle = GridDropDownStyle.Editable,
DropDownEdit = new GridDropdownEditStyleInfo()
{
    ShowButton = false,
}
}
});

```

The following screenshot shows a simple demo of the DropDownList cell type.

ID	Name	Department
174	BIGBOSS1	department3
413	first413_0	department2
459	first459_0	department0
26	first26_0	department2
464	first464_0	department0
130	first130_0	Shipcity Shipcountry
171	first171_0	department0 India
8	first8_0	department1 Australia
392	first392_0	department2 India
492	first492_0	department3 Japan
405	first405_0	department0
305	first305_0	department1
180	first180_0	department0
445	first445_0	department0

GridTree Control DropDownList Cell Type

CheckBox

The CheckBox cell type allows you to choose Boolean options. There are three options that the user can enter into the cell: true, false and null value. The cell value is bound with the IsChecked property of the CheckBox.

The following are the list of properties that are applicable for this cell type.

Properties

Property	Description
IsThreeState	Applying true to this property allows you to enter null value.

The following code shows how to define such a column.

XML

```

<syncfusion:GridTreeColumn MappingName="IsMajor">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridDataStyleInfo CellType="CheckBox" IsThreeState="False"/>
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>

```

C#

```

this.treeGrid.Columns.Add(new GridTreeColumn("IsMajor")
{
    StyleInfo = new GridStyleInfo()
    {
        CellType = "CheckBox",
        IsThreeState = false,
    }
});

```

The following screenshot shows a simple demo of CheckBox cell type.

Title	First Name	Last Name	Department	IsMajor
Management				
Vice President	Andrew	Fuller	Management	<input checked="" type="checkbox"/>
GM	Janet	Leverling	Management	<input checked="" type="checkbox"/>
Manager	Steven	Buchanan	Management	<input checked="" type="checkbox"/>
Accounts				
Accounts Manager	Nancy	Davolio	Accounts	<input checked="" type="checkbox"/>
Accountant	Margaret	Peacock	Accounts	<input checked="" type="checkbox"/>
Accountant	Michael	Suyama	Accounts	<input checked="" type="checkbox"/>
Accountant	Robert	King	Accounts	<input checked="" type="checkbox"/>
Sales				
Sales Manager	Laura	Callahan	Sales	<input checked="" type="checkbox"/>
Sales Representative	Anne	Dodsworth	Sales	<input checked="" type="checkbox"/>
Sales Representative	Albert	Hellstern	Sales	<input checked="" type="checkbox"/>
Sales Representative	Tim	Smith	Sales	<input checked="" type="checkbox"/>
Sales Representative	Justin	Brid	Sales	<input checked="" type="checkbox"/>

GridTree Control CheckBox Cell Type

CurrencyEdit

The CurrencyEdit cell type allows you to represent monetary values to maintain accuracy in calculations. It strips the currency sign in the cell and attempt to parse only the number from the input. Also, by using the GridCurrencyEditStyleInfo class we can customize the CurrencyEdit properties and appearance. Use the GridStyleInfo properties given below to customize these cells.

Properties

GridStyleInfo	Description
Cell Type	Sets to "CurrencyEdit".
CurrencyDecimalDigits	Number of decimal places in the 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.
MinValue	Least value can be set to the cell

MaxValue	Maximum value can be set to the cell
MinValidation	This Enum property is used to control the minimum value in Key Press and Lost focus modes.
MaxValidation	This Enum property is used to control the maximum value in Key Press and Lost focus modes.
IsScrollingOnCircle	This property allows you to increase/ decrease the cell value by using Up/Down keys in keyboard.

The following code snippet explains how to define a CurrencyEdit column.

XML

```
<syncfusion:GridTreeColumn Width="130" MappingName="Salary">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridDataColumnStyle CellType="CurrencyEdit" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("Salary")
{
    StyleInfo = new GridStyleInfo() { CellType = "CurrencyEdit" }
});
```

ID	FirstName	Salary
174	BIGBOSS1	\$200,000.00
413	first413_0	\$30,000.00
459	first459_0	\$80,000.00
26	first26_0	\$70,000.00
464	first464_0	\$60,000.00
130	first130_0	\$100,000.00
171	first171_0	\$70,000.00
8	first8_0	\$30,000.00
392	first392_0	\$50,000.00
492	first492_0	\$60,000.00
405	first405_0	\$110,000.00
305	first305_0	\$90,000.00
180	first180_0	\$110,000.00
445	first445_0	\$60,000.00

GridTree Control CurrencyEdit Cell Type

IntegerEdit

The IntegerEdit cell type is a specialized cell type that restricts the data entry to integer values. The following table contains the style properties specific to this cell type.

Properties

GridStyleInfo	Description
---------------	-------------

Cell Type	Sets to IntegerEdit.
NumberGroupSeparator	String that separates groups of digits.
NumberGroupSizes	Number of digits in each group.
MinValue	Least value can be set to the cell.
MaxValue	Maximum value can be set to the cell.
MinValidation	This Enum property is used to control the minimum value in Key Press and Lost focus modes.
MaxValidation	This Enum property is used to control the maximum value in Key Press and Lost focus modes.
IsScrollingOnCircle	This property allows you to increase/ decrease the cell value by using Up/Down keys in keyboard.

The following code demonstrates how to define an IntegerEdit column.

XML

```
<syncfusion:GridTreeColumn HeaderText="ID" MappingName="Employee ID">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridStyleInfo CellType="IntegerEdit" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("Employee ID")
{
    HeaderText="ID",
    StyleInfo = new GridStyleInfo() { CellType = "IntegerEdit" }
});
```

First Name	Last Name	Employee ID
Andrew	Fuller	2
Nancy	Davolio	1
Janet	Leverling	3
Tim	Smith	11
Caroline	Patterson	12
Margaret	Peacock	4
Steven	Buchanan	5
Michael	Suyama	6
Robert	King	7
Anne	Dodsworth	9
Laura	Callahan	8
Albert	Hellstern	10
Justin	Brid	13
Xavier	Martin	14
Laurent	Pereira	15

GridTree Control IntegerEdit Cell Type

DoubleEdit

The DoubleEdit cell type allows you to enter only values that are *double* into the cell. Thus it can be used to display System.Double type values. Also, by using the GridDoubleEditStyleInfo class you can customize the DoubleEdit properties and appearance. The style properties that affect this cell are given below.

Properties

GridStyleInfo	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
IsScrollingOnCircle	When value reaches the max value then the value will be set to the minimum value.
MinValue	Least value can be set to the cell
MaxValue	Maximum value can be set to the cell
MinValidation	This Enum property is used to control the minimum value in Key Press and Lost focus modes.
MaxValidation	This Enum property is used to control the maximum value in Key Press and Lost focus modes.
IsScrollingOnCircle	This property allows you to increase/ decrease the cell value by using Up/Down keys in keyboard.

The following code sample demonstrates how to define a DoubleEdit column.

XML

```
<syncfusion:GridTreeColumn MappingName="Salary" HeaderText="Amount">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridDataColumnStyle CellType="DoubleEdit"/>
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("Salary")
{
    HeaderText = "Amount",
    StyleInfo = new GridStyleInfo() { CellType = "DoubleEdit" }
});
```

ID	FirstName	Amount
174	BIGBOSS1	200,000.00
413	first413_0	30,000.00
459	first459_0	80,000.00
26	first26_0	70,000.00
464	first464_0	60,000.00
130	first130_0	100,000.00
171	first171_0	70,000.00
8	first8_0	30,000.00
392	first392_0	50,000.00
492	first492_0	60,000.00
405	first405_0	110,000.00
305	first305_0	90,000.00
180	first180_0	110,000.00
445	first445_0	60,000.00
355	first355_0	30,000.00

DoubleEdit Cell Type

PercentEdit

The PercentEdit cell type restricts the data entry to percentage values only. The following table describes the style properties available with this cell type.

Properties

GridStyleInfo	Description
Cell Type	Sets to PercentEdit.
PercentEditMode	Indicates how the the text can be edited in the PercentEdit cells.Possible values: <i>PercentMode</i> and <i>DoubleMode</i> .
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.
MinValue	Least value can be set to the cell
MaxValue	Maximum value can be set to the cell
MinValidation	This Enum property is used to control the minimum value in Key Press and Lost focus modes.
MaxValidation	This Enum property is used to control the maximum value in Key Press and Lost focus modes.
IsScrollingOnCircle	This property allows you to increase/ decrease the cell value by using Up/Down keys in keyboard.

The following code sample demonstrates how to define a PercentEdit column.

XML

```
<syncfusion:GridTreeColumn MappingName="Hike">
<syncfusion:GridTreeColumn.StyleInfo>
<syncfusion:GridStyleInfo HorizontalAlignment="Right" CellType="PercentEdit"
>
<syncfusion:GridStyleInfo.PercentEdit>
<syncfusion:GridPercentEditStyleInfo IsScrollingOnCircle="False" />
</syncfusion:GridStyleInfo.PercentEdit>
</syncfusion:GridStyleInfo>
</syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
GridTreeColumn percentColumn = new GridTreeColumn("Hike")
{
    StyleInfo = new GridStyleInfo()
    {
        CellType = "PercentEdit",
        PercentEdit = new GridPercentEditStyleInfo()
        {
            IsScrollingOnCircle=false,
        }
    }
};
treeGrid.Columns.Add(percentColumn);
```

ID	FirstName	Completeness %
68	first68_0	34.00 %
75	first75_0	68.00 %
55	first55_0	75.00 %
88	first88_0	75.00 %
73	first73_0	68.00 %
20	first20_0	73.00 %
67	first67_0	73.00 %
83	first83_0	73.00 %
60	first60_0	73.00 %
72	first72_0	68.00 %
78	first78_0	72.00 %
85	first85_0	34.00 %
3	first3_0	85.00 %
13	first13_0	3.00 %

GridTree Control PercentEdit Cell Type*[DateTimeEdit](#)*

The DateTimeEdit cells incorporate the DateTimeEdit controls in the grid cells that will help you interactively set a date and time value. The style properties in the following table are applicable to this cell type.

[DateTime Edit Cell Properties](#)

GridStyleInfo	Description
---------------	-------------

CellType	Sets to DateTimeEdit.
DateTimePattern	Sets the date-time pattern. (The next table lists the available patterns with examples.)
MaxDateTime, MinDateTime	Sets the maximum and minimum values for a DateTime cell.
IsCalendarEnabled	Enables the calendar pop-up when set to <i>true</i> .
IsWatchEnabled	Enables the watch pop-up when set to <i>true</i> .
NoneDateText	Specifies the text to be displayed when no date is set.
IsButtonPopUpEnabled	By setting true/false you can disable the button to change the date-time value.
IsEditable	By setting true//false you can enable/disable editing of the cell
IsEmptyDateEnabled	You can leave a cell with an empty value, when you set this property to <i>true</i> .
IsEnabledRepeatButton	The RepeatButton can be disabled by applying <i>false</i> to this property.
IsVisibleRepeatButton	The visibility of the Repeat Button can be enabled by setting the <i>IsEnabledRepeatButton</i> property to <i>true</i> .
NoneDateText	Setting a string value, displays when there is no date set.
PopupDelay	Setting time limit to this property opens the PopUp in the given time delay.
RepeatButtonBackground	Applying a color to this property sets as background of RepeatButton.

DateTime Patterns for DateTime Edit Cells

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

The following code sample demonstrates how to define a column with DateTime Edit cells.

XML

```
<syncfusion:GridTreeColumn HeaderText="DOB" MappingName="Birth Date">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridStyleInfo CellType="DateTimeEdit" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("Birth Date")
{
    HeaderText="DOB",
    StyleInfo = new GridStyleInfo() { CellType = "DateTimeEdit" }
});
```

First Name	Last Name	DOB
Andrew	Fuller	2/19/1942
Nancy	Davolio	12/8/1948
Janet	Leverling	8/30/1963
Tim	Smith	6/6/1973
Caroline	Patterson	
Margaret	Peacock	
Steven	Buchanan	
Michael	Suyama	
Robert	King	
Anne	Dodsworth	
Laura	Callahan	
Albert	Hellstern	
Justin	Brid	
Xavier	Martin	11/30/1960
Laurent	Pereira	12/9/1965

DateTime Edit Cell Type

MaskEdit

The MaskEdit cell type allows you to create specially formatted text cells that conform to an edit mask that you specify. The Style.MaskEdit.Mask property holds the mask string, which controls the format of the input text. The MaskEdit cells are useful when the user wants to display some formatted text such as a social security number, telephone number, etc.

The following table lists the style properties specific to this cell type.

Properties

Property	Description
CellType	Sets to MaskEdit
Mask	Sets to a string value to mask the cell value.

MaxLength	Number of maximum char allowed to entered in the cell.
MinLength	Number of minimum char allowed to entered in the cell.
StringValidation	When the cell value or mask is set to null then the cell is validated based on this property value.

The following code sample demonstrates how to define a MaskEdit column.

XML

```
<syncfusion:GridTreeColumn MappingName="LastName">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridDataStyleInfo CellType="MaskEdit" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("LastName")
{
    HeaderText = "Last Name",
    StyleInfo = new GridStyleInfo() { CellType = "MaskEdit" }
});
```

ID	FirstName	LastName
174	BIGBOSS1	(+37) 24-235752
413	first413_0	(+82) 14-237488
459	first459_0	(+42) 30-231480
26	first26_0	(+8) 24-237603
464	first464_0	(+72) 42-233413
130	first130_0	(+42) 23-230109
171	first171_0	(+15) 11-238485
8	first8_0	(+6) 17-230773
392	first392_0	(+92) 16-237633
492	first492_0	(+27) 37-230339
405	first405_0	(+98) 42-230597
305	first305_0	(+92) 16-238910
180	first180_0	(+65) 13-233582
445	first445_0	(+16) 28-235656
355	first355_0	(+79) 16-232593

MaskEdit Cell Type

[UpDownEdit](#)

The UpDownEdit cell type hosts an UpDownEdit control which contains a pair of arrow buttons that increases or decreases the cell value. The style properties applicable to this cell type are provided in the following table.

[UpDownEdit Cell Properties](#)

GridStyleInfo}	Description
----------------	-------------

Cell Type	Sets 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.
NegativeForeground	Color to differentiate the negative value.

The following sample demonstrates how to define a column with UpDownEdit cells.

XML

```
<syncfusion:GridTreeColumn MappingName="Rating">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridStyleInfo CellType="UpDownEdit" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
GridTreeColumn updownColumn = new GridTreeColumn("Rating")
{
    StyleInfo = new GridStyleInfo()
    {
        CellType = "UpDownEdit",
        UpDownEdit = new GridUpDownEditStyleInfo()
        {
            NegativeForeground = Brushes.Red
        }
    }
};
treeGrid.Columns.Add(updownColumn);
```

ID	FirstName	Count
▲68	first68_0	34.00
▲75	first75_0	68.00
55	first55_0	75.00
88	first88_0	75.00
▲73	first73_0	68.00
20	first20_0	73.00
67	first67_0	73.00
83	first83_0	73.00
60	first60_0	73.00
▲72	first72_0	68.00
78	first78_0	72.00
▲85	first85_0	34.00
▲3	first3_0	85.00
13	first13_0	3.00

UpDownEdit Cell Type

[TimeSpanEdit](#)

The TimeSpanEdit cell type is used to display time value in the Day:Hour:Min:Sec format and also in custom format. The fields can be incremented and decremented by using the up and down arrow keys.

[TimeSpanEdit properties](#)

Properties	Description
AllowNull	Allows to set null value.
Format	Specifies the format to display the time.
ShowArrowButtons	Displays the arrow buttons to change the value by mouse.

The following code sample demonstrates how to define a column with TimeSpanEdit cells.

XML

```
<syncfusion:GridTreeColumn HeaderText="Time Avail" MappingName="Time">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridDataColumnStyle CellType="TimeSpanEdit" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("Time")
{
    HeaderText = "Time Avail",
    StyleInfo = new GridStyleInfo() { CellType = "TimeSpanEdit" }
});
```

ID	FirstName	Time Avail
34	BIGBOSS1	10.14:15:05
68	first68_0	18.12:05:59
75	first75_0	12.04:30:06
55	first55_0	4.02:22:26
88	first88_0	9.22:41:41
73	first73_0	23.22:51:51
20	first20_0	2.10:53:41
67	first67_0	4.18:15:50
83	first83_0	28.11:21:00
60	first60_0	8.20:43:27
72	first72_0	2.05:05:18
78	first78_0	20.02:47:46
85	first85_0	14.09:26:39
3	first3_0	29.19:40:57

TimeSpanEdit Cell Type

ImageCell

The ImageCell type is used to load images inside the graphic cells. To load the graphic image cell in the GridTree control, you have to set the CellType as ImageCell and the CellValue as BitmapImage.

Image CellType properties

Property	Description
Image	Sets the image path to display.
ImageContentAlignment	Sets the alignment of the image (Left or Right).
ImageHeight	Sets the height of the image.
ImageWidth	Sets the width of the image.
ImageMargins	Sets the margin of the image.

The following code sample demonstrates how to define a column with Image cells.

XML

```
<syncfusion:GridTreeColumn MappingName="Image" HeaderText="Progress">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridStyleInfo CellType="ImageCell" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeColumn("Image")
{
    HeaderText = "Progress",
    StyleInfo = new GridStyleInfo() { CellType = "ImageCell" }
});
```

ID	FirstName	Progress	ReportsTo
34	BIGBOSS1		-1
68	first68_0		34
75	first75_0		68
55	first55_0		75
88	first88_0		75
73	first73_0		68

Image Cell Type

Hyperlink

This cell type allows you to perform an operation when click on the cell like navigation from your application to a site or a window.

The following are the list of events that are available for this cell type.

Events

Events	Description	Arguments
CellRequestNavigate	The event rises when the mouse clicks the hyperlink cell value. You can perform operation that need to be done when a hyperlink is clicked.	ColumnIndexâ€™contains the current mouse click hyperlink cell column index.RowIndexâ€™contains the current mouse click hyperlink cell row index.Uriâ€™the Uri where you want to navigate should be applied here.

The following code shows how to define such a column.

XML

```
<syncfusion:GridTreeColumn HeaderText="Department" MappingName="Department">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridStyleInfo CellType="Hyperlink" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
this.treeGrid.Columns.Add(new GridTreeColumn("Department")
{
    StyleInfo = new GridStyleInfo()
    {
        CellType = "Hyperlink",
    }
});
```


The following screenshot shows a simple demo of Hyperlink cell.

Title	First Name	Last Name	Department
Management			
Vice President	Andrew	Fuller	Management
GM	Janet	Leverling	Management
Manager	Steven	Buchanan	Management
Accounts			
Accounts Manager	Nancy	Davolio	Accounts
Accountant	Margaret	Peacock	Accounts
Accountant	Michael	Suyama	Accounts
Accountant	Robert	King	Accounts
Sales			
Sales Manager	Laura	Callahan	Sales
Sales Representative	Anne	Dodsworth	Sales
Sales Representative	Albert	Hellstern	Sales

GridTree Control Hyperlink Cell Type

Button

The Button cell type allows you to load Button control in each cell of the GridTree control. To load a Button in the GridTree control cell, choose Button cell type. You can perform button click action as in the Button control.

The following table shows the events that are available for this cell type.

Events

Events}	Description	Arguments
CellButtonClick	Fires when you click on the button cell.	It contains the following arguments. ColumnIndexâ€”contains the column index where the mouse clicks on the button cell. RowIndexâ€”contains the row index where the mouse clicks on the button cell.

The following code snippet shows a simple demo of Button cell type.

XML

```
<syncfusion:GridTreeColumn HeaderText="Department" MappingName="Department">
  <syncfusion:GridTreeColumn.StyleInfo>
    <syncfusion:GridStyleInfo CellType="Button" />
  </syncfusion:GridTreeColumn.StyleInfo>
</syncfusion:GridTreeColumn>
```

C#

```
this.treeGrid.Columns.Add(new GridTreeColumn("Department")
{
    StyleInfo = new GridStyleInfo()
{
```

```
CellType = "Button",
}
});
```

The following screenshot shows a simple demo of Button cell type.

Title	First Name	Last Name	Department
Management			
Vice President	Andrew	Fuller	Management
GM	Janet	Leverling	Management
Manager	Steven	Buchanan	Management
Accounts			
Accounts Manager	Nancy	Davolio	Accounts
Accountant	Margaret	Peacock	Accounts
Accountant	Michael	Suyama	Accounts
Accountant	Robert	King	Accounts
Sales			
Sales Manager	Laura	Callahan	Sales
Sales Representative	Anne	Dodsworth	Sales
Sales Representative	Albert	Hellstern	Sales
Sales Representative	Tim	Smith	Sales
Sales Representative	Justin	Brid	Sales

Button Cell Type

Unbound Columns

The GridTree control supports unbound columns in which the data is calculated automatically according to a specified formula, format, or custom data from users.

The GridTree control provides support for adding additional columns which is not in underlying business object. Such additional columns are called as unbound columns, as they do not belong to the data source. These unbound fields can be used when you want to add some additional or custom information to the nodes.

You can create an unbound column by instantiating the GridTreeUnboundColumn class, which is a derivative of GridTreeColumn.

Also, you can utilize the features that are available in the bound columns, such as sorting, cell type and customization.

Properties

Property	Description	Type	Data Type
Expression	Allows the user to set formula to compute the value for the unbound column based on the other column values.	Dependency Property	String

Methods

Method Name	Description	Parameters	Return Type
-------------	-------------	------------	-------------

GetUnboundColumnValue	Gets the unbound column value from a specific record and column.	(object Record, GridTreeUnboundColumn Column)	Object
-----------------------	--	---	--------

Events

Event Name	Description	Arguments
QueryUnboundCellInfo	This event fires when the style of an unbound column is created. Using this event you can modify the styles of the unbound column.	(RowColumnIndex Cell, GridTreeUnboundColumn Column, GridStyleInfo Style)
QueryUnboundColumnValue	This event fires, before the value of the unbound column is evaluated by using Expression and Format. If the user sets the value in this event then the value will not be evaluated using Expression and Format. If the user cancels this event, then the value is not set in the appropriate cell.	

The following code helps you to display the custom values in the unbound columns:

C#

```
treeGrid.InternalGrid.QueryUnboundColumnValue += new GridTreeQueryUnboundColumnEventHandler(InternalGrid_QueryUnboundColumnValue);
void InternalGrid_QueryUnboundColumnValue(object sender, GridTreeUnboundColumnEventArgs Args)
{
    Args.Style.CellValue = "Value";
}
```

When the user handles this event and applies cell value then the values calculated by Expression or Format is not displayed in the unbound column.

Format

A Format is a string that contains a format to display data of a column in a specified unbound column, as formatted.

This property is used to apply a format to display a value with other column values in the unbound column.

Some example of Format is given below:

- Format="{Cost:c}"
- Format="{Cost:c} of {Model}"

The following code snippet shows a simple usage of Format.

XML

```

<syncfusion:GridTreeUnboundColumn Format="{Cost:c}"
HeaderText="Cost"
MappingName="CarCost">
<syncfusion:GridTreeUnboundColumn.StyleInfo>
<syncfusion:GridStyleInfo HorizontalAlignment="Right" />
</syncfusion:GridTreeUnboundColumn.StyleInfo>
</syncfusion:GridTreeUnboundColumn>
<syncfusion:GridTreeUnboundColumn Format="{Cost:c} of {Model}"
HeaderText="Cost Per Model"
MappingName="ModelCost">
<syncfusion:GridTreeUnboundColumn.StyleInfo>
<syncfusion:GridStyleInfo HorizontalAlignment="Right" />
</syncfusion:GridTreeUnboundColumn.StyleInfo>
</syncfusion:GridTreeUnboundColumn>

```

C#

```

treeGrid.Columns.Add(new GridTreeUnboundColumn() {
MappingName = "CarCost", HeaderText = "Cost", Format = "{Cost:c}" });
treeGrid.Columns.Add(new GridTreeUnboundColumn() {
MappingName = "ModelCost", HeaderText = "Cost Per Model", Format = "{Cost:c}
of {Model}" });

```



Title	ID	Cost	Cost Per Model
BMW			
M3	9	\$68,550.00	\$68,550.00 of BMW-M3
Z4	10	\$48,650.00	\$48,650.00 of BMW-Z4
M5	11	\$85,700.00	\$85,700.00 of BMW-M5
Audi			
A3	12	\$30,850.00	\$30,850.00 of Audi-A3
A5	13	\$44,700.00	\$44,700.00 of Audi-A5
Q7	14	\$51,450.00	\$51,450.00 of Audi-Q7
R8	15	\$136,800.00	\$136,800.00 of Audi-R8
Porsche			
911	16	\$185,000.00	\$185,000.00 of Porsche-911
977	18	\$195,000.00	\$195,000.00 of Porsche-997
Volkswagen			
Beetle	21	\$20,390.00	\$20,390.00 of Volkswagen-Be...
Jetta	22	\$19,545.00	\$19,545.00 of Volkswagen-Je...

Using Formats in the GridTree Control*Expression*

An Expression is a string contains formula to calculate values to display in the unbound columns.

This property is used to set an expression formula for the unbound column, based on which the data is populated in the unbound column. Expression consists of MappingName, constants, and operators.

The following table lists the operators that are supported with example for each.

Operators

Expression	Syntax	Description	Example Usage
------------	--------	-------------	---------------

Mod	%	Divides the first argument by the second argument and returns the remainder.	[UnitPrice] % 10
Multiplication, Division	*,/	Multiplies/divides first argument by the second argument.	[QunatityPerUnit] * [UnitsInStock]
Addition, Subtraction	+,-	Adds the first argument with the second argument or subtracts the second argument from the first one.	[UnitsInStock]+[Quantity]
Or	OR	Returns 1, if either the first argument or the second one returns <i>true</i> .	[Val]=50 OR [Val]=100
And	AND	Returns 1, if both the parameters return <i>true</i> .	[Val]< 50 AND [Val]>100
Less than	<	Returns <i>true</i> , if the first parameter is less than the second one.	[OrderID] < 2000
Greater than	>	Returns <i>true</i> , if the first parameter is greater than the second one.	[OrderID] > 2500
Less than Or Equal to	<=	Returns <i>true</i> , if the first parameter is less than or equal to the second one.	[OrderID] <= 2050
Greater than Or Equal to	>=	Returns <i>true</i> , if the first parameter is greater than or equal to the second one.	[OrderID] >= 2056
Equal	=	Returns <i>true</i> , if both the arguments have the same value.	[CustomerID] = 90
Not Equal to	<>	Returns <i>true</i> , if both the arguments does not have the same value.	[CustomerID] <> 95
StartsWith	StartsWith	Returns <i>true</i> , if the value starts with the given string.	ProductName StartsWith
EndsWith	EndsWith	Returns <i>true</i> , if the value ends with the specified string.	ProductName EndsWith i
Contains	Contains	Returns <i>true</i> , if the value contains the specified string.	ProductName Contains

The following code snippet shows a simple way to use expressions.

XML

```
<syncfusion:GridTreeUnboundColumn Expression="Quantity*Cost"
HeaderText="Grand Total"
MappingName="GrandTotal"
PercentWidth="2">
<syncfusion:GridTreeUnboundColumn.StyleInfo>
<syncfusion:GridStyleInfo HorizontalAlignment="Right" CellType="CurrencyEdit"
"/>
</syncfusion:GridTreeUnboundColumn.StyleInfo>
</syncfusion:GridTreeUnboundColumn>
```

C#

```
treeGrid.Columns.Add(new GridTreeUnboundColumn() { MappingName = "GrandTotal",
Expression = "Quantity*Cost" });
```

Title	Cost	Quantity	Grand Total
BMW			
M3	\$68,550.00	5	\$342,750.00
Z4	\$48,650.00	15	\$729,750.00
M5	\$85,700.00	2	\$171,400.00
Audi			
A3	\$30,850.00	15	\$462,750.00
A5	\$44,700.00	16	\$715,200.00
Q7	\$51,450.00	19	\$977,550.00
R8	\$136,800.00	24	\$3,283,200.00
Porsche			
911	\$185,000.00	14	\$2,590,000.00
977	\$195,000.00	18	\$3,510,000.00

Using Expressions in the GridTree Control

Interactive Features in WPF GridTreeControl (Classic)

This section elaborates on the following run time interactive features:

- Selection
- Sorting
- Column Sizing

Selection Support

GridTree control has the following two types of selection support:

- Whole Node selection-Whole node selections involve selecting the the entire row when a cell is clicked, and is enabled by setting the EnableNodeSelection property to true. This is the default node selection type in the Grid Tree.
- Cell Range selection-The cell range selection support allows the selection of cell ranges within the GridTree control. This support is enabled by setting EnableNodeSelection property to *false*.

Disabling the Selection

To disable all selection support in the GridTree control, set GridTreeControl.EnableSelections to *false*.

Selection Features

GridTree control does not use the selection support inherited from the Grid control, because the selections in the Grid Tree need to be persisted, as the nodes are expanded/collapsed and sorted. The GridTreeNode.IsSelected property indicates whether the node is selected or not and the GridTreeNode.SelectedColumns property contains the names of the columns selected for the node. You can access selected nodes by using the GridTreeControl.SelectedNodes property.

The following code example illustrates cell range selections in the Grid Tree.

C#

```
foreach (GridTreeNode node in treeGrid.SelectedNodes)
{
    foreach (string columnName in node.SelectedColumns)
    {
        Console.WriteLine("{0} ", treeGrid.InternalGrid.GetValueFromNode(columnName, node));
    }
    Console.WriteLine();
}
```

Sorting

Sorting will arrange the records either in ascending or descending order of the selected field values. The GridTree control allows you to sort the data against one or more columns. The number of columns on which the sorting can be applied is unlimited.

Properties

Property	Description	Type	Data Type
AllowSort	Enables/disables the sorting feature in the GridTree control.	Dependency property	Boolean
EnableMultiColumnSorting	Enables/disables the multicolumn sorting.	Dependency property	Boolean
SortingOptions	It is an enum property used to prevent the sorting on property changes.	Dependency property	Enum of type GridTreeSortingOptions with the following named constants: DefaultDisableSortingOnPropertyChange

EnableTriStateSorting	Enables/disables the tri-state sorting.	Dependency	Boolean
-----------------------	---	------------	---------

Methods

Method	Description	Parameters	Parameter Description	Return Type
SortTree	Used to sort a particular column.	Overloads: 1) (string colName, ListSortDirection direction) 2) (string colName, ListSortDirection direction, bool clearSort)	ColNameâ€”name of the column to be sorted.directionâ€”describes the direction of the sort (ascending or descending).clearSortâ€”checks whether any existing sort should be clear before the new sort is applied.	Void

The sorting can be enabled by using the AllowSort property. The following code example explains this.

C#

```
this.treeGrid.AllowSort = true;
```

XML

```
<syncfusion:GridTreeControl Name="treeGrid"
AllowSort="True"
ExpandStateAtStartup="RootNodesExpanded"
VisualStyle="Metro">
```

Sorting Options

The GridTree control supports the following sorting options:

- Multicolumn Sorting
- Tri-state Sorting
- Custom Sorting

Multicolumn Sorting

The GridTree control provides support to perform multicolumn sorting by holding down the Ctrl key and clicking the left mouse button.

The multicolumn sorting can be enabled by using the EnableMultiColumnSorting property. The following code example illustrates this.

C#

```
this.treeGrid.EnableMultiColumnSorting = true;
```


XML

```
<syncfusion:GridTreeControl Name="treeGrid"
AllowSort="True"
EnableMultiColumnSorting="True"
ExpandStateAtStartup="RootNodesExpanded"
VisualStyle="Metro">
```

The following screenshot shows the GridTree control enabled with multicolumn sorting where the First Name column is sorted in ascending order and the Last Name column is sorted in descending order.

First Name ▲	Last Name ▼	Title	Department
BIGBOSS1	last34913	title4	department2
first34994_0	last34994	title3	department0
first40715_0	last40715	title4	department2
first21752_0	last21752	title2	department1
first30126_0	last30126	title0	department1
first40427_0	last40427	title4	department0
first11428_0	last11428	title4	department1
first35132_0	last35132	title0	department2
first48239_0	last48239	title1	department3
first4945_0	last4945	title4	department3
first66174_0	last66174	title3	department3
first76368_0	last76368	title4	department0

Multicolumn Sorting in GridTree Control

Tri-state Sorting

The GridTree control also supports the tri-state sorting. There are three states in this sorting:

- Ascending—sorts the records in an increasing order.
- Descending—sorts the records in a decreasing order.
- No Sorting—it retains the records in their original state.

This sorting can be enabled by using the EnableTriStateSorting property. The no sorting state will occur after the descending state. The following code example illustrates how to enable the tri-state sorting.

C#

```
this.treeGrid.EnableTriStateSorting = true;
```

XML

```
<syncfusion:GridTreeControl Name="treeGrid"
AllowSort="True"
EnableMultiColumnSorting="True"
EnableTriStateSorting="True"
ExpandStateAtStartup="RootNodesExpanded"
VisualStyle="Metro">
```

Custom Sorting

The GridTree control also features the custom sorting. The following additional members of GridTreeControlImpl allow you to access sort information and provide support for the custom sorting.

- `public bool IsPropertySorted(string propertyName, out SortState state)`—determines whether a particular column is sorted.
- `public IComparer SortComparer`—allows to perform the custom sorting. The default implementation assumes the underlying node items implement the `IComparable` technique and uses that implementation for the sorting comparisons within the columns. If your objects are not sortable using this technique, then you need to provide a sort comparer that properly sorts the grid nodes items based on the sort property values.
- `public string SortProperty`—it is a string that holds the column name to be sorted. You can specify a sort direction by appending a space followed by either `ASC` or `DESC`. In addition, you can specify the multicolumn sorting by passing several columns separated by commas. For example, `"Price ASC, Weight DESC"`, which will indicate to sort the Price column in ascending order, and then sort the Weight column in descending order.
- `public List SortStates`—it is a list of sort states for the columns currently sorted in the GridTree control. The `SortState` class contains information regarding the direction and the property sorted, and it exposes static helper methods, which takes care of the changes between `SortStates` and `SortProperty`.

Column Sizing

The GridTree control supports auto sizing its columns such that the display of the tree occupies the entire width of the client area available in the Grid Tree. The sizing is done by columns occupying certain percentages of the available space. To implement this feature, the Grid Tree must be free to size with its parent.

Note: You cannot set the `Width` or `HorizontalAlignment` properties of the Grid Tree when this feature has been enabled.

You can enable this feature in two ways.

The first is to set the `GridTreeControl.PercentSizingBehavior` to any value except `"None"`.

The second action is to populate the `GridTreeControl.Columns` collection, and to set the `GridTreeColumn.PercentWidth` property for each of the columns that have to be auto sized as the Grid Tree is sized.

The following code example illustrates these settings.

XML

```
<syncfusion:GridTreeControl Name="gridTreeControl2" Grid.Row="1"
RequestTreeItems="gridTreeControl2_RequestTreeItems"
PercentSizingBehavior="SizeUntouchedColumns">
  <syncfusion:GridTreeControl.Columns>
    <syncfusion:GridTreeColumn MappingName="Title" Width="180"/>
    <syncfusion:GridTreeColumn MappingName="FirstName" PercentWidth="1"/>
    <syncfusion:GridTreeColumn MappingName="LastName" PercentWidth="1"/>
  </syncfusion:GridTreeControl.Columns>
</syncfusion:GridTreeControl>
```

After setting the above properties, the Grid Tree will display three columns with the “Title” column having a fixed width of 180. The other two columns would be equally sized to fill the remaining client area.

When the PercentWeight property is enabled, the column will occupy a certain percentage of the remaining client area (after all the fixed-sized columns have been allocated). This percentage is calculated by dividing the PercentWeight property by the sum of all the PercentWeight’s present in the Columns collection.

For example, in the preceding code, if you want the width of the LastName column to be three times the width of the FirstName column, then you have to set its PercentWeight property to 3.

The PercentSizingBehavior property provides the following options to size the columns. They are:

- **None**—No automatic sizing will be done. The displayed column width will either be the default value (gridTreeControl1.DefaultColumnWidth), or the width specified in the GridTreeColumn.Width property. This is the default behavior.
- **SizeUntouchedColumns**—This option will autosize the columns which have the PercentWidth property set, as long as the user does not explicitly change the width of this column through the UI. If the user changes the column size, the column width will not be changed as the Grid Tree is sized.
- **NoSizingIfAnyTouched**—If the user sets the size of any column through the UI, all auto sizing stops and the column size will not be changed as the Grid Tree is sized.
- **SizeAlwaysPercent**—This option will not allow the user to size any column. The columns will always use the percentage sizing to determine their size as the Grid Tree is sized.

Update Mode

The UpdateMode property defines the behavior when updating data to the underlying collection while editing. Below are the three UpdateMode behaviors supported by the GridTree control.

Lost Focus—Update the values to the underlying collection when the current cell is out of focus or moved to another cell. Validation also will take place on lost focus of the cell.

PropertyChanged—Update the values to the underlying collection when a single value of a property is changed. Validation will also take place for every change.

RowCachedMode—Update the values to the underlying collection when the focus is moved to the next row. Validation will take place on lost focus of the cell.

Use Case Scenarios

UpdateMode is mainly used to control the value updates to the bound collection.

Properties

Property	Description}	Type	Data Type
UpdateMode	Defines the behavior when updating data to the underlying collection while editing.	Dependency	UpdateMode

Adding UpdateMode to an Application

The following code snippet explains how to set UpdateMode in XAML.

XML

```
<syncfusion:GridTreeControl
Name="treeGrid"
UpdateMode="LostFocus"
NotifyPropertyChanges="True"
ItemsSource="{Binding Source={StaticResource theWindowDataSource}}"
ChildPropertyName="Children"
VisualStyle="Office14Blue">
```

The following code sample shows how to set UpdateMode in C#.

C#

```
this.treeGrid.UpdateMode = UpdateMode.LostFocus;
```

Appearance

Styles Support

This section will elaborate on different style settings.

Visual Styles

The GridTreeControl.VisualStyle property is used to specify the visual style for the Grid Tree. This property is also bound through the Grid Tree Template to the SkinStorage.VisualStyle property, so it can participate in the general themed appearance of all Syncfusion WPF controls. You can also turn off the visual style support in the GridTree control by setting the SupportVisualStyles property to false.

Style Object

In addition to the GridTreeControl.VisualStyle property, there is a GridTreeControl.LevelStyles collection that enables you to specify a GridStyleInfo object to customize the appearance of all the cells at a particular level.

The following code example illustrates how to apply LevelStyles to the Grid Tree.

C#

```
// Set some miscellaneous level colors so they are easily seen (just set up to 6 levels...).
byte k = 150;
byte k1 = 250;
for (int i = -1; i < 7; ++i)
{
    GridStyleInfo style = new GridStyleInfo();
    style.Background = new SolidColorBrush(Color.FromArgb(255, 239, k1, k));
    gridTreeControl1.LevelStyles.Add(style);
    k += 15;
    k1 -= 15;
}
```

The following screen shot shows the back color set on a level-by-level basis, with the header cells styles set by using the VisualStyle property.

ID	FirstName	LastName	Department	Title	Salary	ReportsTo
174	BIGBOSS1	last174	department3	title1	200000	-1
173	first173_0	last173	department0	title2	80000	174
497	first497_0	last497	department0	title1	80000	173
434	first434_0	last434	department3	title4	70000	497
159	first159_0	last159	department1	title2	100000	434
442	first442_0	last442	department2	title3	70000	434
120	first120_0	last120	department3	title1	110000	434
371	first371_0	last371	department2	title4	60000	497
140	first140_0	last140	department3	title2	40000	497
131	first131_0	last131	department0	title3	110000	140
87	first87_0	last87	department1	title0	100000	140
348	first348_0	last348	department2	title4	50000	140
150	first150_0	last150	department0	title2	40000	140
258	first258_0	last258	department0	title0	40000	497
167	first167_0	last167	department1	title0	60000	497
204	first204_0	last204	department3	title4	50000	174
148	first148_0	last148	department1	title4	100000	204

Level Styles

Grid cell background is customized on a level-by-level basis.

Setting Style for the Column

You can also control the appearance of cells in a particular column by setting the GridStyleInfo values held in the GridTreeColumn.StyleInfo property for a particular column.

The following code example illustrates how to set this property for a column in the Grid Tree.

C#

```
GridTreeColumn tc = new GridTreeColumn("Department", "Department", 100);
tc.StyleInfo.Background = Brushes.Azure;
gridTreeControl1.Columns.Add(tc);
```

The following screen shot shows the StyleInfo property applied to the “Department” column in the Grid Tree.

First Name	Last Name	Department	Salary
BIGBOSS1	last174	department3	200,000
first173_0	last173	department0	80,000
first497_0	last497	department0	80,000
first434_0	last434	department3	70,000
first159_0	last159	department1	100,000
first442_0	last442	department2	70,000
first120_0	last120	department3	110,000
first371_0	last371	department2	60,000
first140_0	last140	department3	40,000
first258_0	last258	department0	40,000
first167_0	last167	department1	60,000
first204_0	last204	department3	50,000
first431_0	last431	department0	70,000
first413_0	last413	department2	30,000

Column Styles

Grid cell background for the Department column is customized.

Setting Style for a Cell

To specify the style for a particular cell, you need to handle the QueryCellInfo event on the embedded GridTreeControlImpl. The following code example illustrates this.

Here is the code for coloring a particular cell.

C#

```
// Subscribe to the event.
gridTreeControl1.Model.QueryCellInfo += new
GridQueryCellInfoEventHandler(Model_QueryCellInfo);
// Event handler
void Model_QueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    if (e.Cell.RowIndex > 0) //skip header
    {
        // Get the node.
        GridTreeNode node =
        gridTreeControl1.InternalGrid.GetNodeAtRowIndex(e.Cell.RowIndex);
        if (node != null && node.Item != null)
        {
            // Cast it to the appropriate type.
            Employee emp = node.Item as Employee;
            if (emp != null)
            {
                // Pick out the cell by employee and column that you want to style.
                // For example, here we color the Department of the employee whose ID is
                159.
                if (emp.ID == 159)
                {
                    string name =
                    gridTreeControl1.InternalGrid.ColumnIndexToName(e.Cell.ColumnIndex);
                    if (name == "Department")
                    {
                        e.Style.Background = Brushes.Red;
                        e.Handled = true;
                    }
                }
            }
        }
    }
}
```

First Name	Last Name	Department	Salary
BigBoss1	last174	department3	200,000
first173_0	last173	department0	80,000
first497_0	last497	department0	80,000
first434_0	last434	department3	70,000
first159_0	last159	department1	100,000
first442_0	last442	department2	70,000
first120_0	last120	department3	110,000
first371_0	last371	department2	60,000
first77_0	last77	department1	80,000
first391_0	last391	department3	50,000
first16_0	last16	department3	30,000

Cell Style

Thus, the background of a grid cell can be customized.

Note: Level styles are the lowest in precedence, followed by column styles, and then followed by cell - specific styles set in QueryCellInfo.

Customizing the ExpandCell

The GridTreeControl.InternalGrid has a ExpandGlyphType property that enables you to customize the appearance of the Expand cell. It also has a GridTreeControl.InternalGrid.SetExpandBrushesAndPen method that is used to provide the brushes used to draw the expand glyphs. The ExpandGlyph property has the following settings.

GridTree control Property

GridTree control Property	Description
Triangle	The glyph is displayed as a triangle. The color of the triangle is controlled either through the GridTreeControl.VisualStyle property or by explicitly calling the SetExpandBrushesAndPen method. This is the default setting.
PlusMinus	The glyph is displayed as a little square containing either a plus sign or a minus sign. The color of the glyph is controlled either through the GridTreeControl.VisualStyle property or by explicitly calling the SetExpandBrushesAndPen method.
PlusMinusLines	The glyph is displayed as a little square containing either a plus sign or a minus sign. In addition to the plus-minus squares, tree lines are drawn. The displayed colors are controlled either through the GridTreeControl.VisualStyle property or by explicitly calling the SetExpandBrushesAndPen method.
Themed	The Themed button seen in the GridData control is displayed. Its appearance is completely determined by the GridTreeControl.VisualStyle property.
Custom	The Grid Tree will not draw its own glyph. Instead, an event on the cell renderer is triggered, so that you can provide a Geometry object at that point which the renderer will draw to display your glyph. The color of the

	glyph is controlled either through the GridTreeControl.VisualStyle property or by explicitly calling the SetExpandBrushesAndPen method.
--	---

Here are some sample code snippets indicating how you can subscribe to the required event, and use the event handler to provide a PathGeometry object that will be drawn.

PathGeometry Class-Represents a complex shape that may be composed of arcs, curves, ellipses, lines, and rectangles.

C#

```
// Subscribe to the event.
GridTreeExpandCellRenderer renderer =
treeGrid.InternalGrid.CellRenderers["ExpanderCell"] as
GridTreeExpandCellRenderer;
if (renderer != null)
{
    renderer.GlyphDrawing += new
GridTreeGlyphDrawingHandler(renderer_GlyphDrawing);
}
// Sample event handler.
void renderer_GlyphDrawing(object sender, GridTreeGlyphDrawingEventArgs
args)
{
    // Draw a little circle.
    GridTreeExpandCellRenderer renderer = sender as GridTreeExpandCellRenderer;
    if (renderer != null)
    {
        // Code that draws a circle.
        PathGeometry pg = args.Geometry;
        Point pt0 = args.StartPoint;
        if (args.IsHot)
        {
            pt0.Offset(1, 5);
        }
        else
        {
            pt0.Offset(0, 4);
        }
        PathFigure pf = new PathFigure();
        pf.StartPoint = pt0;
        pf.IsClosed = false;
        pf.IsFilled = !args.Opened;
        // Radius
        double r = args.IsHot ? 4 : 5;
        // Make it essentially a full circle.
        pt0.Offset(0, .001);
        ArcSegment seg = new ArcSegment(pt0, new Size(r, r), 0, true,
SweepDirection.Clockwise, true);
        pf.Segments.Add(seg);
        pg.Figures.Add(pf);
    }
}
```


Node Images

You can display an image next to the expand glyph in the expand cell of the Grid Tree by setting the `GridTreeControl.SupportNodeImages` property to true. When this property is set to true, the Grid Tree will raise the `RequestNodeImage` event that allows you to provide an image for a given node. The `EventArgs` will provide you with the `GridTreeNode` object, and then the image can be set using the `NodeImage` property based on the given tree node.

The following code example illustrates how to handle the `RequestNodeImage` event.

C#

```
void treeGrid_RequestNodeImage(object sender,
GridTreeRequestNodeImageEventArgs args)
{
    args.NodeImage = employees.GetItemBitmap(args.Item as Employee);
}
```

Here is a screen shot that shows custom glyphs and node images.



Custom Glyphs and Node Images

Blendability

The GridTree control supports setting its styles through XAML and Microsoft Expression Blend.

Use Case Scenarios

This feature allows the GridTree control to be customized through Microsoft Expression Blend 3 or 4.

Properties

Property}	Description	Type	Data Type
StyleManager	Used to Customize the GridTreeControl Appearance	Dependency	GridTreeStyleManager

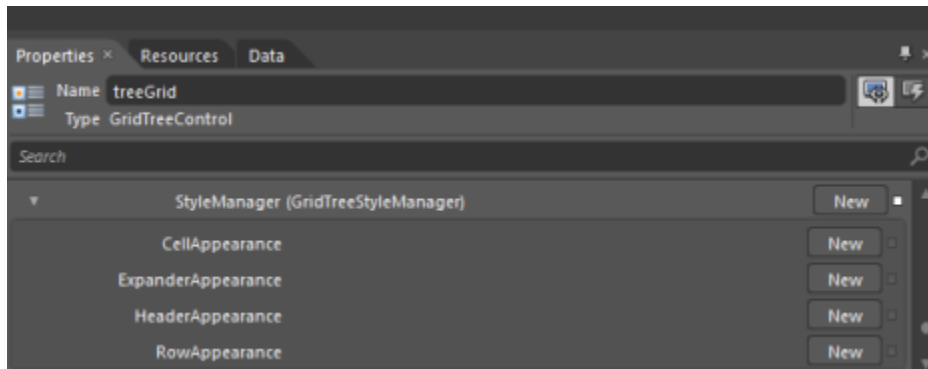
Sample Link

The Blend Styling demo in the sample browser is purely customized in XAML through the `GridTreeStyleManager` class. To access the Blend Styling demo:

1. Open the Syncfusion Essential Studio dashboard.
2. Select User Interface Edition.
3. Select WPF and click the orange Run Samples button on the right side of the screen.
4. Select GridTree control.
5. In the Styles menu item, select the Blend Styling demo.

Adding Styles to an Application

The appearance of the GridTree control can be customized using Microsoft Expression Blend 3 or 4®. This can be achieved through the StyleManager property of type GridTreeStyleManager. The properties required to customize the appearance are defined in the GridTreeStyleManager class.



Blend Property Window Showing StyleManager Properties

GridTreeStyleManager properties are organized under different groups, each representing a specific area of the GridTree control.

- CellAppearance
- ExpanderAppearance
- HeaderAppearance
- RowAppearance

The screenshot displays a hierarchical tree structure of authors and their books. Annotations with orange arrows point to specific visual elements:

- Header Appearance:** Points to the header row with columns: Author Name, ISBN, Title, Publisher Name.
- Row Appearance:** Points to a row where the entire row is highlighted (e.g., Alexandre Dumas - The Titans).
- Cell Appearance:** Points to a specific cell within a row (e.g., The White Dragon by Anne McCaffrey).
- Expander Appearance:** Points to the expand/collapse arrow icon next to an author's name.

Author Name	ISBN	Title	Publisher Name
▲ Alexandre Dumas			
Alexandre Dumas	0-123-1233-0	The Three Musketeers	Grosset & Dunlap
Alexandre Dumas	0-125-3344-1	The Black Tulip	P. F. Collier & Son
Alexandre Dumas	0-124-5544-X	The Titans	Harper
▶ James Clavell			
▲ Anne McCaffrey			
Anne McCaffrey	0-129-4567-1	Dragonsong	Atheneum
Anne McCaffrey	0-129-4912-0	Dragonsinger	Atheneum
Anne McCaffrey	0-130-2939-4	The White Dragon	Ballantine Books
▲ Robert Louis Stevenson			
Robert Louis Stevenson	0-134-3945-7	A Child's Garden of Verses	Scribner
Robert Louis Stevenson	0-129-4912-0	Treasure Island	J. W. Lovell Co.
Robert Louis Stevenson	0-130-2939-4	Kidnapped	Mead Dodd
Robert Louis Stevenson	0-134-3945-7	A Child's Garden of Verses	Scribner
Robert Louis Stevenson	0-135-2222-2	Treasure Island	J. W. Lovell Co.
Robert Louis Stevenson	0-136-3956-1	Kidnapped	Mead Dodd
▲ Anne Rice			

GridTreeStyleManager Properties

Cell Appearance

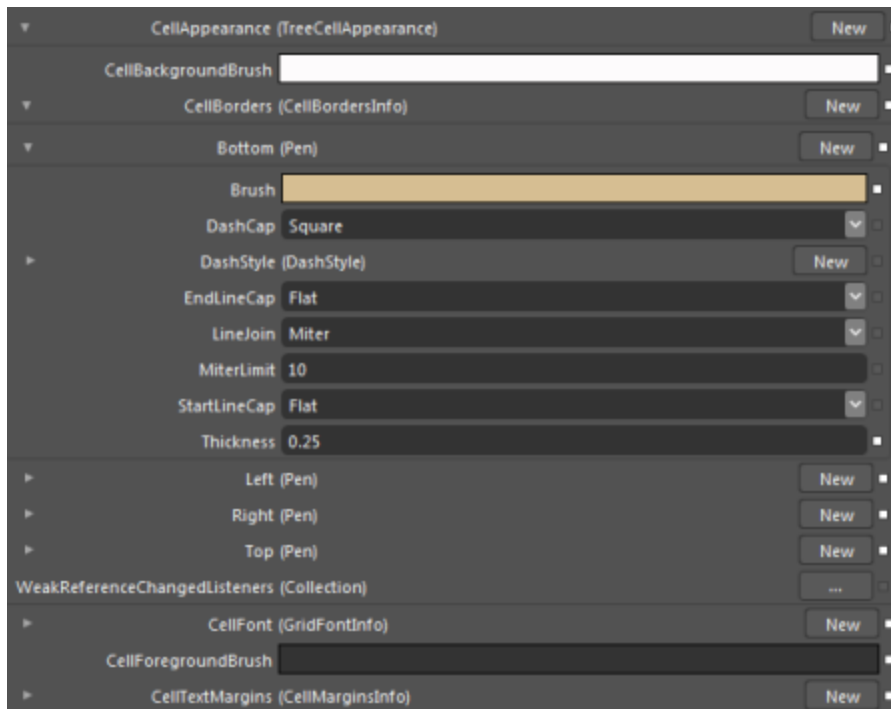
In the cell group, cells can be customized by specifying their margins, borders, etc. The following table shows the properties defined in this group.

Properties

Property	Description
CellBackgroundBrush	Used to change the background color of the cell
CellBorders	CellBorderInfo type property used to change the cell borders values (border color, thickness)
CellFont	GridFontInfo type property used to describe the font styles of the cell.
CellForegroundBrush	Used to change the foreground color of the cell
CellTextMargin	CellMarginsInfo type property describes cell text margins in the cell



Before Cell Appearance Applied



After Cell Appearance Applied

XML

```
<sf:GridTreeStyleManager.CellAppearance>
  <sf:TreeCellAppearance
    CellBackgroundBrush="#FFFDFBFC"
    CellForegroundBrush="#FF333333">
    <sf:TreeCellAppearance.CellFont>
      <sf:GridFontInfo FontFamily="Rockwell" FontSize="10" />
    </sf:TreeCellAppearance.CellFont>
    <sf:TreeCellAppearance.CellTextMargins>
      <sf:CellMarginsInfo Bottom="2" Left="2" Right="2" Top="2" />
    </sf:TreeCellAppearance.CellTextMargins>
    <sf:TreeCellAppearance.CellBorders>
      <sf:CellBordersInfo>
        <sf:CellBordersInfo.Top>
          <Pen Brush="#FFD6BE92" Thickness="0.25" />
        </sf:CellBordersInfo.Top>
        <sf:CellBordersInfo.Right>
          <Pen Brush="#FFD6BE92" Thickness="0.25" />
        </sf:CellBordersInfo.Right>
        <sf:CellBordersInfo.Left>
          <Pen Brush="#FFD6BE92" Thickness="0.25" />
        </sf:CellBordersInfo.Left>
        <sf:CellBordersInfo.Bottom>
          <Pen Brush="#FFD6BE92" Thickness="0.25" />
        </sf:CellBordersInfo.Bottom>
      </sf:CellBordersInfo>
    </sf:TreeCellAppearance.CellBorders>
  </sf:TreeCellAppearance>
</sf:GridTreeStyleManager.CellAppearance>
```

Author Name	ISBN	Title	Publisher Name
▲ Alexandre Dumas			
Alexandre Dumas	0-123-1233-0	The Three Musketeers	Grosset & Dunlap
Alexandre Dumas	0-125-3344-1	The Black Tulip	P. F. Collier & Son
Alexandre Dumas	0-124-5544-X	The Titans	Harper
▶ James Clavell			
▲ Anne McCaffrey			
Anne McCaffrey	0-129-4567-1	Dragonsong	Atheneum
Anne McCaffrey	0-129-4912-0	Dragonsinger	Atheneum
Anne McCaffrey	0-130-2939-4	The White Dragon	Sollantine Books
▲ Robert Louis Stevenson			
Robert Louis Stevenson	0-134-3945-7	A Child's Garden of Verses	Scribner
Robert Louis Stevenson	0-129-4912-0	Treasure Island	J. W. Lovell Co.
Robert Louis Stevenson	0-130-2939-4	Kidnapped	Mead Dodd
Robert Louis Stevenson	0-134-3945-7	A Child's Garden of Verses	Scribner
Robert Louis Stevenson	0-135-2222-2	Treasure Island	J. W. Lovell Co.
Robert Louis Stevenson	0-136-3956-1	Kidnapped	Mead Dodd
▲ Anne Rice			

Cell
Background,
BorderBrush,
Font, FontSize
are Changed.

Cell Appearance Applied to GridTree Control

Expander Appearance

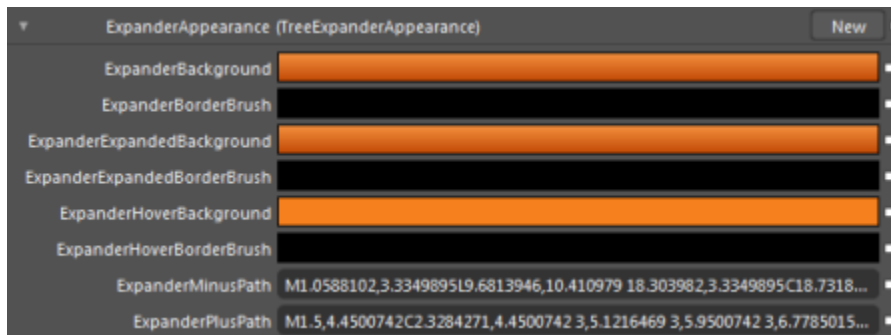
In the expander appearance group, the properties required to customize expand and collapse buttons are included. The following table shows the properties defined in this group.

Properties

Property	Description}}
ExpanderBackground	Used to change the background color of the expander
ExpanderBorderBrush	Used to change the border color of the expander
ExpanderExpandedBackground	Used to change the expanded expander background color
ExpanderExpandedBorderBrush	Used to change the expanded expander border color
ExpanderHoverBackground	Used to change the background color of the expander while the pointer hovers on the expander
ExpanderHoverBorderBrush	Used to change the border color while the mouse hovers on the expander
ExpanderMinusPath	Geometry type used to display different types of minus glyph
ExpanderPlusPath	Geometry type used to display different types of plus glyph



Before Expander Appearance Applied



After Expander Appearance Applied

XML

```
<sf:GridTreeStyleManager.ExpanderAppearance>
<sf:TreeExpanderAppearance ExpanderBorderBrush="Black"
ExpanderExpandedBorderBrush="Black"
ExpanderHoverBackground="#FFF5801D"
ExpanderHoverBorderBrush="Black"
ExpanderMinusPath="M1.0588102,3.3349895L9.6813946,10.410979 18.303982,3.3349895C18.731884,3.3349895 19.118088,3.7399883 19.281094,4.3580003 19.445393,4.9749742 19.354794,5.6889944 19.051689,6.1619864L10.429102,19.646975C10.2323,19.954988 9.959897,20.13199 9.6813946,20.13199 9.4019918,20.13199 9.1295891,19.954988 8.9326878,19.646975L0.31020308,6.1619864C0.10809803,5.8450003 0,5.4219956 0,4.9909959 0,4.7779832 0.026299,4.5619798 0.081700325,4.3580003 0.24469995,3.7399883 0.63000345,3.3349895 1.0588102,3.3349895z M9.7325096,0C11.450365,0 12.84301,1.3382103 12.84301,2.9904997 12.84301,4.6427898 11.450365,5.9809999 9.7325096,5.9809999 8.0146551,5.9809999 6.6220098,4.6427898 6.6220098,2.9904997 6.6220098,1.3382103 8.0146551,0 9.7325096,0z"
ExpanderPlusPath="M1.5,4.4500742C2.3284271,4.4500742 3,5.1216469 3,5.9500742 3,6.7785015 2.3284271,7.4500742 1.5,7.4500742 0.67157292,7.4500742 0,6.7785015 0,5.9500742 0,5.1216469 0.67157292,4.4500742 1.5,4.4500742z M2.8990631,0.00036717192C3.0453069,0.005159697,3.1827641,0.056897067,3.284497,0.15714154L8.5183868,5.3089428C8.6242905,5.4130292,8.6701775,5.5520487,8.6654148,5.6966238L8.6658916,5.6966238C8.6662092,5.7011499 8.6654148,5.7055159 8.6656513,5.710042 8.6654148,5.714488 8.6662092,5.718854 8.6658916,5.7233L8.6654148,5.7233C8.6701775,5.867878,8.6242905,6.0069752,8.5183868,6.1111407L3.284497,11.262862C3.0519648,11.491915 2.6327903,11.467778 2.3486581,11.208635 2.3486581,11.208635 2.2061534,11.101613 1.9369454,10.833023 1.6678151,10.56459 1.7572079,10.396356 1.9367857,10.137055 2.1164434,9.8779926 4.7354136,6.097723 4.7354136,6.097723 4.9301553,5.7890387 4.933569,5.630013 4.7354136,5.3222814 4.7354136,5.3222814 2.1164434,1.5420915 1.9367857,1.2827913 1.7572079,1.0237284
```

```

1.6678151,0.85541332 1.9369454,0.58698106 2.2061534,0.31847125 2.3486581,0.
21136875 2.3486581,0.21136881 2.5084825,0.065645903 2.711035,-
0.0057947943 2.8990631,0.00036717192z">
<sf:TreeExpanderAppearance.ExpanderBackground>
<LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
<GradientStop Offset="0" Color="#FFF5913F" />
<GradientStop Offset="1" Color="#FFC14804" />
</LinearGradientBrush>
</sf:TreeExpanderAppearance.ExpanderBackground>
<sf:TreeExpanderAppearance.ExpanderExpandedBackground>
<LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
<GradientStop Color="#FFF5913F" />
<GradientStop Offset="1" Color="#FFC14804" />
</LinearGradientBrush>
</sf:TreeExpanderAppearance.ExpanderExpandedBackground>
</sf:TreeExpanderAppearance>
</sf:GridTreeStyleManager.ExpanderAppearance>

```

Author Name	ISBN	Title	Publisher Name
Alexandre Dumas			
Alexandre Dumas	0-123-1233-0	The Three Musketeers	Crosset & Dunlap
Alexandre Dumas	0-125-3344-1	The Black Tulip	P. F. Collier & Son
Alexandre Dumas	0-124-5544-X	The Titans	Harper
James Clavell			
Anne McCaffrey			
Anne McCaffrey	0-129-4567-1	Dragonsong	Atheneum
Anne McCaffrey	0-129-4812-0	Dragonsinger	Atheneum
Anne McCaffrey	0-130-2939-4	The White Dragon	Ballantine Books
Robert Louis Stevenson			
Robert Louis Stevenson	0-134-3845-7	A Child's Garden of Verses	Scribner
Robert Louis Stevenson	0-129-4812-0	Treasure Island	J. W. Lovell Co.
Robert Louis Stevenson	0-130-2939-4	Kidnapped	Mead Dodd
Robert Louis Stevenson	0-134-3845-7	A Child's Garden of Verses	Scribner
Robert Louis Stevenson	0-135-2222-2	Treasure Island	J. W. Lovell Co.
Robert Louis Stevenson	0-136-3856-1	Kidnapped	Mead Dodd
Anne Rice			

Expander Appearance Changed.

Expander Appearance Applied to the GridTree Control

Header Appearance

In the header appearance options, properties required to customize the header are defined.

The following table shows the properties defined in this group.

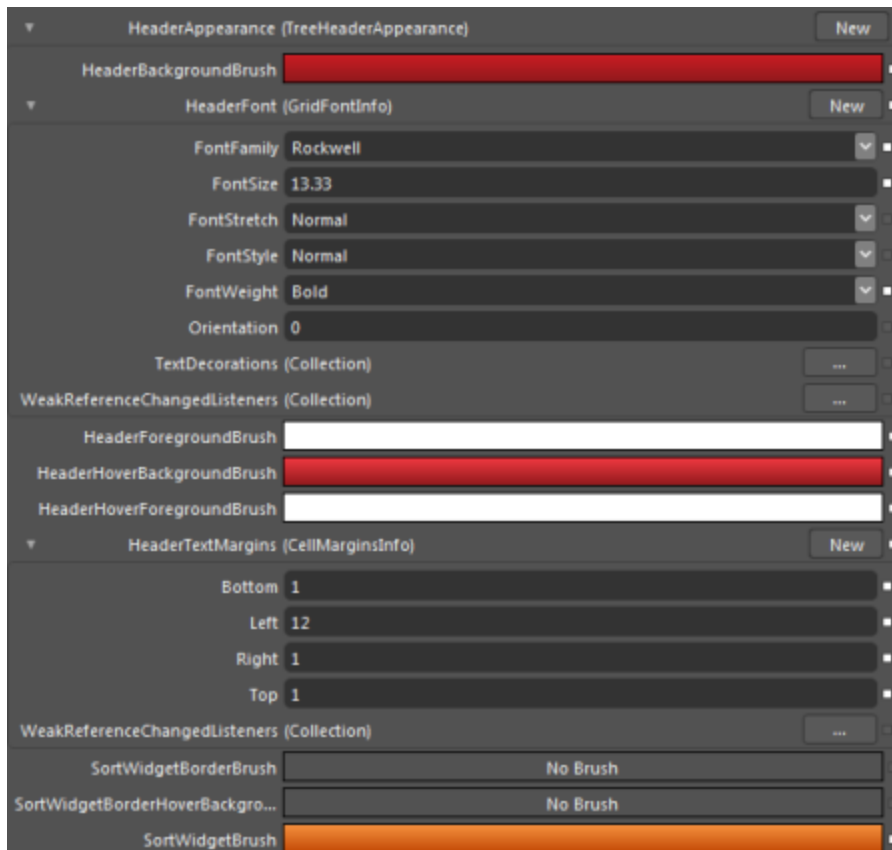
Properties

Property	Description
HeaderBackgroundBrush	Used to change the background color of the header
HeaderFont	GridFontInfo type property used to describe the font styles of the cell

HeaderForegroundBrush	Used to change the foreground color of the header
HeaderHoverBackgroundBrush	Used to change the background color of the header on mouse hover
HeaderHoverForegroundBrush	Used to change the header foreground color on mouse hover
HeaderTextMargins	CellMarginsInfo type property describes text margin in the header
SortWidgetBorderBrush	Used to change the sort icon border color
SortWidgetBorderHoverBackgroundBrush	Used to change the sort icon border background color
SortWidgetBrush	Used to change the sort icon background color



Before Header Appearance Applied



After Header Appearance Applied

XML

```
<sf:GridTreeStyleManager.HeaderAppearance>
  <sf:TreeHeaderAppearance
    HeaderForegroundBrush="White"
    HeaderHoverForegroundBrush="White">
    <sf:TreeHeaderAppearance.HeaderFont>
      <sf:GridFontInfo FontFamily="Rockwell"
        FontSize="13.33"
        FontWeight="Bold" />
    </sf:TreeHeaderAppearance.HeaderFont>
    <sf:TreeHeaderAppearance.HeaderHoverBackgroundBrush>
      <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
        <GradientStop Offset="0" Color="#FFF33A42" />
        <GradientStop Offset="1" Color="#FF8E191C" />
      </LinearGradientBrush>
    </sf:TreeHeaderAppearance.HeaderHoverBackgroundBrush>
    <sf:TreeHeaderAppearance.HeaderBackgroundBrush>
      <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
        <GradientStop Offset="0" Color="#FFCB1C23" />
        <GradientStop Offset="1" Color="#FF8E191C" />
      </LinearGradientBrush>
    </sf:TreeHeaderAppearance.HeaderBackgroundBrush>
    <sf:TreeHeaderAppearance.SortWidgetBrush>
      <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
        <GradientStop Offset="0" Color="#FFF5913F" />
        <GradientStop Offset="1" Color="#FFC14804" />
      </LinearGradientBrush>
    </sf:TreeHeaderAppearance.SortWidgetBrush>
  </sf:TreeHeaderAppearance>
</sf:GridTreeStyleManager.HeaderAppearance>
```

```

</LinearGradientBrush>
</sf:TreeHeaderAppearance.SortWidgetBrush>
<sf:TreeHeaderAppearance.HeaderTextMargins>
<sf:CellMarginsInfo Bottom="1"
Left="12"
Right="1"
Top="1" />
</sf:TreeHeaderAppearance.HeaderTextMargins>
</sf:TreeHeaderAppearance>
</sf:GridTreeStyleManager.HeaderAppearance>

```

Author Name	ISBN	Title	Publisher Name	Header Apperance Changed
Alexandre Dumas				
Alexandre Dumas	0-123-1233-0	The Three Musketeers	Grosset & Dunlap	
Alexandre Dumas	0-125-3344-1	The Black Tulip	P. F. Collier & Son	
Alexandre Dumas	0-124-5544-X	The Titans	Harper	
James Clavell				
Anne McCaffrey				
Anne McCaffrey	0-129-4567-1	Dragonsong	Atheneum	
Anne McCaffrey	0-129-4912-0	Dragonsinger	Atheneum	
Anne McCaffrey	0-130-2939-4	The White Dragon	Ballantine Books	
Robert Louis Stevenson				
Robert Louis Stevenson	0-134-3945-7	A Child's Garden of Verses	Scribner	
Robert Louis Stevenson	0-129-4912-0	Treasure Island	J. W. Lovell Co.	
Robert Louis Stevenson	0-130-2939-4	Kidnapped	Mead Dodd	
Robert Louis Stevenson	0-134-3945-7	A Child's Garden of Verses	Scribner	
Robert Louis Stevenson	0-135-2222-2	Treasure Island	J. W. Lovell Co.	
Robert Louis Stevenson	0-136-3956-1	Kidnapped	Mead Dodd	
Anne Rice				

Header Appearance Applied to the GridTreeControl

Row Appearance

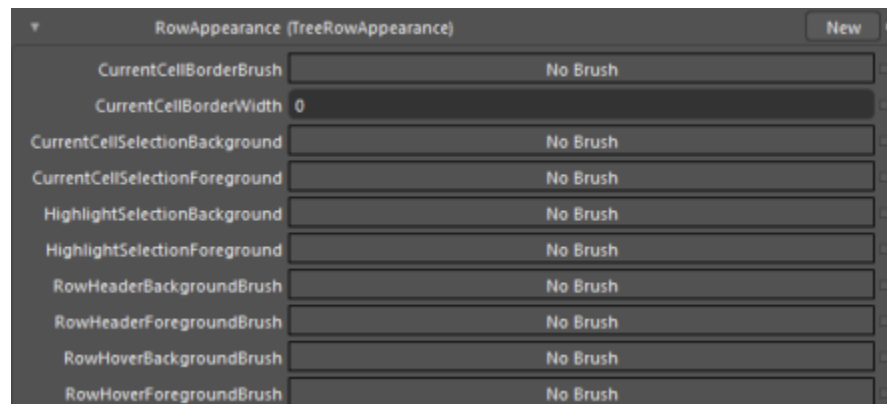
Properties required to customize the grid rows are defined in the row appearance options.

The following table shows the properties defined in this group.

Properties

Property	Description
CurrentCellBorderBrush	Used to change the current cell border color
CurrentCellBorderWidth	Used to change the current cell border thickness
CurrentCellSelectionBackground	Used to change the current cell selection background color
CurrentCellSelectionForeground	Used to change the current cell selection foreground color
HeaderHoverForegroundBrush	Used to change the header hover foreground color
HighlightSelectionBackground	Used to change the highlight selection background color
HighlightSelectionForeground	Used to change the highlight selection foreground color
RowHeaderBackgroundBrush	Used to change the row header background color

RowHeaderForegroundBrush	Used to change the row header foreground color
RowHoverBackgroundBrush	Used to change the hover color of the GridTree control
RowHoverForegroundBrush	Used to change the hover foreground color of the GridTree control



Before Row Appearance Applied



After Row Appearance Applied

XML

```

<sf:GridTreeStyleManager.RowAppearance>
  <sf:TreeRowAppearance CurrentCellSelectionBackground="#FFB31B20"
    CurrentCellSelectionForeground="White"
    HighlightSelectionBackground="#FFCDBEA3"
    HighlightSelectionForeground="#FFB71B21"
    RowHeaderForegroundBrush="White"
    RowHoverBackgroundBrush="#FFF9BBBB"
    RowHoverForegroundBrush="Black">
    <sf:TreeRowAppearance.RowHeaderBackgroundBrush>
      <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
        <GradientStop Offset="0" Color="#FFCB1C23" />
        <GradientStop Offset="1" Color="#FF8E191C" />
      </LinearGradientBrush>
    </sf:TreeRowAppearance.RowHeaderBackgroundBrush>
  </sf:TreeRowAppearance>
</sf:GridTreeStyleManager.RowAppearance>

```

	Author Name	ISBN	Title	Publisher Name	
♥	Alexandre Dumas				
	Alexandre Dumas	0-123-1233-0	The Three Musketeers	Grosset & Dunlap	
	Alexandre Dumas	0-125-3344-1	The Black Tulip	P. F. Collier & Son	
	Alexandre Dumas	0-124-5544-X	The Titans	Harper	HighlightSelection Background
♥	James Clavell				
♥	Anne McCaffrey				
	Anne McCaffrey	0-129-4567-1	Dragonsong	Atheneum	RowHover Background
	Anne McCaffrey	0-129-4912-0	Dragonsinger	Atheneum	
	Anne McCaffrey	0-130-2838-4	The White Dragon	Ballantine Books	
♥	Robert Louis Stevenson				
	Robert Louis Stevenson	0-134-3945-7	A Child's Garden of Verses	Scribner	
	Robert Louis Stevenson	0-129-4912-0	Treasure Island	J. W. Lovell Co.	
	Robert Louis Stevenson	0-130-2838-4	Kidnapped	Mead Dodd	
	Robert Louis Stevenson	0-134-3945-7	A Child's Garden of Verses	Scribner	
	Robert Louis Stevenson	0-135-2222-2	Treasure Island	J. W. Lovell Co.	
	Robert Louis Stevenson	0-136-3956-1	Kidnapped	Mead Dodd	
♥	Anne Rice				

Row Appearance Applied to the GridTree Control

Serialization

The GridTree control's state can be serialized and deserialized in XML format. All the styles and properties that reflect the state of the grid can be serialized.

The following options in the GridTree control can be serialized and deserialized.

- Width Options
- Sort Options
- Selection Options
- Visual Styles
- Editing Options
- Resizing Options

Use Case Scenarios

Serialization is used to retain the application state when the application is closed and reopened.

Methods

Method	Description	Parameters	Type}	Return Type
Serialize	This method serializes the properties to XML.	(String filename)	Public	void
SerializeAsString	This method serializes the properties as an XML string.	NA	Public	String
SerializeToStream	This method serializes properties as a stream.	(TextWriter textWriter)	Public	void
Deserialize	This method deserialize the XML file.	(String filename)	Public	void

DeserializeFromString	This method is used to deserialize the properties from the XML string.	String(fileName)	Public	void
DeserializeFromStream	This method is used to deserialize the property from the stream.	TextReader(textReader)	Public	void

[Sample Link](#)

The Serialization Demo sample in the sample browser illustrates the GridTree control's serialization feature.

1. Open the Syncfusion Essential Studio Dashboard.
2. Select User Interface Edition.
3. Select WPF and click the orange Run Samples button on the right side of the window.
4. Select GridTree Control.
5. Under the Serialization menu item, select Serialization Demo.

Adding Serialization/Deserialization to an Application

[Serializing](#)

There are three methods of serialization/deserialization available in the GridTree control.

- XML string
- XML file
- XML stream

[API Usage](#)

[Serializing as an XML String](#)

The following code illustrates how to serialize the GridTree control as an XML string.

C#

```
string result=this.treeGrid.InternalGrid.SerializeAsString();
```

[Serializing as an XML File](#)

The following code illustrates how to serialize the GridTree control as an XML file.

C#

```
this.treeGrid.InternalGrid.Serialize("newChanges.xml");
```

[Serializing as an XML Stream](#)

The following code illustrates how to serialize the GridTree control as an XML stream.

C#

```
TextWriter sw=new StreamWriter("newChanges.xml");
this.treeGrid.InternalGrid.SerializeToStream(sw);
```

Deserializing

There are three methods to deserializing forms:

- XML string
- XML file
- XML stream

API Usage

Deserialize from XML String

The following code illustrates how to deserialize from an XML string.

C#

```
//the result should be an XML string saved during the serialization process.  
this.treeGrid.InternalGrid.DeserializeFromString(result);
```

Deserialize from XML File

The following code illustrates how to deserialize an XML file.

C#

```
//newChanges.xml file should be the XML file saved during the serialization  
process.  
this.treeGrid.InternalGrid.Deserialize("newChanges.xml");
```

Deserialize from XML Stream

The following code illustrates how to deserialize an XML stream.

C#

```
//newChanges.txt file should be the text file saved during the serialization  
process.  
TextReader sr = new StreamReader("newChanges.txt");  
this.treeGrid.InternalGrid.DeserializeFromStream(sr);
```

TreeViewAdv (Classic)

WPF TreeViewAdv (Classic) Overview

TreeViewAdv control displays hierarchical data in a tree structure, and has items that can be expanded and collapsed. TreeViewAdv includes all essential features as well as certain advanced features that make the control unique and extra-ordinary. Properties of the TreeViewAdv control enable users to achieve desired layouts and outputs by using both XAML and C# code with ease.

Features

- Provides support to add any number of items to the control.
- Developed using UI Virtualization; enabling enhanced performance.
- Data binding support.
- Select multiple items using the CTRL+ SHIFT keys.
- Multiple item drag-and-drops within the control and to other TreeViewAdv controls.
- Built-in animations for Expand and Collapse operations with adjustable animation delay.

- Fake Drag Indicator to simulate where items may be placed during drag-and-drop operations.
- Show or Hide root lines and change the brush applied to the lines.
- Edit items at runtime as a folder using the F2 key or mouse.
- Customize the complete look and feel of the control.
- Custom template support.
- Sort TreeViewAdv items at run time.
- Add images to TreeViewItemAdv to identify expanded or collapsed states.
- Add images as left and right image sources to the TreeViewItemAdv.

Getting Started with WPF TreeViewAdv (Classic)

This section explains how to create Tree layout using [TreeViewAdv](#) control.

Assembly deployment

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

Refer to this [documentation](#) to find more details about installing nuget packages in a WPF application.

Add TreeViewAdv to Project

The [TreeViewAdv](#) control can be added to project by the following ways.

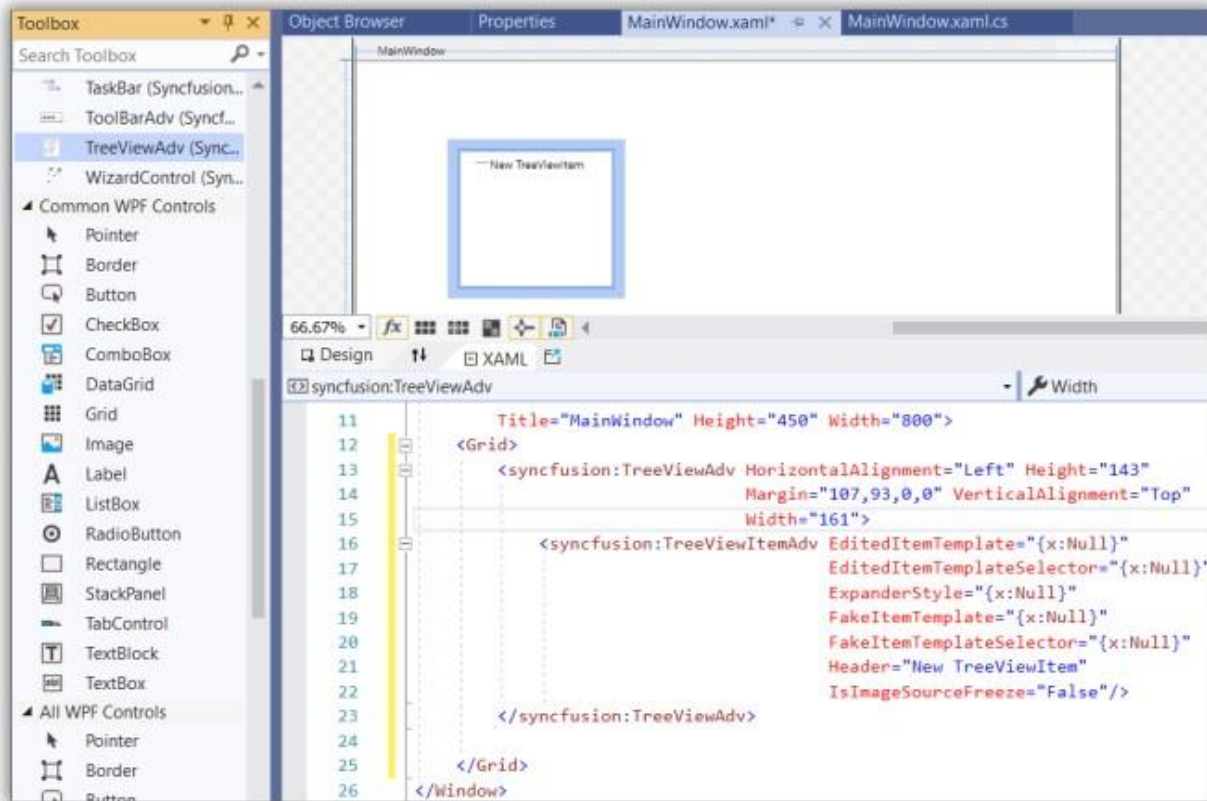
1. Adding TreeViewAdv by designer.
2. Adding TreeViewAdv by XAML.
3. Adding TreeViewAdv by C#.

Adding WPF TreeViewAdv via designer

1) The [TreeViewAdv](#) 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.Tools.Wpf*

Syncfusion.Shared.Wpf



2) Set the properties for TreeViewAdv in design mode using the SmartTag feature.

Adding WPF TreeViewAdv via XAML

To add the [TreeViewAdv](#) 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.Tools.Wpf* *Syncfusion.Shared.Wpf* 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the TreeViewAdv in XAML page.

XML

```

<Window x:Class="TreeViewAdv_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:TreeViewAdv_sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<syncfusion:TreeViewAdv Height="160" Width="160"
HorizontalAlignment="Center"
VerticalAlignment="Center" syncfusion:DockingManager.DockToFill="True" />
</Grid>
</Window>

```


Adding WPF TreeViewAdv via C#

To add the [TreeViewAdv](#) 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.Tools.Wpf* *Syncfusion.Shared.Wpf* 3) Include the required namespace.

C#

```
using Syncfusion.Windows.Tools.Controls;
```

4) Create an instance of [TreeViewAdv](#), and add it to the window.

C#

```
//Initializing TreeViewAdv
TreeViewAdv treeView = new TreeViewAdv();
//Setting Height,Width and HorizontalAlignment
treeView.Height = 150;
treeView.Width = 150;
treeView.HorizontalAlignment = HorizontalAlignment.Center;
//Adding control in MainWindow
this.Content = treeView;
```

Adding TreeView item to TreeViewAdv control

The TreeViewItem is added to a [TreeViewAdv](#) control either by using XAML or C# codes. The following code example lets you to create and add treeview items to the TreeViewAdv using [TreeViewItemAdv](#).

XML

```
<Window x:Class="WpfApplication1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525">
<Grid>
<syncfusion:TreeViewAdv x:Name="Tree">
<syncfusion:TreeViewItemAdv Header="WPF" />
<syncfusion:TreeViewItemAdv Header="Winrt" />
<syncfusion:TreeViewItemAdv Header="Silverlight" />
<syncfusion:TreeViewItemAdv Header="WindowPhone" />
<syncfusion:TreeViewItemAdv Header="UniversalWindows" />
</syncfusion:TreeViewAdv>
</Grid>
</Window>
```

C#

```
public partial class MainWindow : Window
{
public MainWindow()
{
InitializeComponent();
TreeViewAdv treeView = new TreeViewAdv() { Name = "Tree" };
treeView.Height = 150;
treeView.Width = 150;
treeView.HorizontalAlignment = HorizontalAlignment.Center;
```

```
TreeViewItemAdv item1 = new TreeViewItemAdv();
item1.Header = "WPF";
TreeViewItemAdv item2 = new TreeViewItemAdv();
item2.Header = "Winrt";
TreeViewItemAdv item3 = new TreeViewItemAdv();
item3.Header = "Silverlight";
TreeViewItemAdv item4 = new TreeViewItemAdv();
item4.Header = "WindowPhone";
TreeViewItemAdv item5 = new TreeViewItemAdv();
item5.Header = "UniversalWindows";
treeView.Items.Add(item1);
treeView.Items.Add(item2);
treeView.Items.Add(item3);
treeView.Items.Add(item4);
treeView.Items.Add(item5);
this.Content = treeView;
}
```

Set VisualStyle

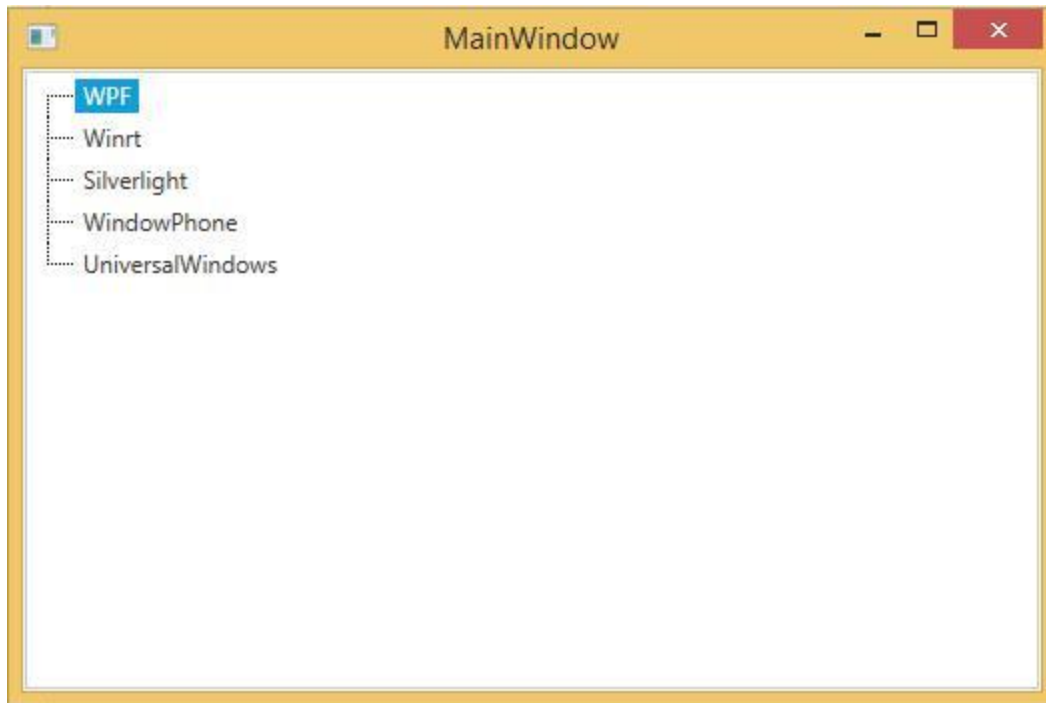
[TreeViewAdv](#) supports various visual styles by using the SkinStorage. To apply Visual Studio style on the current layout, refer the below code to apply the value Metro to the VisualStyle property of the SkinStorage for the Window

XML

```
<Window x:Class="WpfApplication1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525">
<Grid>
<syncfusion:TreeViewAdv syncfusion:SkinStorage.VisualStudio="Metro"
x:Name="Tree">
<syncfusion:TreeViewItemAdv Header="WPF" />
<syncfusion:TreeViewItemAdv Header="Winrt" />
<syncfusion:TreeViewItemAdv Header="Silverlight" />
<syncfusion:TreeViewItemAdv Header="WindowPhone" />
<syncfusion:TreeViewItemAdv Header="UniversalWindows"/>
</syncfusion:TreeViewAdv>
</Grid>
</Window>
```

C#

```
TreeViewAdv treeView = new TreeViewAdv();
SkinStorage.SetVisualStyle(treeView, "Metro");
this.Content = treeView;
```



Setting ItemsSource for TreeviewAdv

The following code snippet sets a collection used to generate the Items of the [TreeViewAdv](#) control.

XML

```
<Window x:Class="WpfApplication1.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"
syncfusion:SkinStorage.VisualStudio="Metro"
xmlns:local="clr-namespace:WpfApplication1"
Title="MainWindow" Height="350" Width="525">
<Window.Resources>
<local:TechnologyList x:Key="technologyList" />
</Window.Resources>
<Grid>
<syncfusion:TreeViewAdv x:Name="Tree" ItemsSource="{StaticResource
technologyList}" />
</syncfusion:TreeViewAdv>
</Grid>
</Window>
```

C#

```
public class TechnologyList : ObservableCollection<string>
{
    public TechnologyList()
    {
        this.Add("WPF");
        this.Add("Winrt");
        this.Add("Silverlight");
        this.Add("WindowPhone");
        this.Add("UniversalWindows");
    }
}
```

```
}  
}
```

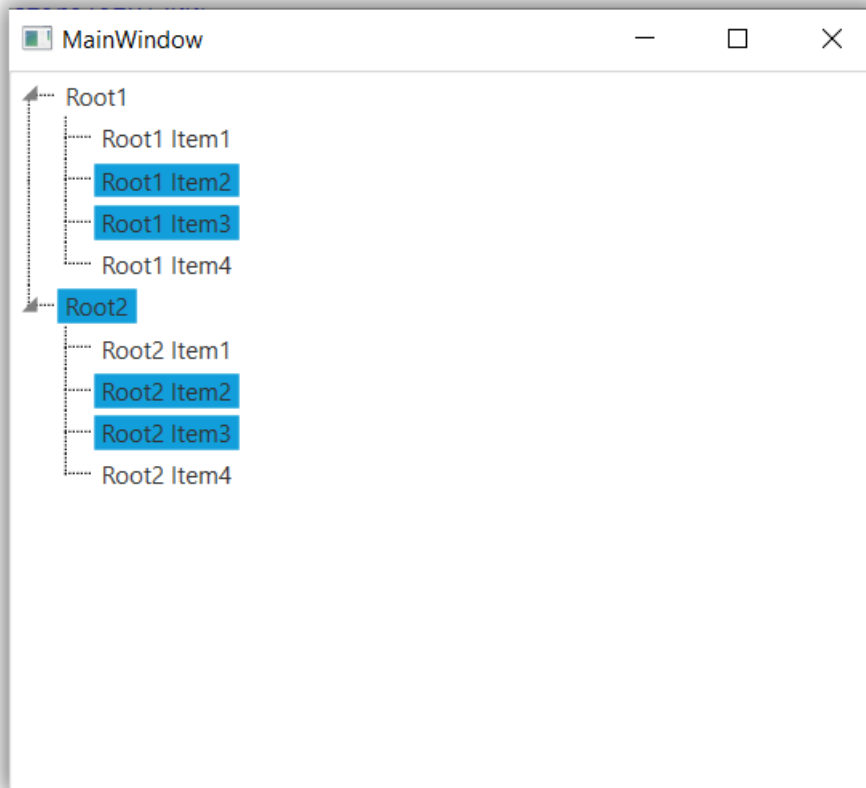
VB.NET

```
Public Class TechnologyList  
Inherits ObservableCollection(Of String)  
Public Sub New()  
Me.Add("WPF")  
Me.Add("Winrt")  
Me.Add("Silverlight")  
Me.Add("WindowPhone")  
Me.Add("UniversalWindows")  
End Sub  
End Class
```



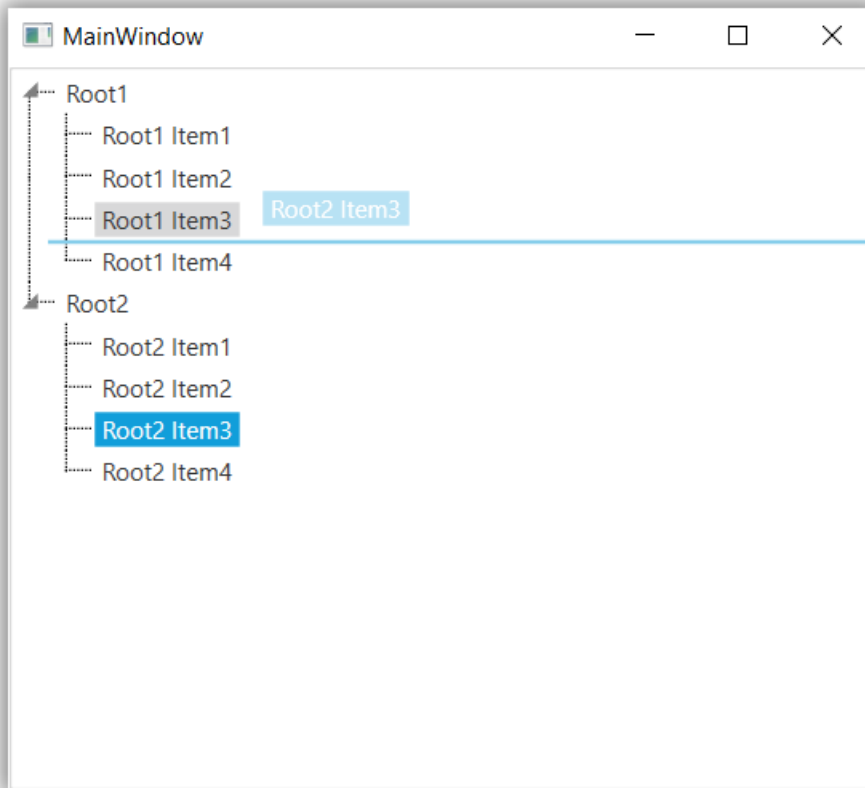
Multiple Selection in TreeViewAdv

The [TreeViewAdv](#) control supports selecting multiple items by using the CTRL or SHIFT keys. The selected items are dragged to any item or node within the same control or to another TreeViewAdv control. This is achieved by enabling the [AllowMultiSelect](#) property. Click [here](#) to get detailed information on AllowMultiSelect function.



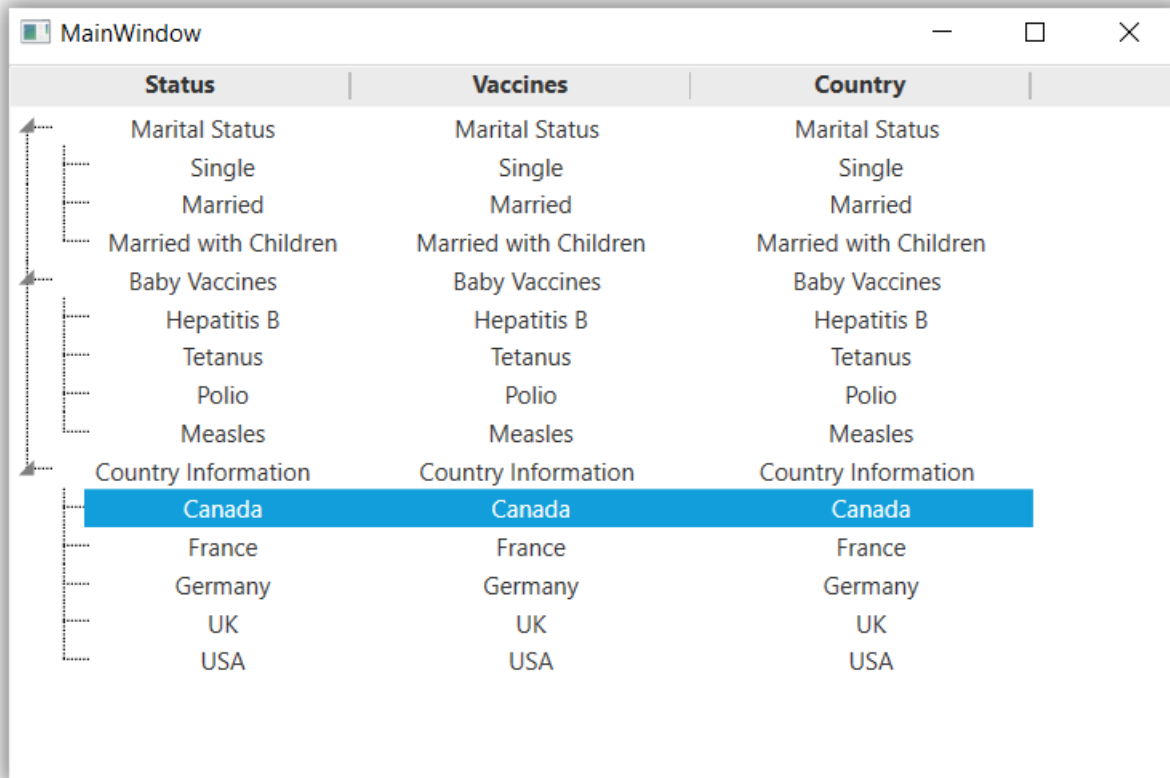
Drag and Drop in TreeViewAdv

By enabling the [AllowDragDrop](#) property TreeViewAdv control allows us to drag TreeView items from one location to another. By using the [DraggingContainerOpacity](#) property, we can change the opacity value of the dragged element. It is useful to be able to view the content behind the dragged element. Click [here](#) to get detailed information on drag and drop of TreeViewAdv items.



MultiColumn TreeView in TreeViewAdv

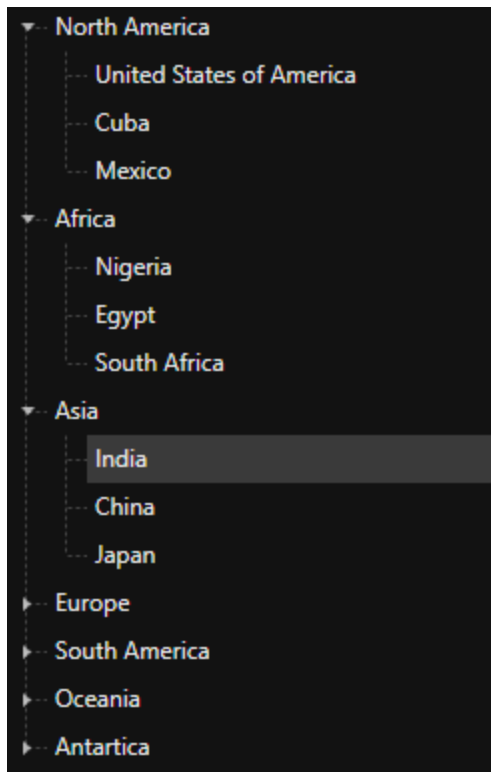
TreeView control can be created with multiple columns by setting the [MultiColumnEnable](#) property to true. Click [here](#) to get detailed information on how to create MultiColumn TreeViewAdv.



Theme

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

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



Populating with Data in WPF TreeViewAdv (Classic)

This section explains about populating TreeViewAdv with TreeView items using several ways.

- Through XAML
- Through Programmatically
- Through DataBinding

Through XAML

TreeViewAdv can be created in XAML as follows:

XML

```
<syncfusion:TreeViewAdv>
<syncfusion:TreeViewItemAdv Header="Root1">
<syncfusion:TreeViewItemAdv Header="SubItem1"/>
<syncfusion:TreeViewItemAdv Header="SubItem2"/>
<syncfusion:TreeViewItemAdv Header="SubItem2"/>
<syncfusion:TreeViewItemAdv Header="SubItem2"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Root2">
<syncfusion:TreeViewItemAdv Header="SubItem1"/>
<syncfusion:TreeViewItemAdv Header="SubItem2"/>
<syncfusion:TreeViewItemAdv Header="SubItem2"/>
<syncfusion:TreeViewItemAdv Header="SubItem2"/>
</syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```


Through programmatically

Include the following namespace to the using directives list to create TreeViewAdv in

C#

```
using Syncfusion.Windows.Tools.Controls;
```

VB.NET

```
Imports Syncfusion.Windows.Tools.Controls
```

You can create the TreeViewAdv as follows:

C#

```
TreeViewAdv treeviewAdv = new TreeViewAdv();
TreeViewItemAdv root1 = new TreeViewItemAdv() { Header = "Root1" };
TreeViewItemAdv subitem11 = new TreeViewItemAdv() { Header = "SubItem1" };
TreeViewItemAdv subitem12 = new TreeViewItemAdv() { Header = "SubItem2" };
TreeViewItemAdv subitem13 = new TreeViewItemAdv() { Header = "SubItem3" };
TreeViewItemAdv subitem14 = new TreeViewItemAdv() { Header = "SubItem4" };
root1.Items.Add(subitem11);
root1.Items.Add(subitem12);
root1.Items.Add(subitem13);
root1.Items.Add(subitem14);
TreeViewItemAdv root2 = new TreeViewItemAdv() { Header = "Root1" };
TreeViewItemAdv subitem21 = new TreeViewItemAdv() { Header = "SubItem1" };
TreeViewItemAdv subitem22 = new TreeViewItemAdv() { Header = "SubItem2" };
TreeViewItemAdv subitem23 = new TreeViewItemAdv() { Header = "SubItem3" };
TreeViewItemAdv subitem24 = new TreeViewItemAdv() { Header = "SubItem4" };
root2.Items.Add(subitem21);
root2.Items.Add(subitem22);
root2.Items.Add(subitem23);
root2.Items.Add(subitem24);
treeviewAdv.Items.Add(root1);
treeviewAdv.Items.Add(root2);
//TreeViewAdv added the child of grid.
layout.Children.Add(treeviewAdv);
```

VB.NET

```
Dim treeviewAdv As New TreeViewAdv()
Dim root1 As New TreeViewItemAdv() With {.Header = "Root1"}
Dim subitem11 As New TreeViewItemAdv() With {.Header = "SubItem1"}
Dim subitem12 As New TreeViewItemAdv() With {.Header = "SubItem2"}
Dim subitem13 As New TreeViewItemAdv() With {.Header = "SubItem3"}
Dim subitem14 As New TreeViewItemAdv() With {.Header = "SubItem4"}
root1.Items.Add(subitem11)
root1.Items.Add(subitem12)
root1.Items.Add(subitem13)
root1.Items.Add(subitem14)
Dim root2 As New TreeViewItemAdv() With {.Header = "Root1"}
Dim subitem21 As New TreeViewItemAdv() With {.Header = "SubItem1"}
Dim subitem22 As New TreeViewItemAdv() With {.Header = "SubItem2"}
Dim subitem23 As New TreeViewItemAdv() With {.Header = "SubItem3"}
Dim subitem24 As New TreeViewItemAdv() With {.Header = "SubItem4"}
```

```
root2.Items.Add(subitem21)
root2.Items.Add(subitem22)
root2.Items.Add(subitem23)
root2.Items.Add(subitem24)
treeviewAdv.Items.Add(root1)
treeviewAdv.Items.Add(root2)
'TreeViewAdv added the child of grid.
layout.Children.Add(treeviewAdv)
```

Through DataBinding

DataBinding is a powerful mechanism to auto-update data between the business model and the user interface.

Binding to XML

XML file can also be used as ItemsSource for the TreeViewAdv. The following example illustrates this:

1. Create the XML file with the following details and name it as Data.xml:

XML

```
<Products>
<Product Name="Tools" >
<Feature Name="Ribbon" >
<Feature Name="Office2010UI"/>
<Feature Name="Data Binding Support"/>
</Feature>
<Feature Name="Docking Manager">
<Feature Name="Maximization"/>
<Feature Name="State Persistence"/>
</Feature>
<Feature Name="TreeView">
<Feature Name="Editing"/>
<Feature Name="Sorting"/>
</Feature>
<Feature Name="Data Editors" >
<Feature Name="Watermark Text" />
<Feature Name="Extended Value Scrolling" />
</Feature>
</Product>
</Products>
```

2. Add the XmlDataProvider for the above XML document as follows:

XML

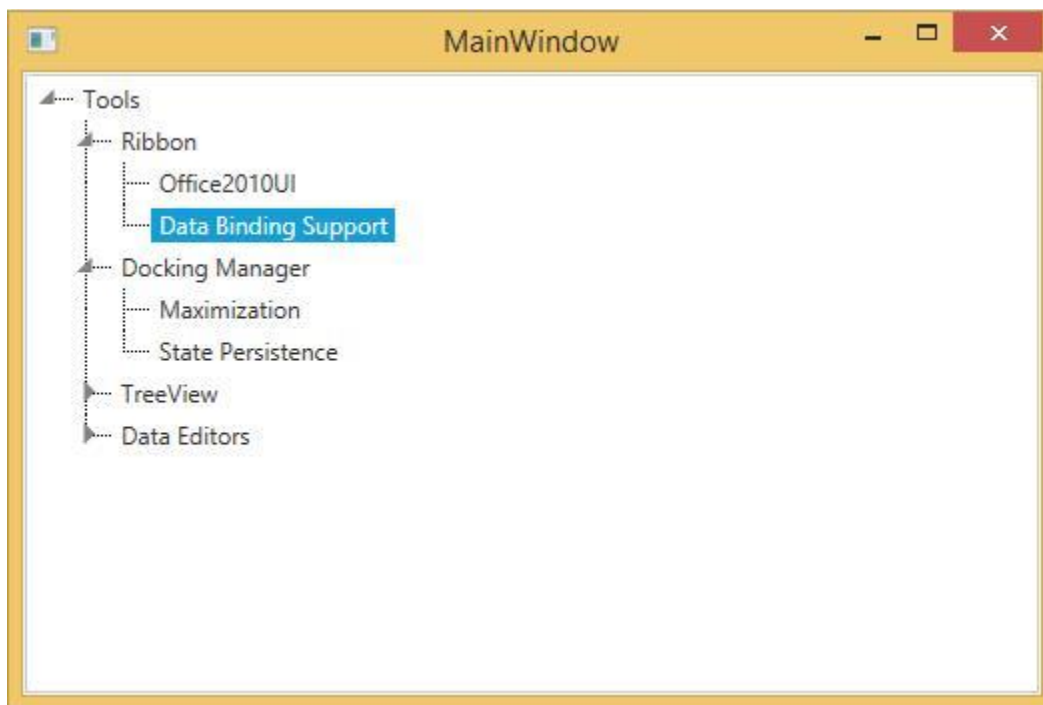
```
<Window.Resources>
<XmlDataProvider Source="Data.xml" x:Key="xmlSource" XPath="Products"/>
</Window.Resources>
```

3. Set the ItemsSource property for the TreeViewAdv as follows:

XML

```
<syncfusion:TreeViewAdv ItemsSource="{Binding Source={StaticResource  
xmlSource}, XPath=Product}" >  
<syncfusion:TreeViewAdv.ItemTemplate>  
<HierarchicalDataTemplate ItemsSource="{Binding XPath=Feature}">  
<TextBlock Text="{Binding XPath=@Name}" />  
</HierarchicalDataTemplate>  
</syncfusion:TreeViewAdv.ItemTemplate>  
</syncfusion:TreeViewAdv>
```

4. TreeViewAdv will be created as follows:



[View Sample in GitHub](#)

Binding to object

TreeViewAdv supports object binding. The following example illustrates data binding:

1. Create a class that acts as a model for TreeViewAdv as given in the following code snippet:

C#

```
public class Model  
{  
    public Model()  
    {  
        SubItems = new ObservableCollection<Model>();  
    }  
    public string Header { get; set; }  
    public bool IsCheckable { get; set; }  
    public ObservableCollection<Model> SubItems { get; set; }  
}
```

```
}

```

VB.NET

```
Public Class Model
Public Sub New()
SubItems = New ObservableCollection(Of Model) ()
End Sub
Public Property Header() As String
Public Property IsCheckable() As Boolean
Public Property SubItems() As ObservableCollection(Of Model)
End Class

```

2. Create a ViewModel class and initialize the items as given in the following code snippet:

C#

```
public class ViewModel
{
public ViewModel()
{
TreeItems = new ObservableCollection<Model>();
PopulateData();
}
public ObservableCollection<Model> TreeItems { get; set; }
private void PopulateData()
{
Model Root1 = new Model() { Header = "Root1" };
PopulateSubItems(Root1);
TreeItems.Add(Root1);
Model Root2 = new Model() { Header = "Root2" };
PopulateSubItems(Root2);
TreeItems.Add(Root2);
}
private void PopulateSubItems(Model Root)
{
Model SubItem1 = new Model() { Header = Root.Header + " Item1" };
Model SubItem2 = new Model() { Header = Root.Header + " Item2" };
Model SubItem3 = new Model() { Header = Root.Header + " Item3" };
Model SubItem4 = new Model() { Header = Root.Header + " Item4" };
Root.SubItems.Add(SubItem1);
Root.SubItems.Add(SubItem2);
Root.SubItems.Add(SubItem3);
Root.SubItems.Add(SubItem4);
}
}

```

VB.NET

```
Public Class ViewModel
Public Sub New()
TreeItems = New ObservableCollection(Of Model) ()
PopulateData()
End Sub

```

```
Public Property TreeItems() As ObservableCollection(Of Model)
Private Sub PopulateData()
Dim Root1 As New Model() With {.Header = "Root1"}
PopulateSubItems(Root1)
TreeItems.Add(Root1)
Dim Root2 As New Model() With {.Header = "Root2"}
PopulateSubItems(Root2)
TreeItems.Add(Root2)
End Sub
Private Sub PopulateSubItems(ByVal Root As Model)
Dim SubItem1 As New Model() With {.Header = Root.Header & " Item1"}
Dim SubItem2 As New Model() With {.Header = Root.Header & " Item2"}
Dim SubItem3 As New Model() With {.Header = Root.Header & " Item3"}
Dim SubItem4 As New Model() With {.Header = Root.Header & " Item4"}
Root.SubItems.Add(SubItem1)
Root.SubItems.Add(SubItem2)
Root.SubItems.Add(SubItem3)
Root.SubItems.Add(SubItem4)
End Sub
End Class
```

3. Create a ViewModel instance and use it as DataContext for the Root Window as given in the following code snippet:

XML

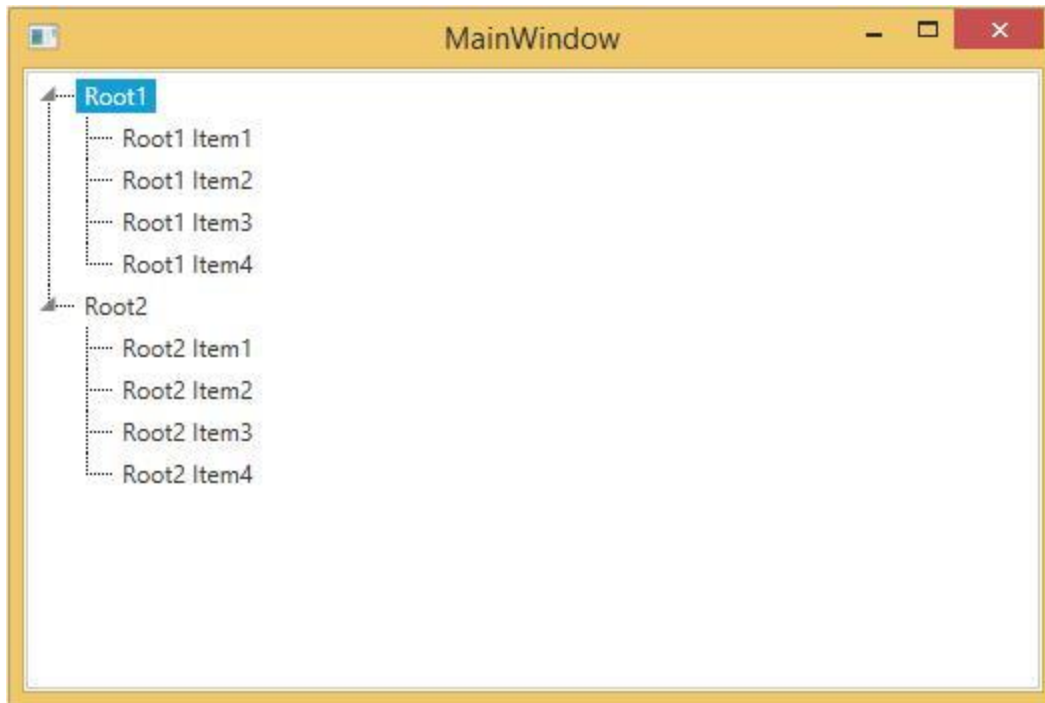
```
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
```

4. Configure the ItemsSource and ItemTemplate of the TreeViewAdv as given below:

XML

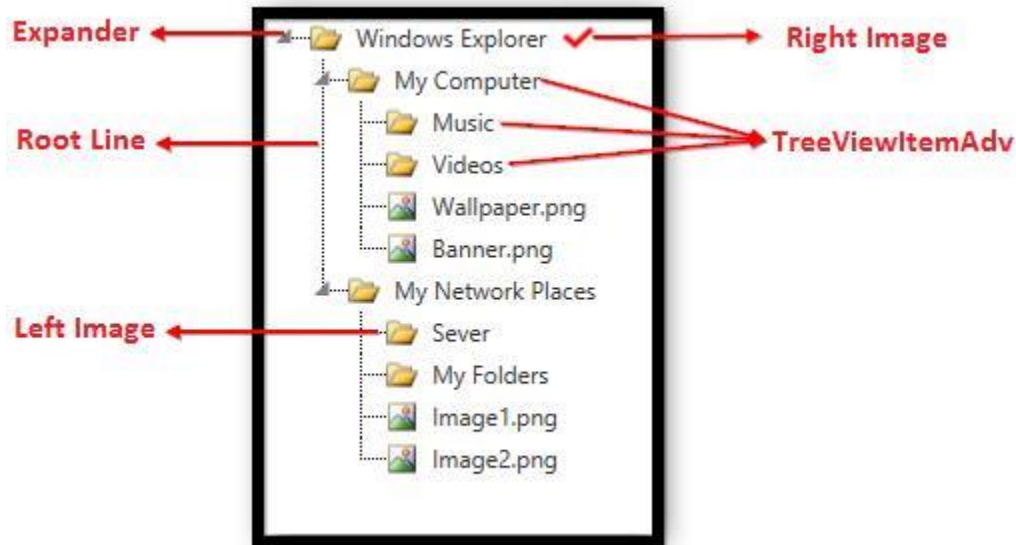
```
<syncfusion:TreeViewAdv ItemsSource="{Binding TreeItems}">
<syncfusion:TreeViewAdv.ItemTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding SubItems}">
<TextBlock Text="{Binding Header}" />
</HierarchicalDataTemplate>
</syncfusion:TreeViewAdv.ItemTemplate>
</syncfusion:TreeViewAdv>
```

5. The TreeViewAdv will be created as follows:



Visual Structure in WPF TreeViewAdv (Classic)

TreeViewAdv control displays hierarchical data in a tree structure, and has items that can be expanded and collapsed



The elements of the control are described below:

- **TreeViewItemAdv** Contents
- **Expander** – Used to Expand/Collapse the TreeViewItem
- **Root Line** - This line is used to enhance the visual of the connection between TreeViewItems and its parent.
- **LeftImage** - Displays the image in front of the TreeViewItem.

- **RightImage** – Displays the image to the right of the TreeViewItemAdv

Images for items in WPF TreeViewAdv (Classic)

Node image

To add images to the left and right corner of the TreeViewItemAdv by using the LeftImageSource and RightImageSource properties of the TreeViewItemAdv class.

- LeftImageSource—this is a dependency property, which gets or sets ImageSource to left image.
- RightImageSource—this is dependency property, which gets or sets RightSource to right image.

XML

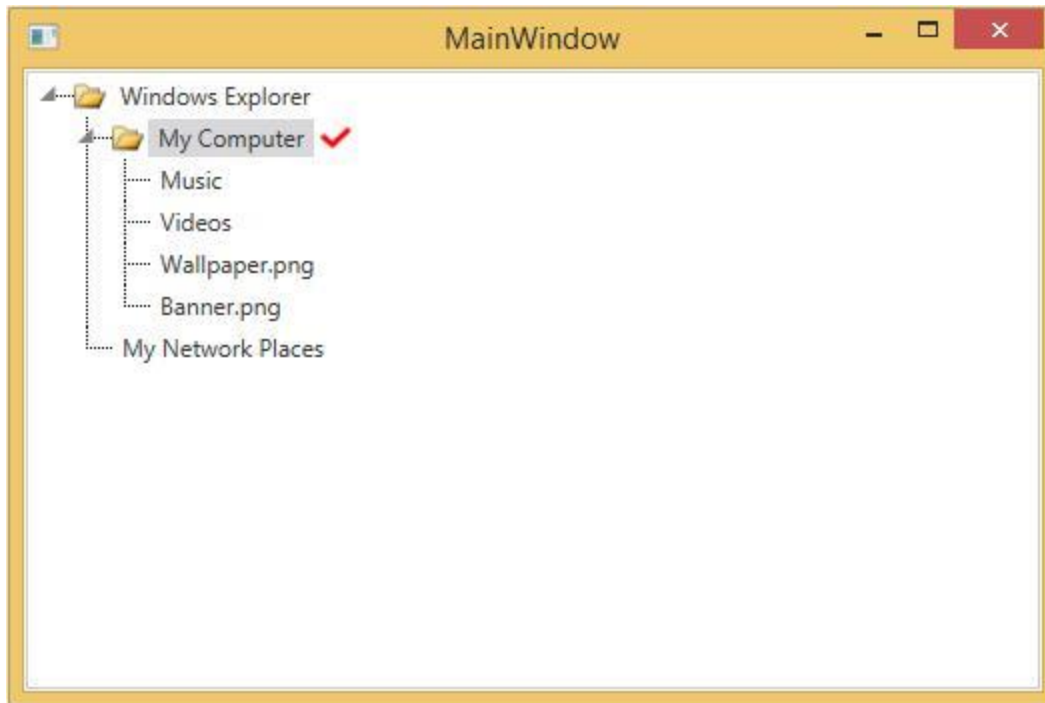
```
<syncfusion:TreeViewAdv >
<syncfusion:TreeViewItemAdv x:Name="item1" Header="Windows Explorer"
LeftImageSource="Folder_Expanded.png" >
<syncfusion:TreeViewItemAdv Header="My Computer" x:Name="item2"
LeftImageSource="Folder_Expanded.png" RightImageSource="Right tick.png" >
<syncfusion:TreeViewItemAdv Header="Music" />
<syncfusion:TreeViewItemAdv Header="Videos" />
<syncfusion:TreeViewItemAdv Header="Wallpaper.png" />
<syncfusion:TreeViewItemAdv Header="Banner.png" />
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="My Network Places" >
</syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

C#

```
//Set image for left
item1.LeftImageSource = new BitmapImage(new Uri("/Folder_Expanded.png",
UriKind.RelativeOrAbsolute));
//Set image for right
item2.RightImageSource = new BitmapImage(new Uri("/Right tick.png",
UriKind.RelativeOrAbsolute));
```

VB.NET

```
'Set image for left
item1.LeftImageSource = New BitmapImage(New Uri("/Folder_Expanded.png",
UriKind.RelativeOrAbsolute))
'Set image for right
item2.RightImageSource = New BitmapImage(New Uri("/Right tick.png",
UriKind.RelativeOrAbsolute))
```



Customize image size in TreeViewItemAdv

TreeViewAdv allow user to set the height and width of the TreeViewItemAdv images. The ImageHeight property sets the height of the image and the ImageWidth property sets the width of the images. The default value of the properties is NaN. Here is the code for setting these properties.

XML

```
<syncfusion:TreeViewAdv >
  <syncfusion:TreeViewItemAdv x:Name="item1" ImageHeight="25" ImageWidth="25"
  Header="Windows Explorer" LeftImageSource="Folder_Expanded.png" >
    <syncfusion:TreeViewItemAdv Header="My Computer" x:Name="item2"
    LeftImageSource="Folder_Expanded.png" RightImageSource="Right tick.png" >
      <syncfusion:TreeViewItemAdv Header="Music" />
      <syncfusion:TreeViewItemAdv Header="Videos" />
      <syncfusion:TreeViewItemAdv Header="Wallpaper.png" />
      <syncfusion:TreeViewItemAdv Header="Banner.png" />
    </syncfusion:TreeViewItemAdv>
    <syncfusion:TreeViewItemAdv Header="My Network Places" >
    </syncfusion:TreeViewItemAdv>
  </syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

C#

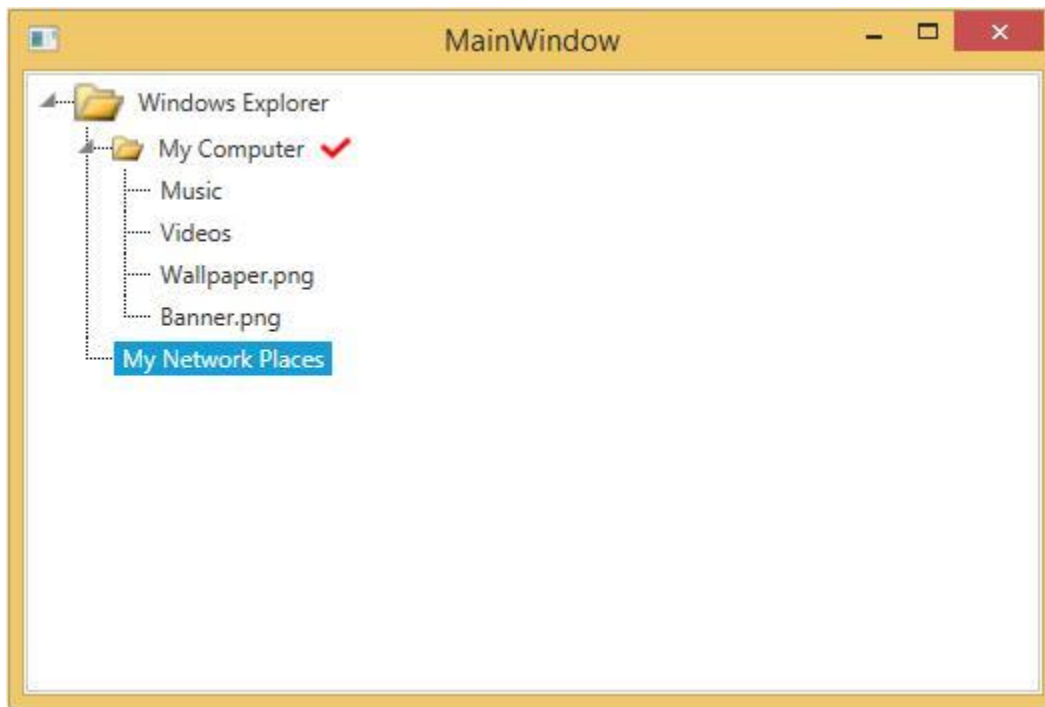
```
//Set image height
item1.ImageHeight = 25;
//Set image width
item1.ImageWidth = 25;
```

VB.NET

```
'Set image height
```



```
item1.ImageHeight = 25  
'Set image width  
item1.ImageWidth = 25
```



Expand the item in WPF TreeViewAdv (Classic)

TreeViewAdv allows to expand or collapse each node by setting [IsExpanded](#) property.

XML

```
<Grid x:Name="Grid">  
  <syncfusion:TreeViewAdv Name="treeViewAdv">  
    <syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital Status">  
      <syncfusion:TreeViewItemAdv Header="Single"/>  
      <syncfusion:TreeViewItemAdv Header="Married"/>  
      <syncfusion:TreeViewItemAdv Header="Married with Children"/>  
    </syncfusion:TreeViewItemAdv>  
    <syncfusion:TreeViewItemAdv Header="Baby Vaccines">  
      <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>  
      <syncfusion:TreeViewItemAdv Header="Tetanus"/>  
      <syncfusion:TreeViewItemAdv Header="Polio"/>  
      <syncfusion:TreeViewItemAdv Header="Measles"/>  
    </syncfusion:TreeViewItemAdv>  
    <syncfusion:TreeViewItemAdv Header="Country Information"/>  
  </syncfusion:TreeViewAdv>  
</Grid>
```

C#

```
public MainWindow()  
{  
  InitializeComponent();  
  TreeViewAdv treeViewAdv = new TreeViewAdv();
```

```

TreeViewItemAdv root1 = new TreeViewItemAdv() { Header = "Marital Status" };
TreeViewItemAdv subitem11 = new TreeViewItemAdv() { Header = "Single" };
TreeViewItemAdv subitem12 = new TreeViewItemAdv() { Header = "Married" };
TreeViewItemAdv subitem13 = new TreeViewItemAdv() { Header = "Married with
Children" };
root1.Items.Add(subitem11);
root1.Items.Add(subitem12);
root1.Items.Add(subitem13);
TreeViewItemAdv root2 = new TreeViewItemAdv() { Header = "Baby Vaccines" };
TreeViewItemAdv subitem21 = new TreeViewItemAdv() { Header = "Hepatitis B"
};
TreeViewItemAdv subitem22 = new TreeViewItemAdv() { Header = "Tetanus" };
TreeViewItemAdv subitem23 = new TreeViewItemAdv() { Header = "Polio" };
TreeViewItemAdv subitem24 = new TreeViewItemAdv() { Header = "Measles" };
root2.Items.Add(subitem21);
root2.Items.Add(subitem22);
root2.Items.Add(subitem23);
root2.Items.Add(subitem24);
TreeViewItemAdv root3 = new TreeViewItemAdv() { Header = "Baby Vaccines" };
treeViewAdv.Items.Add(root1);
treeViewAdv.Items.Add(root2);
treeViewAdv.Items.Add(root3);
Grid.Children.Add(treeViewAdv);
root1.IsExpanded = true;
}

```

VB.NET

```

Public Sub New()
InitializeComponent()
Dim treeViewAdv As TreeViewAdv = New TreeViewAdv()
Dim root1 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header =
"Marital Status"}
Dim subitem11 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header =
"Single"}
Dim subitem12 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header =
"Married"}
Dim subitem13 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header =
"Married with Children"}
root1.Items.Add(subitem11)
root1.Items.Add(subitem12)
root1.Items.Add(subitem13)
Dim root2 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header = "Baby
Vaccines"}
Dim subitem21 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header =
"Hepatitis B"}
Dim subitem22 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header =
"Tetanus"}
Dim subitem23 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header =
"Polio"}
Dim subitem24 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header =
"Measles"}
root2.Items.Add(subitem21)
root2.Items.Add(subitem22)
root2.Items.Add(subitem23)
root2.Items.Add(subitem24)

```

```
Dim root3 As TreeViewItemAdv = New TreeViewItemAdv() With {.Header = "Baby  
Vaccines"}  
treeViewAdv.Items.Add(root1)  
treeViewAdv.Items.Add(root2)  
treeViewAdv.Items.Add(root3)  
Grid.Children.Add(treeViewAdv)  
root1.IsExpanded = True  
End Sub
```



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

Animation type

The type of animation that is generated while expanding or collapsing the TreeViewAdv is controlled by using the [AnimationType](#) property. This property includes the following options.

- **Fade** — Fade animation applies when expand or collapse the nodes.
- **None** — Expands or collapses the nodes without animation.
- **Slide** — Slide animation applies when expand or collapse node.

The default value of this property is set to **Slide**.

XML

```
<Grid x:Name="Grid">  
  <syncfusion:TreeViewAdv Name="treeViewAdv" AnimationType="Fade">  
    <syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital Status">  
      <syncfusion:TreeViewItemAdv Header="Single"/>  
      <syncfusion:TreeViewItemAdv Header="Married"/>  
      <syncfusion:TreeViewItemAdv Header="Married with Children"/>  
    </syncfusion:TreeViewItemAdv>  
    <syncfusion:TreeViewItemAdv Header="Baby Vaccines">
```

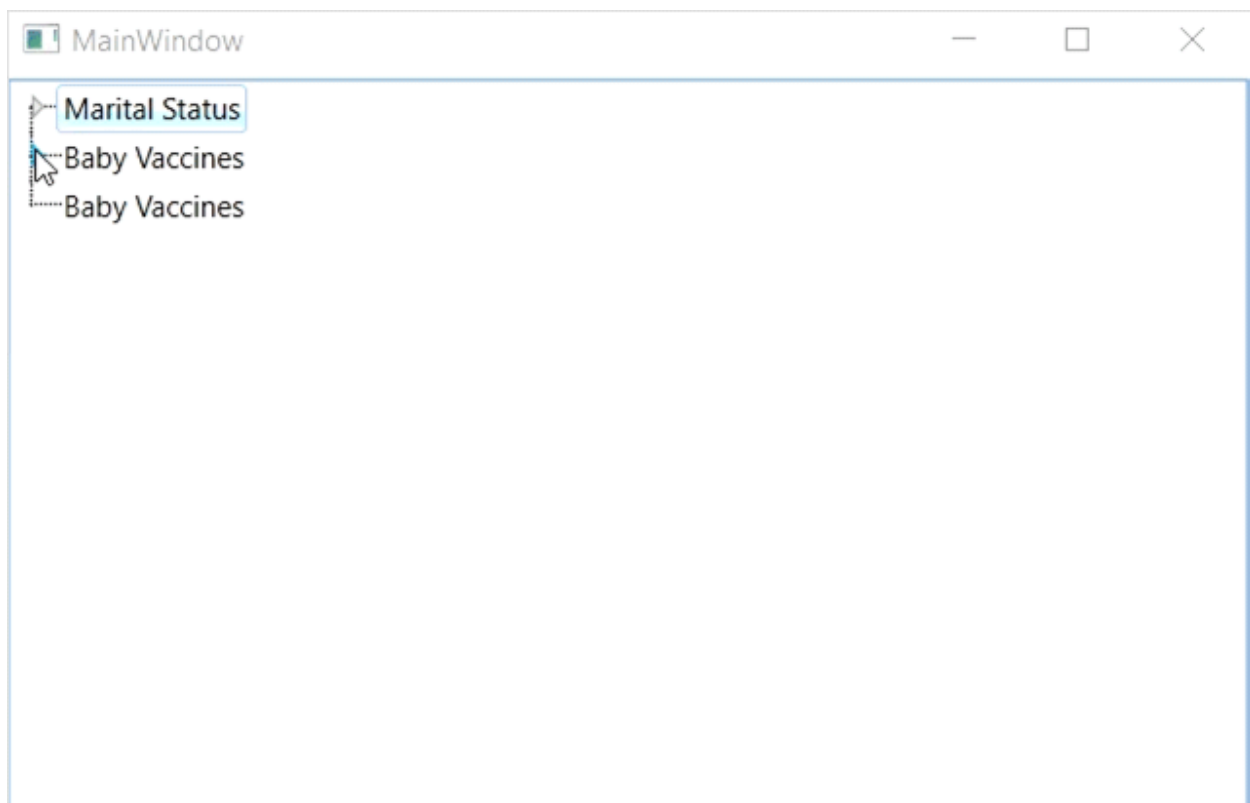
```
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Country Information"/>
</syncfusion:TreeViewAdv>
</Grid>
```

C#

```
treeViewAdv.AnimationType = AnimationType.Fade;
```

VB.NET

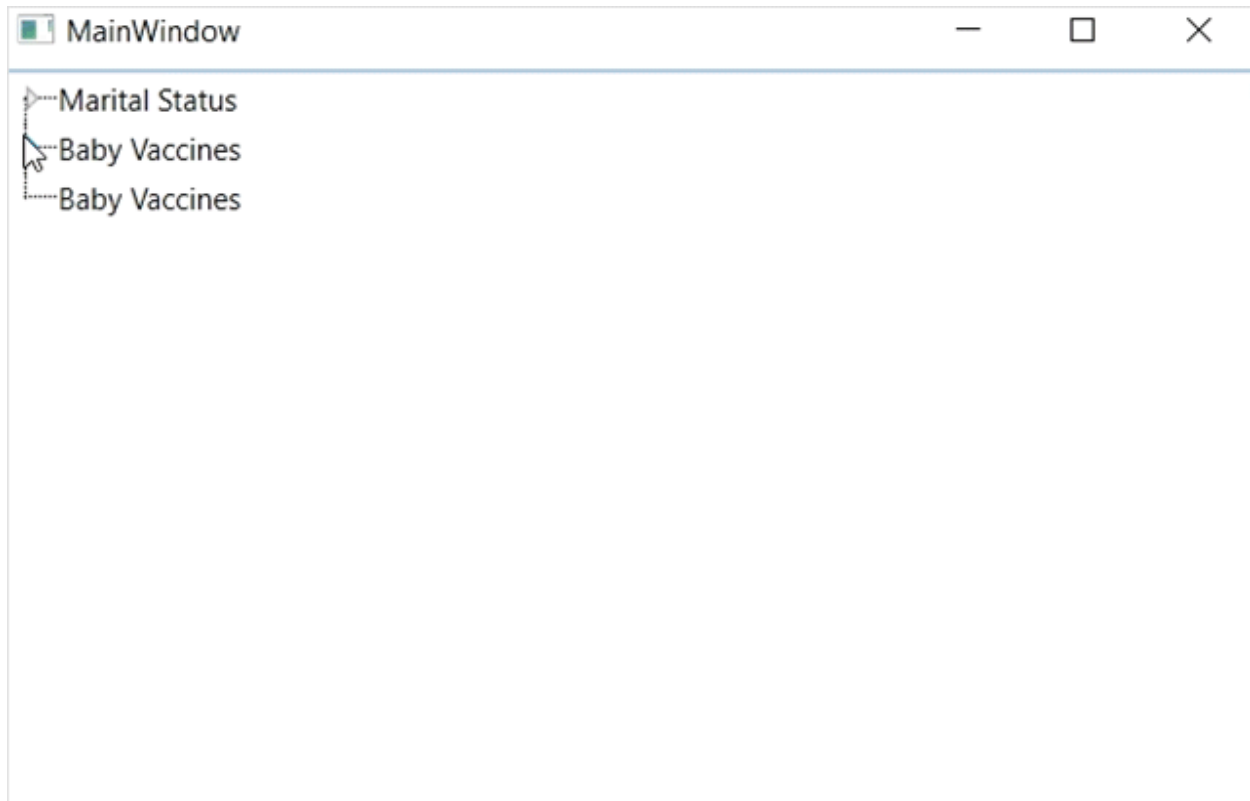
```
treeViewAdv.AnimationType = AnimationType.Fade
```

**C#**

```
treeViewAdv.AnimationType = AnimationType.Slide;
```

VB.NET

```
treeViewAdv.AnimationType = AnimationType.Slide
```



Animation speed

The speed of animation that is generated while expanding or collapsing the TreeViewAdv is controlled by using the [AnimationSpeed](#) property. The default value of this property is set to 1.

XML

```
<Grid x:Name="Grid">
  <syncfusion:TreeViewAdv Name="treeViewAdv" AnimationSpeed="2">
    <syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital Status">
      <syncfusion:TreeViewItemAdv Header="Single"/>
      <syncfusion:TreeViewItemAdv Header="Married"/>
      <syncfusion:TreeViewItemAdv Header="Married with Children"/>
    </syncfusion:TreeViewItemAdv>
    <syncfusion:TreeViewItemAdv Header="Baby Vaccines">
      <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
      <syncfusion:TreeViewItemAdv Header="Tetanus"/>
      <syncfusion:TreeViewItemAdv Header="Polio"/>
      <syncfusion:TreeViewItemAdv Header="Measles"/>
    </syncfusion:TreeViewItemAdv>
    <syncfusion:TreeViewItemAdv Header="Country Information"/>
  </syncfusion:TreeViewAdv>
</Grid>
```

C#

```
treeViewAdv.AnimationSpeed = 2;
```

VB.NET

```
' Set animation speed
treeViewAdv.AnimationSpeed = 2
```

Expand animation

The Expand or Collapse operation in a TreeViewItemAdv leads to an animated action. This animation is controlled by using the [ExpandAnimation](#) property of TreeViewItemAdv. The animation is also applied to the child items. The following code example illustrates how to set this property.

XML

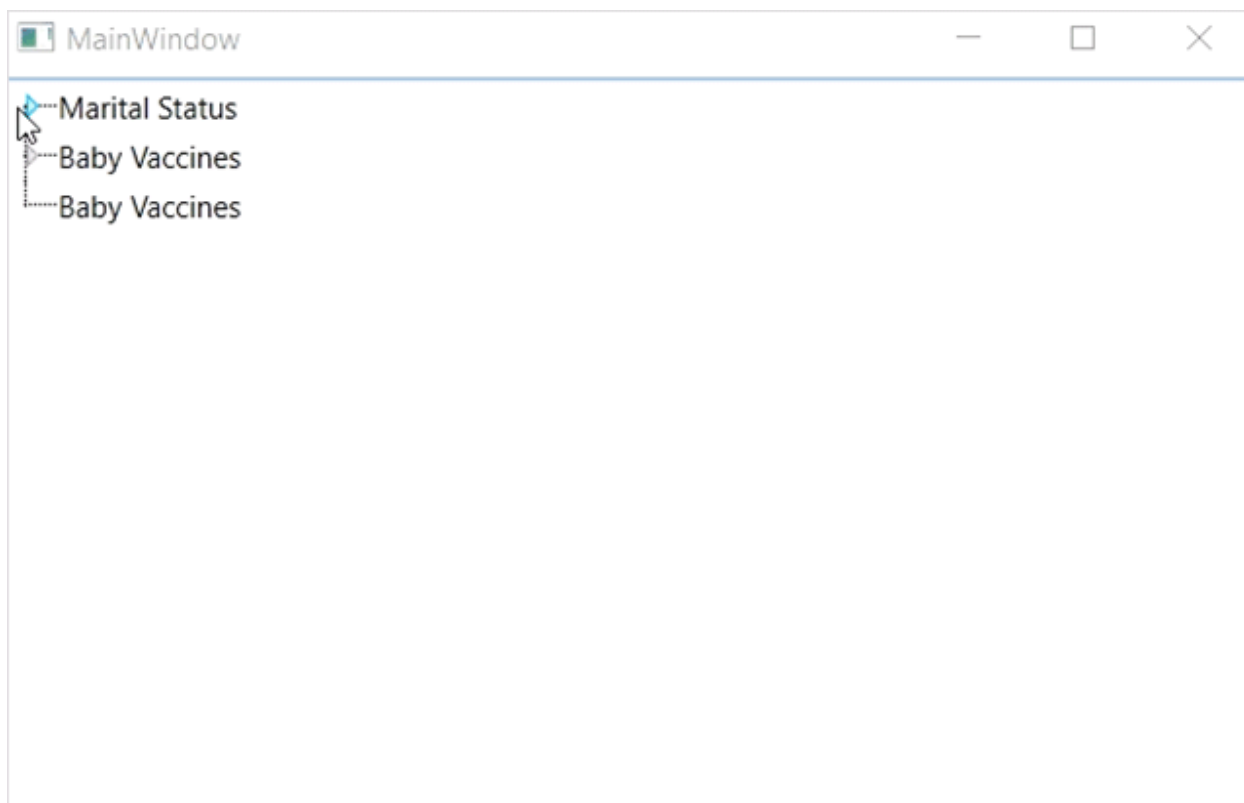
```
<Window.Resources>
<DoubleAnimation x:Key="ExpandAnimation" Duration="0:0:2"/>
</Window.Resources>
<Grid x:Name="Grid">
<syncfusion:TreeViewAdv Name="treeViewAdv" AnimationSpeed="2">
<syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital Status"
ExpandAnimation="{StaticResource ExpandAnimation}">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Baby Vaccines">
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Country Information"/>
</syncfusion:TreeViewAdv>
</Grid>
```

C#

```
DoubleAnimation expandanimation=new DoubleAnimation(){Duration=new
Duration(new TimeSpan(0,0,2))};
treeViewItemAdv.ExpandAnimation = expandanimation;
```

VB.NET

```
Dim expandanimation As New DoubleAnimation() With {.Duration = New
Duration(New TimeSpan(0,0,2))}
treeViewItemAdv.ExpandAnimation = expandanimation
```



Expand/Collapse Images in WPF TreeViewAdv (Classic)

The tree nodes in TreeViewAdv control can have sub nodes or items. On clicking the parent node, it expands to show child nodes. Hence, the nodes switch between collapsed and expanded states, on mouse-click. TreeViewAdv control lets you specify separate images for the expanded and collapsed nodes. You can add images to the TreeViewItemAdv to indicate the state of an item by using the ExpandedImageSource and CollapsedImageSource properties. Images are displayed based on the state of the item.

The following are the two possible states for any item:

- Expanded
- Collapsed

The Expanded state is identified by setting the ExpandedImageSource property and the Collapsed state is identified by setting the CollapsedImageSource property as follows.

XML

```
<syncfusion:TreeViewAdv >
  <syncfusion:TreeViewItemAdv Header="My Computer" x:Name="item1"
    CollapsedImageSource="folder.png">
    <syncfusion:TreeViewItemAdv Header="Music" />
    <syncfusion:TreeViewItemAdv Header="Videos" />
    <syncfusion:TreeViewItemAdv Header="Wallpaper.png" />
    <syncfusion:TreeViewItemAdv Header="Banner.png" />
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="My Network Places" x:Name="item2"
    ExpandedImageSource="Folder_Expanded.png" >
```

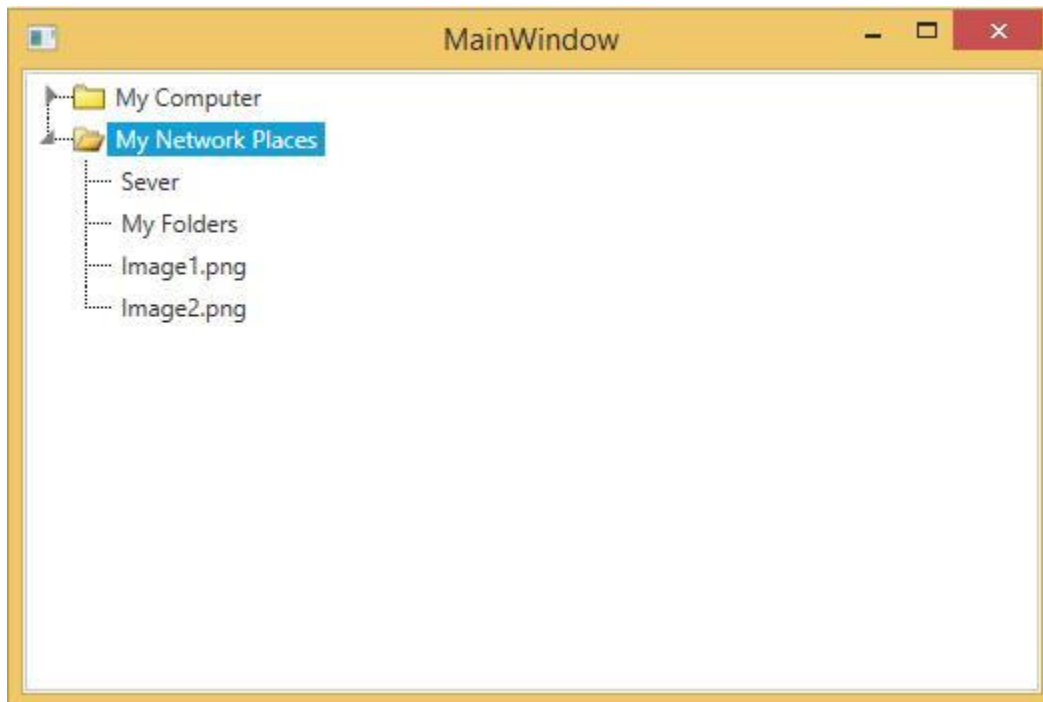
```
<syncfusion:TreeViewItemAdv Header="Sever" />
<syncfusion:TreeViewItemAdv Header="My Folders" />
<syncfusion:TreeViewItemAdv Header="Image1.png" />
<syncfusion:TreeViewItemAdv Header="Image2.png" />
</syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

C#

```
// Set image for collapsed button
item1.CollapsedImageSource = new BitmapImage(new Uri("/folder.png",
UriKind.Relative));
// Set image for expand button
item2.ExpandedImageSource = new BitmapImage(new Uri("/Folder_Expanded.png",
UriKind.Relative));
```

VB.NET

```
' Set image for collapsed button
item1.CollapsedImageSource = New BitmapImage(New Uri("/folder.png",
UriKind.Relative))
' Set image for expand button
item2.ExpandedImageSource = New BitmapImage(New Uri("/Folder_Expanded.png",
UriKind.Relative))
```



Node editing in WPF TreeViewAdv (Classic)

TreeViewAdv allow user to edit/non-edit the tree node items at runtime, by enable/disable IsEditable property. Default value of this property is true. At runtime, you can edit a node, by simply clicking the node text

The following code example illustrates how to set this property.

XML

```
<syncfusion:TreeViewAdv Name="treeViewAdv" >
  <!-- Able to edit this node because IsEditable is true-->
  <syncfusion:TreeViewItemAdv Name="item1" Header="Marital Status">
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
  </syncfusion:TreeViewItemAdv>
  <!-- Unable to edit this node because IsEditable is false-->
  <syncfusion:TreeViewItemAdv Header="Baby Vaccines" Name="item2"
  IsEditable="False">
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
  </syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

C#

```
//Set IsEditable is false
item2.IsEditable = false;
```

VB.NET

```
'Set IsEditable is false
item2.IsEditable = False
```



Setting node in EditMode

We can also set a node to be in edit mode when the tree view loads. This is done using the `IsInEditMode` property. When this property is set to **true**, the particular node will be in edit mode when it loads

XML

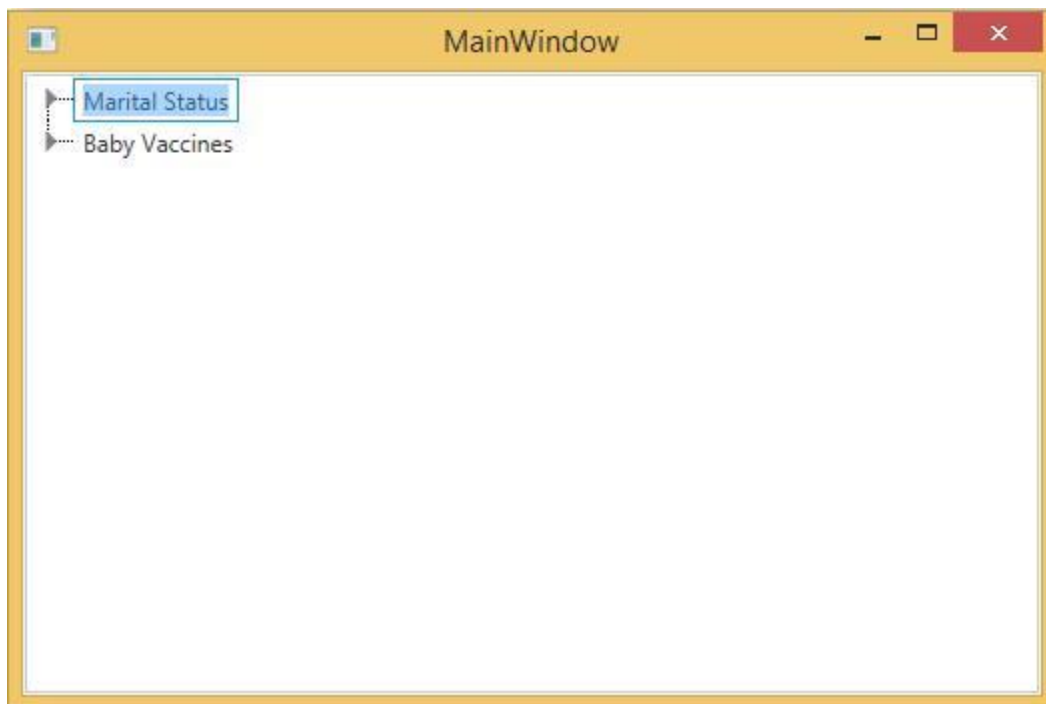
```
<syncfusion:TreeViewAdv Name="treeViewAdv" >
<syncfusion:TreeViewItemAdv Name="item1" Header="Marital Status"
IsInEditMode="True">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Baby Vaccines" Name="item2" >
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

C#

```
//Set edit mode
item1.IsInEditMode = true;
```

VB.NET

```
'Set edit mode
item1.IsInEditMode = True
```



Node editing event

The following events are handled when the tree node is in Edit Mode.

- BeforeItemEdit—Occurs when the IsInEditMode property changes. This event is handled before the TreeViewItemAdv enters the edit mode.
- AfterItemEdit—Occurs when the IsInEditMode property changes. This event is handled after the edit operations are completed.
- EditKeyUp—Occurs when a key is raised, when the item in edit mode. This event is handled when the item is in edit mode.
- EditKeyDown—Occurs when a key is raised, when the item is in edit mode. This event is handled when the item is in edit mode.

XML

```
<syncfusion:TreeViewAdv Name="treeViewAdv" >
  <syncfusion:TreeViewItemAdv Name="treeviewitem" Header="Marital Status"
    BeforeItemEdit="treeviewitem_BeforeItemEdit"
    AfterItemEdit="treeviewitem_AfterItemEdit"
    EditKeyUp="treeviewitem_EditKeyUp" EditKeyDown="treeviewitem_EditKeyDown" >
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Baby Vaccines" Name="item2" >
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
  </syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

C#

```
private void treeviewitem_BeforeItemEdit(object sender,
EditModeChangeEventArgs e)
{
    Debug.WriteLine("BeforeItemEdit: old( " + e.OldValue + "), new( " +
e.NewValue + ")");
}
private void treeviewitem_AfterItemEdit(object sender,
EditModeChangeEventArgs e)
{
    Debug.WriteLine("AfterItemEdit: old( " + e.OldValue + "), new( " +
e.NewValue + ")");
}
private void treeviewitem_EditKeyUp(object sender, KeyEventArgs e)
{
    Debug.WriteLine("Up: " + e.Key);
}
private void treeviewitem_EditKeyDown(object sender, KeyEventArgs e)
{
    Debug.WriteLine("Down: " + e.Key);
}
```

VB.NET

```

Private Sub treeviewitem_BeforeItemEdit(ByVal sender As Object, ByVal e As
EditModeChangeEventArgs)
Debug.WriteLine("BeforeItemEdit: old( " & e.OldValue & "), new( " &
e.NewValue & ")")
End Sub
Private Sub treeviewitem_AfterItemEdit(ByVal sender As Object, ByVal e As
EditModeChangeEventArgs)
Debug.WriteLine("AfterItemEdit: old( " & e.OldValue & "), new( " &
e.NewValue & ")")
End Sub
Private Sub treeviewitem_EditKeyUp(ByVal sender As Object, ByVal e As
KeyEventArgs)
Debug.WriteLine("Up: " & e.Key)
End Sub
Private Sub treeviewitem_EditKeyDown(ByVal sender As Object, ByVal e As
KeyEventArgs)
Debug.WriteLine("Down: " & e.Key)
End Sub

```

Root lines for Items in WPF TreeViewAdv (Classic)

The TreeViewAdv displays root lines, which link the nodes of a tree structure. These TreeViewAdv root lines are displayed or hidden by using the ShowRootLines property of the class TreeViewAdv. To set this property, use the below code

XML

```

<!-- Adding TreeViewAdv With show root lines -->
<syncfusion:TreeViewAdv Name="treeViewAdv" ShowRootLines="False">
<!-- Adding TreeViewItemAdv -->
<syncfusion:TreeViewItemAdv Header="Marital Status">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Baby Vaccines">
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Country Information"/>
</syncfusion:TreeViewAdv>

```

C#

```

// Show root lines
treeViewAdv.ShowRootLines = false;

```

VB.NET

```

' Show root lines
treeViewAdv.ShowRootLines = False

```



Selecting a Node in WPF TreeViewAdv (Classic)

TreeViewAdv allow user to select single or multiple node by setting AllowMultiSelect property.

- Selecting an item through programmatically
- Enable to allow Multiple Selection in TreeViewAdv
- Selecting an items through AddNodeToSelectedItems Collection

Selecting an item through programmatically

You can show a node to be selected, at runtime, when the TreeViewAdv control is loaded, by setting the IsSelected property of a particular node to **true**. Here is the code snippet to enable this property.

XML

```
<!-- Adding TreeViewAdv with selected -->
<syncfusion:TreeViewAdv Name="treeViewAdv">
  <!-- Adding TreeViewItemAdv -->
  <syncfusion:TreeViewItemAdv Name="treeViewItemAdv1" IsSelected="True"
  Header="Marital Status">
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Name="treeViewItemAdv2" Header="Baby Vaccines">
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Country Information">
    <syncfusion:TreeViewItemAdv Header="Canada"/>
    <syncfusion:TreeViewItemAdv Header="France"/>
  </syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

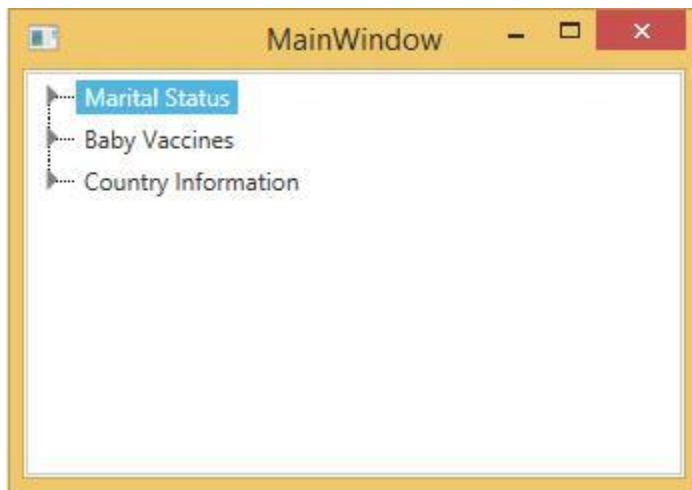
```
<syncfusion:TreeViewItemAdv Header="Germany"/>
<syncfusion:TreeViewItemAdv Header="UK"/>
<syncfusion:TreeViewItemAdv Header="USA"/>
</syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

C#

```
//Set selection
treeViewItemAdv1.IsSelected = true;
```

VB.NET

```
'Set selection
treeViewItemAdv1.IsSelected = True
```

**Enable to allow multiple selection in TreeViewAdv**

The TreeViewAdv control supports selecting multiple items by using the CTRL or SHIFT keys. The selected items are dragged to any item or node within the same control or to another TreeViewAdv control. This is achieved by enabling the AllowMultiSelect property.

The following code example can be used to set this property.

XML

```
<!-- Adding TreeViewAdv with multiselection of items -->
<syncfusion:TreeViewAdv AllowMultiSelect="True" Name="treeViewAdv">
<!-- Adding TreeViewItemAdv -->
<syncfusion:TreeViewItemAdv Header="Marital Status">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Baby Vaccines">
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
```

```

</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Country Information"/>
</syncfusion:TreeViewAdv>

```

C#

```

// Enable multi select of items
treeViewAdv.AllowMultiSelect = true;

```

VB.NET

```

' Enable multi select of items
treeViewAdv.AllowMultiSelect = True

```



Selecting an item through [AddNodeToSelectedItems](#) collection

TreeViewAdv control provides support to select multiple treeview items programmatically. This is achieved by using the `AddNodeToSelectedItems` method which is used to select multiple items by iterating through each and every item in the TreeViewAdv control.

Here are the code snippets to select multiple items in the TreeViewAdv control using C# code.

C#

```

foreach (TreeViewItemAdv items in this.treeView.Items)
{
    treeView.AddNodeToSelectedItems(items);
}

```

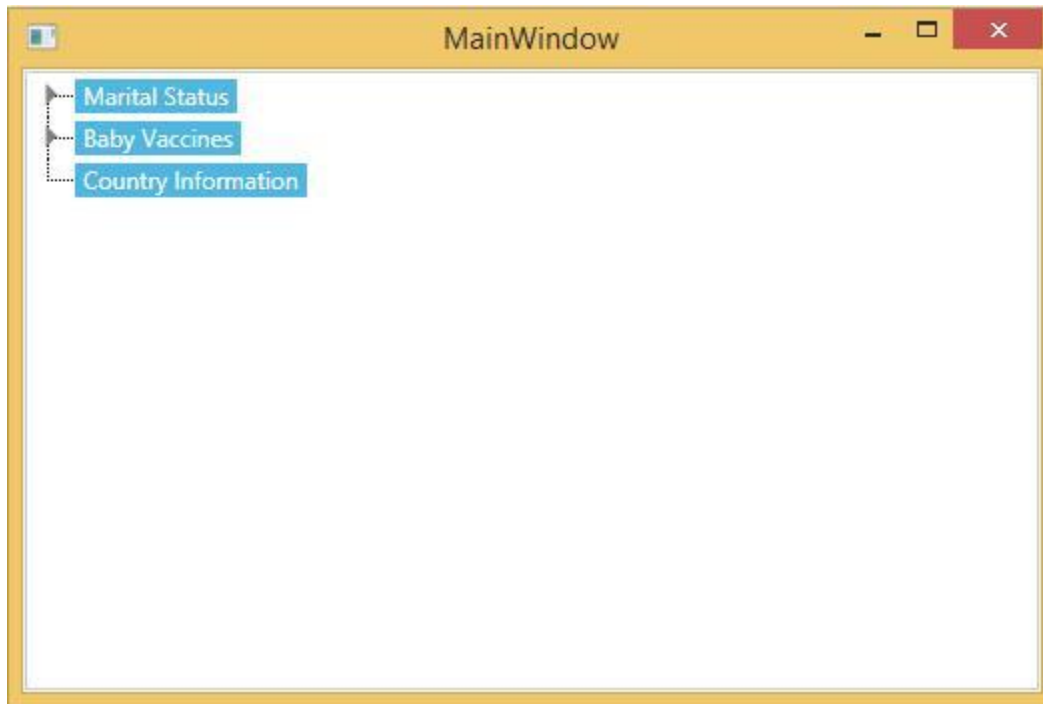
VB.NET

```

For Each items As TreeViewItemAdv In Me.treeView.Items

```

```
treeView.AddNodeToSelectedItems(items)  
Next items
```



Sorting TreeViewItemAdv in WPF TreeViewAdv (Classic)

TreeViewAdv has the advanced ability to sort the TreeViewItemAdv items at run time. The **Sorting** property of the control allows you to specify the direction of sorting. The sorting options are as follows.

- Ascending
- Descending
- None (default)

XML

```
<syncfusion:TreeViewAdv Name="treeViewAdv" Sorting="Ascending" >  
  <!-- Adding TreeViewItemAdv -->  
  <syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital  
Status">  
    <syncfusion:TreeViewItemAdv Header="Single"/>  
    <syncfusion:TreeViewItemAdv Header="Married"/>  
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>  
  </syncfusion:TreeViewItemAdv>  
  <syncfusion:TreeViewItemAdv Header="Baby Vaccines">  
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>  
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>  
    <syncfusion:TreeViewItemAdv Header="Polio"/>  
    <syncfusion:TreeViewItemAdv Header="Measles"/>  
  </syncfusion:TreeViewItemAdv>  
  <syncfusion:TreeViewItemAdv Header="Country Information">  
    <syncfusion:TreeViewItemAdv Header="Canada"/>  
    <syncfusion:TreeViewItemAdv Header="France"/>  
  </syncfusion:TreeViewItemAdv>  
</syncfusion:TreeViewAdv>
```



```
<syncfusion:TreeViewItemAdv Header="Germany"/>
<syncfusion:TreeViewItemAdv Header="UK"/>
<syncfusion:TreeViewItemAdv Header="USA"/>
</syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

C#

```
//Sort the item
treeViewAdv.Sorting = SortDirection.Ascending;
```

VB.NET

```
'Sort the item
treeViewAdv.Sorting = SortDirection.Ascending
```

Sorting field

Using **SortingField** property, you can specify a sorting criteria. This is dependency property, which gets or sets the property name being used as the sorting criteria. The default value is **Header**.

XML

```
<syncfusion:TreeViewAdv Name="treeViewAdv" Sorting="Ascending"
    SortingField="Header" >
    <!-- Adding TreeViewItemAdv -->
    <syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital
    Status">
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
    </syncfusion:TreeViewItemAdv>
    <syncfusion:TreeViewItemAdv Header="Baby Vaccines">
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
    </syncfusion:TreeViewItemAdv>
    <syncfusion:TreeViewItemAdv Header="Country Information">
    <syncfusion:TreeViewItemAdv Header="Canada"/>
    <syncfusion:TreeViewItemAdv Header="France"/>
    <syncfusion:TreeViewItemAdv Header="Germany"/>
    <syncfusion:TreeViewItemAdv Header="UK"/>
    <syncfusion:TreeViewItemAdv Header="USA"/>
    </syncfusion:TreeViewItemAdv>
    </syncfusion:TreeViewAdv>
```

C#

```
//Sort the item based on criteria
treeViewAdv.SortingField = "Header";
```

VB.NET

```
'Sort the item based on criteria
```

```
treeViewAdv.SortingField = "Header"
```



Dragging TreeView items in WPF TreeViewAdv (Classic)

TreeViewAdv control enables to drag TreeView items from one location to another. This is done by enabling the AllowDragDrop property.

Use the following code to enable this property

XML

```
<syncfusion:TreeViewAdv Name="treeViewAdv" AllowDragDrop="True" >
  <syncfusion:TreeViewItemAdv Name="treeviewitem" Header="Marital Status" >
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Baby Vaccines" Name="item2" >
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Country Information" />
</syncfusion:TreeViewAdv>
```

C#

```
//set allow drag and drop
treeViewAdv.AllowDragDrop = true;
```

VB.NET

```
'set allow drag and drop
treeViewAdv.AllowDragDrop = True
```



Transparent dragging image

The TreeViewAdv control provides support to change the opacity of an element being dragged. By using the DraggingContainerOpacity property, we can change the opacity value of the dragged element. It is useful to be able to view the content behind the dragged element.

XML

```
<syncfusion:TreeViewAdv Name="treeViewAdv" AllowDragDrop="True"
DraggingContainerOpacity="0.4" >
  <syncfusion:TreeViewItemAdv Name="treeviewitem" Header="Marital Status" >
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Baby Vaccines" Name="item2" >
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Country Information" />
</syncfusion:TreeViewAdv>
```

C#

```
//set DraggingContainerOpacity
treeViewAdv.DraggingContainerOpacity = 0.4;
```

VB.NET

```
'set DraggingContainerOpacity
treeViewAdv.DraggingContainerOpacity = 0.4
```

**Fake drag indicator**

Fake Drag Indicator to indicate where TreeViewItemAdv may be placed during drag and drop operations. This is achieved by enabling the “IsFakeDragIndicator” property.

The following code example can be used to set this property

XML

```
<syncfusion:TreeViewAdv Name="treeViewAdv" IsFakeDragIndicator="True">
  <syncfusion:TreeViewItemAdv Name="treeviewitem" Header="Marital Status" >
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Baby Vaccines" Name="item2" >
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Country Information" />
</syncfusion:TreeViewAdv>
```

C#

```
//set FakeDragIndicator is true
treeViewAdv.IsFakeDragIndicator = true;
```

VB.NET

```
'set FakeDragIndicator is true
treeViewAdv.IsFakeDragIndicator = True
```

**TreeViewVirtualization in WPF TreeViewAdv (Classic)**

The TreeViewVirtualization feature enables users to reduce the loading time of TreeViewItems regardless of the items count.

VirtualizationMode:

Sets the virtualization mode for TreeViewItems. If VirtualizationMode is set to Normal, virtualization logic is handled internally. If VirtualizationMode is set to Extended, then the class used for creating the business object for the TreeViewItem has to implement the IVirtualTree interface.

VirtualizationMode is Normal

Virtualization is processed internally (without IVirtualTree).

XML

```
<syncfusion:TreeViewAdv Name="treeViewAdv" ItemsSource="{Binding MyItems}"
IsVirtualizing="True" VirtualizationMode="Normal" >
</syncfusion:TreeViewAdv>
```

C#

```
treeViewAdv.IsVirtualizing = true;
treeViewAdv.VirtualizationMode = VirtualizationMode.Normal;
```

VB.NET

```
treeViewAdv.IsVirtualizing = True
treeViewAdv.VirtualizationMode = VirtualizationMode.Normal
```

VirtualizationMode is extended

IVirtualTree interface has to be implemented by the class creating the business object for TreeViewItem.

XML

```
<syncfusion:TreeViewAdv Name="treeViewAdv" AllowDragDrop="True"
IsVirtualizing="True" VirtualizationMode="Extended" >
<syncfusion:TreeViewAdv.DataContext>
<local:ViewModel />
</syncfusion:TreeViewAdv.DataContext>
<syncfusion:TreeViewAdv.ItemTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding Items}">
<TextBlock Text="{Binding Header}"/>
</HierarchicalDataTemplate>
</syncfusion:TreeViewAdv.ItemTemplate>
</syncfusion:TreeViewAdv>
```

C#

```
treeViewAdv.IsVirtualizing = true;
treeViewAdv.VirtualizationMode = VirtualizationMode.Extended;
```

VB.NET

```
treeViewAdv.IsVirtualizing = True
treeViewAdv.VirtualizationMode = VirtualizationMode.Extended
```

Implementing IVirtualTree interface

ViewModel class:

C#

```
public class ViewModel
{
    public ObservableCollection<MyTreeView> MyItems
    {
        get;
        set;
    }
    public ViewModel()
    {
        MyItems = new ObservableCollection<MyTreeView>();
        for (int i = 0; i < 1000; i++)
        {
            MyTreeView myitem = new MyTreeView() { Header = "Module " +
            i.ToString() };
            for (int j = 0; j < 100; j++)
            {
                MyTreeView _myitem = new MyTreeView() { Header = "Sub Module " +
                j.ToString() };
            }
        }
    }
}
```

```
myitem.Items.Add(_myitem);  
}  
MyItems.Add(myitem);  
}  
}  
}
```

VB.NET

```
Public Class ViewModel  
Public Property MyItems() As ObservableCollection(Of MyTreeView)  
Public Sub New()  
MyItems = New ObservableCollection(Of MyTreeView)()  
For i As Integer = 0 To 999  
Dim myitem As New MyTreeView() With {.Header = "Module " & i.ToString()}  
For j As Integer = 0 To 99  
Dim _myitem As New MyTreeView() With {.Header = "Sub Module " &  
j.ToString()}  
myitem.Items.Add(_myitem)  
Next j  
MyItems.Add(myitem)  
Next i  
End Sub  
End Class
```

Model class with IVirtualTree

C#

```
public class Model : IVirtualTree  
{  
public Model()  
{  
items = new ObservableCollection<Model>();  
}  
private string header;  
public string Header  
{  
Get  
{  
return header;  
}  
Set  
{  
header = value;  
}  
}  
private ObservableCollection<Model> items;  
public ObservableCollection<Model> Items  
{  
Get  
{  
return items;  
}  
Set  
{
```

```

items = value;
}
}
public double ExtentHeight
{
    get;
    set;
}
public bool IsExpanded
{
    get;
    set;
}
public int ItemsCount
{
    get;
    set;
}
public IVirtualTree Parent
{
    get;
    set;
}
public bool IsSelected
{
    get;
    set;
}
}

```

VB.NET

```

Public Class Model
    Implements IVirtualTree
    Public Sub New()
        items_Renamed = New ObservableCollection(Of Model) ()
    End Sub
    Private header_Renamed As String
    Public ReadOnly Property Header() As String
        [Get]
        If True Then
            Return header_Renamed
        End If
        [Set]
        If True Then
            header_Renamed = value
        End If
    End Property
    Private items_Renamed As ObservableCollection(Of Model)
    Public ReadOnly Property Items() As ObservableCollection(Of Model)
        [Get]
        If True Then
            Return items_Renamed
        End If
        [Set]
        If True Then

```



```

items_Renamed = value
End If
End Property
Public Property ExtentHeight() As Double
Public Property IsExpanded() As Boolean
Public Property ItemsCount() As Integer
Public Property Parent() As IVirtualTree
Public Property IsSelected() As Boolean
End Class

```

LoadOnDemand in WPF TreeViewAdv (Classic)

The LoadOnDemand feature enables users to load items dynamically when a particular TreeViewItem is expanded. Hence the items are loaded on demand and reduce the loading time.

1. To load items on demand:
 - **LoadOnDemand** event is used to load the sub-items when a particular item is expanded.
 - **LoadingHeader** is used to display the text while sub-items are being loaded.
 - **IsLoadOnDemand** value has to be set to true if the item is to be loaded on demand.

XML

```

<syncfusion:TreeViewAdv x:Name="treeViewAdv"
LoadOnDemand="treeViewAdv_LoadOnDemand">
<syncfusion:TreeViewItemAdv Header="Root1" IsLoadOnDemand="True"
LoadingHeader="Loading....." />
<syncfusion:TreeViewItemAdv Header="Root2" IsLoadOnDemand="True"
LoadingHeader="Loading....." />
</syncfusion:TreeViewAdv>

```

C#

```

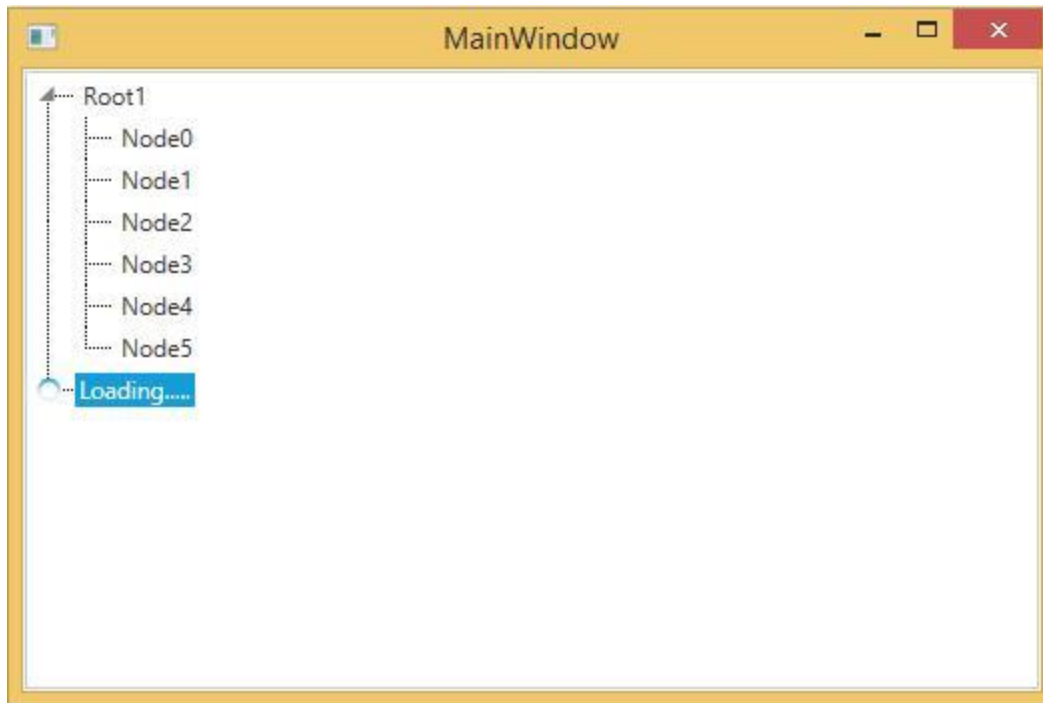
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        timer = new DispatcherTimer();
        timer.Interval = new TimeSpan(0, 0, 0, 0, 800);
        timer.Tick += new EventHandler(timer_Tick);
    }
    private void timer_Tick(object sender, EventArgs e)
    {
        if (loadingitem != null)
        {
            for (int i = 0; i <= 5; i++)
            {
                loadingitem.Items.Add(new TreeViewItemAdv() { Header = "Node" + i });
            }
            loadingitem.IsLoadOnDemand = false;
            timer.Stop();
        }
    }
    private DispatcherTimer timer;
    private TreeViewItemAdv loadingitem;
}

```

```
private void treeViewAdv_LoadOnDemand(object sender, LoadonDemandEventArgs
args)
{
    loadingitem = args.TreeViewItem as TreeViewItemAdv;
    timer.Start();
}
}
```

VB.NET

```
Partial Public Class MainWindow
Inherits Window
Public Sub New()
InitializeComponent()
timer = New DispatcherTimer()
timer.Interval = New TimeSpan(0, 0, 0, 0, 800)
AddHandler timer.Tick, AddressOf timer_Tick
End Sub
Private Sub timer_Tick(ByVal sender As Object, ByVal e As EventArgs)
If loadingitem IsNot Nothing Then
For i As Integer = 0 To 5
loadingitem.Items.Add(New TreeViewItemAdv() With {.Header = "Node" & i})
Next i
loadingitem.IsLoadOnDemand = False
timer.Stop()
End If
End Sub
Private timer As DispatcherTimer
Private loadingitem As TreeViewItemAdv
Private Sub treeViewAdv_LoadOnDemand(ByVal sender As Object, ByVal args As
LoadonDemandEventArgs)
loadingitem = TryCast(args.TreeViewItem, TreeViewItemAdv)
timer.Start()
End Sub
End Class
```



Creating a MultiColumnTreeView in WPF TreeViewAdv (Classic)

TreeView control can be created with multiple columns by setting the `MultiColumnEnable` property to **true**. This is dependency property, which gets or sets the value defining whether items are in multicolumn mode. The default value is **false**

XML

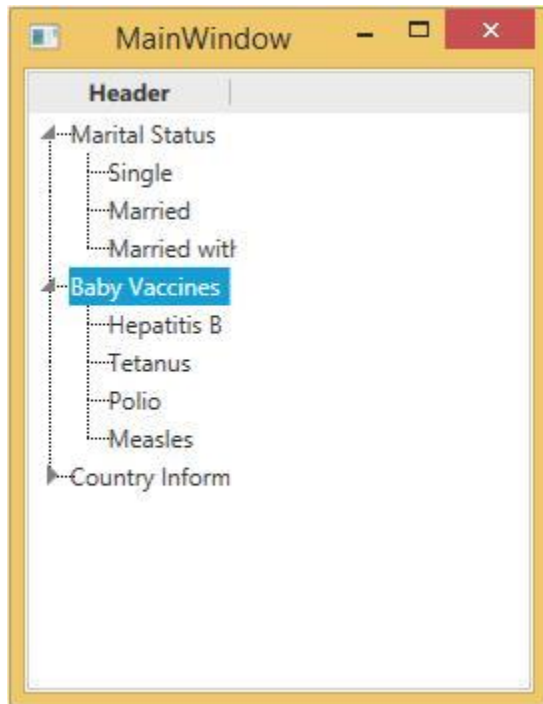
```
<!-- Adding TreeViewAdv with Enabling multiple column -->
<syncfusion:TreeViewAdv Name="treeViewAdv" MultiColumnEnable="True">
  <!-- Adding TreeViewItemAdv -->
  <syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital Status">
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Baby Vaccines">
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Country Information">
    <syncfusion:TreeViewItemAdv Header="Canada"/>
    <syncfusion:TreeViewItemAdv Header="France"/>
    <syncfusion:TreeViewItemAdv Header="Germany"/>
    <syncfusion:TreeViewItemAdv Header="UK"/>
    <syncfusion:TreeViewItemAdv Header="USA"/>
  </syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

C#

```
//Enable multiple column enable
treeViewAdv.MultiColumnEnable = true;
```

VB.NET

```
'Enable multiple column enable
treeViewAdv.MultiColumnEnable = True
```



Header for MultiColumn

TreeViewAdv allow user to set headers for individual columns using the Columns property. All the columns are defined in TreeViewColumnCollections.

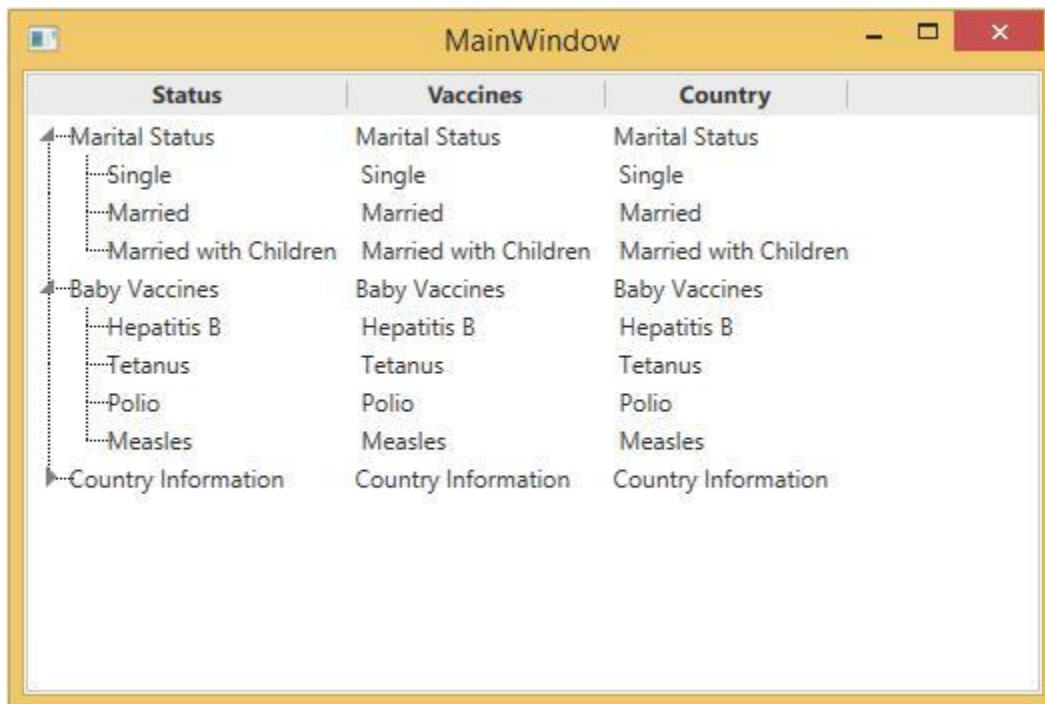
XML

```
<!-- Adding TreeViewAdv with Enabling multiple column -->
<syncfusion:TreeViewAdv Name="treeViewAdv" MultiColumnEnable="True">
  <!-- Adding TreeViewItemAdv -->
  <syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital Status">
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Baby Vaccines">
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Country Information">
    <syncfusion:TreeViewItemAdv Header="Canada"/>
    <syncfusion:TreeViewItemAdv Header="France"/>
    <syncfusion:TreeViewItemAdv Header="Germany"/>
  </syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
```

```

<syncfusion:TreeViewItemAdv Header="UK"/>
<syncfusion:TreeViewItemAdv Header="USA"/>
</syncfusion:TreeViewItemAdv>
<!-- Adding header -->
<syncfusion:TreeViewAdv.Columns>
<syncfusion:TreeViewColumnCollection>
<syncfusion:TreeViewColumn Width="150" Header="Status"
DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
<syncfusion:TreeViewColumn Width="100" Header="Vaccines"
DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
<syncfusion:TreeViewColumn Width="50" Header="Country"
DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
</syncfusion:TreeViewColumnCollection>
</syncfusion:TreeViewAdv.Columns>
</syncfusion:TreeViewAdv>

```



[View sample in GitHub](#)

Auto-Resize of columns in Multicolumn TreeView

The width property of TreeViewColumn has changed from a double type to a GridLength type that can be set to the Auto or * values. When the column width is set as Auto, the desired size or minimum size will be set as the width of the column. When the width is set as *, the remaining space that is available in the window will be set as the width of the column. Similarly, the value can be set as 0.3, 2, 128, etc.

XML

```

<!-- Adding TreeViewAdv with Enabling multiple column -->
<syncfusion:TreeViewAdv Name="treeViewAdv" MultiColumnEnable="True">
<!-- Adding TreeViewItemAdv -->

```

```

<syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="111">
<syncfusion:TreeViewItemAdv Header="211"/>
<syncfusion:TreeViewItemAdv Header="212"/>
<syncfusion:TreeViewItemAdv Header="213"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Steve">
<syncfusion:TreeViewItemAdv Header="Charles"/>
<syncfusion:TreeViewItemAdv Header="Greg"/>
<syncfusion:TreeViewItemAdv Header="Danielle"/>
<syncfusion:TreeViewItemAdv Header="Ethan"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="President">
<syncfusion:TreeViewItemAdv Header="TeamLead"/>
<syncfusion:TreeViewItemAdv Header="TeamLead"/>
<syncfusion:TreeViewItemAdv Header="ProductLead"/>
<syncfusion:TreeViewItemAdv Header="ProductManager"/>
</syncfusion:TreeViewItemAdv>
<!-- Adding header -->
<syncfusion:TreeViewAdv.Columns>
<syncfusion:TreeViewColumnCollection>
<syncfusion:TreeViewColumn Width="90" Header="ID"
DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
<syncfusion:TreeViewColumn Width="Auto" Header="FirstName"
DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
<syncfusion:TreeViewColumn Width="200" Header="Role"
DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
</syncfusion:TreeViewColumnCollection>
</syncfusion:TreeViewAdv.Columns>
</syncfusion:TreeViewAdv>

```



Allowing reordering columns

TreeViewAdv control now provides support to reorder the columns in the TreeViewAdv control by using the AllowsColumnReorder property. Enable this property using the below code.

XML

```
<!-- Adding TreeViewAdv with Enabling multiple column -->
<syncfusion:TreeViewAdv Name="treeViewAdv" MultiColumnEnable="True">
  <!-- Adding TreeViewItemAdv -->
  <syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="111">
    <syncfusion:TreeViewItemAdv Header="211"/>
    <syncfusion:TreeViewItemAdv Header="212"/>
    <syncfusion:TreeViewItemAdv Header="213"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Steve">
    <syncfusion:TreeViewItemAdv Header="Charles"/>
    <syncfusion:TreeViewItemAdv Header="Greg"/>
    <syncfusion:TreeViewItemAdv Header="Danielle"/>
    <syncfusion:TreeViewItemAdv Header="Ethan"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="President">
    <syncfusion:TreeViewItemAdv Header="TeamLead"/>
    <syncfusion:TreeViewItemAdv Header="TeamLead"/>
    <syncfusion:TreeViewItemAdv Header="ProductLead"/>
    <syncfusion:TreeViewItemAdv Header="ProductManager"/>
  </syncfusion:TreeViewItemAdv>
  <!-- Adding header -->
  <syncfusion:TreeViewAdv.Columns>
    <syncfusion:TreeViewColumnCollection>
      <syncfusion:TreeViewColumn Width="200" Header="ID"
        DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
        AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
      <syncfusion:TreeViewColumn Width="Auto" Header="FirstName"
        DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
        AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
      <syncfusion:TreeViewColumn Width="*" Header="Role"
        DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
        AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
    </syncfusion:TreeViewColumnCollection>
  </syncfusion:TreeViewAdv.Columns>
</syncfusion:TreeViewAdv>
```

Binding SelectedItem in MVVM Pattern in WPF TreeViewAdv (Classic)

A new set of properties namely SelectedTreeItem have been defined to bind SelectedItem value for TreeViewAdv in MVVM pattern.

Use case scenario

Users can get the SelectedItem using SelectedTreeItem property in order to enable two way binding between SelectedItem property in View and SelectedTreeItem property of TreeViewAdv, without modifying the actual SelectedItem property of TreeViewAdv.

Property	Description
SelectedTreeItem	This can be used to bind the Selected TreeView Item to a property of MVVM sample.

Samples Location:

The samples are located in the following location:

[TreeView_SelectedTreeIndex.Zip](#)

Customizing Data Templates in WPF TreeViewAdv (Classic)

This section explains about customizing the TreeViewItemAdv using DataTemplate

ItemTemplate

The user can customize the business object that has to be displayed as TreeViewItemAdv using the ItemTemplate of TreeViewAdv. Since TreeViewAdv displays the hierarchical data, the HierarchicalDataTemplate is used to define the ItemTemplate. In the following example, business object is displayed as CheckBox

We have generate Business object

Model class:**C#**

```
public class Model : NotificationObject
{
    public Model()
    {
        Models = new ObservableCollection<Model>();
    }
    private string caption = string.Empty;
    public string Caption
    {
        Get
        {
            return caption;
        }
        Set
        {
            caption = value;
            this.RaisePropertyChanged(() => this.Caption);
        }
    }
    private bool isChecked = false;
    public bool IsChecked
    {
        Get
        {
            return isChecked;
        }
        Set
        {
            isChecked = value;
            this.RaisePropertyChanged(() => this.IsChecked);
        }
    }
    private bool isCheckable = false;
    public bool IsCheckable
    {

```



```

Get
{
    return isCheckedable;
}
Set
{
    isCheckedable = value;
    this.RaisePropertyChanged(() => this.IsCheckedable);
}
}
private ObservableCollection<Model> models = null;
public ObservableCollection<Model> Models
{
    Get
    {
        return models;
    }
    Set
    {
        models = value;
        this.RaisePropertyChanged(() => this.Models);
    }
}
}

```

VB.NET

```

Public Class Model
    Inherits NotificationObject
    Public Sub New()
        Models = New ObservableCollection(Of Model) ()
    End Sub
    'INSTANT VB NOTE: The variable caption was renamed since Visual Basic does
    'not allow variables and other class members to have the same name:
    Private caption_Renamed As String = String.Empty
    Public ReadOnly Property Caption() As String
    [Get]
    If True Then
        Return caption_Renamed
    End If
    [Set]
    If True Then
        caption_Renamed = value
        Me.RaisePropertyChanged(Function() Me.Caption)
    End If
    End Property
    Private isChecked As Boolean = False
    Public ReadOnly Property IsChecked() As Boolean
    [Get]
    If True Then
        Return isChecked
    End If
    [Set]
    If True Then
        isChecked = value
        Me.RaisePropertyChanged(Function() Me.IsChecked)
    End If
    End Property

```

```

End If
End Property
Private isCheckedable As Boolean = False
Public ReadOnly Property IsCheckable() As Boolean
[Get]
If True Then
Return isCheckedable
End If
[Set]
If True Then
isCheckedable = value
Me.RaisePropertyChanged(Function() Me.IsCheckable)
End If
End Property
Public ReadOnly Property Models() As ObservableCollection(Of Model)
[Get]
If True Then
Return models_Renamed
.
End If
[Set]
If True Then
models_Renamed = value
Me.RaisePropertyChanged(Function() Me.Models)
End If
End Property
End Class

```

ViewModel class:

C#

```

public class ViewModel : NotificationObject
{
public ViewModel()
{
TreeItems = new ObservableCollection<Model>();
Model model1 = new Model() { Caption = "WPF" };
Model model2 = new Model() { Caption = "Silverlight" };
Model model3 = new Model() { Caption = "ASP.Net" };
Model model4 = new Model() { Caption = "ASP.Net MVC" };
Model mainmodel1 = new Model() { Caption = "User Interface" };
mainmodel1.Models.Add(model1);
mainmodel1.Models.Add(model2);
mainmodel1.Models.Add(model3);
mainmodel1.Models.Add(model4);
Model model5 = new Model() { Caption = "WPF", IsCheckable=true };
Model model6 = new Model() { Caption = "Silverlight", IsCheckable = true };
Model model7 = new Model() { Caption = "ASP.Net", IsCheckable = true };
Model model8 = new Model() { Caption = "ASP.Net MVC", IsCheckable = true };
Model mainmodel2 = new Model() { Caption = "Business Intelligence",
IsCheckable = true };
mainmodel2.Models.Add(model5);
mainmodel2.Models.Add(model6);
mainmodel2.Models.Add(model7);
mainmodel2.Models.Add(model8);
}
}

```

```

Model model9 = new Model() { Caption = "WPF" };
Model mode20 = new Model() { Caption = "Silverlight" };
Model mode21 = new Model() { Caption = "ASP.Net" };
Model mainmodel3 = new Model() { Caption = "Reporting" };
mainmodel3.Models.Add(model9);
mainmodel3.Models.Add(mode20);
mainmodel3.Models.Add(mode21);
Model mod = new Model() { Caption = "Syncfusion Essential Studio" };
mod.Models.Add(mainmodel1);
mod.Models.Add(mainmodel2);
mod.Models.Add(mainmodel3);
TreeItems.Add(mod);
}
public ObservableCollection<Model> _treeitems;
public ObservableCollection<Model> TreeItems
{
    Get
    {
        return _treeitems;
    }
    Set
    {
        _treeitems = value;
    }
}
}

```

VB.NET

```

Public Class ViewModel
    Inherits NotificationObject
    Public Sub New()
        TreeItems = New ObservableCollection(Of Model)()
        Dim model1 As New Model() With {.Caption = "WPF"}
        Dim model2 As New Model() With {.Caption = "Silverlight"}
        Dim model3 As New Model() With {.Caption = "ASP.Net"}
        Dim model4 As New Model() With {.Caption = "ASP.Net MVC"}
        Dim mainmodel1 As New Model() With {.Caption = "User Interface"}
        mainmodel1.Models.Add(model1)
        mainmodel1.Models.Add(model2)
        mainmodel1.Models.Add(model3)
        mainmodel1.Models.Add(model4)
        Dim model5 As New Model() With {
            .Caption = "WPF",
            .IsCheckable=True
        }
        Dim model6 As New Model() With {
            .Caption = "Silverlight",
            .IsCheckable = True
        }
        Dim model7 As New Model() With {
            .Caption = "ASP.Net",
            .IsCheckable = True
        }
        Dim model8 As New Model() With {
            .Caption = "ASP.Net MVC",

```

```

    .IsCheckable = True
}
Dim mainmodel2 As New Model() With {
    .Caption = "Business Intelligence",
    .IsCheckable = True
}
mainmodel2.Models.Add(model5)
mainmodel2.Models.Add(model6)
mainmodel2.Models.Add(model7)
mainmodel2.Models.Add(model8)
Dim model9 As New Model() With {.Caption = "WPF"}
Dim mode20 As New Model() With {.Caption = "Silverlight"}
Dim mode21 As New Model() With {.Caption = "ASP.Net"}
Dim mainmodel3 As New Model() With {.Caption = "Reporting"}
mainmodel3.Models.Add(model9)
mainmodel3.Models.Add(mode20)
mainmodel3.Models.Add(mode21)
Dim [mod] As New Model() With {.Caption = "Syncfusion Essential Studio"}
[mod].Models.Add(mainmodel1)
[mod].Models.Add(mainmodel2)
[mod].Models.Add(mainmodel3)
TreeItems.Add([mod])
End Sub
Public _treeitems As ObservableCollection(Of Model)
Public ReadOnly Property TreeItems() As ObservableCollection(Of Model)
[Get]
If True Then
Return _treeitems
End If
[Set]
If True Then
_treeitems = value
End If
End Property
End Class

```

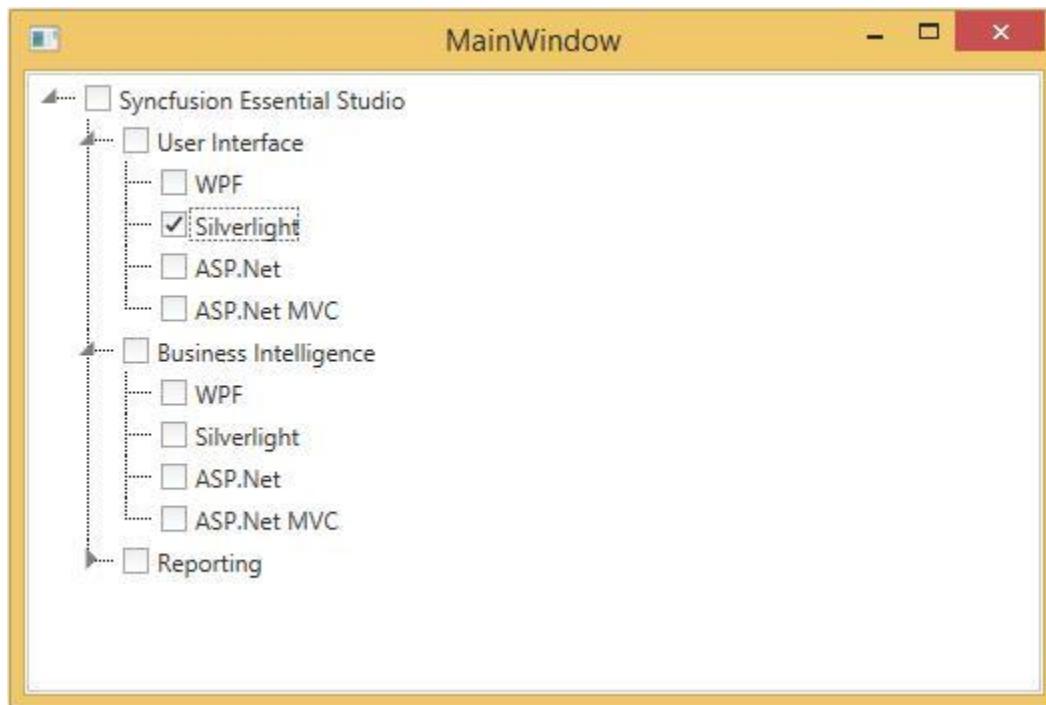
XML

```

<Window x:Class="ItemTemplateSample.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:ItemTemplateSample"
syncfusion:SkinStorage.VisualStudio="Metro"
Title="MainWindow" Height="350" Width="525">
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
<syncfusion:TreeViewAdv ItemsSource="{Binding TreeItems}" >
<syncfusion:TreeViewAdv.ItemTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding Models}">
<CheckBox Content="{Binding Caption}" IsChecked="{Binding
Path=IsChecked,Mode=TwoWay}" />
</HierarchicalDataTemplate>
</syncfusion:TreeViewAdv.ItemTemplate>

```

```
</syncfusion:TreeViewAdv>
</Grid>
</Window>
```



Item template selector

Different templates can be used for items based on specific constraints using the `ItemTemplateSelector`.

The following example illustrates this:

1. Create the template selector as shown in the following code snippet:

C#

```
public class TreeViewAdvItemTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        Window window = Application.Current.MainWindow;
        if (((Model)item).IsCheckable)
        {
            return window.Resources["CheckableTemplate"] as DataTemplate;
        }
        else
        {
            return window.Resources["NormalTemplate"] as DataTemplate;
        }
    }
}
```

VB.NET

```
Public Class TreeViewAdvItemTemplateSelector
    Inherits DataTemplateSelector
    Public Overrides Function SelectTemplate(ByVal item As Object, ByVal
        container As DependencyObject) As DataTemplate
        Dim window As Window = Application.Current.MainWindow
        If DirectCast(item, Model).IsCheckable Then
            Return TryCast(window.Resources("CheckableTemplate"), DataTemplate)
        Else
            Return TryCast(window.Resources("NormalTemplate"), DataTemplate)
        End If
    End Function
End Class
```

2. Define the Data templates in the Window's resources as follows:

XML

```
<HierarchicalDataTemplate ItemsSource="{Binding Models}"
    x:Key="CheckableTemplate">
    <CheckBox Content="{Binding Caption}" />
</HierarchicalDataTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding Models}"
    x:Key="NormalTemplate">
    <TextBlock Text="{Binding Caption}" />
</HierarchicalDataTemplate>
```

3. Create the instance for the template selector in the Window's resources as follows:

XML

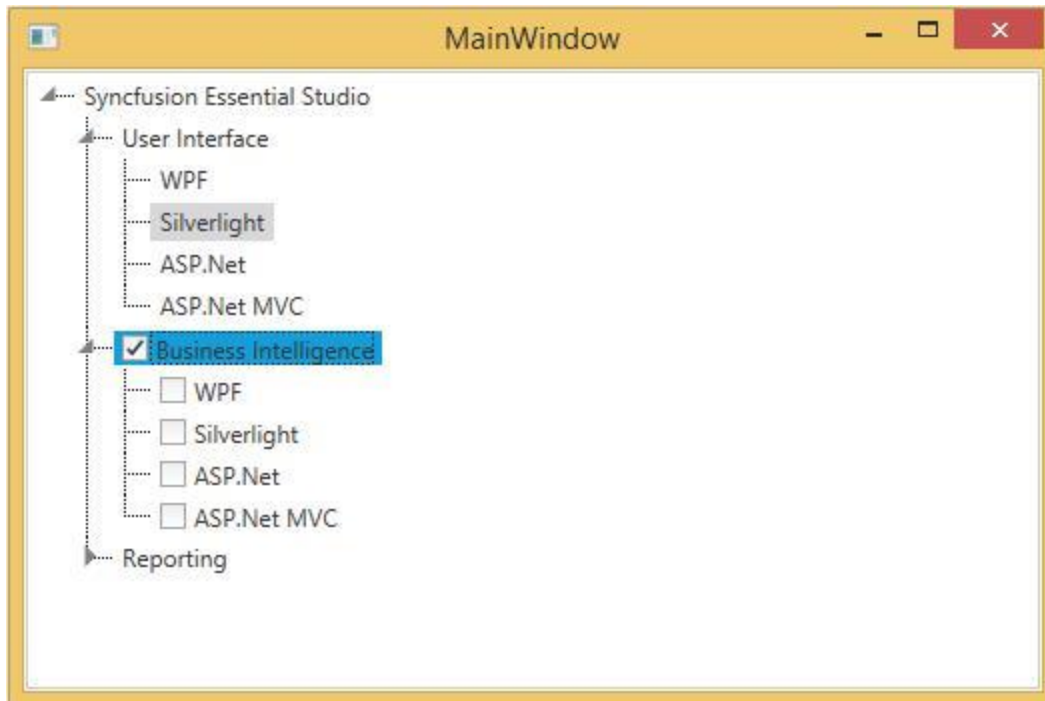
```
<local:TreeViewAdvItemTemplateSelector
    x:Key="treeViewItemTemplateSelector"/>
```

4. Use the template selector to choose the template for the TreeViewAdv as follows:

XML

```
<syncfusion:TreeViewAdv ItemsSource="{Binding TreeItems}"
    ItemTemplateSelector="{StaticResource treeViewItemTemplateSelector}" >
</syncfusion:TreeViewAdv>
```

The TreeViewAdv generates as shown in the following screenshot:



Edit template

The user can modify the template while editing the TreeViewItemAdv. The following example illustrates the process of changing the template:

1. Create the DataTemplate instance for the EditTemplate as follows:

XML

```
<DataTemplate x:Key="EditTemplate">
  <TextBox Text="{Binding Header}" FontStyle="Italic" FontWeight="Bold" />
</DataTemplate>
```

2. Set the EditedItemTemplate for the TreeViewAdv to the above template as follows:

XML

```
<syncfusion:TreeViewAdv ItemsSource="{Binding TreeItems}"
  EditedItemTemplate="{StaticResource EditTemplate}" >
  <syncfusion:TreeViewAdv.ItemTemplate>
    <HierarchicalDataTemplate ItemsSource="{Binding Models}">
      <TextBlock Text="{Binding Caption}" />
    </HierarchicalDataTemplate>
  </syncfusion:TreeViewAdv.ItemTemplate>
</syncfusion:TreeViewAdv>
```

While editing the TreeViewItemAdv appears as shown in the following screen shot:



Edit template selector

The user can choose the template at runtime for editing the TreeViewAdv.

The following example explains how to choose the template at runtime:

1. Create the template selector as given in the following code snippet:

C#

```
public class TreeViewAdvEditTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        Window window = Application.Current.MainWindow;
        if (((Model)item).IsCheckable)
        {
            return window.Resources["CheckableEditTemplate"] as DataTemplate;
        }
        Else
        {
            return window.Resources["NormalEditTemplate"] as DataTemplate;
        }
    }
}
```

VB.NET

```
Public Class TreeViewAdvEditTemplateSelector
    Inherits DataTemplateSelector
    Public Overrides Function SelectTemplate(ByVal item As Object, ByVal container As DependencyObject) As DataTemplate
        Dim window As Window = Application.Current.MainWindow
```



```
If DirectCast(item, Model).IsCheckable Then
Return TryCast(window.Resources("CheckableEditTemplate"), DataTemplate)
End If
[Else]
If True Then
Return TryCast(window.Resources("NormalEditTemplate"), DataTemplate)
End If
End Function
End Class
```

2. Define the Data templates in the Window's resources as follows:

XML

```
<DataTemplate x:Key="CheckableEditTemplate">
<TextBox Text="{Binding Header}" FontStyle="Italic" FontWeight="Bold"
Foreground="Blue"/>
</DataTemplate>
<DataTemplate x:Key="NormalEditTemplate">
<TextBox Text="{Binding Header}" FontStyle="Italic" FontWeight="Bold"
Foreground="Green"/>
</DataTemplate>
```

3. Create the instance for the template selector in the Window's resources as follows:

XML

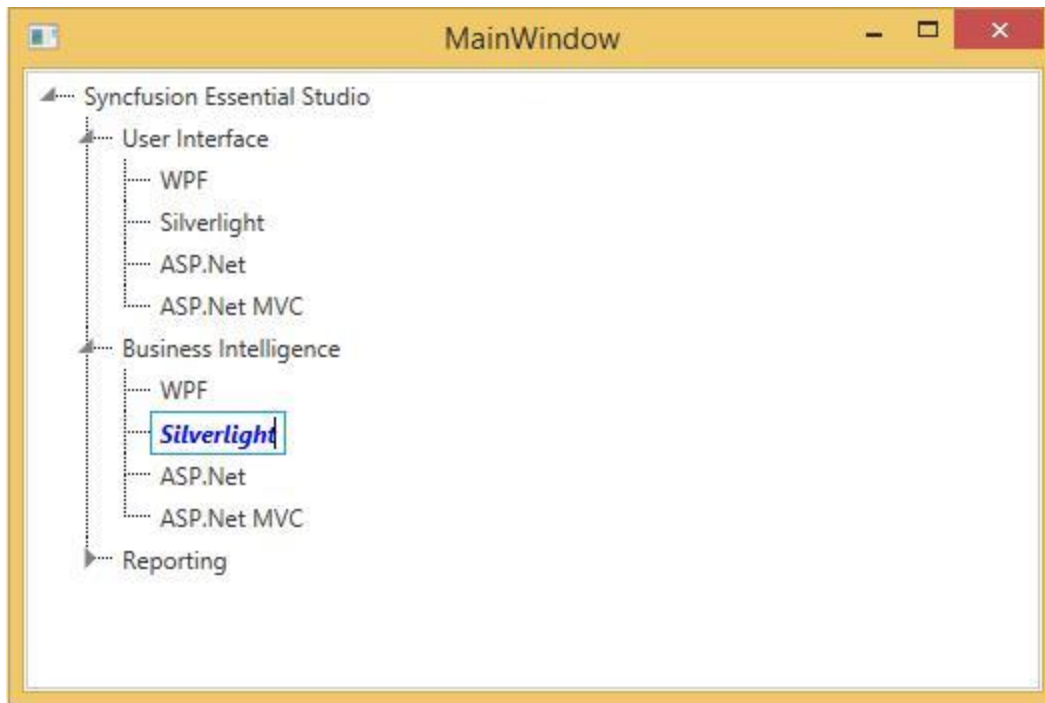
```
<local:TreeViewAdvEditTemplateSelector
x:Key="TreeViewAdvEditTemplateSelector"/>
```

4. Use the template selector to choose the template for the TreeViewAdv as follows:

XML

```
<syncfusion:TreeViewAdv ItemsSource="{Binding TreeItems}"
EditedItemTemplateSelector="{StaticResource
TreeViewAdvEditTemplateSelector}" >
<syncfusion:TreeViewAdv.ItemTemplate>
<HierarchicalDataTemplate ItemsSource="{Binding Models}">
<TextBlock Text="{Binding Caption}" />
</HierarchicalDataTemplate>
</syncfusion:TreeViewAdv.ItemTemplate>
</syncfusion:TreeViewAdv>
```

The TreeViewAdv generates as shown in the following screenshot:



Header template

User can customize the header of the treeview item by using HeaderTemplate using the below code snippet.

XML

```
<Window x:Class="ItemTemplateSample.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"
syncfusion:SkinStorage.VisualStudio="Metro"
Title="MainWindow" Height="350" Width="525">
<Window.Resources>
<DataTemplate x:Key="HeaderTemplate">
<StackPanel Orientation="Horizontal">
<TextBlock FontWeight="Bold" Text="Marital Status" />
</StackPanel>
</DataTemplate>
</Window.Resources>
<Grid>
<!-- Adding TreeViewAdv with HeaderTemplate -->
<syncfusion:TreeViewAdv Name="treeViewAdv">
<!-- Adding TreeViewItemAdv -->
<syncfusion:TreeViewItemAdv Name="treeViewItemAdv1"
HeaderTemplate="{StaticResource HeaderTemplate}">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
</Grid>
</Window>
```



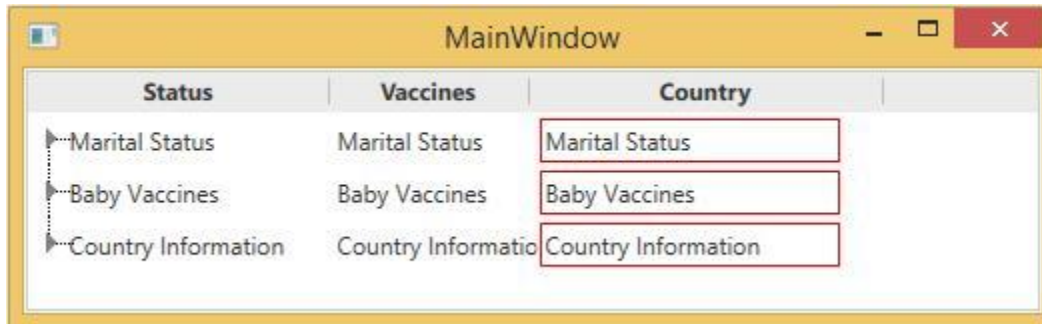
Cell template

TreeViewAdv allow user to customize the items under a column header by defining a cell template for the TreeViewColumns. To create a cell template use the below code

XML

```
<!-- Adding TreeViewAdv with Enabling multiple column -->
<syncfusion:TreeViewAdv Name="treeViewAdv" MultiColumnEnable="True">
  <!-- Adding TreeViewItemAdv -->
  <syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital Status">
    <syncfusion:TreeViewItemAdv Header="Single"/>
    <syncfusion:TreeViewItemAdv Header="Married"/>
    <syncfusion:TreeViewItemAdv Header="Married with Children"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Baby Vaccines">
    <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
    <syncfusion:TreeViewItemAdv Header="Tetanus"/>
    <syncfusion:TreeViewItemAdv Header="Polio"/>
    <syncfusion:TreeViewItemAdv Header="Measles"/>
  </syncfusion:TreeViewItemAdv>
  <syncfusion:TreeViewItemAdv Header="Country Information">
    <syncfusion:TreeViewItemAdv Header="Canada"/>
    <syncfusion:TreeViewItemAdv Header="France"/>
    <syncfusion:TreeViewItemAdv Header="Germany"/>
    <syncfusion:TreeViewItemAdv Header="UK"/>
    <syncfusion:TreeViewItemAdv Header="USA"/>
  </syncfusion:TreeViewItemAdv>
  <!-- Adding header -->
  <syncfusion:TreeViewAdv.Columns>
    <syncfusion:TreeViewColumnCollection>
      <syncfusion:TreeViewColumn Width="150" Header="Status"
        DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
          AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
      <syncfusion:TreeViewColumn Width="100" Header="Vaccines"
        DisplayMemberBinding="{Binding Path=Header, RelativeSource={RelativeSource
          AncestorType={x:Type syncfusion:TreeViewItemAdv}}}" />
      <syncfusion:TreeViewColumn Width="50" Header="Country">
        <!-- Cell Template -->
        <syncfusion:TreeViewColumn.CellTemplate>
          <DataTemplate>
            <Border Margin="1" Width="150" BorderBrush="Red" BorderThickness="1">
```

```
<TextBlock Margin="2" Text="{Binding Path=Header,
RelativeSource={RelativeSource AncestorType={x:Type
syncfusion:TreeViewItemAdv}}}" />
</Border>
</DataTemplate>
</syncfusion:TreeViewColumn.CellTemplate>
</syncfusion:TreeViewColumn>
</syncfusion:TreeViewColumnCollection>
</syncfusion:TreeViewAdv.Columns>
</syncfusion:TreeViewAdv>
```



Appearance in WPF TreeViewAdv (Classic)

This section deals with the appearance of TreeViewAdv control and contains the following topics:

Customizing the appearance of the TreeViewAdv

The TreeViewAdv appearance is customized by using the appearance properties available in the control. You can set the color for the Foreground, Background, Selected Item Foreground, Selected Item Background, MouseOver Foreground and MouseOver Background of TreeViewAdv control.

- **SelectedBackground:** Gets or sets the background color of the selected treeview item
- **SelectedForeground:** Gets or sets the foreground color of the selected treeview item
- **MouseOverForeground:** Gets or sets the foreground color of the treeview item over which the mouse pointer moves
- **MouseOverBackground:** Gets or sets the background color of the treeview item over which the mouse pointer moves
- **SelectionUnfocussedBackground:** Gets or sets the background color of the selected treeview item when the item loses focus

The following code example illustrates the above property settings.

XML

```
<!-- Adding TreeViewAdv With Brushes -->
<syncfusion:TreeViewAdv MouseOverBackground="Aqua"
MouseOverForeground="Magenta" Name="treeViewAdv" SelectedBackground="Orange"
SelectedForeground="Red" SelectionUnfocussedBackcolor="Gold">
<!-- Adding TreeViewItemAdv -->
<syncfusion:TreeViewItemAdv Header="Marital Status">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
```

```
<syncfusion:TreeViewItemAdv Header="Baby Vaccines">
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Country Information"/>
</syncfusion:TreeViewAdv>
```

C#

```
// Set MouseOverBackground property
treeViewAdv.MouseOverBackground = Brushes.Aqua;
// Set MouseOverForeground property
treeViewAdv.MouseOverForeground = Brushes.Magenta;
// Set SelectedBackground property
treeViewAdv.SelectedBackground = Brushes.Orange;
// Set SelectedForeground property
treeViewAdv.SelectedForeground = Brushes.Red;
// Set SelectionUnfocussedBackColor property
treeViewAdv.SelectionUnfocussedBackColor = Brushes.Gold;
```

VB.NET

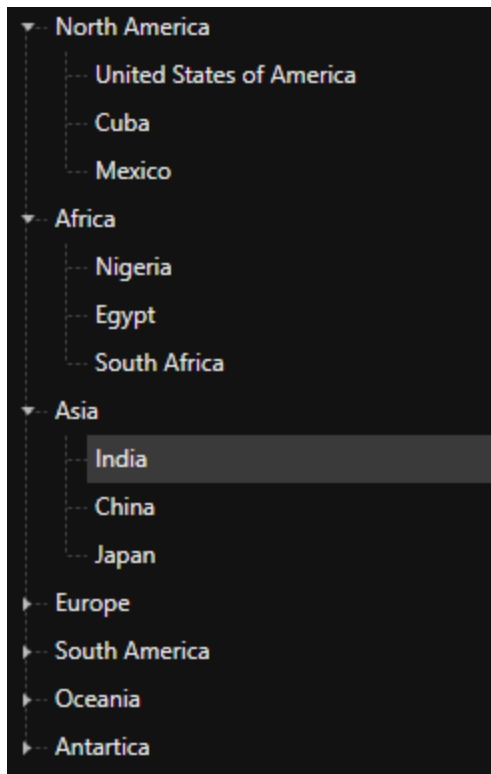
```
' Set MouseOverBackground property
treeViewAdv.MouseOverBackground = Brushes.Aqua
' Set MouseOverForeground property
treeViewAdv.MouseOverForeground = Brushes.Magenta
' Set SelectedBackground property
treeViewAdv.SelectedBackground = Brushes.Orange
' Set SelectedForeground property
treeViewAdv.SelectedForeground = Brushes.Red
' Set SelectionUnfocussedBackColor property
treeViewAdv.SelectionUnfocussedBackColor = Brushes.Gold
```



Theme

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

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



Customizing root lines

You can customize the Root lines color and pen of the TreeViewAdv.

Line color

The color of the root lines, which connect different nodes in a TreeViewAdv control is changed by using the LineBrush property. Use the following code example to set the color of the root lines.

XML

```
<!-- Adding TreeViewAdv With show root lines and line brush -->
<syncfusion:TreeViewAdv Name="treeViewAdv" LineBrush="Red"
ShowRootLines="True">
<!-- Adding TreeViewItemAdv -->
<syncfusion:TreeViewItemAdv Header="Marital Status">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Baby Vaccines">
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Country Information"/>
</syncfusion:TreeViewAdv>
```

C#

```
// Show root lines
```

```
treeViewAdv.ShowRootLines = true;  
// Set line brush  
treeViewAdv.LineBrush = Brushes.Red;
```

VB.NET

```
' Show root lines  
treeViewAdv.ShowRootLines = True  
' Set line brush  
treeViewAdv.LineBrush = Brushes.Red
```



Line pen

The root lines which connect different nodes in a TreeViewAdv control are customized by using the LinePen property. This property specifies the pen color for a node line. To set the LinePen property, refer the below code

XML

```
<!-- Adding TreeViewAdv With show root lines and line pen -->  
<syncfusion:TreeViewAdv Name="treeViewAdv" ShowRootLines="True">  
  <syncfusion:TreeViewAdv.LinePen>  
    <Pen Brush="Red" Thickness="1"/>  
  </syncfusion:TreeViewAdv.LinePen>  
<!-- Adding TreeViewItemAdv -->  
<syncfusion:TreeViewItemAdv Header="Marital Status">  
  <syncfusion:TreeViewItemAdv Header="Single"/>  
  <syncfusion:TreeViewItemAdv Header="Married"/>  
  <syncfusion:TreeViewItemAdv Header="Married with Children"/>  
</syncfusion:TreeViewItemAdv>  
<syncfusion:TreeViewItemAdv Header="Baby Vaccines">  
  <syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
```



```
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Country Information"/>
</syncfusion:TreeViewAdv>
```

C#

```
// Show root lines
treeViewAdv.ShowRootLines = true;
// Set line Pen
treeViewAdv.LinePen = new Pen(Brushes.Red, 1);
```

VB.NET

```
' Show root lines
treeViewAdv.ShowRootLines = True
' Set line Pen
treeViewAdv.LinePen = New Pen(Brushes.Red, 1)
```

**Styles in WPF TreeViewAdv (Classic)**

This section deals with the following Styles supported by TreeViewAdv control

Setting drag indicator style

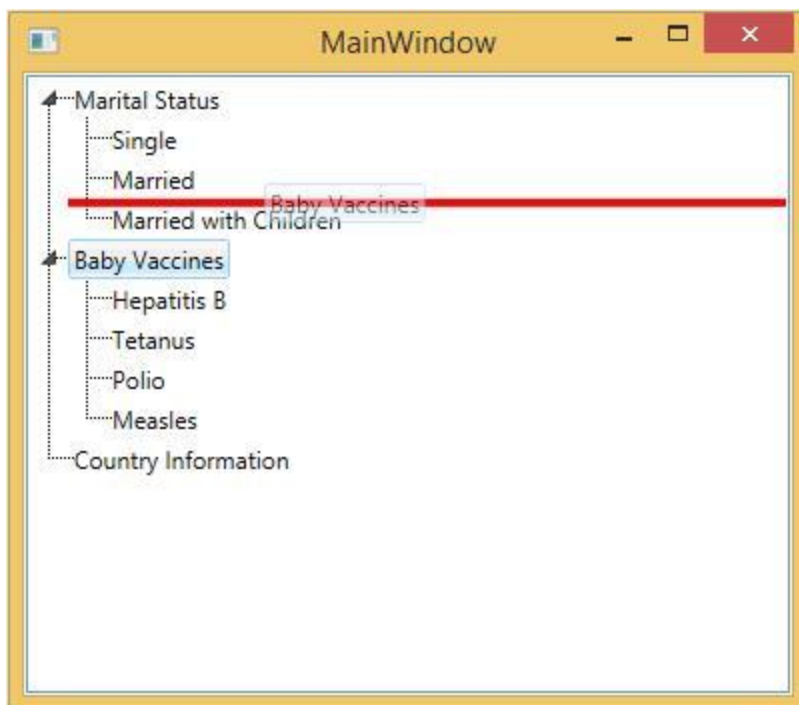
TreeViewAdv allow user to customize the style of the Drag Indicator which is used to indicate the drag-and-drop operation in progress by using the DragIndicatorStyle property. The following code example illustrates how to set this property.

XML

```

<Window x:Class="ItemTemplateSample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="400">
<Window.Resources>
<!-- Creating the style for DragIndicator -->
<Style x:Key="Drag_Marker" TargetType="{x:Type
syncfusion:TemplatedAdornerInternalControl}">
<Setter Property="HorizontalAlignment" Value="Left"/>
<Setter Property="VerticalAlignment" Value="Top"/>
<Setter Property="SnapsToDevicePixels" Value="False"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type
syncfusion:TemplatedAdornerInternalControl}">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Rectangle Grid.Column="0" Height="4" Fill="Red"/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
<Grid>
<!-- Adding TreeViewAdv with DragIndicatorStyle -->
<syncfusion:TreeViewAdv DragIndicatorStyle="{StaticResource Drag_Marker}"
Name="treeViewAdv">
<!-- Adding TreeViewItemAdv -->
<syncfusion:TreeViewItemAdv Header="Marital Status">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Baby Vaccines">
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Country Information"/>
</syncfusion:TreeViewAdv>
</Grid>
</Window>

```



Setting expand style

The `ExpanderStyle` property enables to customize the appearance and style of expansion of the `TreeViewAdv` during the Expand or Collapse operation. The following code example illustrates how to set this property.

XML

```
<Window x:Class="ItemTemplateSample.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:ItemTemplateSample"
Title="MainWindow" Height="350" Width="400">
<Window.Resources>
<!-- Template for TreeViewAdvExpander -->
<ControlTemplate x:Key="MyExpanderTemplateKey" TargetType="{x:Type
Expander}">
<ToggleButton Name="Expander" ClickMode="Press" IsChecked="{Binding
Path=IsExpanded, RelativeSource={RelativeSource TemplatedParent}}">
<ToggleButton.Style>
<Style TargetType="ToggleButton">
<Setter Property="FrameworkElement.Focusable" Value="False"/>
<Setter Property="FrameworkElement.Width" Value="19"/>
<Setter Property="FrameworkElement.Height" Value="13"/>
<Setter Property="Control.Template">
<Setter.Value>
<ControlTemplate TargetType="ToggleButton">
<Border Height="10" Width="10" BorderBrush="Black" BorderThickness="1">
<Border Name="BackgroundBorder" Background="Blue"/>
</Border>
<ControlTemplate.Triggers>
Trigger Property="ToggleButton.IsChecked" Value="True">
<Setter Property="Background" TargetName="BackgroundBorder" Value="Red"/>

```

```
</Trigger>
</ControlTemplate.Triggers>
/ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ToggleButton.Style>
</ToggleButton>
</ControlTemplate>
<!-- Style for TreeViewAdvExpander -->
<Style x:Key="MyexpanderStyle" TargetType="{x:Type Expander}">
<Setter Property="Template" Value="{StaticResource MyExpanderTemplateKey}"/>
</Style>
</Window.Resources>
<Grid>
<!-- Adding TreeViewAdv with expand animation -->
<syncfusion:TreeViewAdv Name="treeViewAdv" ExpanderStyle="{DynamicResource
MyexpanderStyle}">
<!-- Adding TreeViewItemAdv -->
<syncfusion:TreeViewItemAdv Name="treeViewItemAdv" Header="Marital Status">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Baby Vaccines">
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Header="Country Information"/>
</syncfusion:TreeViewAdv>
</Grid>
</Window>
```



Setting item container style

To set the style for the item container, use `ItemContainerStyle` property. This dependency property can be applied to `TreeViewAdv`, as well as `TreeViewItemAdv`.

The following example can be used to set this property.

XML

```
<Window.Resources>
<Style x:Key="TreeViewItemStyle" TargetType="{x:Type
syncfusion:TreeViewItemAdv}">
<Setter Property="LeftImageSource" Value="folder.png" />
<Setter Property="ImageHeight" Value="16" />
<Setter Property="ImageWidth" Value="16" />
</Style>
</Window.Resources>
<Grid >
<!-- Adding TreeViewAdv with selected -->
<syncfusion:TreeViewAdv Name="treeViewAdv"
ItemContainerStyle="{StaticResource TreeViewItemStyle}">
<!-- Adding TreeViewItemAdv -->
<syncfusion:TreeViewItemAdv Name="treeViewItemAdv1" IsSelected="True"
Header="Marital Status">
<syncfusion:TreeViewItemAdv Header="Single"/>
<syncfusion:TreeViewItemAdv Header="Married"/>
<syncfusion:TreeViewItemAdv Header="Married with Children"/>
</syncfusion:TreeViewItemAdv>
<syncfusion:TreeViewItemAdv Name="treeViewItemAdv2" Header="Baby Vaccines">
<syncfusion:TreeViewItemAdv Header="Hepatitis B"/>
<syncfusion:TreeViewItemAdv Header="Tetanus"/>
<syncfusion:TreeViewItemAdv Header="Polio"/>
<syncfusion:TreeViewItemAdv Header="Measles"/>
</syncfusion:TreeViewItemAdv>
```

```
<syncfusion:TreeViewItemAdv Header="Country Information">
<syncfusion:TreeViewItemAdv Header="Canada"/>
<syncfusion:TreeViewItemAdv Header="France"/>
<syncfusion:TreeViewItemAdv Header="Germany"/>
<syncfusion:TreeViewItemAdv Header="UK"/>
<syncfusion:TreeViewItemAdv Header="USA"/>
</syncfusion:TreeViewItemAdv>
</syncfusion:TreeViewAdv>
</Grid>
</Window>
```



Schedule (Classic)

WPF Schedule (Classic) Overview

Introduction

SfSchedule is used to provide Outlook-like scheduling. The SfSchedule control allows user to create and manage appointments. It includes features such as viewing daily/all day/spanned appointments, complete customization using custom Templates, efficient performance, Data binding to different sources.

Use Cases

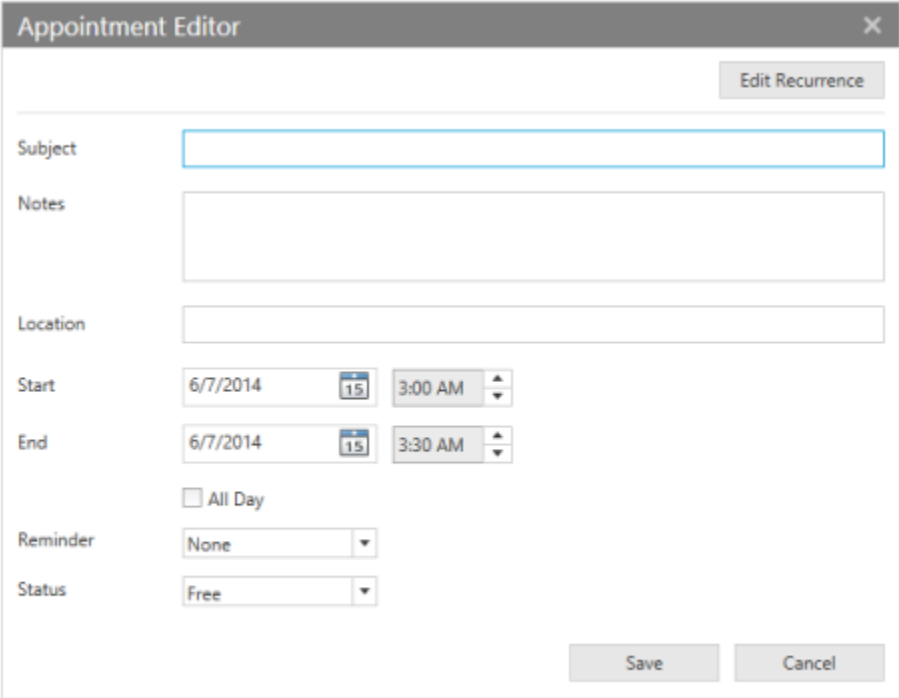
1. Schedule control can be used for public-service employees such as police officers, firefighters and health-care workers — anyone who works an unusual but predictable schedule.
2. In families if a spouse/partner works a shift schedule, then we have entirely different rotation appear on the calendar with family members. This will help us to organize family activities such as vacations and day-care.
3. Schedule control is used in a company for scheduling meetings with a clients.

4. It allows to view all the banking transaction events of a company in a daily, weekly, or monthly overview.
5. We can get a quick look at the call meetings happened in BPO of a single day in any calendar view.
6. Using this we can create single or recurring events and tasks to the employees, and set reminders for upcoming events, and color-code different item types.
7. A Drag and Drop feature allows us to reschedule an event or change its duration easily.
8. When we try to schedule an appointment with a doctor it helps to find the available time of the person and requesting for an appointment based on the availability of the concern doctor. In this way it saves the time.
9. Schedule control can be used in a scenario of categorizing the user appointments based on its importance.

Key Features

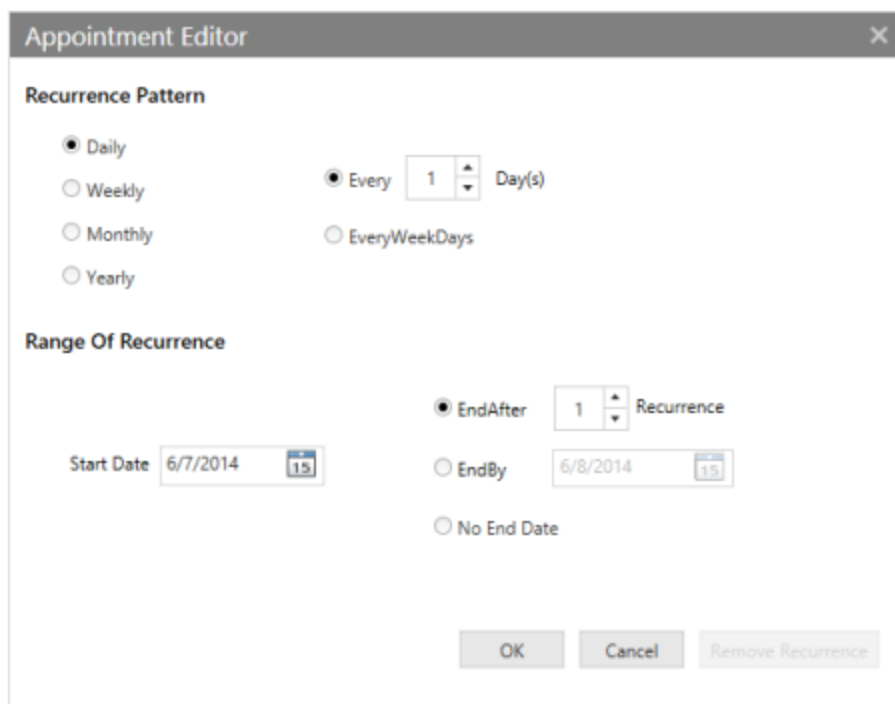
Appointment Editor

The Appointments Editor to be used to create or edit various types of appointments with a specific time or location or using "All Day Events" without a specific time or location on your schedule. Appointment Editor can customize several feature of appointment such as applying reminder and adding recurrence to an appointment. It's also simple to reschedule the added appointments by editing the Start Time and End Time of appointments.



The screenshot shows the 'Appointment Editor' dialog box with the following fields and controls:

- Subject:** A text input field.
- Notes:** A larger text area for additional information.
- Location:** A text input field.
- Start:** A date and time selector showing '6/7/2014' and '3:00 AM'.
- End:** A date and time selector showing '6/7/2014' and '3:30 AM'.
- All Day:** A checkbox labeled 'All Day'.
- Reminder:** A dropdown menu currently set to 'None'.
- Status:** A dropdown menu currently set to 'Free'.
- Buttons:** 'Edit Recurrence' (top right), 'Save' (bottom right), and 'Cancel' (bottom right).

The image shows a dialog box titled "Appointment Editor" with a close button (X) in the top right corner. It contains two main sections: "Recurrence Pattern" and "Range Of Recurrence". In the "Recurrence Pattern" section, there are five radio buttons: "Daily" (selected), "Weekly", "Monthly", "Yearly", and "EveryWeekDays". Next to the "Every" radio button is a spinner box containing the number "1" and the text "Day(s)". In the "Range Of Recurrence" section, there are three radio buttons: "EndAfter" (selected), "EndBy", and "No End Date". Next to the "EndAfter" radio button is a spinner box containing the number "1" and the text "Recurrence". Below these, there are two date pickers: "Start Date" with the value "6/7/2014" and "EndBy" with the value "6/8/2014". At the bottom of the dialog are three buttons: "OK", "Cancel", and "Remove Recurrence".

Built-in Views

Schedule provides 4 different types of viewing the calendar,

- Day
- Week
- Month
- TimeLine

Multi-Resource Support

The Schedule control allows you to define resources that can be assigned to appointments. Resources let you associate additional information with your appointments. The schedule can group appointments based on the resources associated with them.

Recurrence Appointment

You can schedule recurring appointments to repeat daily, weekly, monthly, or yearly. You can customize recurring appointment schedules that repeat for daily, weekly, monthly, or yearly.

Reminders

You can use reminder to organize your appointments in your schedule. Schedule reminds you the particular appointment in the specified time. The remainder time can be set using the `ReminderTime` property of `ScheduleAppointment`.

Rich User Experience

The Schedule control allows you to perform various operations. You can easily drag and drop the appointments from one timeslot to another timeslot. Appointment resizing operation can also be performed as per required start and end time of schedule in an interactive manner and also appointment can be modified through Appointment Editor.

Appointment Mapping

The AppointmentMapping attributes are used to map the properties in the underlying data source to the Schedule appointments. The various attributes of the AppointmentMapping property are as follows.

- Subject
- Location
- StartTime
- EndTime

Getting Started with WPF Schedule (Classic)

This section gives you an overview of how to work with [SfSchedule](#) and also includes a walk-through to configure real-time [SfSchedule](#) command.

Assembly deployment

Refer to the section on [control dependencies](#) for a list of assemblies or NuGet Packages to be used as a guide for using control in any application. Further information on installing the NuGet package can be found in the following link in a WPF application: [How to install nuget packages](#). You can also use [Syncfusion Reference Manager](#) to refer the scheduler's dependent assemblies.

Create a project

In Visual Studio, create a new WPF project to show the features of the scheduler control and add the following namespace to the added assemblies.

Assembly: [Syncfusion.SfSchedule.WPF](#)

Namespace: [Syncfusion.UI.Xaml.Schedule](#)

Add control manually in XAML

To add the control manually in XAML page, follow the given steps:

1. Add the [Syncfusion.SfSchedule.WPF](#) assembly reference to the project.
2. Import WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the [SfSchedule](#) control in XAML page.

XML

```
<Window x:Class="SfScheduleSample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:schedule="http://schemas.syncfusion.com/wpf"
WindowStartupLocation="CenterScreen" WindowStyle="None"
Width="950" Height="600">
<Grid x:Name="LayoutRoot">
<schedule:SfSchedule/>
</Grid>
</Window>
```

Add control manually in C#

To manually attach control to C #, follow the following steps:

1. Add the [Syncfusion.SfSchedule.WPF](#) assembly reference to the project.

2. Import the `SfSchedule` namespace using `Syncfusion.UI.Xaml.Schedule`.
3. Create an `SfSchedule` instance, and add it to the window.

C#

```
using Syncfusion.UI.Xaml.Schedule;
namespace GettingStarted
{
    public partial class MainWindow : Window
    {
        SfSchedule schedule = new SfSchedule();
        this.Content = schedule;
    }
}
```

Run the code above and now you can see the empty scheduler as follows, but without appointments. To view the scheduler appointments, whether local or remote information must be passed on to the scheduler.



Scheduler Views-(Day, Week, WorkWeek, TimeLine and Month)

Scheduler control provides five different views for showing appointments which can be changed by setting [ScheduleType](#) property. By default, the control loads `Day` view.

- Day
- Week
- WorkWeek
- Month
- TimeLine

Day View

Day view is used to view a single day, and by default the current day is shown. Appointments on a specific day will be scheduled on the basis of their duration in the respective time slots.

XML

```
<schedule:SfSchedule ScheduleType="Day"/>
```

C#

```
schedule.ScheduleType = ScheduleType.Day;
```



Week View

Week view is to view all week days of a particular week. Appointments will be scheduled in the corresponding timeslots on the basis of the week dates.

XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleType="Week"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.Week;
```

◀ ▶ June 2014							
	Sunday 15	Monday 16	Tuesday 17	Wednesday 18	Thursday 19	Friday 20	Saturday 21
All Day							
12:00 AM							
30							
01:00 AM							
30							
02:00 AM							
30							
03:00 AM							
30							
04:00 AM							
30							

Work Week View

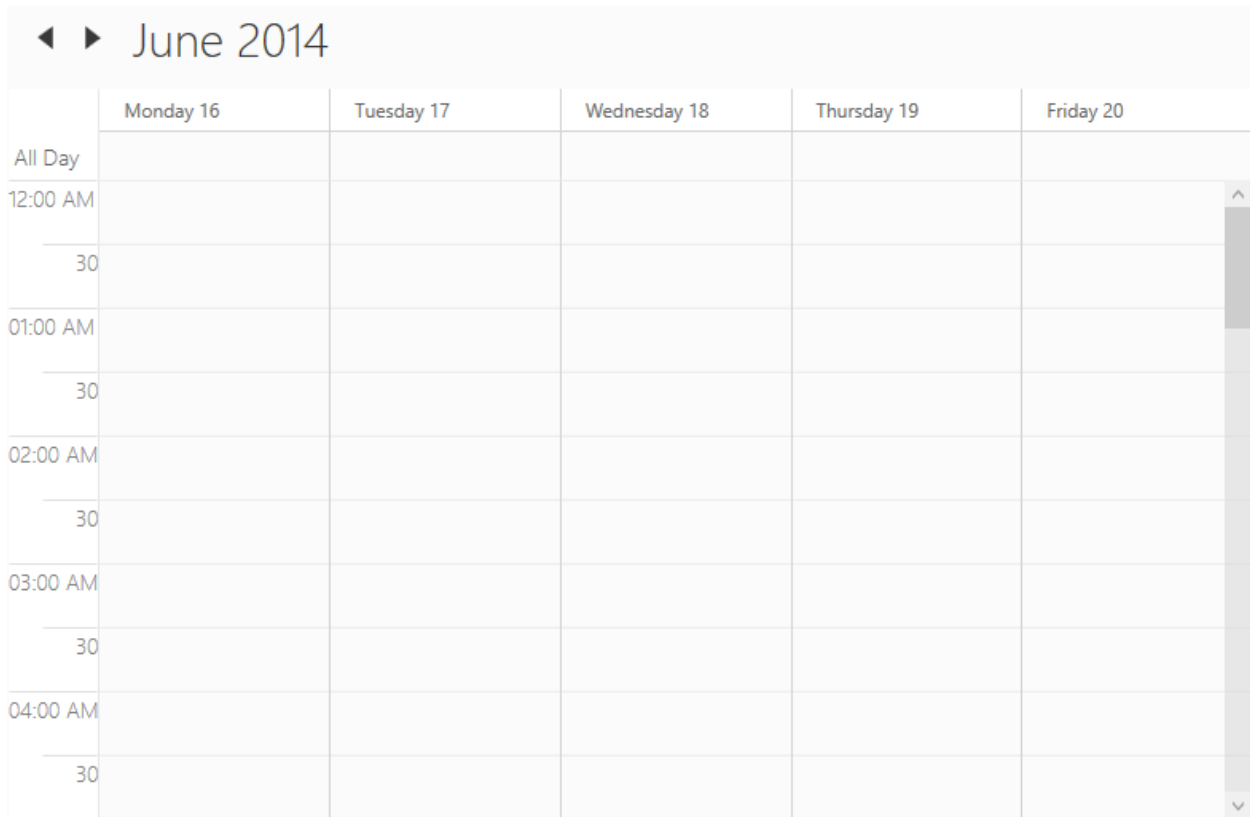
WorkWeek view is to view only working days of a particular week. By default, **Saturday** and **Sunday** are the non-working days. and you can change non-working days of a week using `non-working days` property. Appointments displayed in timeslots with the corresponding day of the week depending on their duration.

XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleType="WorkWeek"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.WorkWeek;
```



Month View

In schedule, **Month** view displays the month of dates similar to calendar and displays appointments for each day in a cell similar to outlook.

XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleType="Month"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.Month;
```

◀ ▶ June 2014						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	01	02	03	04	05

TimeLine View

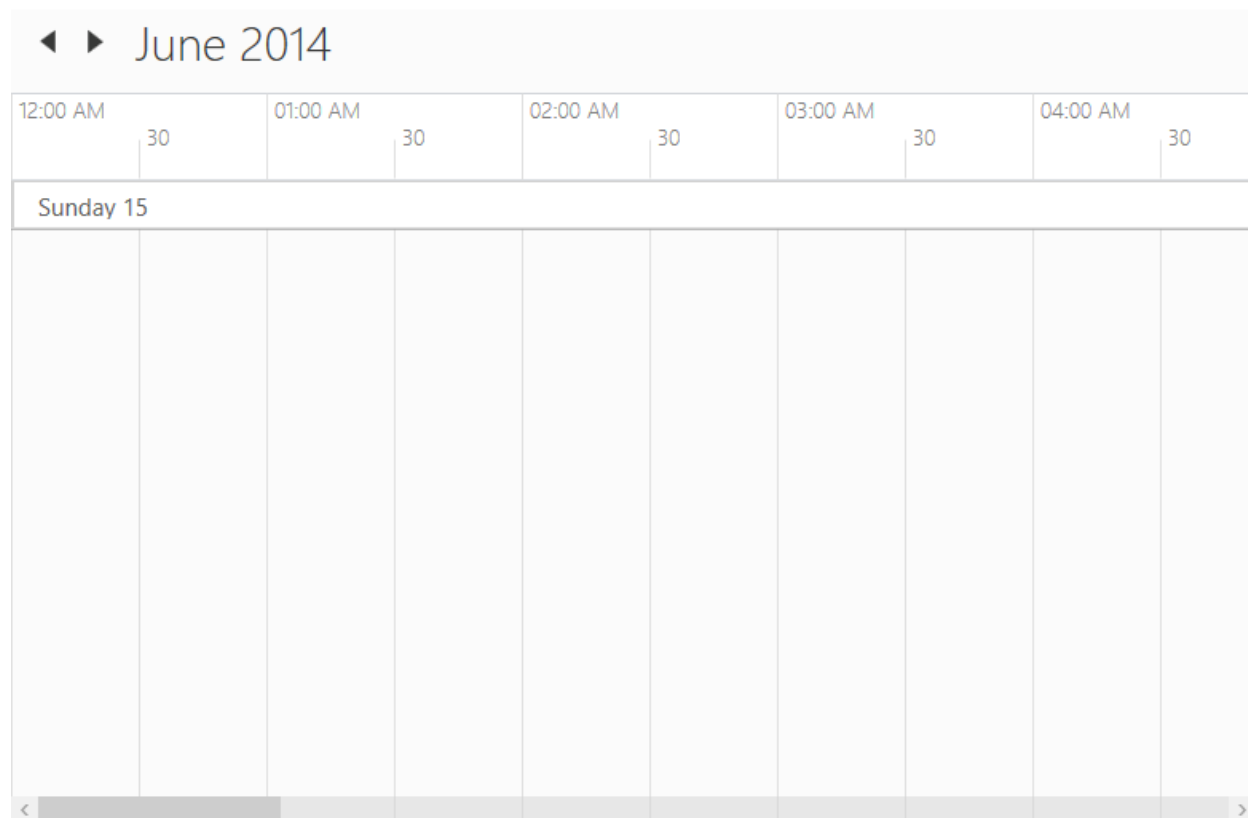
Timeline view displays the dates with the appropriate day count in the horizontal time axis. When moving right or left, you can see the past or future events. With an intuitive drag-and-drop feature, each view shows events accurately through time slots.

XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleType="TimeLine"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.TimeLine;
```



Appointments

Scheduler has a built-in capability to handle the appointment arrangement internally based on the [ScheduleAppointment](#) collections. You need to allocate the collection generated to [Appointments](#) property.

Adding Appointments

ScheduleAppointment is a class that includes the specific of scheduled appointment. It has some basic properties such as [StartTime](#), [EndTime](#), [Subject](#) and some additional information about the appointment can be added with [Notes](#), [Location](#), [AllDay](#), [IsRecursive](#) properties.

XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleType="Month"/>
```

C#

```
ScheduleAppointmentCollection appointmentCollection = new
ScheduleAppointmentCollection();
//Creating new event
ScheduleAppointment clientMeeting = new ScheduleAppointment();
DateTime currentDate = DateTime.Now;
DateTime startTime = new DateTime
(currentDate.Year, currentDate.Month, currentDate.Day, 10, 0, 0);
DateTime endTime = new DateTime (currentDate.Year,
currentDate.Month, currentDate.Day, 12, 0, 0);
clientMeeting.StartTime = startTime;
clientMeeting.EndTime = endTime;
clientMeeting.Color = Color.Blue;
```

```
clientMeeting.Subject = "ClientMeeting";  
appointmentCollection.Add(clientMeeting);  
schedule.Appointments = appointmentCollection;
```

Download demo from [GitHub](#)

Adding Custom data object

You can also map custom appointments data to our scheduler.

Here steps to render meetings using scheduler control with respective custom data properties created in a class `Meeting`.

- Creating custom class to map that objects with `ScheduleAppointment`
- Populating the data objects
- Mapping the data object to `ScheduleAppointment`
- Binding data source for Scheduler control.

Creating custom class to map that object with appointment

You can create a custom class `Meeting` with mandatory fields `From`, `To` and `EventName`.

C#

```
/// <summary>  
/// Represents custom data properties.  
/// </summary>  
public class Meeting  
{  
    public string EventName { get; set; }  
    public string Organizer { get; set; }  
    public string ContactID { get; set; }  
    public int Capacity { get; set; }  
    public DateTime From { get; set; }  
    public DateTime To { get; set; }  
    public Color color { get; set; }  
    public bool AllDay { get; set; }  
}
```

Note: You can inherit this class from `INotifyPropertyChanged` for dynamic changes in custom data.

Populating the data objects

By setting `From` and `To` of `Meeting` class, you can schedule meetings for a specific day. You can also change the subject and color of appointment using `EventName` and `Color` property. You may define the list of custom appointments in a separate class of `ViewModel`.

C#

```
/// <summary>  
/// Represents collection of appointments.  
/// </summary>  
public class ViewModel  
{  
    public ObservableCollection<Meeting> Meetings { get; set; }  
    List<string> eventNameCollection;  
    List<Brush> colorCollection;
```



```
public ViewModel()
{
    Meetings = new ObservableCollection<Meeting>();
    CreateEventNameCollection();
    CreateColorCollection();
    CreateAppointments();
}
/// <summary>
/// Creates meetings and stores in a collection.
/// </summary>
private void CreateAppointments()
{
    Random randomTime = new Random();
    List<Point> randomTimeCollection = GettingTimeRanges();
    DateTime date;
    DateTime DateFrom = DateTime.Now.AddDays(-10);
    DateTime DateTo = DateTime.Now.AddDays(10);
    DateTime dataRangeStart = DateTime.Now.AddDays(-3);
    DateTime dataRangeEnd = DateTime.Now.AddDays(3);
    for (date = DateFrom; date < DateTo; date = date.AddDays(1))
    {
        if ((DateTime.Compare(date, dataRangeStart) > 0) && (DateTime.Compare(date, dataRangeEnd) < 0))
        {
            for (int AdditionalAppointmentIndex = 0; AdditionalAppointmentIndex < 3; AdditionalAppointmentIndex++)
            {
                Meeting meeting = new Meeting();
                int hour =
                    (randomTime.Next((int)randomTimeCollection[AdditionalAppointmentIndex].X,
                    (int)randomTimeCollection[AdditionalAppointmentIndex].Y));
                meeting.From = new DateTime(date.Year, date.Month, date.Day, hour, 0, 0);
                meeting.To = (meeting.From.AddHours(1));
                meeting.EventName = eventNameCollection[randomTime.Next(9)];
                meeting.Color = colorCollection[randomTime.Next(9)];
                if (AdditionalAppointmentIndex % 3 == 0)
                meeting.AllDay = true;
                Meetings.Add(meeting);
            }
        }
        else
        {
            Meeting meeting = new Meeting();
            meeting.From = new DateTime(date.Year, date.Month, date.Day,
            randomTime.Next(9, 11), 0, 0);
            meeting.To = (meeting.From.AddHours(1));
            meeting.EventName = eventNameCollection[randomTime.Next(9)];
            meeting.Color = colorCollection[randomTime.Next(9)];
            Meetings.Add(meeting);
        }
    }
}
/// <summary>
/// Creates event names collection.
/// </summary>
private void CreateEventNameCollection()
{
}
```

```

eventNameCollection = new List<string>();
eventNameCollection.Add("General Meeting");
eventNameCollection.Add("Plan Execution");
eventNameCollection.Add("Project Plan");
eventNameCollection.Add("Consulting");
eventNameCollection.Add("Performance Check");
eventNameCollection.Add("Yoga Therapy");
eventNameCollection.Add("Plan Execution");
eventNameCollection.Add("Project Plan");
eventNameCollection.Add("Consulting");
eventNameCollection.Add("Performance Check");
}
/// <summary>
/// Creates color collection.
/// </summary>
private void CreateColorCollection()
{
    colorCollection = new List<Brush>();
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF339933")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF00ABA9")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFE671B8")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF1BA1E2")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFD80073")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFA2C139")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFA2C139")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFD80073")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF339933")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFE671B8")));
    colorCollection.Add(new
SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF00ABA9")));
}
/// <summary>
/// Gets the time ranges.
/// </summary>
private List<Point> GettingTimeRanges()
{
    List<Point> randomTimeCollection = new List<Point>();
    randomTimeCollection.Add(new Point(9, 11));
    randomTimeCollection.Add(new Point(12, 14));
    randomTimeCollection.Add(new Point(15, 17));
    return randomTimeCollection;
}
}

```

Mapping the data object to schedule appointment

You can map those properties of `Meeting` class with our scheduler control by using [ScheduleAppointmentMapping](#) property.

XML

```
<syncfusion:SfSchedule x:Name="schedule">
  <syncfusion:SfSchedule.AppointmentMapping>
    <syncfusion:ScheduleAppointmentMapping
      AppointmentBackgroundMapping="color"
      EndTimeMapping="To"
      StartTimeMapping="From"
      SubjectMapping="EventName"
      AllDayMapping="AllDay" />
  </syncfusion:SfSchedule.AppointmentMapping>
</syncfusion:SfSchedule>
```

C#

```
ScheduleAppointmentMapping dataMapping = new ScheduleAppointmentMapping();
dataMapping.AppointmentBackgroundMapping = "color";
dataMapping.EndTimeMapping = "To";
dataMapping.StartTimeMapping = "From";
dataMapping.SubjectMapping = "EventName";
dataMapping.AllDayMapping = "AllDay";
schedule.AppointmentMapping = dataMapping;
```

Binding data source for Scheduler

Create meetings of type `ObservableCollection<Meeting>` and assign those appointments collection `Meetings` to the `ItemsSource` property of Scheduler.

XML

```
<syncfusion:SfSchedule x:Name="schedule"
  ItemsSource = "{Binding Meetings}"
  ScheduleType = "Month" >
  <syncfusion:SfSchedule.DataContext>
    <local:ViewModel/>
  </syncfusion:SfSchedule.DataContext>
</syncfusion:SfSchedule>
```

C#

```
ViewModel viewModel = new ViewModel();
schedule.ItemsSource = viewModel.Meetings;
```

Download demo from [GitHub](#).

Views in WPF Schedule (Classic)

Scheduler provides the following different types of views.

- Day view
- Week view

- WorkWeek view
- Timeline View - Refer [scheduler timeline view](#) documentation for more customization details.
- Month View - Refer [scheduler month view](#) documentation for more customization details.

This topic covers customization of day, week and workweek views of the scheduler which shares common properties.

Note: This topic explains all customization using day view. But the same applies for week and workweek views also. if you want to make customization specific to views, then please refer [link](#)

Header date format

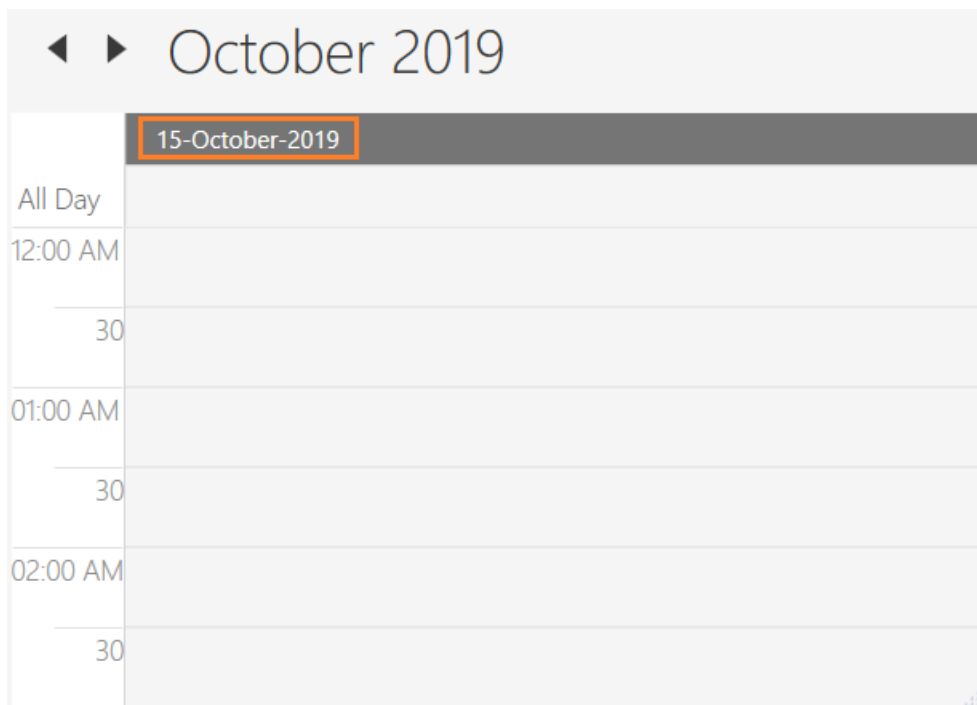
Scheduler supports to customize the default header date format of the day, week, workweek and timeline view by using the [HeaderDateFormat](#) property.

XML

```
<Schedule:SfSchedule HeaderDateFormat="dd-MMMM-yyyy"/>
```

C#

```
this.schedule.HeaderDateFormat = "dd-MMMM-yyyy";
```



Time formatting

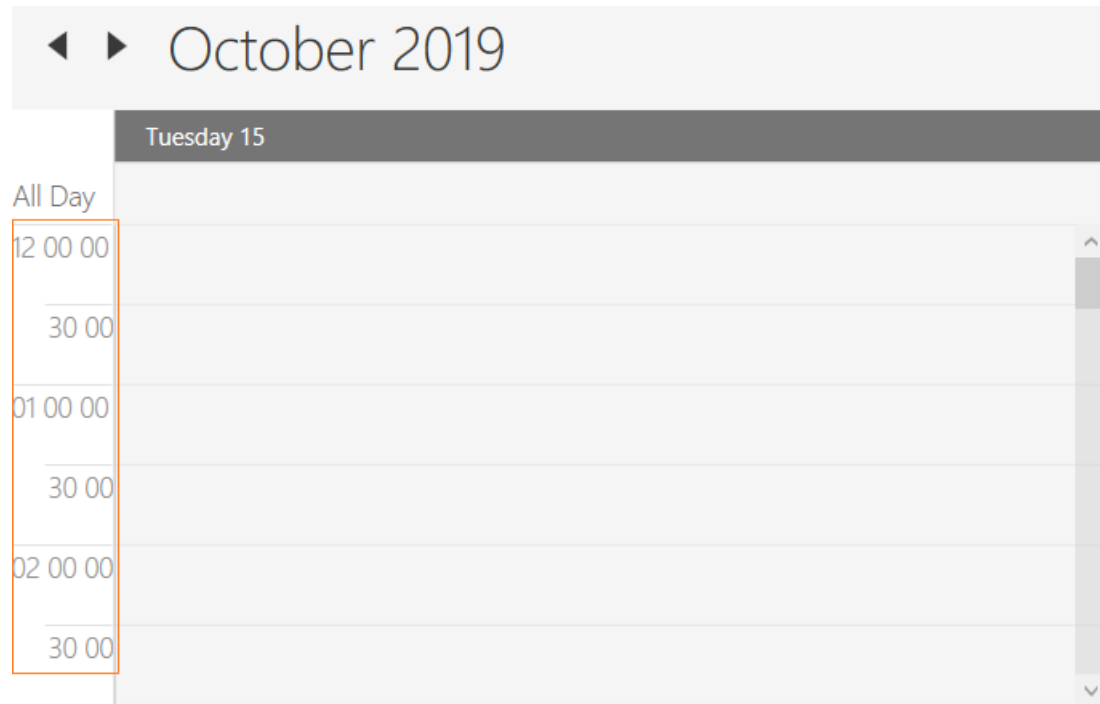
Scheduler supports to customize time format for day, week, workweek and timeline views by using [MajorTickTimeFormat](#) and [MinorTickTimeFormat](#) property.

XML

```
<syncfusion:SfSchedule MajorTickTimeFormat="hh mm ss"
MinorTickTimeFormat="mm ss"/>
```

C#

```
this.schedule.MajorTickTimeFormat = "hh mm ss";
this.schedule.MinorTickTimeFormat = "hh mm ss";
```

**Enable auto formatting**

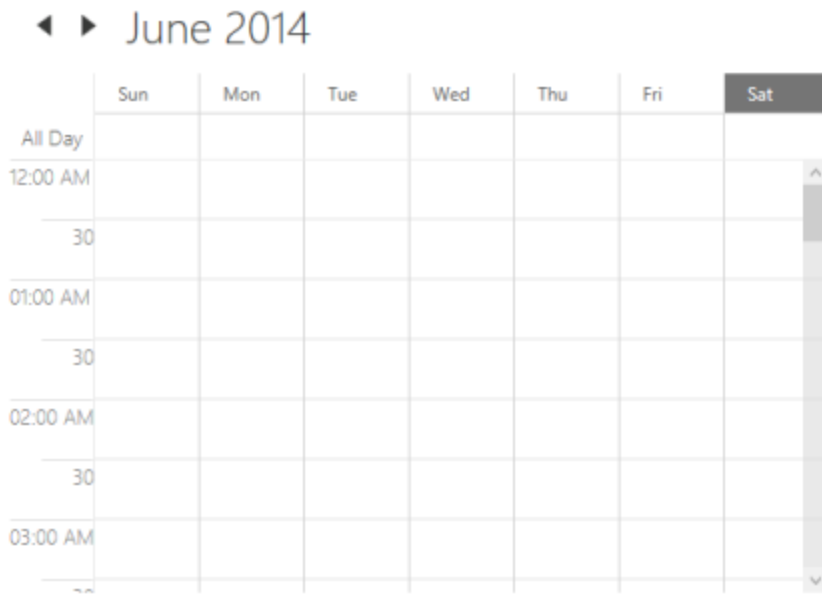
When reducing the size of the schedule in week and month views, headers may be only partially shown. To avoid incompletely displayed headers, automatic formatting can be enabled by setting the [EnableAutoFormat](#) property of the Scheduler control as `true`. If this property enabled, [HeaderDateFormat](#) settings will not apply.

XML

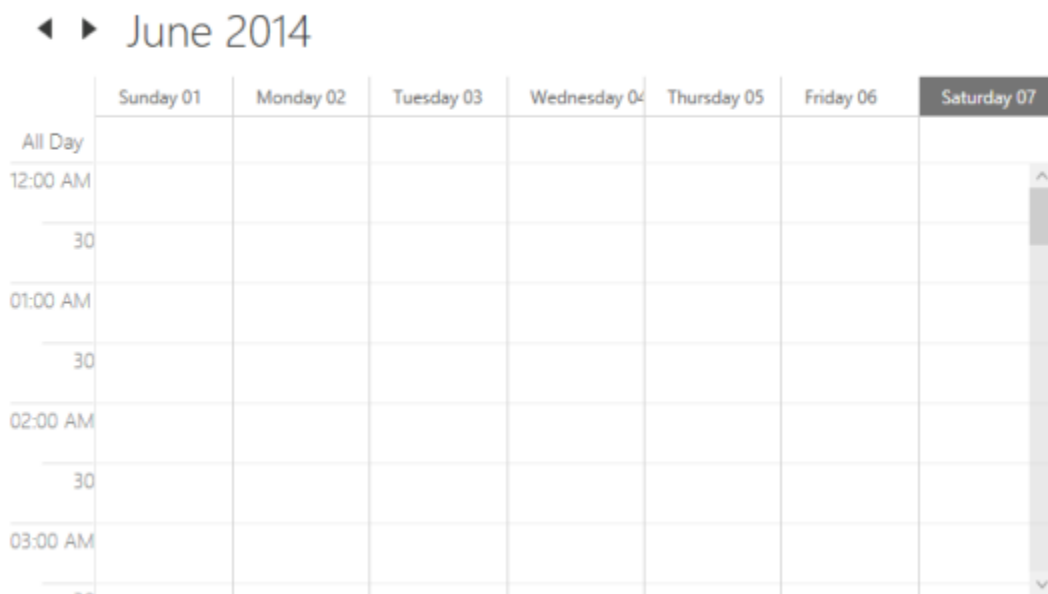
```
<syncfusion:SfSchedule x:Name="schedule" Background="White"
Height="400" Width="500"
EnableAutoFormat="True"
ScheduleType="Week">
</syncfusion:SfSchedule>
```

C#

```
schedule.Background = new SolidColorBrush(Colors.White);
schedule.Height = 400;
schedule.Width = 500;
schedule.EnableAutoFormat = true;
schedule.ScheduleType = ScheduleType.Week;
```



After changing the window size by resizing the window.



Change time interval

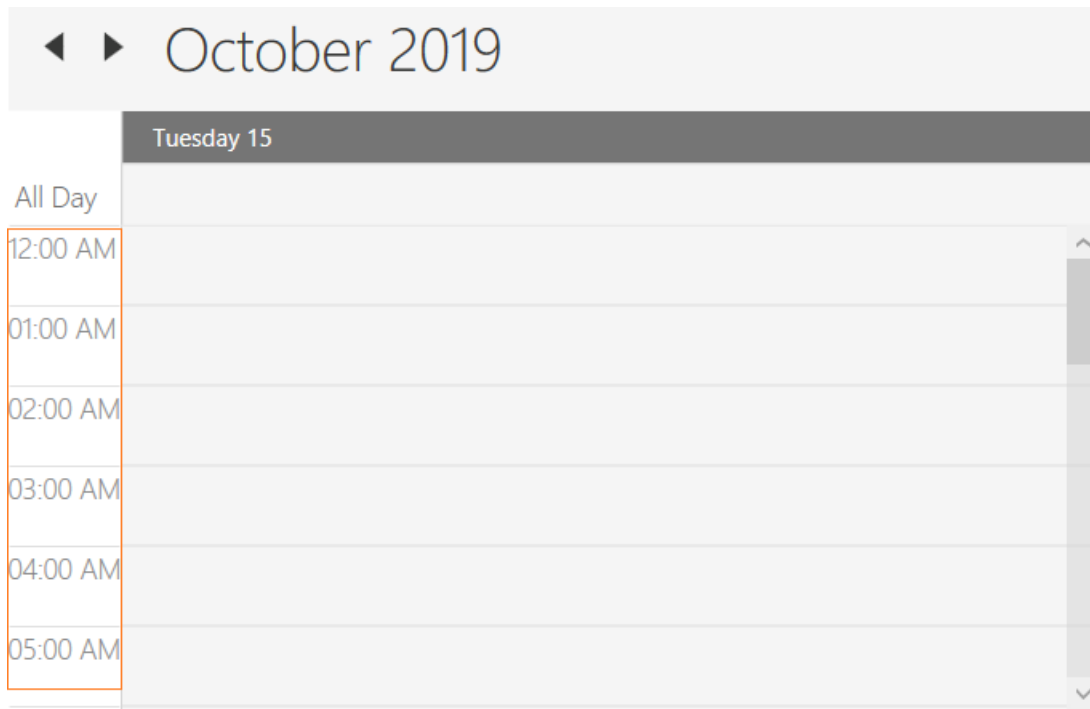
Scheduler supports to change the time interval by using [TimeInterval](#) property.

XML

```
<Schedule:SfSchedule TimeInterval = "OneHour" />
```

C#

```
this.schedule.TimeInterval = TimeInterval.OneHour;
```



Change time interval height

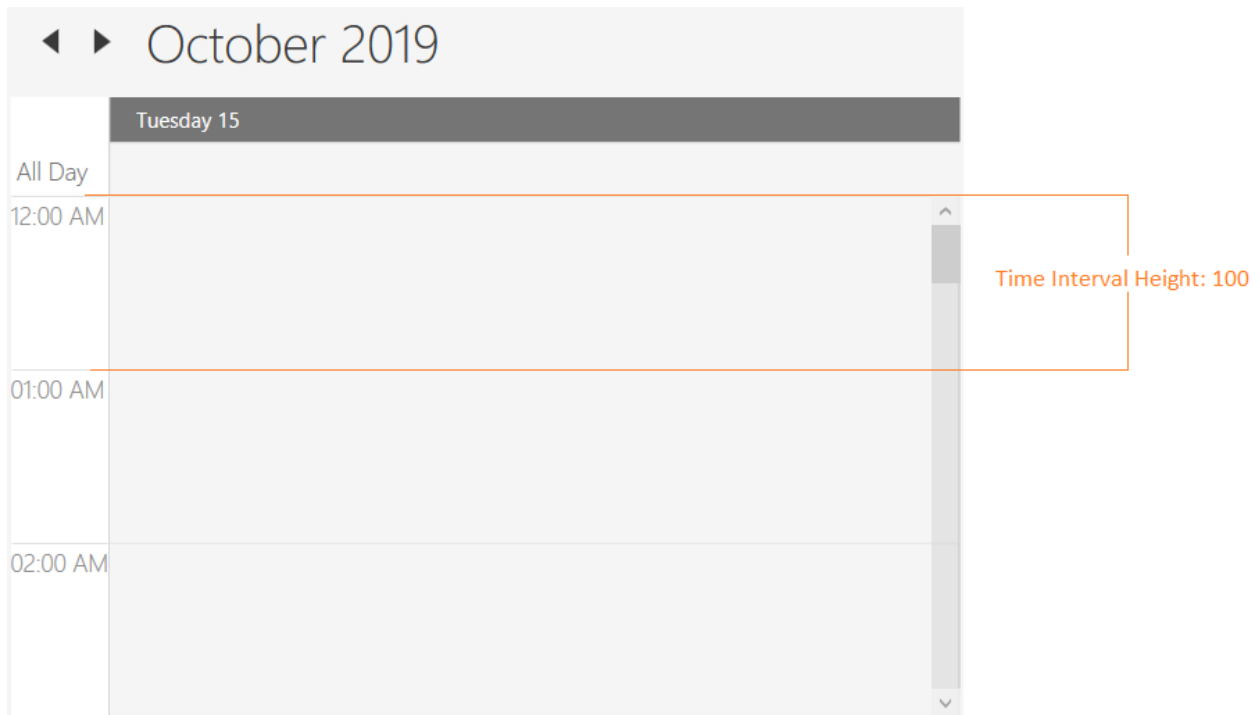
Scheduler supports to change the time interval height by using [IntervalHeight](#) property.

XML

```
<Schedule:SfSchedule IntervalHeight = 100 />
```

C#

```
this.schedule.IntervalHeight = 100;
```



Change between 12-hour and 24-hour format

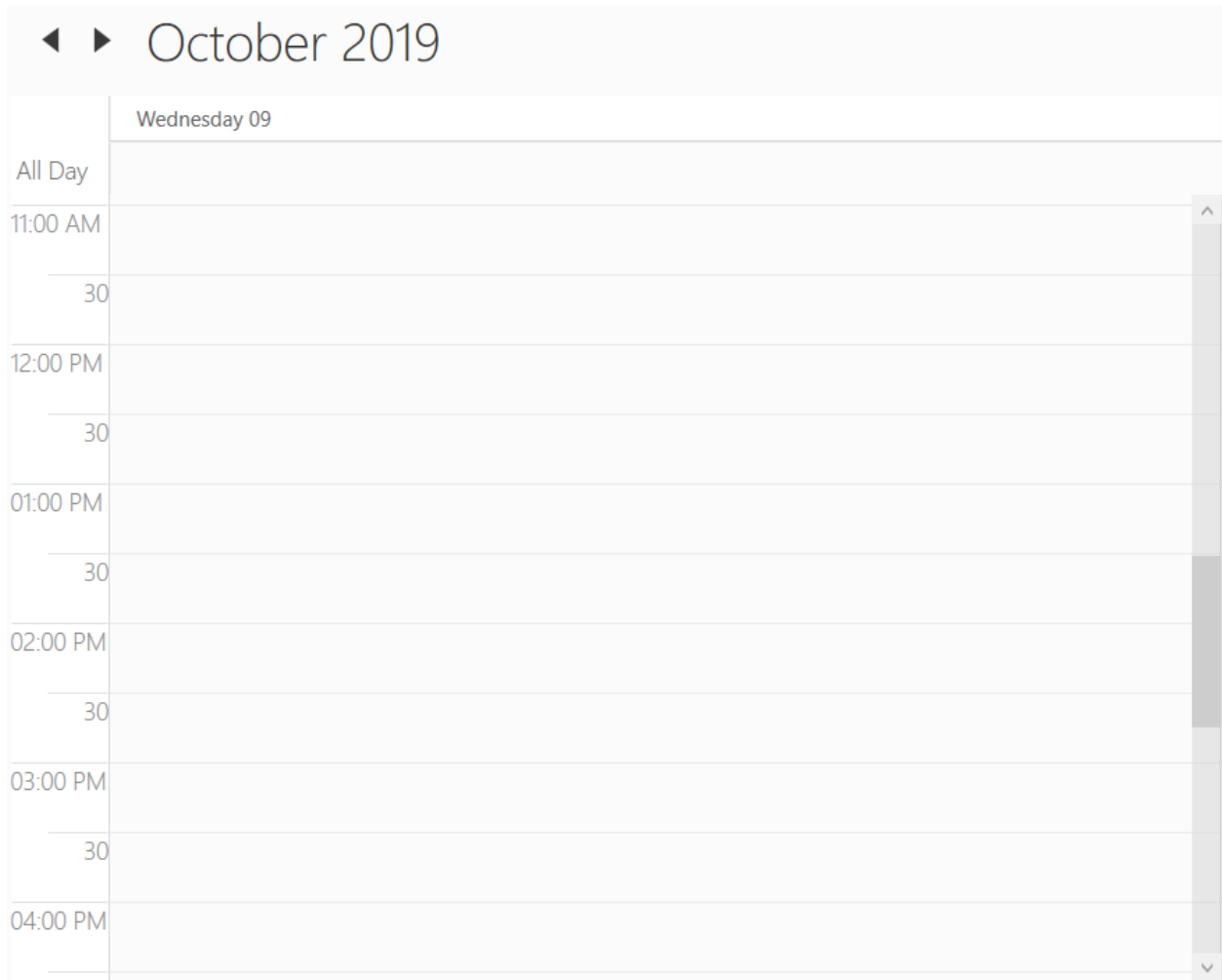
Scheduler supports to change the time format from 12hours to 24 hours by using the [TimeMode](#) property.

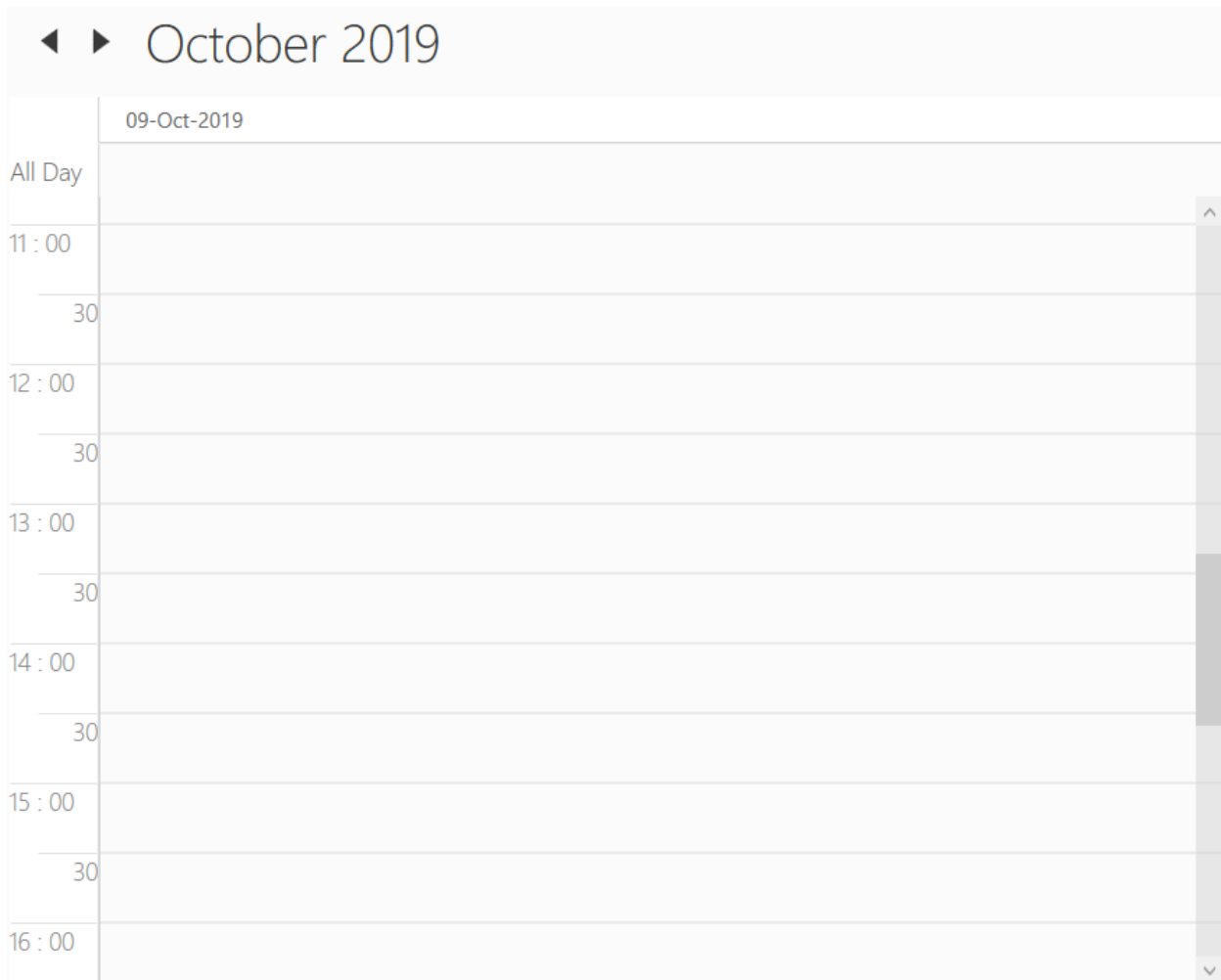
XML

```
<Grid Background="White" Name="grid">  
  <Schedule:SfSchedule ScheduleType="Day" TimeMode="TwelveHours" >  
  </Schedule:SfSchedule>  
</Grid>
```

C#

```
schedule.ScheduleType = ScheduleType.Day;  
schedule.TimeMode = TimeModes.TwelveHours;
```



Change first day of week

Scheduler supports to change the first day of week with any day by using [FirstDayOfWeek](#) property.

Day - **FirstDayOfWeek** of Scheduler is not applicable for day view as it displays only one day.

Week/Month - By default, scheduler control will be rendered with **Sunday** as the first day of the week.

WorkWeek - By default, scheduler control will be rendered with **Monday** as the first day. **Saturday** and **Sunday** has considered as a non working days.

Non-accessible timeslots

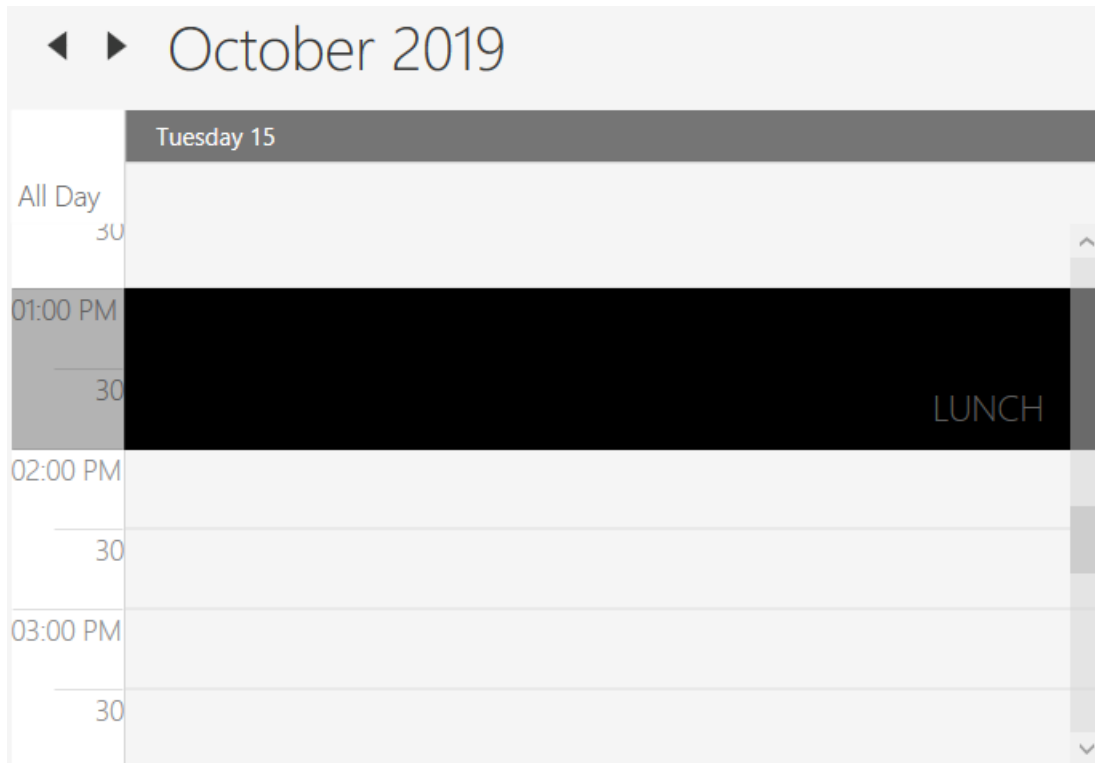
Scheduler supports to mark certain timeslots as non-accessible timeslots by using [NonAccessibleBlocks](#) property. User can't interact over the timeslot marked as non-accessible timeslots.

XML

```
<Schedule:SfSchedule>
  <Schedule:SfSchedule.NonAccessibleBlocks>
    <Schedule:NonAccessibleBlock Background="Black" StartHour="13" EndHour="14"
      Label="Lunch">
    </Schedule:NonAccessibleBlock>
  </Schedule:SfSchedule.NonAccessibleBlocks>
</Schedule:SfSchedule>
```

C#

```
this.schedule.NonAccessibleBlocks.Add(new NonAccessibleBlock() { Background = new SolidColorBrush(Colors.Black), StartHour = 13, EndHour = 14, Label = "Lunch" });
```

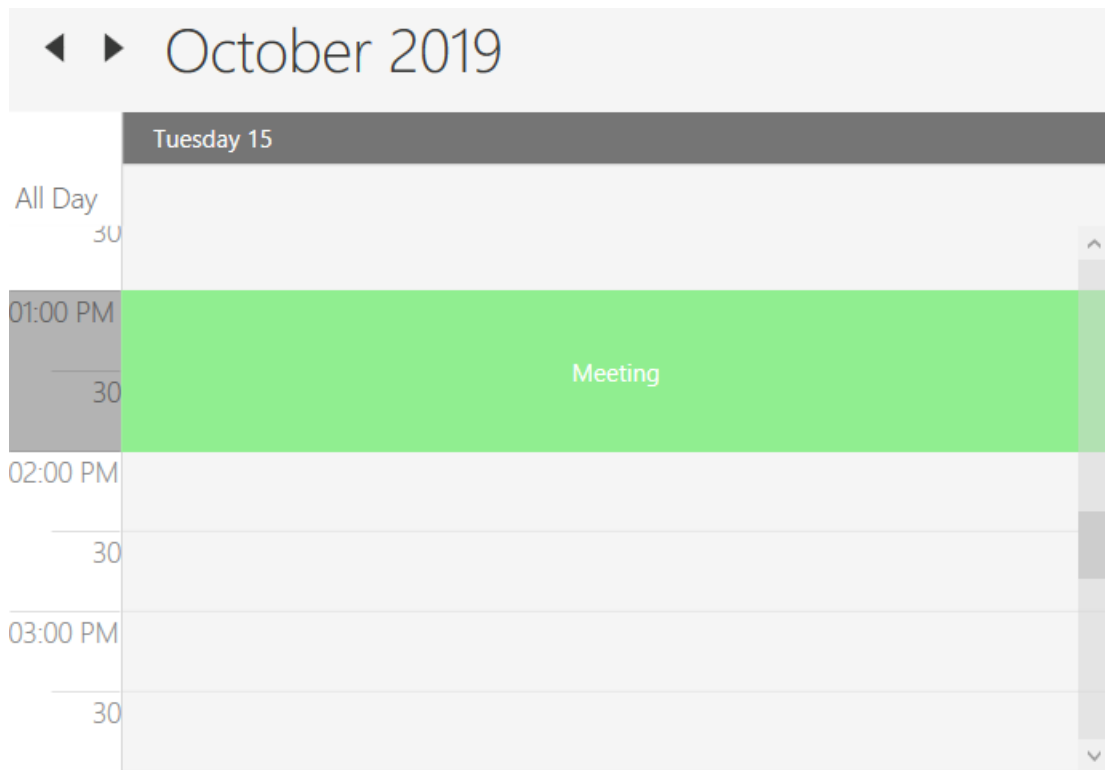


Customize non-accessible timeslots using template

Scheduler supports to customize the non-accessible timeslots by using [NonAccessibleBlockTemplate](#) property.

XML

```
<syncfusion:SfSchedule.NonAccessibleBlockTemplate>
  <DataTemplate>
    <Border Background="{Binding Color}">
      <TextBlock Text="{Binding EventName}" Foreground="White"
        HorizontalAlignment="Center" VerticalAlignment="Center"/>
    </Border>
  </DataTemplate>
</syncfusion:SfSchedule.NonAccessibleBlockTemplate>
```



Change non-working days

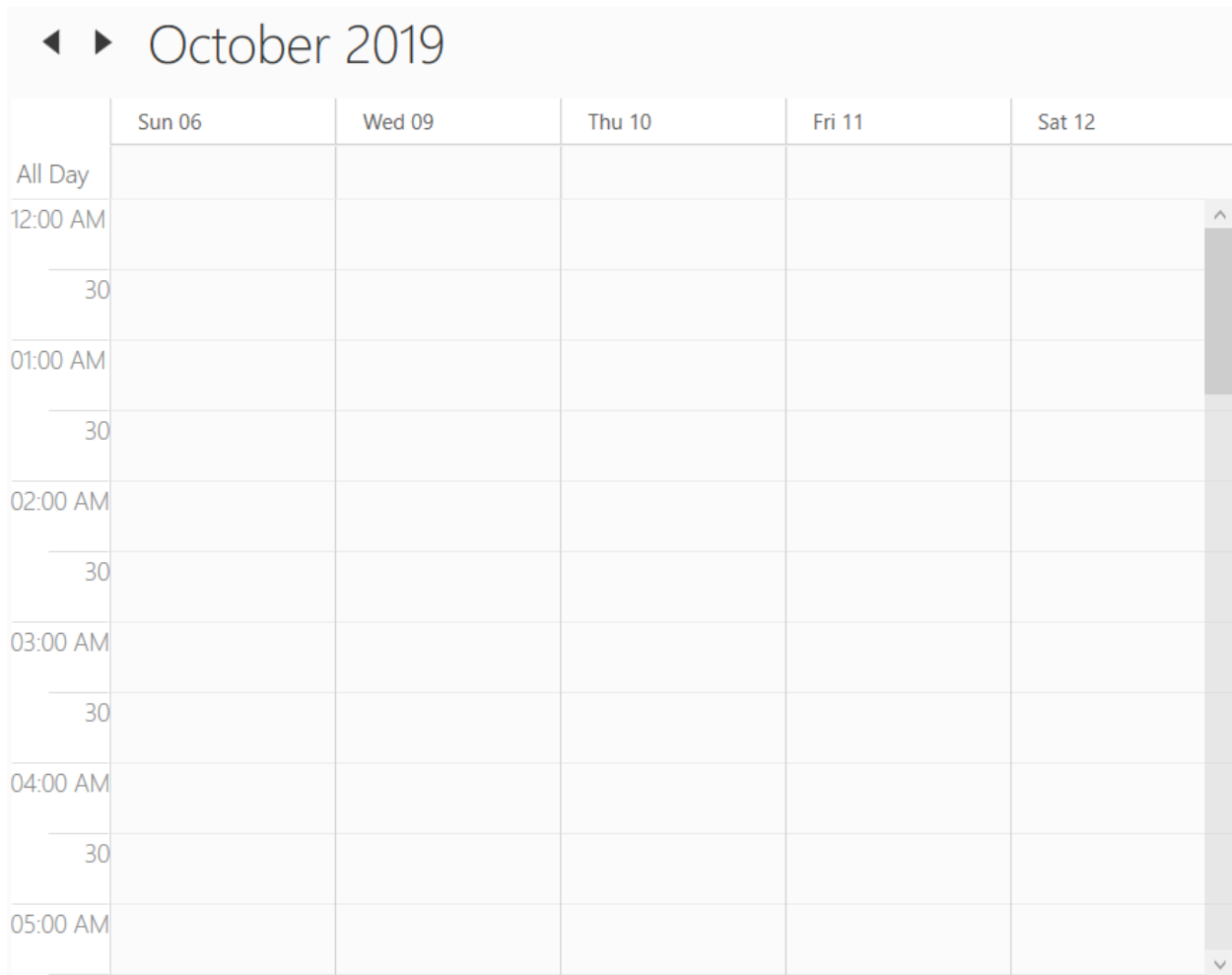
By default Schedule considers **Saturday** and **Sunday** as a non working days. You can change the non-working days using [NonWorkingDays](#) property.

XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleType="WorkWeek"
NonWorkingDays="Monday,Tuesday"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.WorkWeek;
this.schedule.NonWorkingDays = DayOfWeek.Monday.ToString() + "," +
DayOfWeek.Tuesday.ToString();
```



Collapsed hours

Scheduler supports to hide the selected hours by using [CollapsedHours](#) property. [ScheduleCollapsedHours](#) does have the following properties.

[StartHour](#) - To set start time of collapsed hour.

[EndHour](#) - To set end time of collapsed hour.

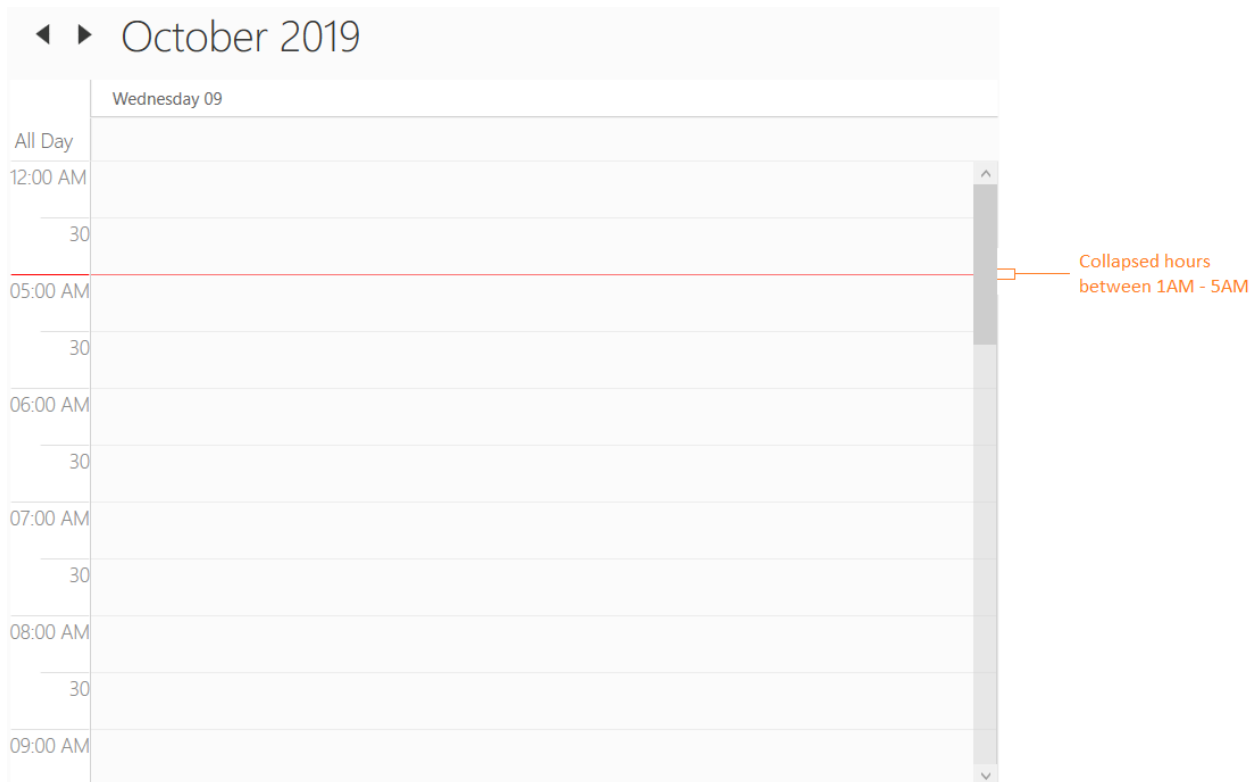
[Background](#) - To set the background of collapsed hours.

XML

```
<schedule:SfSchedule Background="White" x:Name="schedule"
ScheduleType="Week">
  <schedule:SfSchedule.CollapsedHours>
    <schedule:ScheduleCollapsedHour StartHour="1" EndHour="5"
    Background="Red"/>
  </schedule:SfSchedule.CollapsedHours>
</schedule:SfSchedule>
```

C#

```
this.schedule.CollapsedHours.Add(new ScheduleCollapsedHour() { StartHour =
1, EndHour = 5, Background = new SolidColorBrush(Colors.Red) });
```



Change working hours

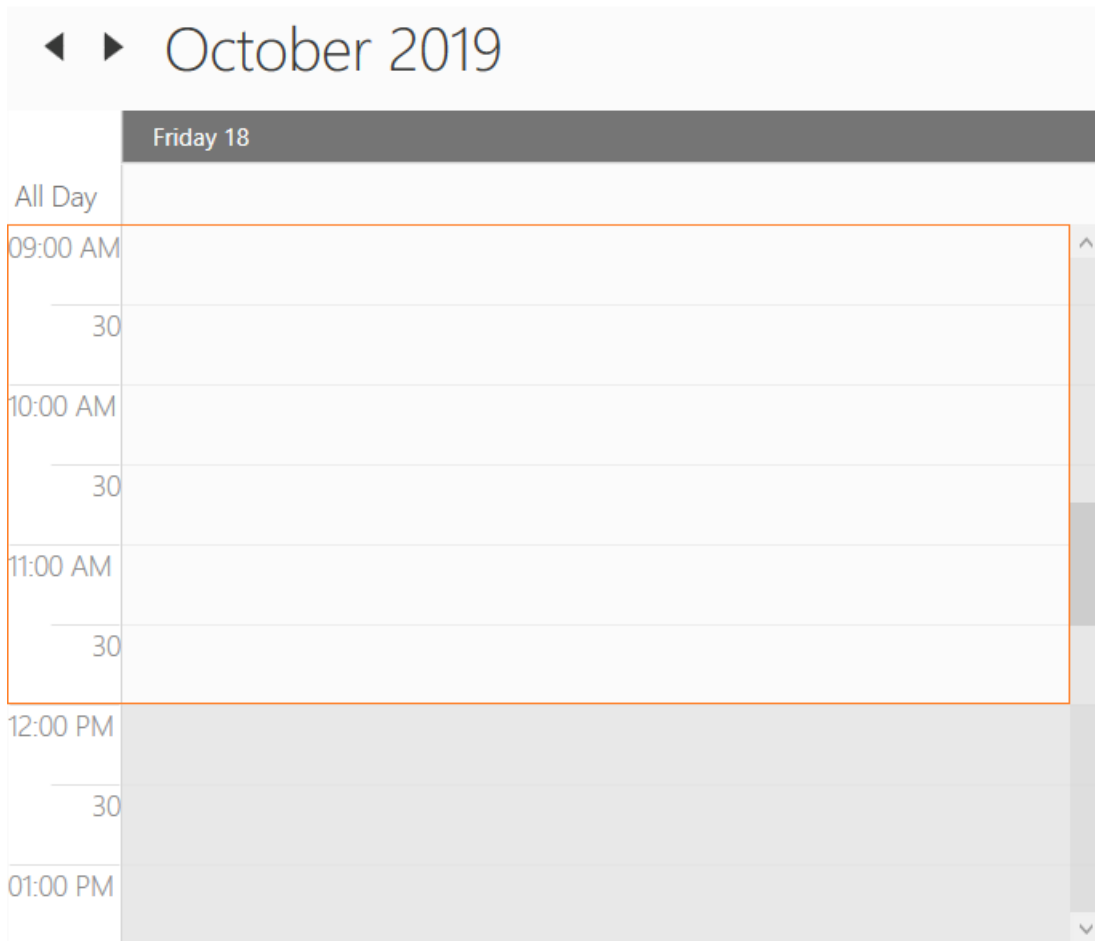
Working hours of Scheduler will be differentiated with non-working hours by separate color using [IsHighLightWorkingHours](#) property for day, week, workweek and timeline views. By default, working hours will be between 09 to 18. You can customize the working hours by setting [WorkStartHour](#) and [WorkEndHour](#) properties.

XML

```
<schedule:SfSchedule x:Name="schedule"
    WorkStartHour="9"
    WorkEndHour="12"
    IsHighLightWorkingHours="True"/>
```

C#

```
this.schedule.WorkStartHour = 9;
this.schedule.WorkEndHour = 12;
this.schedule.IsHighLightWorkingHours = true;
```



Display working hours only

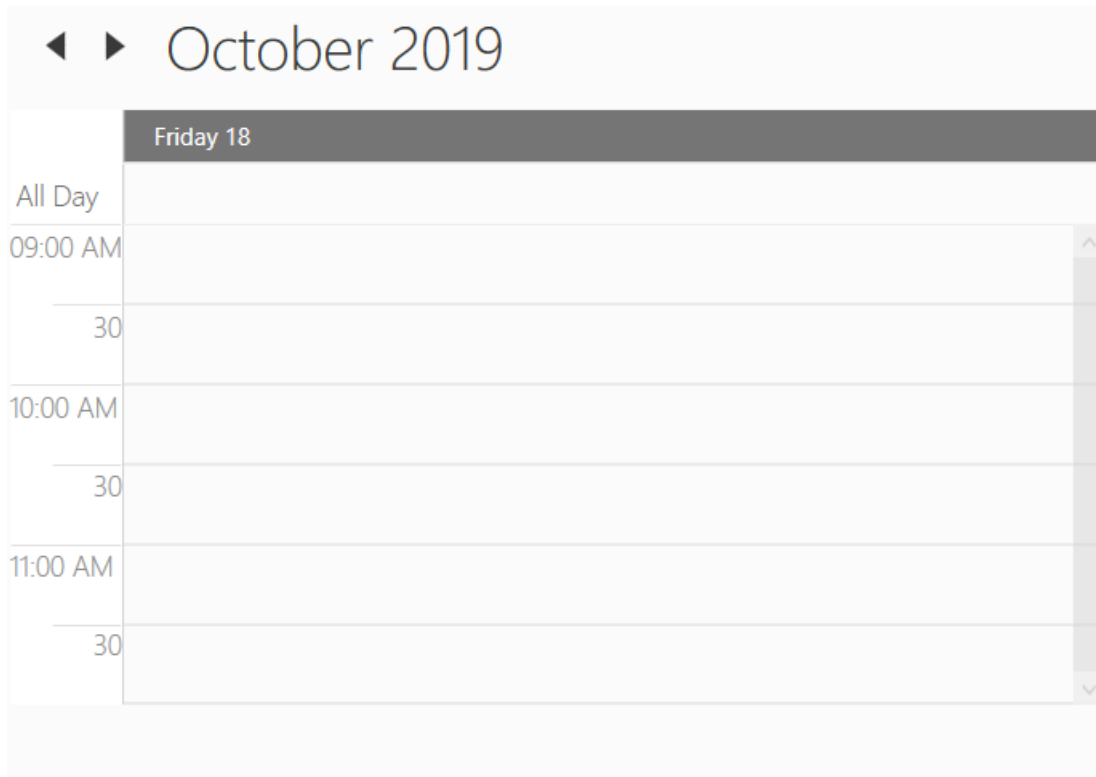
Scheduler supports to display the working hours only by disabling the [ShowNonWorkingHours](#) property.

XML

```
<schedule:SfSchedule x:Name="schedule"
    WorkStartHour="9"
    WorkEndHour="12"
    ShowNonWorkingHours="False"/>
```

C#

```
this.schedule.WorkStartHour = 9;
this.schedule.WorkEndHour = 12;
this.schedule.ShowNonWorkingHours = false;
```



Change non-working hours background

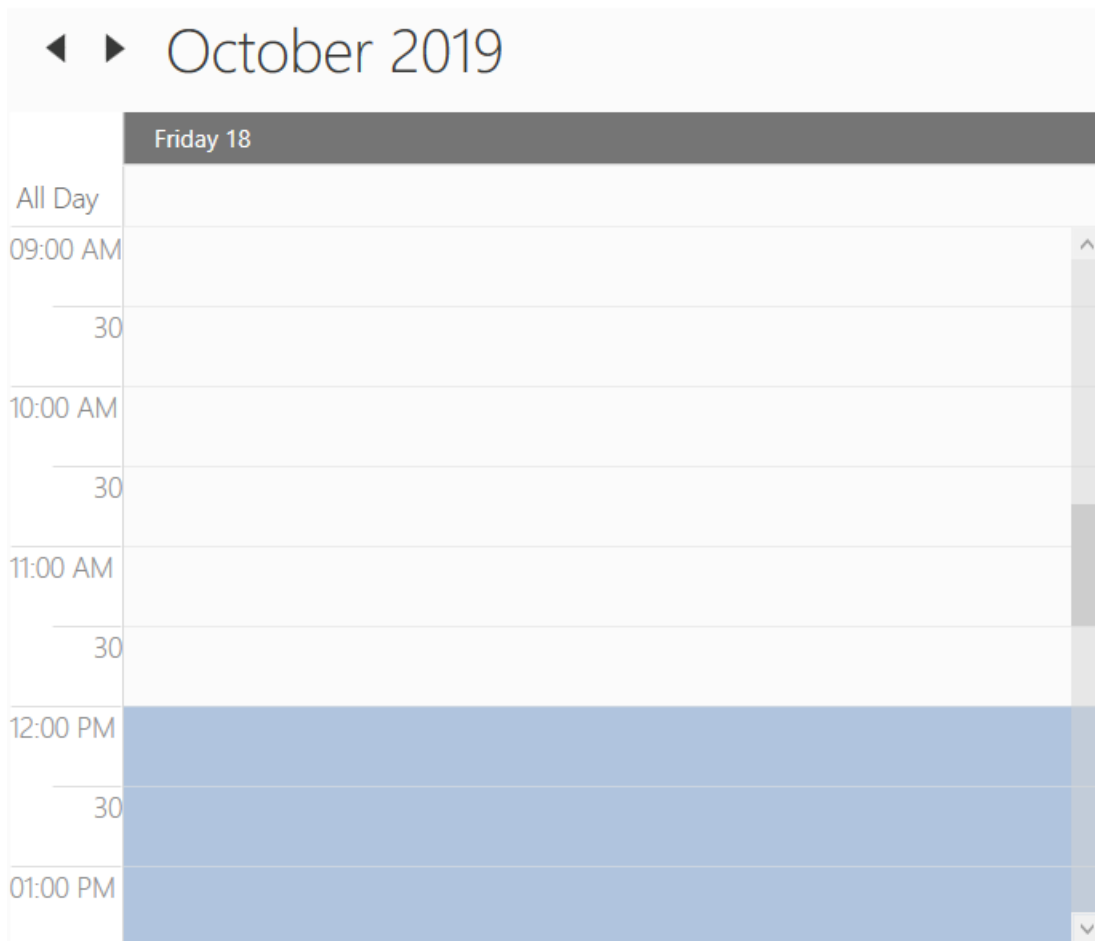
Scheduler supports to change the background color for non-working hours by using [NonWorkingHourBrush](#) property.

XML

```
<schedule:SfSchedule x:Name="schedule"
    WorkStartHour="9"
    WorkEndHour="12"
    NonWorkingHourBrush="LightSteelBlue"
    IsHighLightWorkingHours="True"/>
```

C#

```
this.schedule.WorkStartHour = 9;
this.schedule.WorkEndHour = 12;
this.schedule.NonWorkingHourBrush = Brushes.LightSteelBlue;
this.schedule.IsHighLightWorkingHours = true;
```

Current time indicator

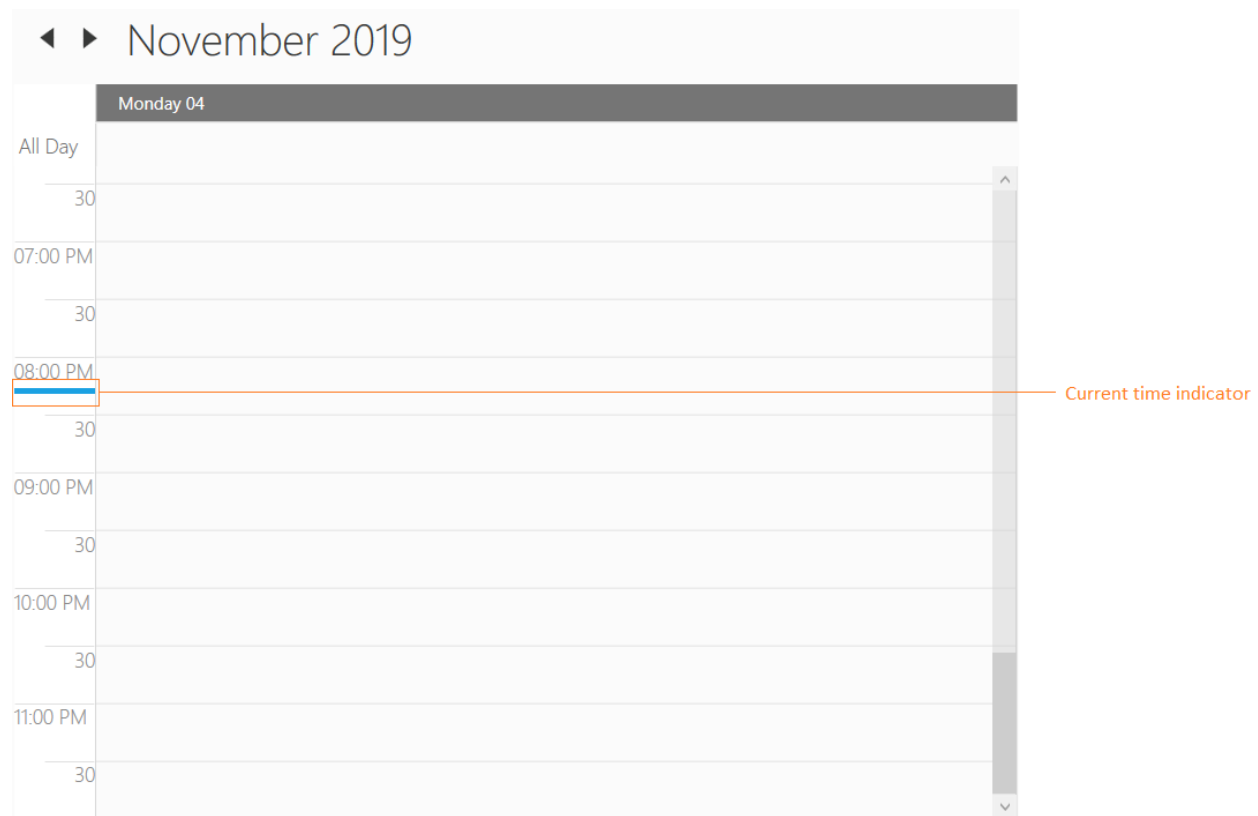
Scheduler supports to display the current time indicator by using the [CurrentTimeIndicatorVisibility](#) property.

XML

```
<syncfusion:SfSchedule ScheduleType="Day"
    CurrentTimeIndicatorVisibility="Visible"/>
```

C#

```
schedule.ScheduleType = ScheduleType.Day;
this.schedule.CurrentTimeIndicatorVisibility = Visibility.Visible;
```



Customize current time indicator

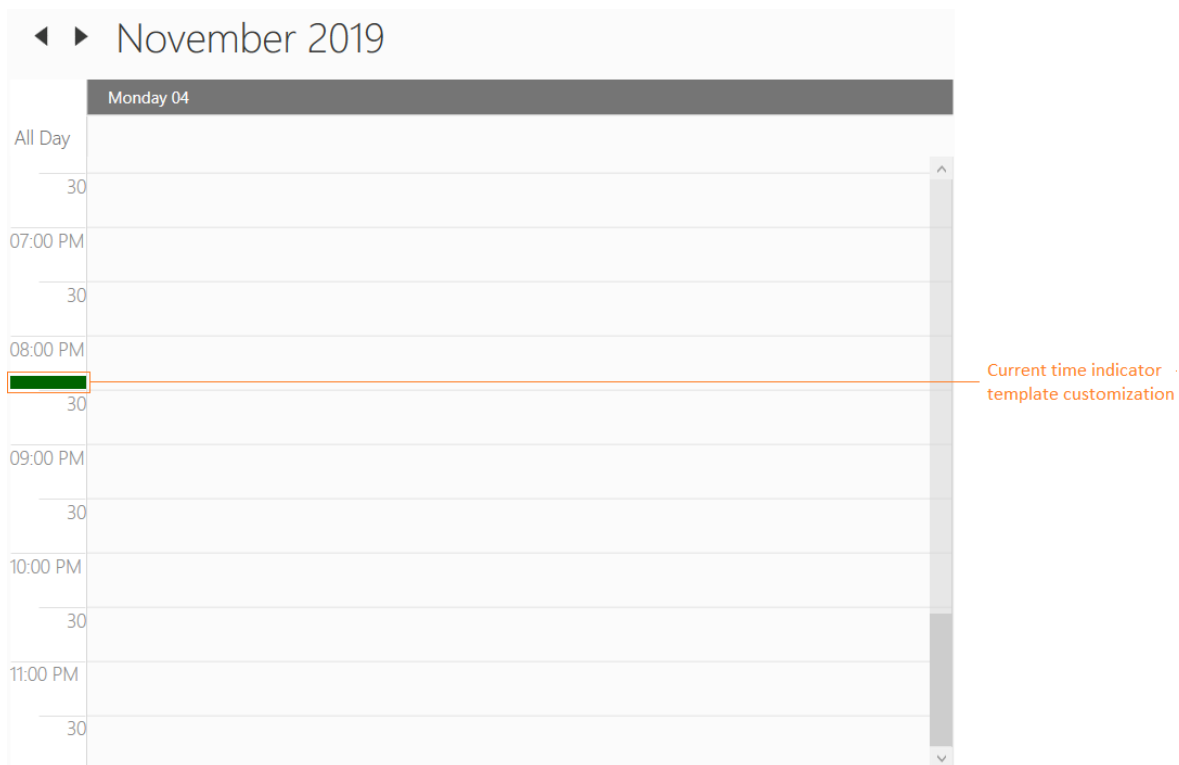
Scheduler supports to customize the current time indicator by using [CurrentTimeIndicatorTemplate](#) property.

XML

```
<Schedule:SfSchedule x:Name="schedule" ScheduleType="Day"
CurrentTimeIndicatorVisibility="Visible">
  <Schedule:SfSchedule.CurrentTimeIndicatorTemplate>
    <DataTemplate>
      <Border Background="DarkGreen" Height="10" Width="100"></Border>
    </DataTemplate>
  </Schedule:SfSchedule.CurrentTimeIndicatorTemplate>
</Schedule:SfSchedule>
```

C#

```
schedule.ScheduleType = ScheduleType.Day;
schedule.CurrentTimeIndicatorVisibility = Visibility.Visible;
schedule.CurrentTimeIndicatorTemplate =
(DataTemplate) this.Resources["CurrentTimeIndicatorTemplate"];
```



Change hours or minutes time label visibility

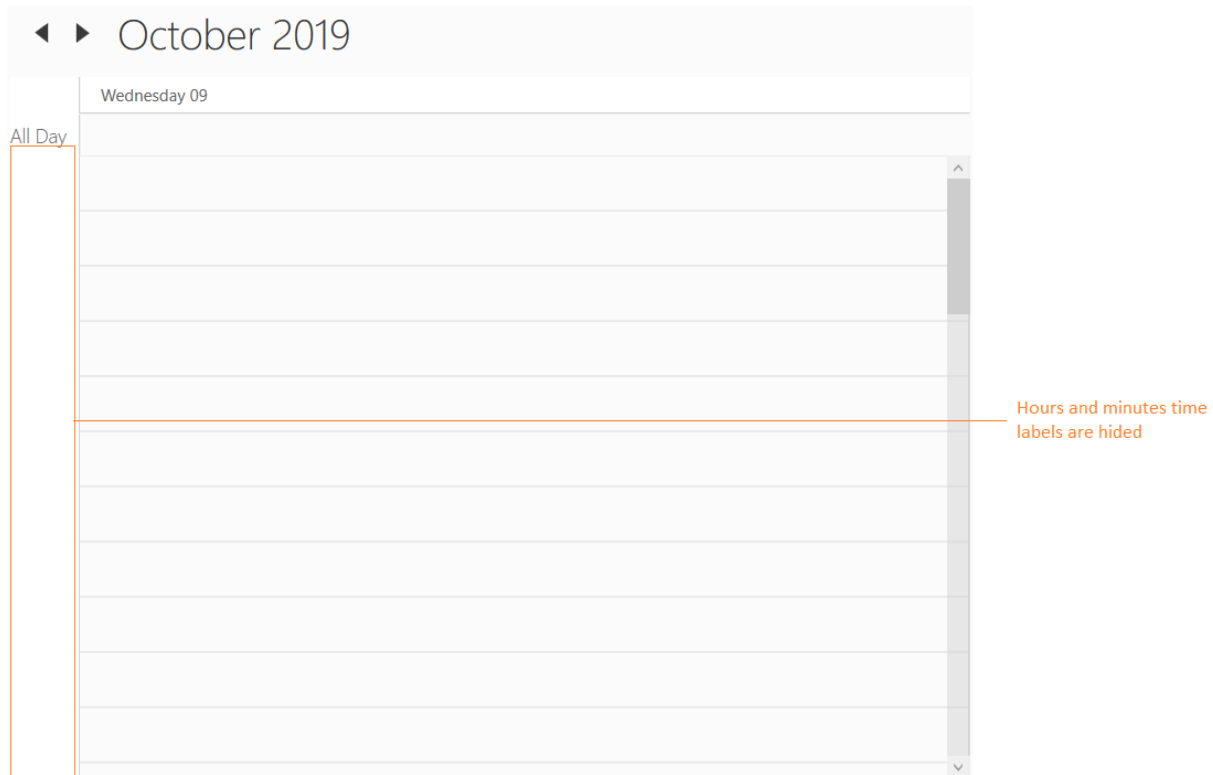
Scheduler supports to visible/collapse the hours and minutes time label visibility by using [MajorTickVisibility](#) and [MinorTickVisibility](#) properties.

XML

```
<Schedule:SfSchedule ScheduleType="Day" MajorTickVisibility="Collapsed"
MinorTickVisibility="Collapsed" />
```

C#

```
this.schedule.ScheduleType = ScheduleType.Day;
this.schedule.MajorTickVisibility = Visibility.Collapsed;
this.schedule.MinorTickVisibility = Visibility.Collapsed;
```



Appearance

Changing time label background

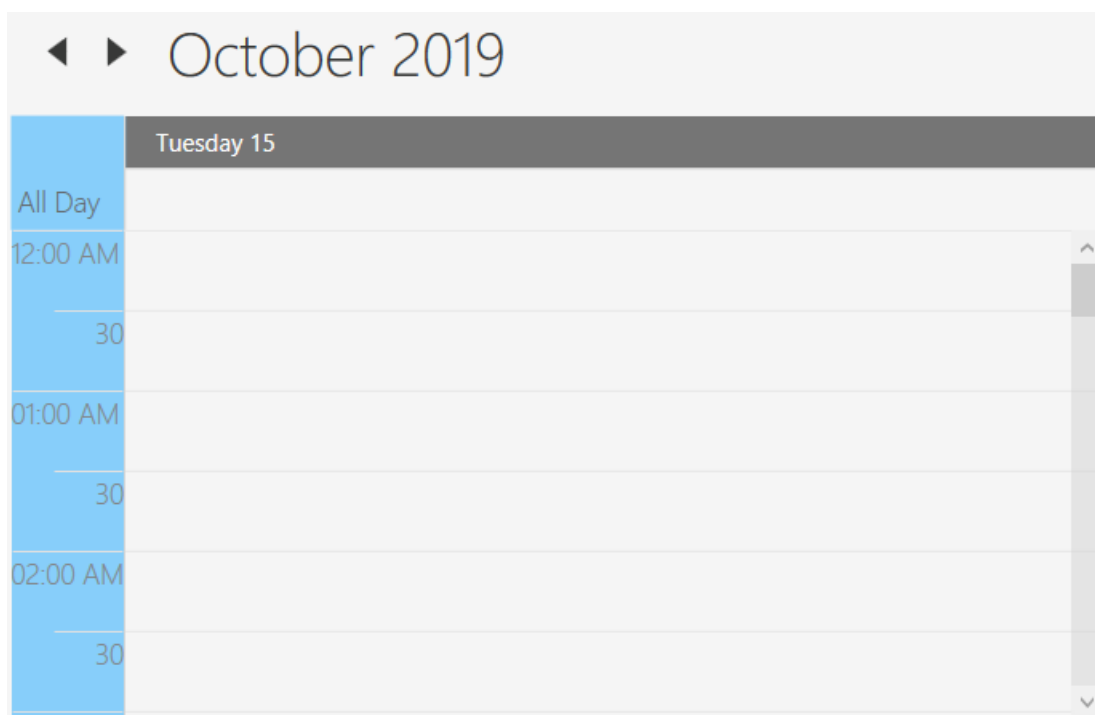
Scheduler supports to change the time slot label background by using [HeaderBackground](#) property.

XML

```
<Schedule:SfSchedule HeaderBackground="LightSkyBlue" />
```

C#

```
this.schedule.HeaderBackground = Brushes.LightSkyBlue;
```



Stroke customization

In Scheduler control, major, minor horizontal and vertical lines drawn in the day, week, workweek and timeline views by using following properties,

Property Table

API Name	Data Type	Description
MajorTickStroke	Brush	Used to customize the major line stroke of the day, week, workweek and timeline views.
MinorTickStroke	Brush	Used to customize the minor line stroke of the day, week, workweek and timeline views.
MajorTickLabelStroke	Brush	Used to customize the major line label stroke in the day, week, workweek and timeline views.
MinorTickLabelStroke	Brush	Used to customize the minor line label stroke of the day, week, workweek and timeline views.
MajorTickStrokeDashArray	DoubleCollection	Used to customize the major line stroke dash array of the day, week, workweek and timeline views.
MinorTickStrokeDashArray	DoubleCollection	Used to customize the minor line stroke dash array of the day, week, workweek and timeline views.
DayViewVerticalLineStroke	Brush	Used to customize the vertical line stroke of the day, week and workweek view.

Changing time label foreground

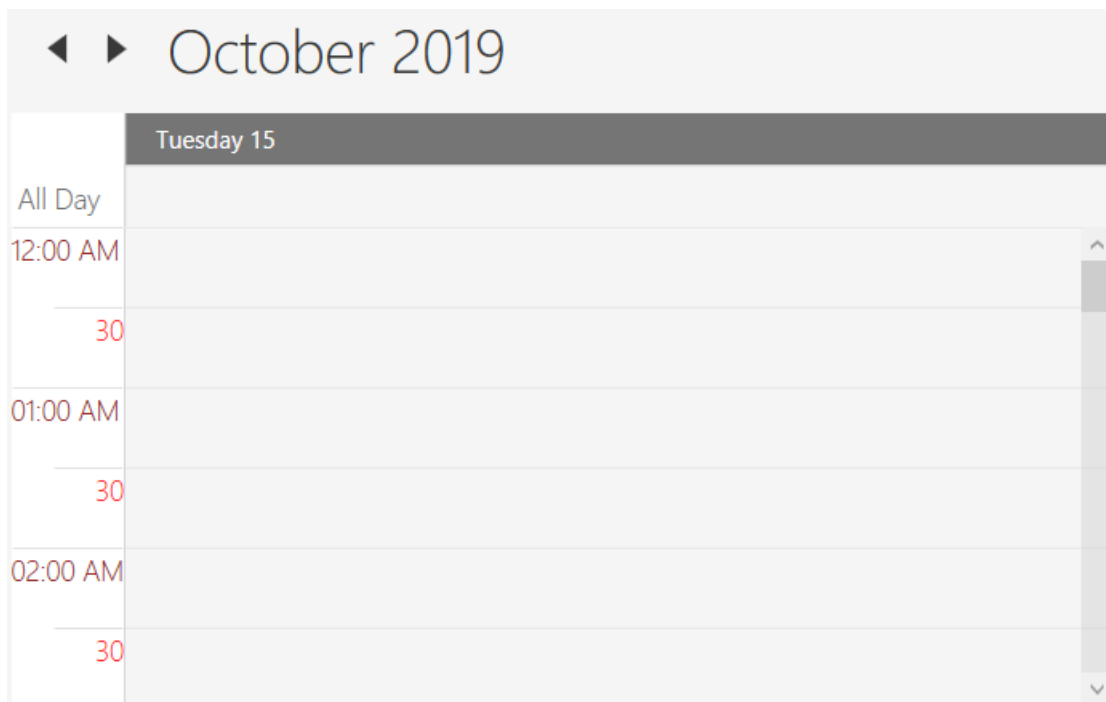
Scheduler supports to change the time label foreground by using [MinorTickLabelStroke](#) and [MajorTickLabelStroke](#) property.

XML

```
<syncfusion:SfSchedule MajorTickLabelStroke="DarkRed"
MinorTickLabelStroke="Red"/>
```

C#

```
this.schedule.MajorTickLabelStroke = Brushes.DarkRed;
this.schedule.MinorTickLabelStroke = Brushes.Red;
```



Changing timeslots line color

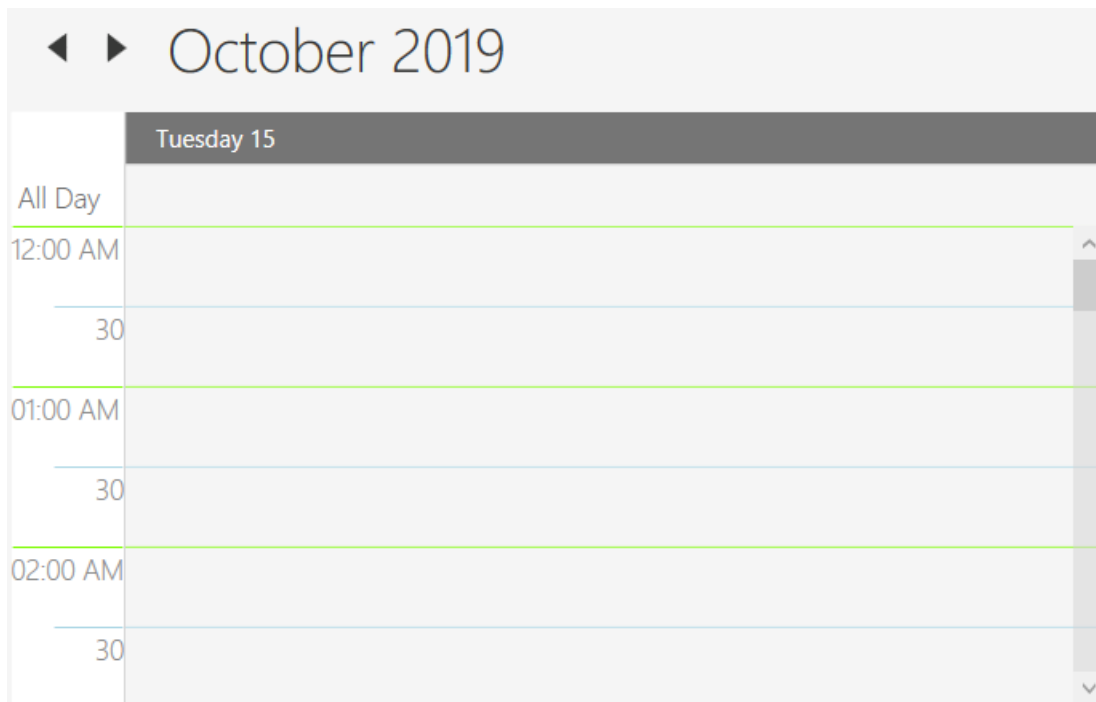
Scheduler supports to change the time slots line color by using [MajorTickStroke](#) and [MinorTickStroke](#) property.

XML

```
<syncfusion:SfSchedule MajorTickStroke="LawnGreen"
MinorTickStroke="LightBlue"/>
```

C#

```
this.schedule.MajorTickStroke = Brushes.LawnGreen;
this.schedule.MinorTickStroke = Brushes.LightBlue;
```



Changing timeslots line style

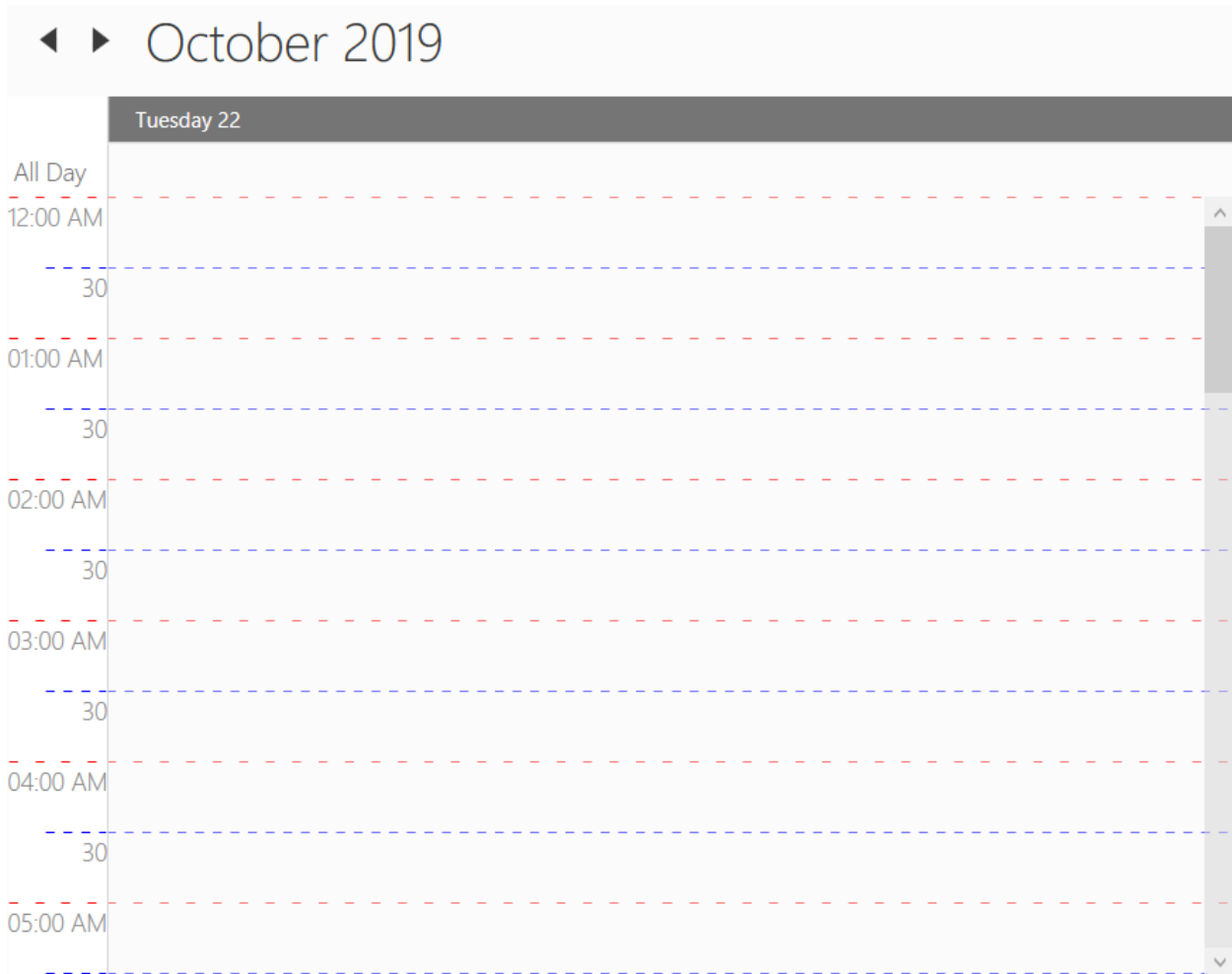
Scheduler supports to customize the major/minor line stroke style of the day, week, workweek and time line views.

XML

```
<syncfusion:SfSchedule MajorTickStroke="Red" MajorTickStrokeDashArray="5,10"
MinorTickStroke="Blue" MinorTickStrokeDashArray="5,5"/>
```

C#

```
this.schedule.MajorTickStroke = Brushes.Red;
this.schedule.MajorTickStrokeDashArray="5,10";
this.schedule.MinorTickStroke = Brushes.Blue;
this.schedule.MinorTickStrokeDashArray="5,5";
```



Changing vertical line color

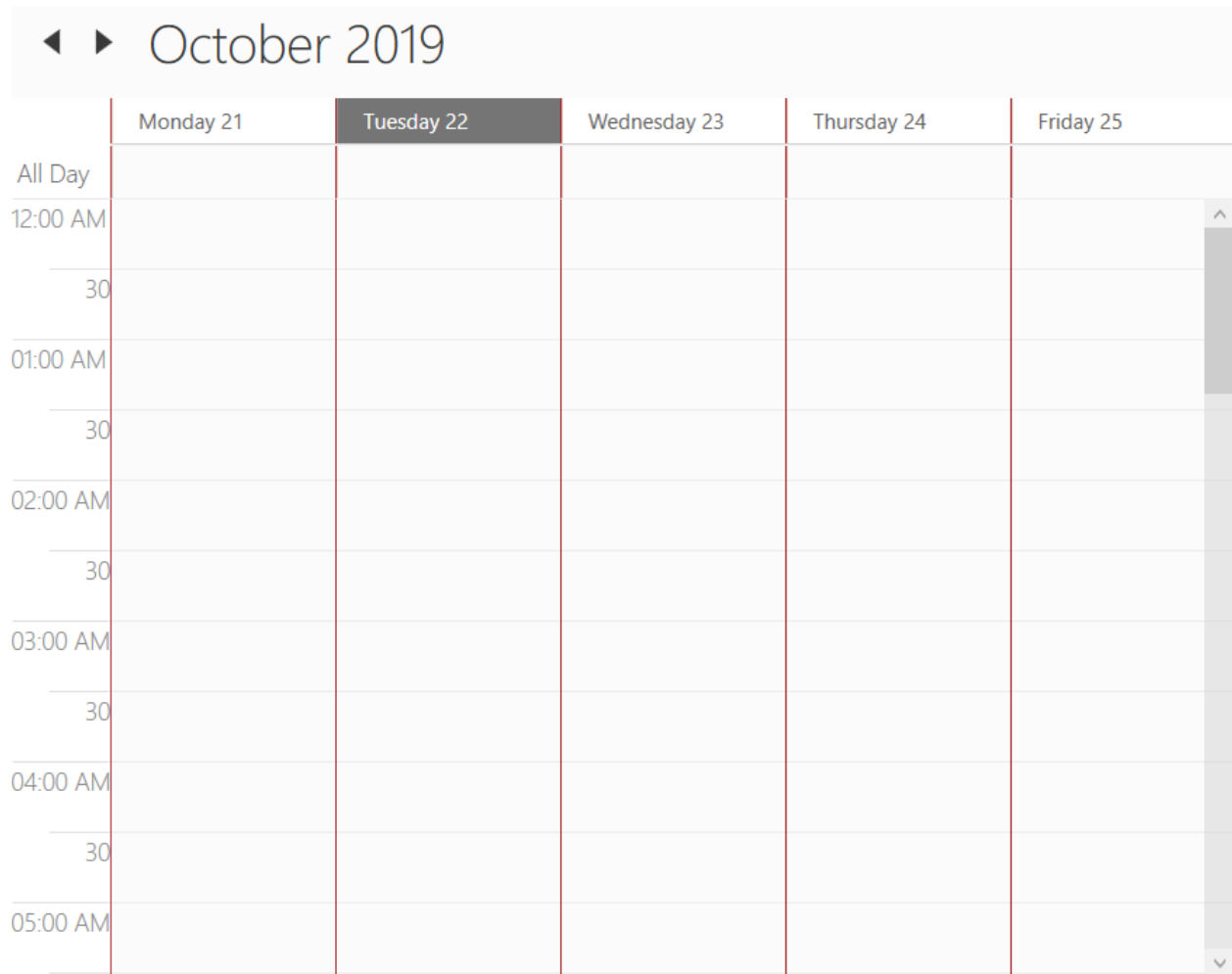
Scheduler supports to change the vertical line for day, week and workweek view by using [DayViewVerticalLineStroke](#).

XML

```
<syncfusion:SfSchedule ScheduleType="WorkWeek"
    DayViewVerticalLineStroke="Brown"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.WorkWeek;
this.schedule.DayViewVerticalLineStroke = Brushes.Brown;
```

Current day highlighting

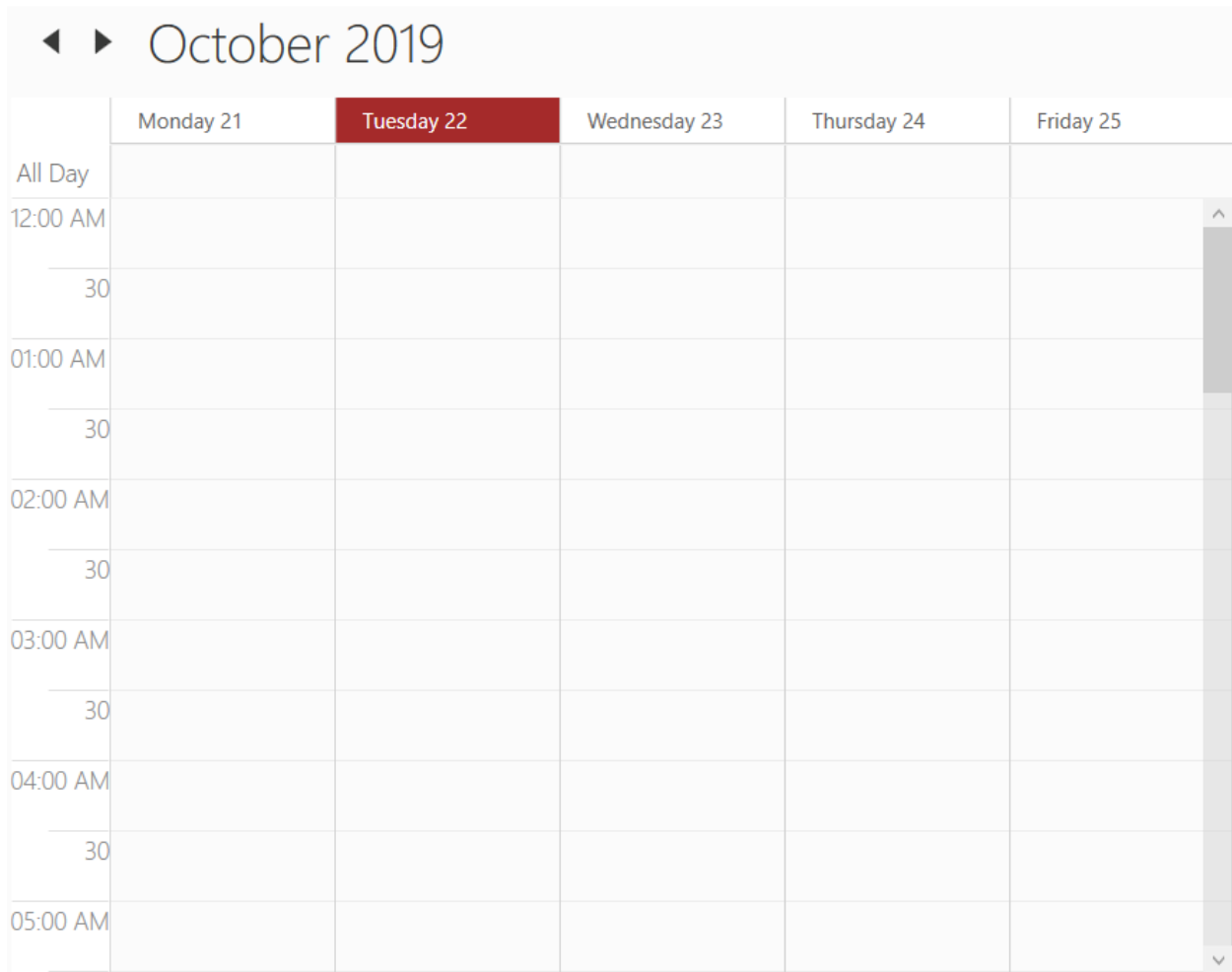
Scheduler supports to change current day background and foreground for all views by using [CurrentDateBackground](#) and [CurrentDateForeground](#) property.

XML

```
<syncfusion:SfSchedule CurrentDateBackground="Brown"
CurrentDateForeground="White"/>
```

C#

```
this.schedule.CurrentDateBackground = Brushes.Brown;
this.schedule.CurrentDateForeground = Brushes.White;
```



Change schedule view settings based on the views at run time

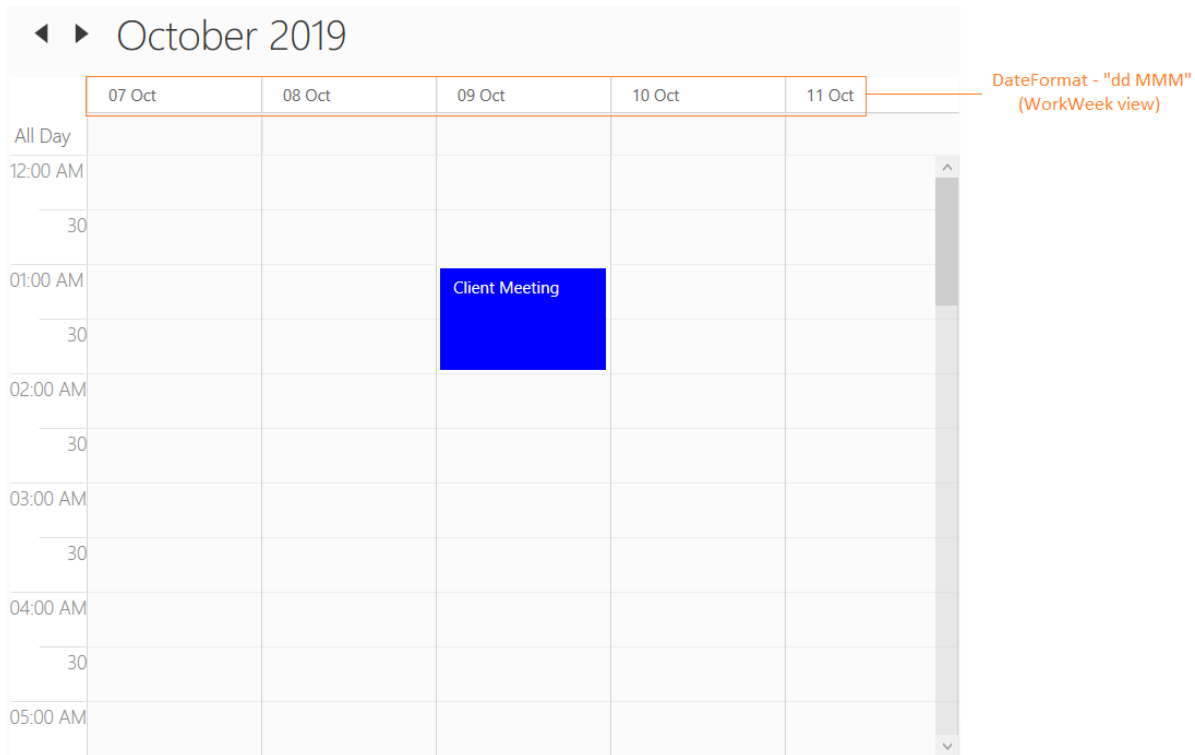
Scheduler supports to notify before that changing the schedule view by using [ScheduleTypeChanging](#) event. By this event, the appearance can be adjusted depending on the view.

For example, if you want to change the header date format based on the schedule view, you can use this event. Please refer the following code example,

C#

```
this.schedule.ScheduleTypeChanging += Schedule_ScheduleTypeChanging;
private void Schedule_ScheduleTypeChanging(object sender,
ScheduleTypeChangingEventArgs e)
{
    switch (e.NewValue)
    {
        case ScheduleType.Day:
            this.schedule.HeaderDateFormat = "dd-MMM-yyyy";
            break;
        case ScheduleType.Week:
            this.schedule.HeaderDateFormat = "dd-MMM";
            break;
        case ScheduleType.WorkWeek:
            this.schedule.HeaderDateFormat = "dd MMM";
            break;
    }
}
```

```
}
}
```



Month View in WPF Schedule (Classic)

Scheduler used to display entire dates of the specific month, current month will be displayed by default initially. **Month** view displays the month of dates similar to calendar and displays appointments for each day in a cell similar to outlook.

Change header date format

Scheduler supports to change header format of the month view by using [MonthHeaderDateFormat](#) property.

XML

```
<Schedule:SfSchedule ScheduleType="Month" MonthHeaderDateFormat="dd/MM"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.Month;
this.schedule.HeaderDateFormat = "dd/MM";
```

◀ ▶ October 2019						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
29/09	30/09	01/10	02/10	03/10	04/10	05/10
06/10	07/10	08/10	09/10	10/10	11/10	12/10
13/10	14/10	15/10	16/10	17/10	18/10	19/10
20/10	21/10	22/10	23/10	24/10	25/10	26/10
27/10	28/10	29/10	30/10	31/10	01/11	02/11

Change header background

Scheduler supports to change the header background by using [HeaderBackground](#) property.

XML

```
<Schedule:SfSchedule ScheduleType="Month" HeaderBackground="Cyan"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.Month;
this.schedule.HeaderBackground = Brushes.Cyan;
```

◀ ▶ October 2019

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
29	30	01	02	03	04	05
06	07	08	09	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	01	02

Change first day of week

Scheduler supports to change the first day of week by using [FirstDayOfWeek](#) property.

XML

```
<Schedule:SfSchedule ScheduleType="Month" FirstDayOfWeek="Tuesday"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.Month;
this.schedule.FirstDayOfWeek = DayOfWeek.Tuesday;
```

◀ ▶ October 2019						
Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Monday
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	01	02	03	04

Change active and inactive month dates background

Scheduler supports to change the active and inactive month background by using [FocusedMonth](#) and [NonFocusedMonth](#) property.

XML

```
<Schedule:SfSchedule ScheduleType="Month" FocusedMonth="AliceBlue"
NonFocusedMonth = "SkyBlue"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.Month;
this.schedule.FocusedMonth = Brushes.AliceBlue;
this.schedule.NonFocusedMonth = Brushes.SkyBlue;
```

◀ ▶ October 2019						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
29	30	01	02	03	04	05
06	07	08	09	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	01	02

Change the border color

Scheduler support to change the header items border color by using [MonthViewLineStroke](#) property.

XML

```
<Schedule:SfSchedule ScheduleType="Month" MonthViewLineStroke="BlueViolet"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.Month;
this.schedule.MonthViewLineStroke = Brushes.BlueViolet;
```

◀ ▶ October 2019						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
29	30	01	02	03	04	05
06	07	08	09	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	01	02

Change the selection background

Scheduler supports to change the selection background by using [CellSelectionBrush](#)

XML

```
<Schedule:SfSchedule ScheduleType="Month" CellSelectionBrush="SkyBlue"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.Month;
this.schedule.CellSelectionBrush = Brushes.SkyBlue;
```


◀ ▶ October 2019						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
29	30	01	02	03	04	05
06	07	08	09	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	01	02

Current day highlighting

Scheduler supports to change current day background and foreground for month view by using [CurrentDateBackground](#) and [CurrentDateForeground](#) property.

XML

```
<syncfusion:SfSchedule ScheduleType="Month" CurrentDateBackground="Brown"
CurrentDateForeground="White"/>
```

C#

```
this.schedule.ScheduleType = ScheduleType.Month;
this.schedule.CurrentDateBackground = Brushes.Brown;
this.schedule.CurrentDateForeground = Brushes.White;
```

◀ ▶ November 2019						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
27	28	29	30	31	01	02
03	04	05	06	07	08	09
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Timeline View in WPF Schedule (Classic)

Timeline view displays the dates with the appropriate day count in the horizontal time axis. When moving right or left, you can see the past or future events. With an intuitive drag-and-drop feature, each view shows events accurately through time slots.

XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleType="TimeLine"/>
```

C#

```
this.schedule.ScheduleType= ScheduleType.TimeLine;
```

Header date format

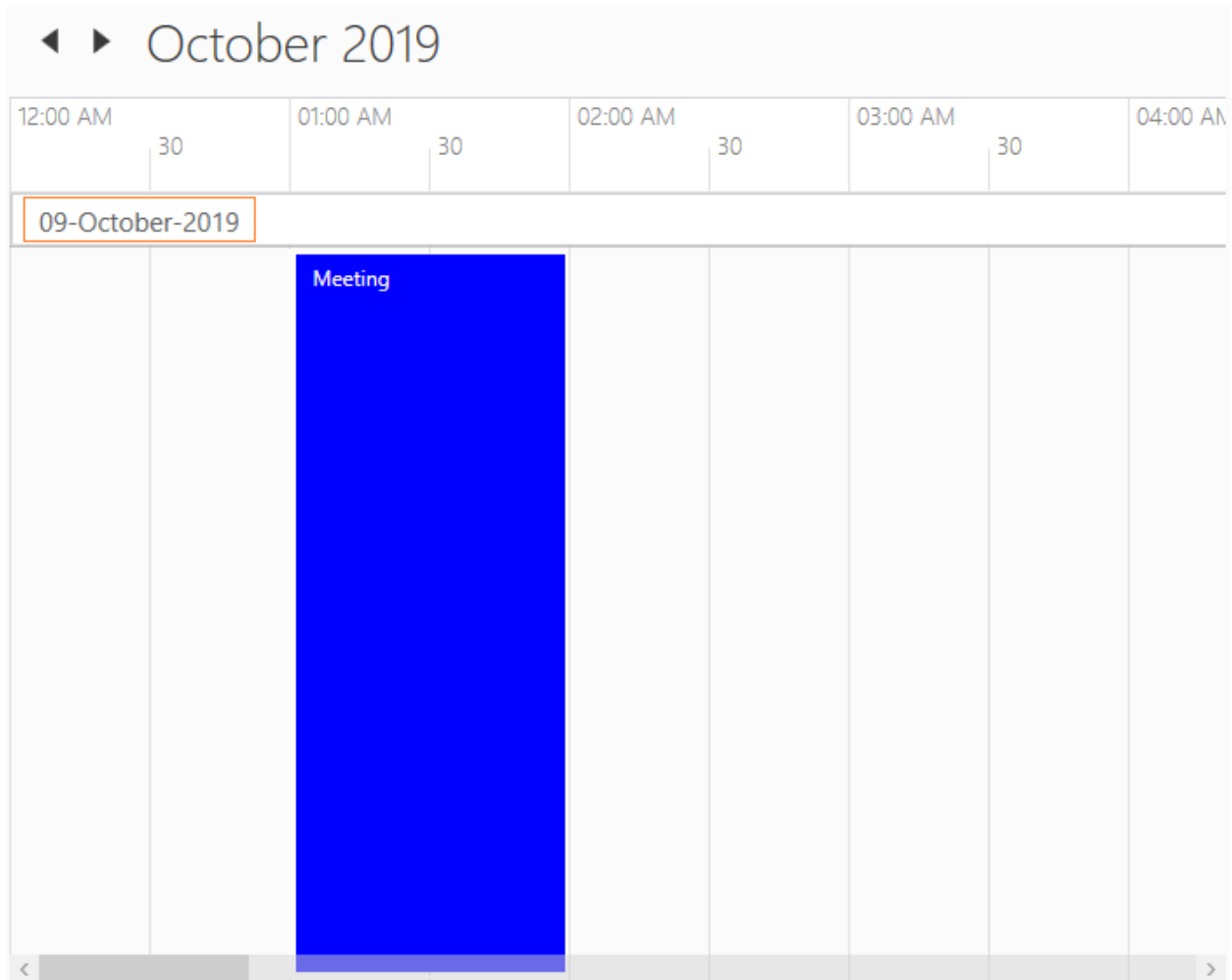
Scheduler supports to customize the header date format of the day, week, workweek and timeline view using [HeaderDateFormat](#) property.

XML

```
<Schedule:SfSchedule HeaderDateFormat="dd-MMMM-yyyy"/>
```

C#

```
this.schedule.HeaderDateFormat = "dd-MMMM-yyyy";
```



Time formatting

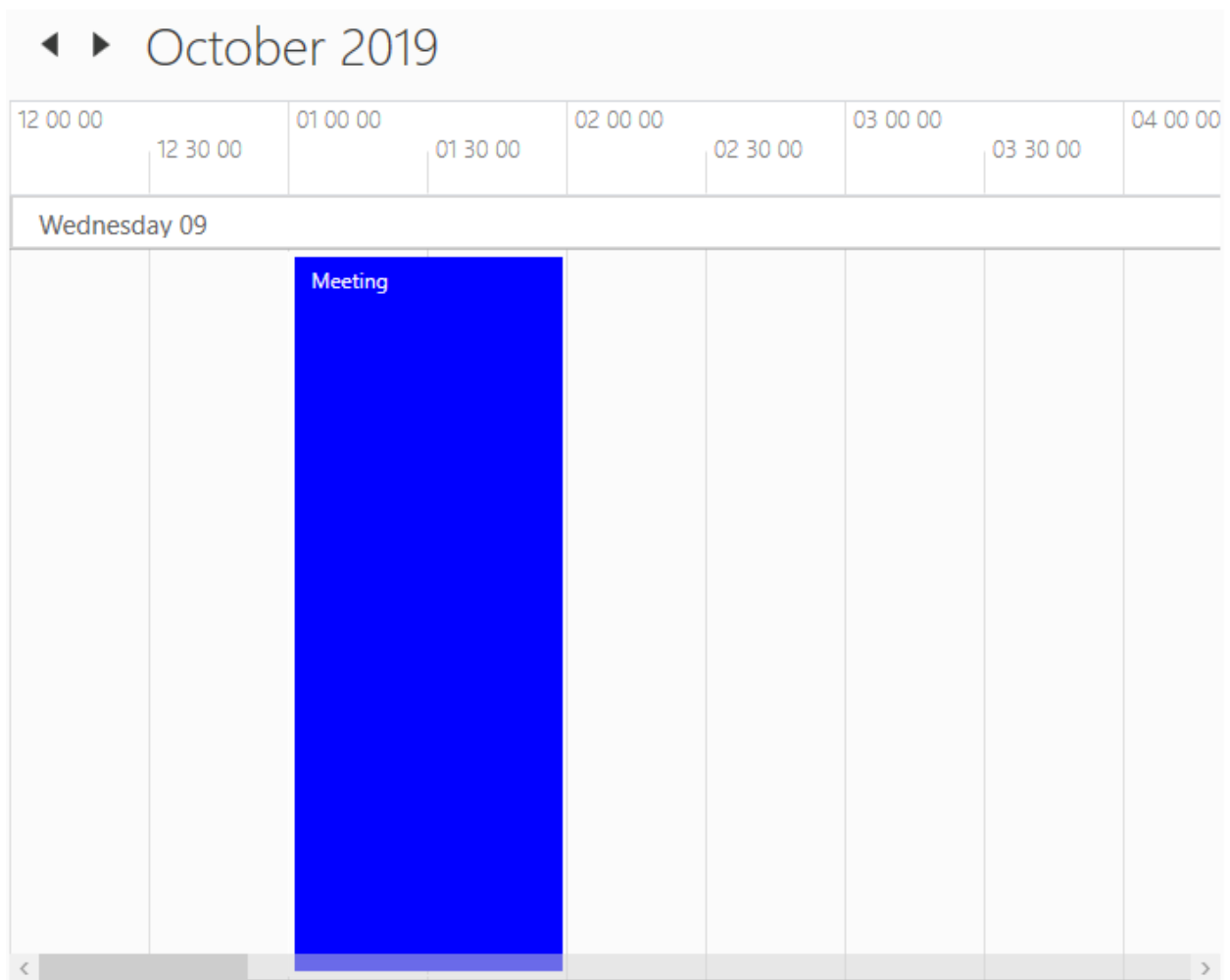
Scheduler supports to customize time format for day, week, workweek and timeline views using [MajorTickTimeFormat](#) and [MinorTickTimeFormat](#) property.

XML

```
<syncfusion:SfSchedule MajorTickTimeFormat="hh mm ss"
MinorTickTimeFormat="mm ss"/>
```

C#

```
this.schedule.MajorTickTimeFormat = "hh mm ss";
this.schedule.MinorTickTimeFormat = "hh mm ss";
```



Change time interval

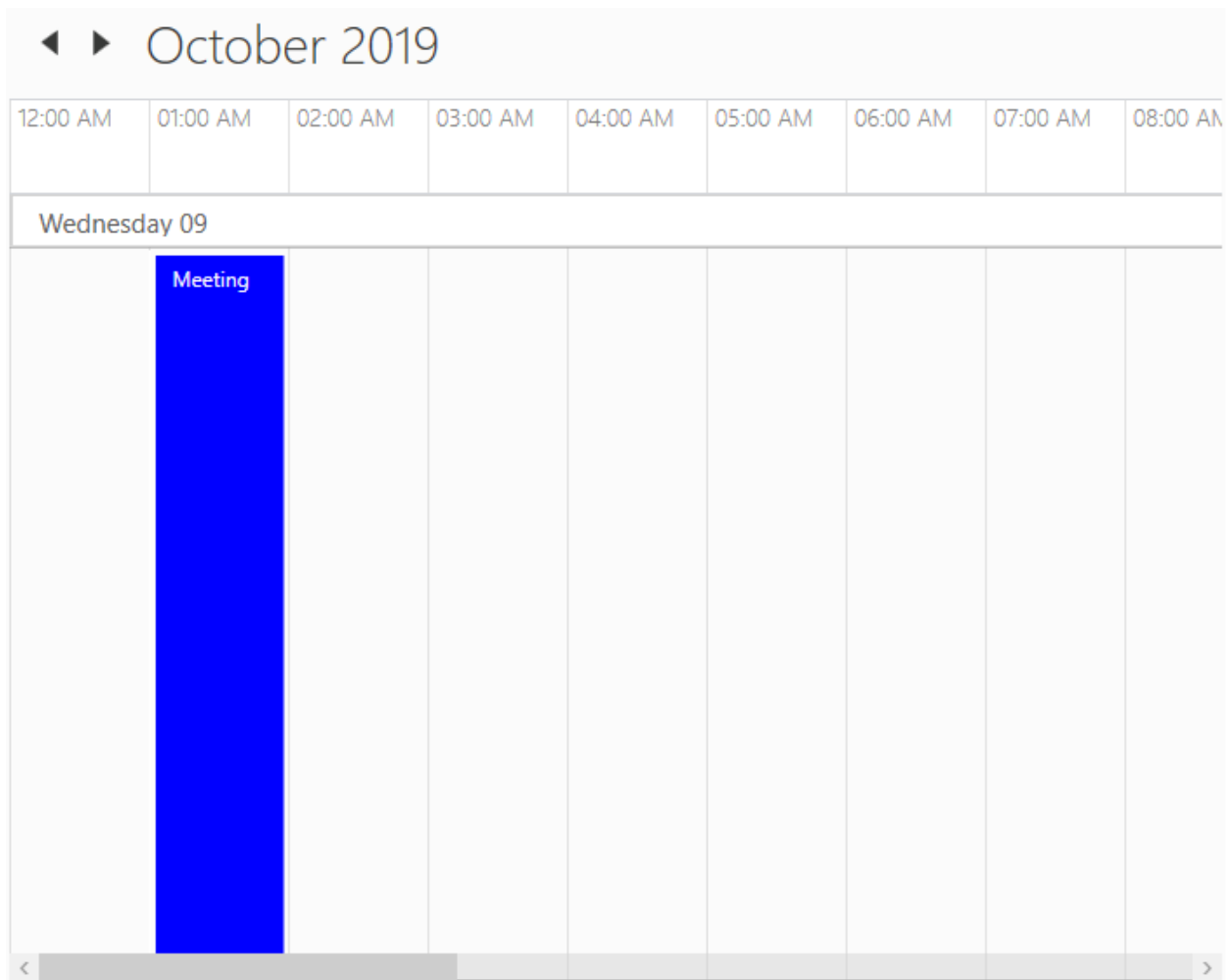
Scheduler supports to change the time interval using [TimeInterval](#) property.

XML

```
<Schedule:SfSchedule TimeInterval = "OneHour" />
```

C#

```
this.schedule.TimeInterval = TimeInterval.OneHour;
```



Change time interval height

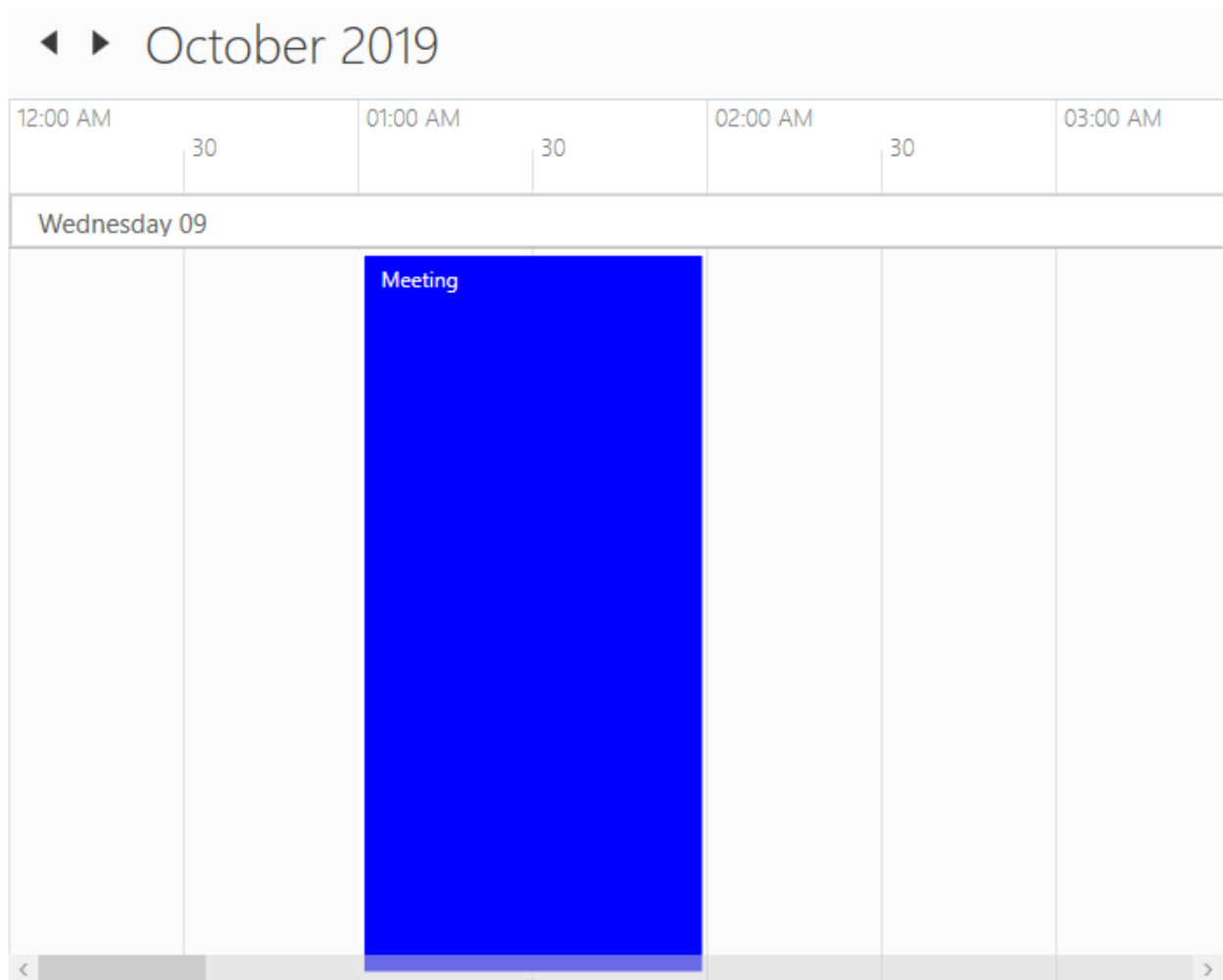
Scheduler supports to change the time interval height using [IntervalHeight](#) property.

XML

```
<Schedule:SfSchedule IntervalHeight = 100 />
```

C#

```
this.schedule.IntervalHeight = 100;
```



Change between 12-hour and 24-hour format

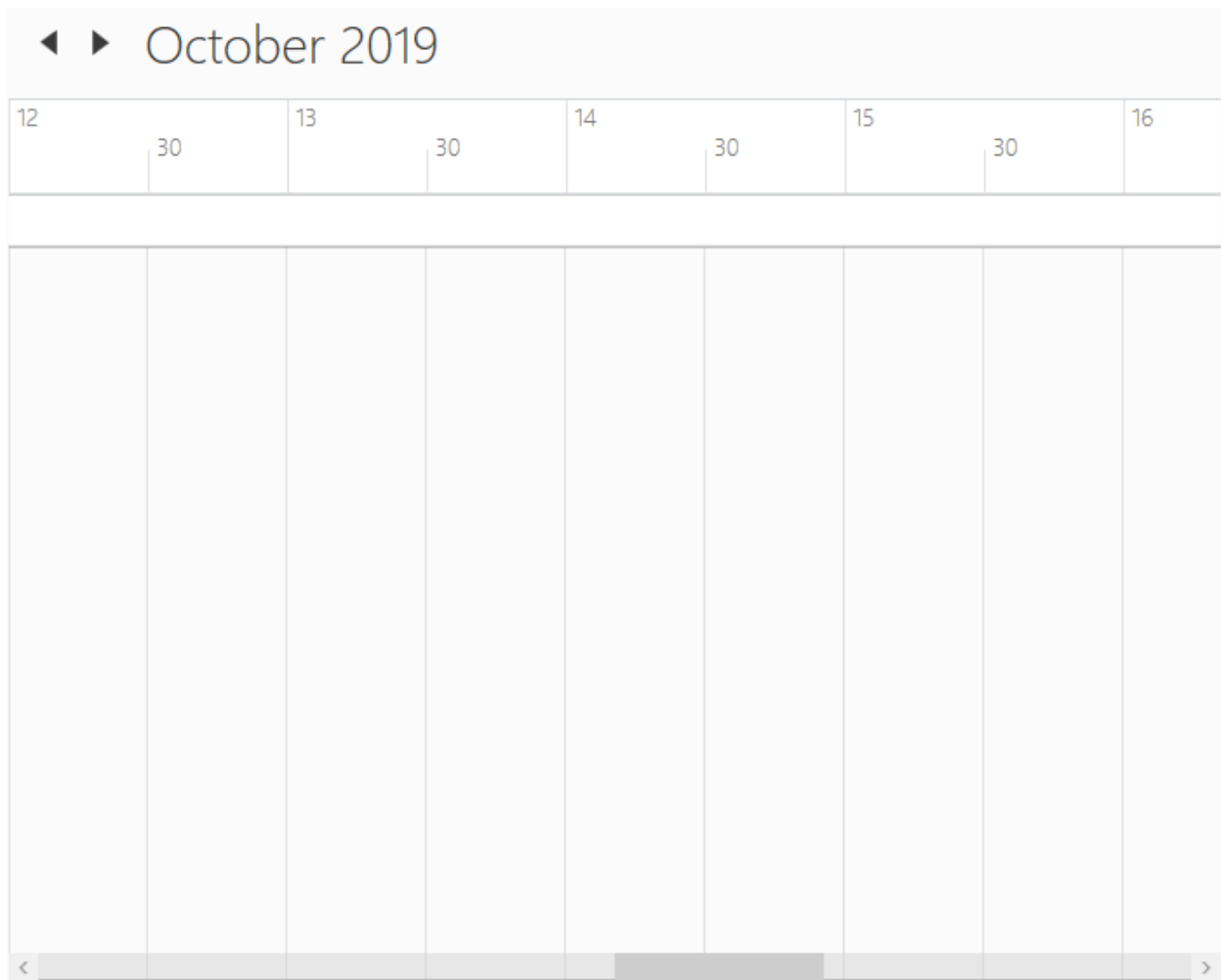
Scheduler supports to change the time format from 12hours to 24 hours using [TimeMode](#) property.

XML

```
<Grid Background="White" Name="grid">
  <Schedule:SfSchedule ScheduleType="TimeLine" TimeMode="TwentyFourHours" >
  </Schedule:SfSchedule>
</Grid>
```

C#

```
schedule.ScheduleType = ScheduleType.TimeLine;
schedule.TimeMode = TimeModes.TwentyFourHours;
```



Non-accessible timeslots

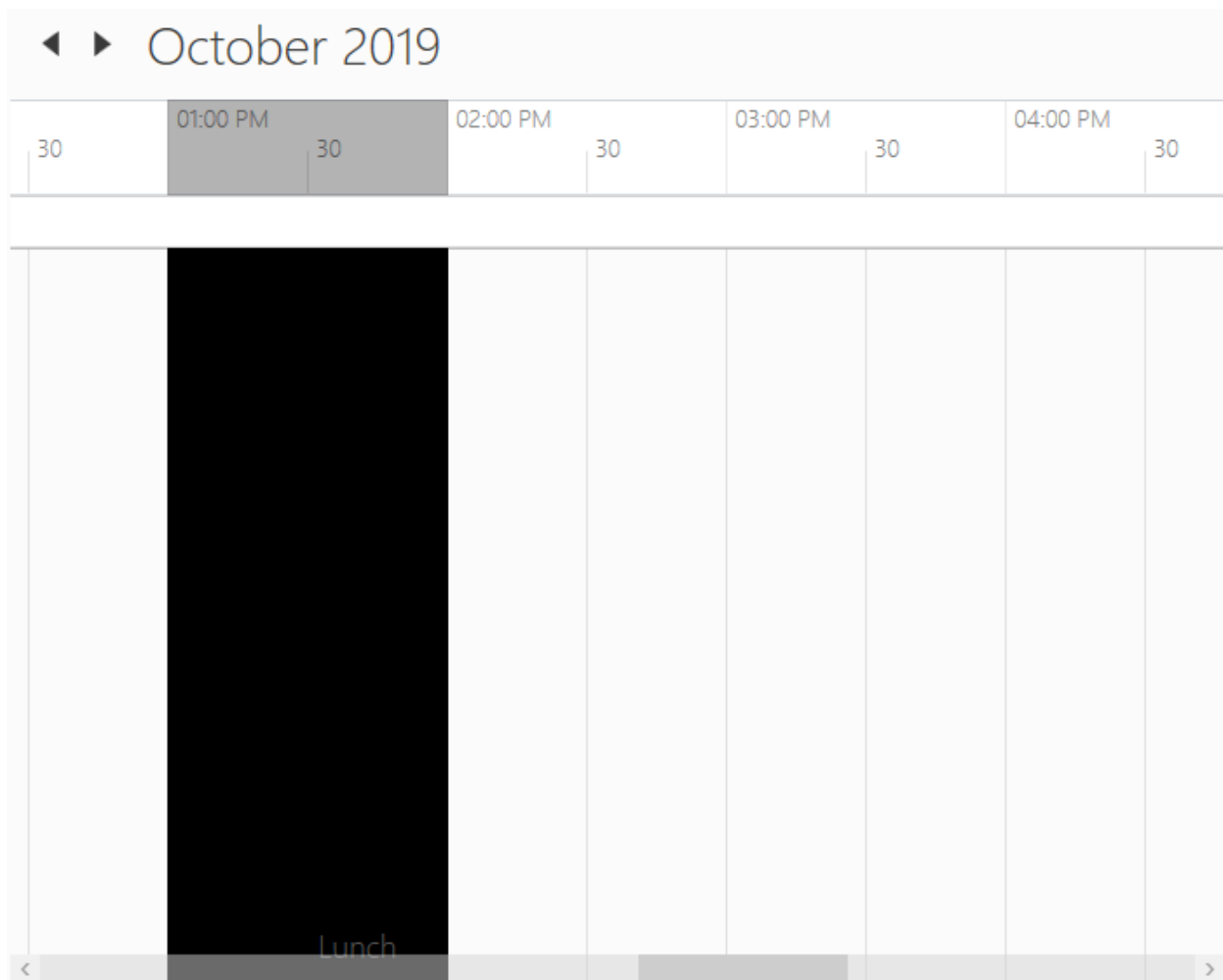
Scheduler supports to mark certain timeslots as non-accessible timeslots using [NonAccessibleBlocks](#) property. User can't interact over the timeslot marked as non-accessible timeslots.

XML

```
<Schedule:SfSchedule>
  <Schedule:SfSchedule.NonAccessibleBlocks>
    <Schedule:NonAccessibleBlock Background="Black" StartHour="13" EndHour="14"
    Label="Lunch">
  </Schedule:NonAccessibleBlock>
  </Schedule:SfSchedule.NonAccessibleBlocks>
</Schedule:SfSchedule>
```

C#

```
this.schedule.NonAccessibleBlocks.Add(new NonAccessibleBlock() { Background
= new SolidColorBrush(Colors.Black), StartHour = 13, EndHour = 14, Label =
"Lunch" });
```

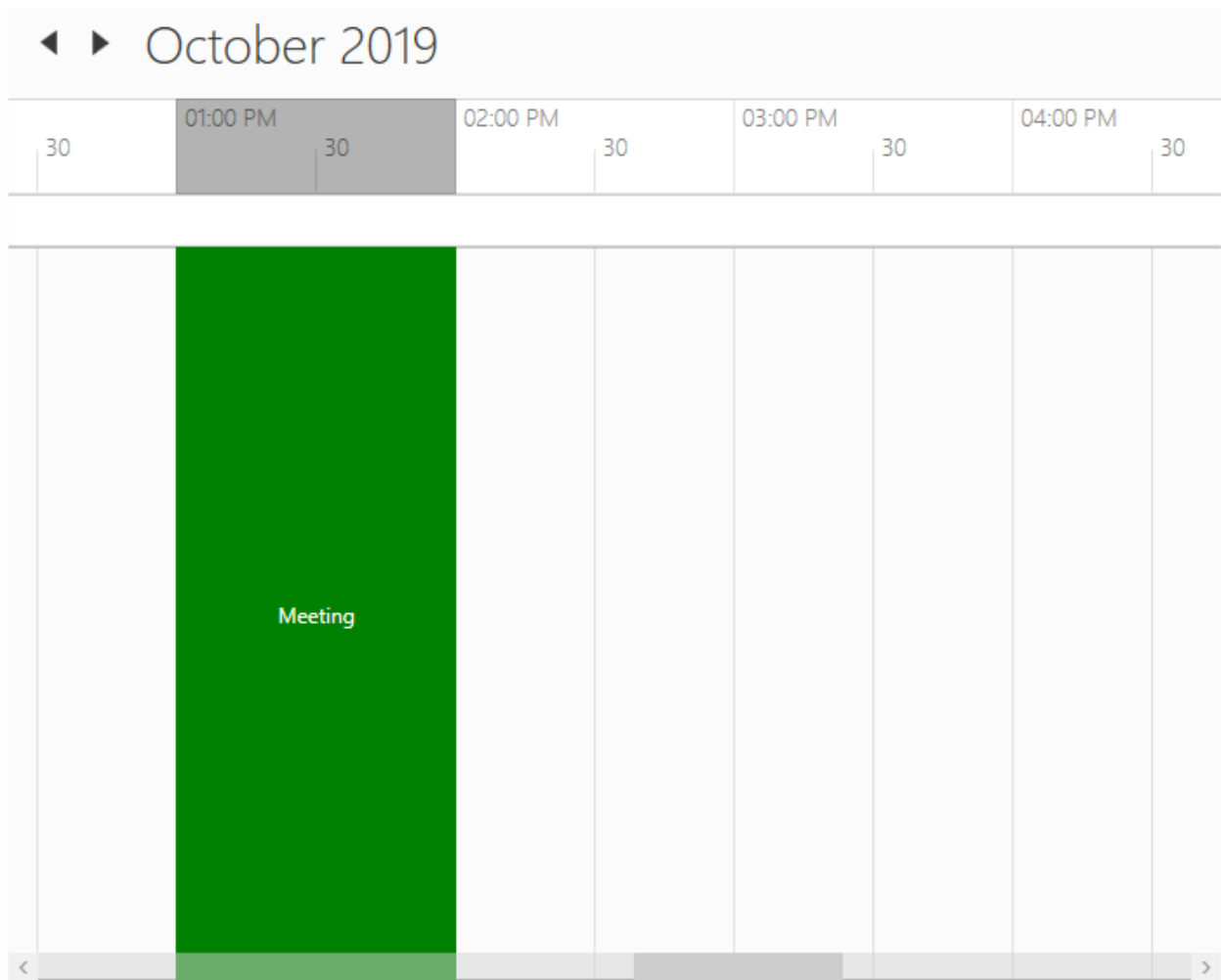


Customize non-accessible timeslots using template

Scheduler supports to customize the non-accessible timeslots using [NonAccessibleBlockTemplate](#) property.

XML

```
<syncfusion:SfSchedule.NonAccessibleBlockTemplate>
  <DataTemplate>
    <Border Background="{Binding Color}">
      <TextBlock Text="{Binding EventName}" Foreground="White"
        HorizontalAlignment="Center" VerticalAlignment="Center"/>
    </Border>
  </DataTemplate>
</syncfusion:SfSchedule.NonAccessibleBlockTemplate>
```

Collapsed hours

Scheduler supports to hide the selected hours by using [CollapsedHours](#) property.

[ScheduleCollapsedHours](#) does have the following properties.

[StartHour](#) - To set start time of collapsed hour.

[EndHour](#) - To set end time of collapsed hour.

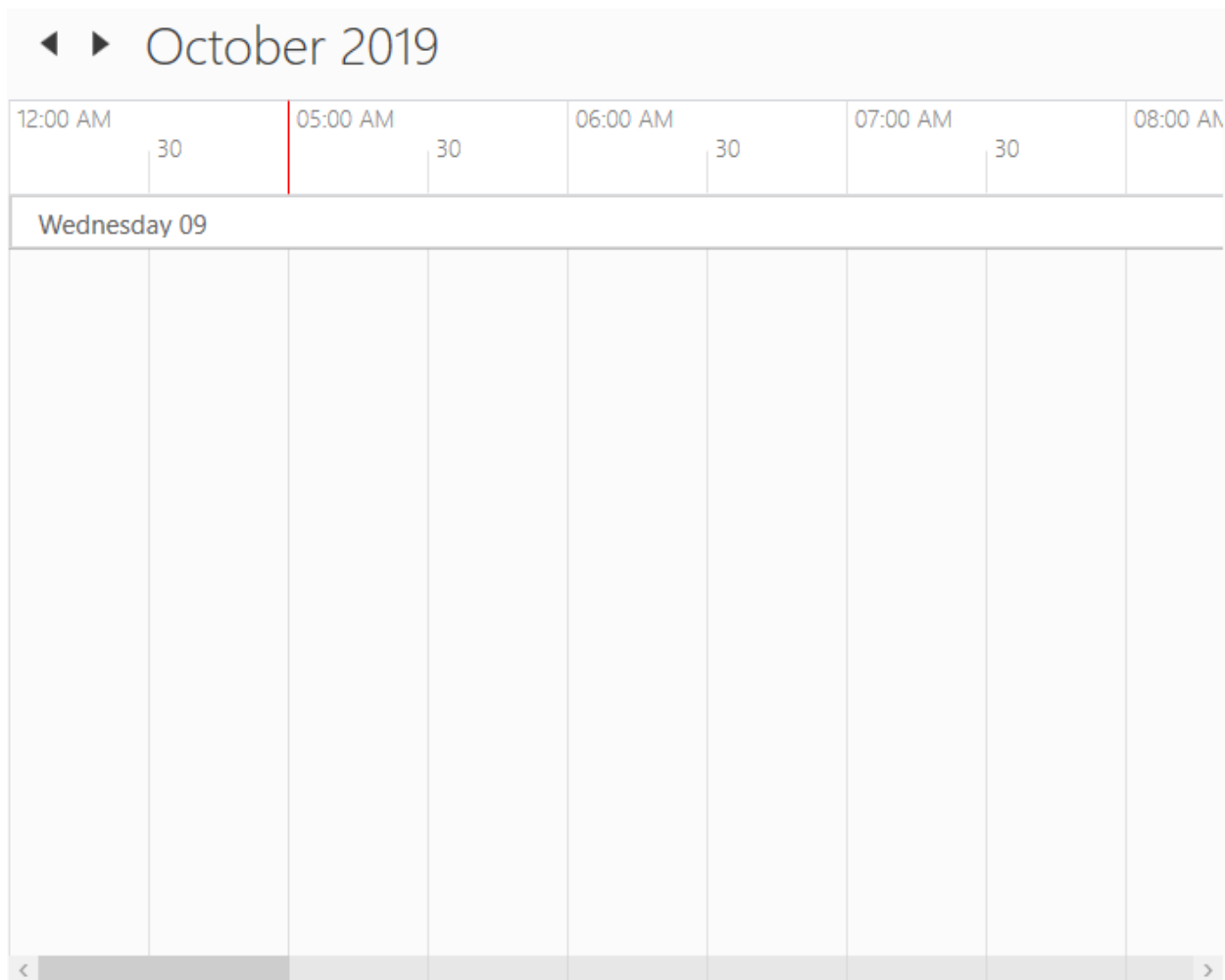
[Background](#) - To set the background of collapsed hours.

XML

```
<schedule:SfSchedule Background="White" x:Name="schedule"
ScheduleType="TimeLine">
<schedule:SfSchedule.CollapsedHours>
<schedule:ScheduleCollapsedHour StartHour="1" EndHour="5"
Background="Red"/>
</schedule:SfSchedule.CollapsedHours>
</schedule:SfSchedule>
```

C#

```
this.schedule.CollapsedHours.Add(new ScheduleCollapsedHour() { StartHour = 1, EndHour = 5, Background = new SolidColorBrush(Colors.Red) });
```



Change working hours

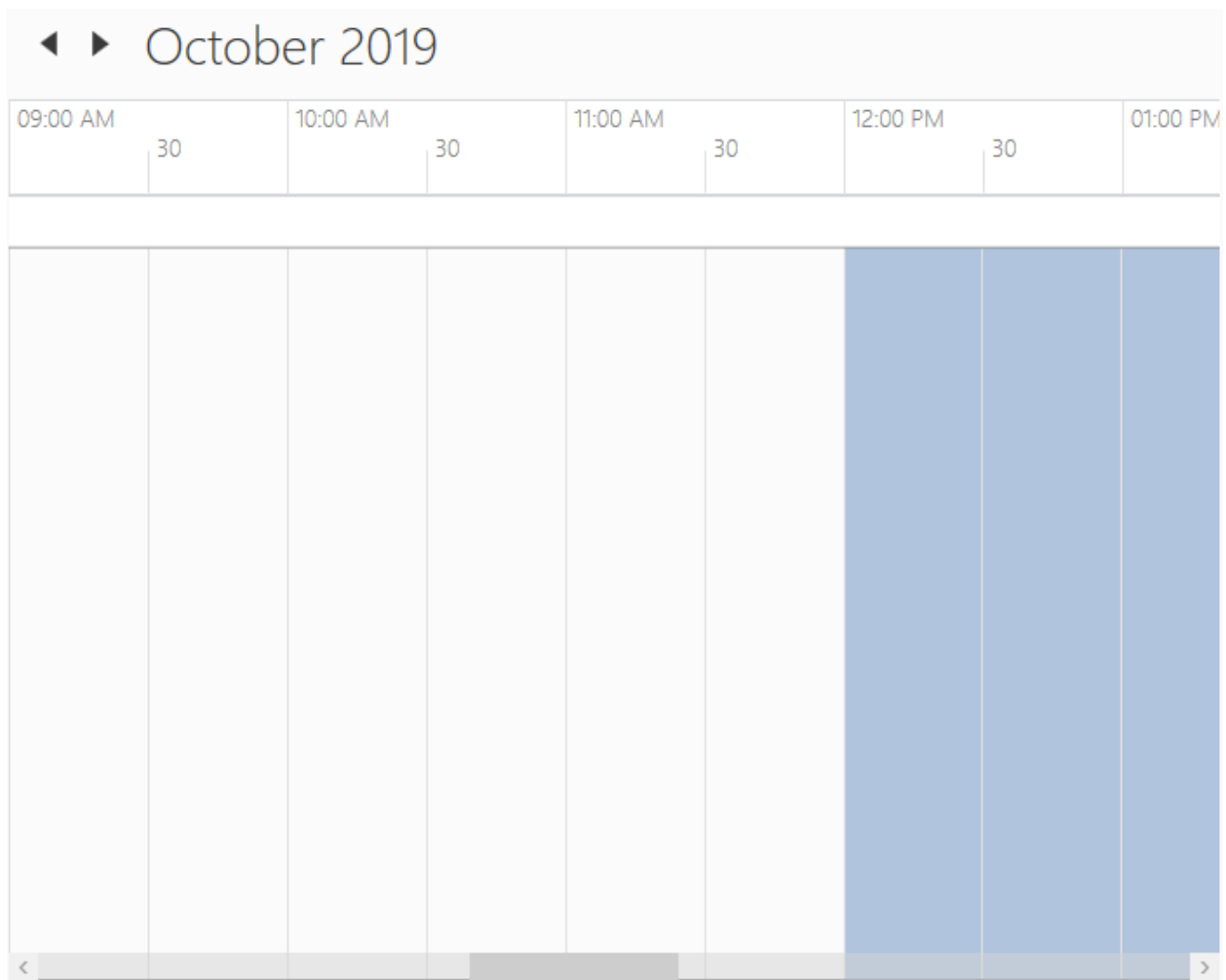
Working hours of Scheduler will be differentiated with non-working hours by separate color using [IsHighLightWorkingHours](#) property for day, week, workweek and timeline views. By default, working hours will be between 09 to 18. You can customize the working hours by setting [WorkStartHour](#) and [WorkEndHour](#) properties.

XML

```
<schedule:SfSchedule x:Name="schedule"
    WorkStartHour="9"
    WorkEndHour="12"
    IsHighLightWorkingHours="True"/>
```

C#

```
this.schedule.WorkStartHour = 9;
this.schedule.WorkEndHour = 12;
this.schedule.IsHighLightWorkingHours = true;
```



Display working hours only

Scheduler supports to display the working hours only by disabling the [ShowNonWorkingHours](#) property.

XML

```
<schedule:SfSchedule x:Name="schedule"
    WorkStartHour="9"
    WorkEndHour="12"
    ShowNonWorkingHours="False"/>
```

C#

```
this.schedule.WorkStartHour = 9;
this.schedule.WorkEndHour = 12;
this.schedule.ShowNonWorkingHours = false;
```

Change non-working hours background

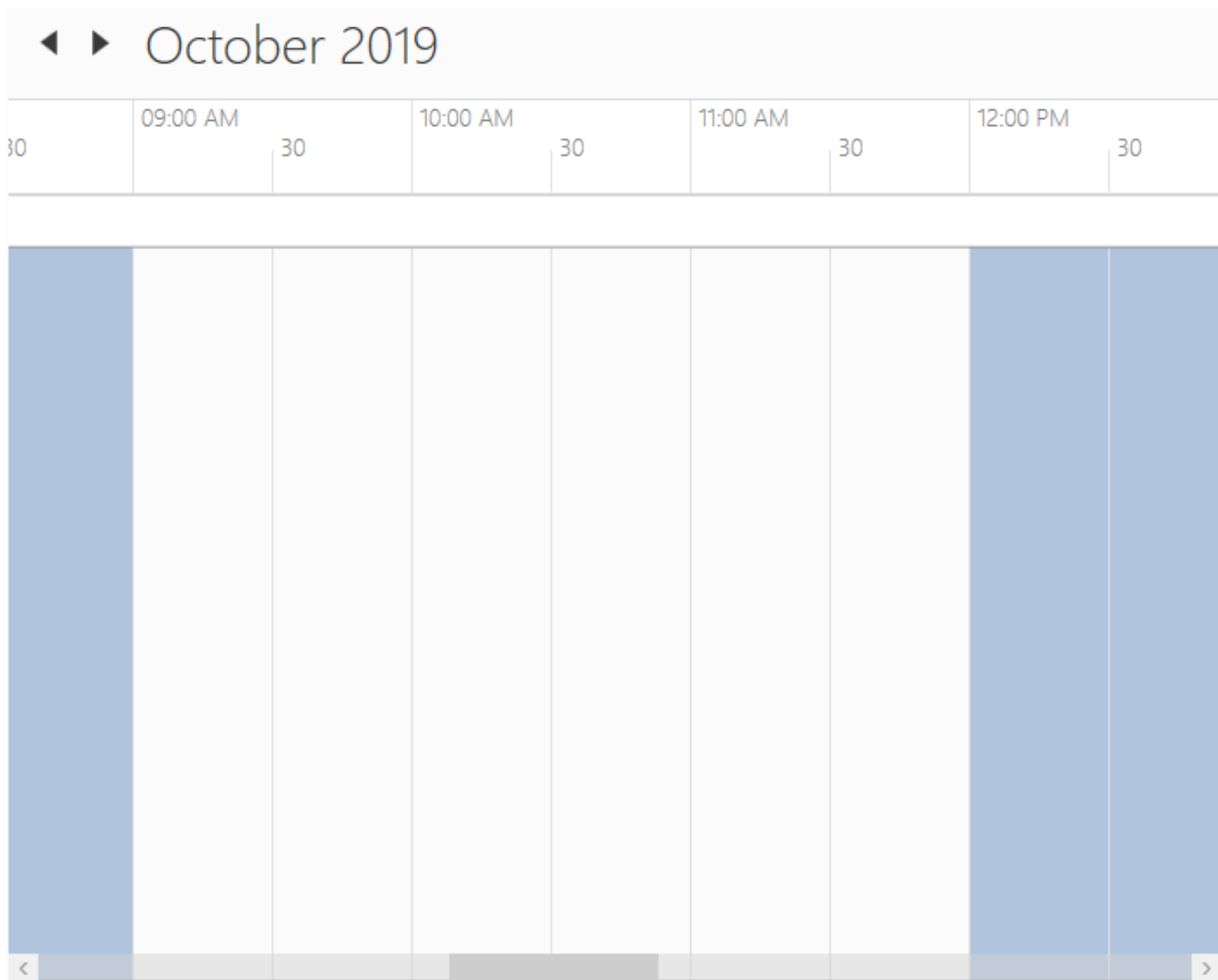
Scheduler supports to change the background color for non-working hours using [NonWorkingHourBrush](#) property.

XML

```
<schedule:SfSchedule x:Name="schedule"
WorkStartHour="9"
WorkEndHour="12"
NonWorkingHourBrush="LightSteelBlue"
IsHighLightWorkingHours="True"/>
```

C#

```
this.schedule.WorkStartHour = 9;
this.schedule.WorkEndHour = 12;
this.schedule.NonWorkingHourBrush = Brushes.LightSteelBlue;
this.schedule.IsHighLightWorkingHours = true;
```



Current time indicator

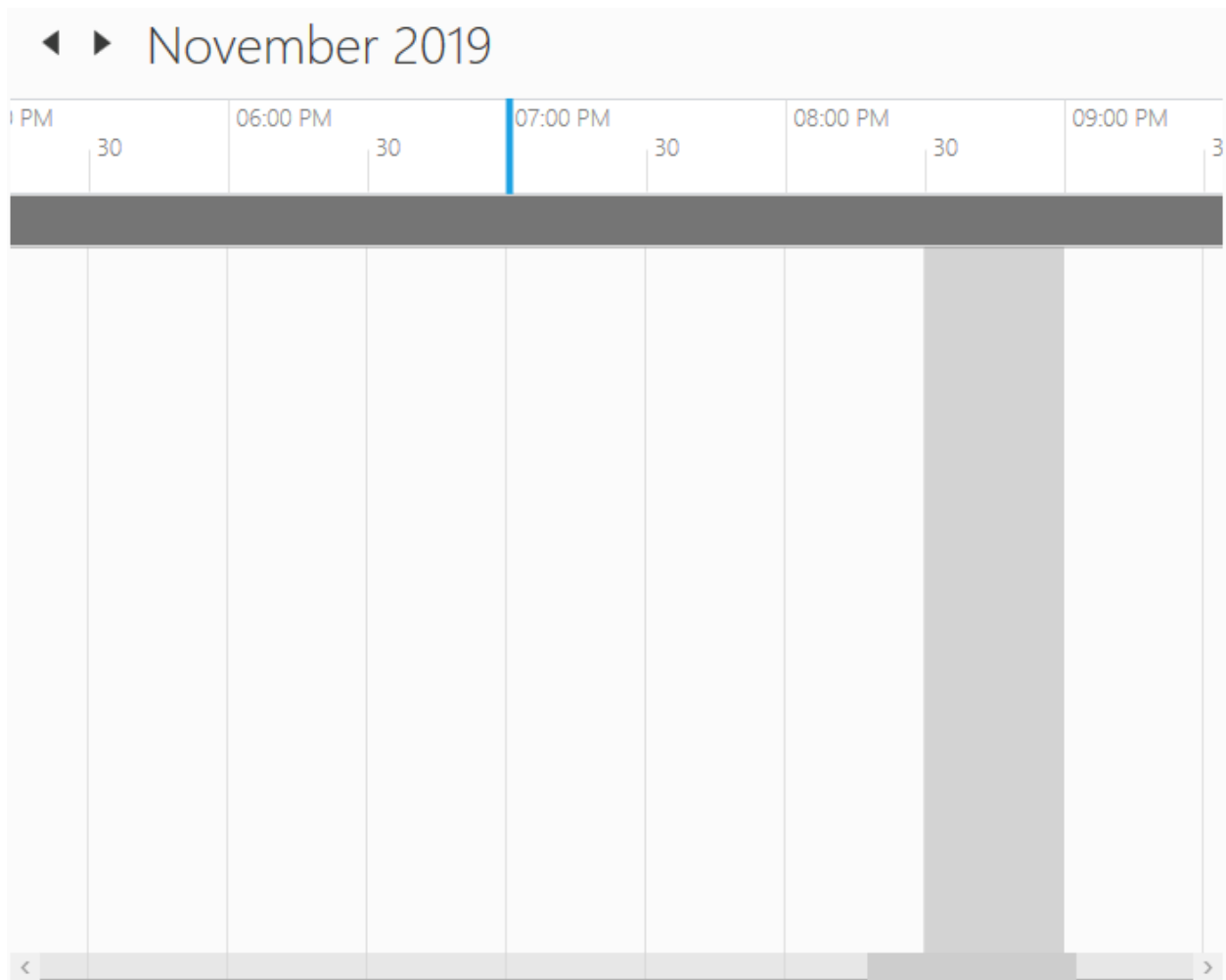
Scheduler supports to display the current time indicator using the [CurrentTimeIndicatorVisibility](#) property.

XML

```
<syncfusion:SfSchedule ScheduleType="TimeLine"
    CurrentTimeIndicatorVisibility="Visible"/>
```

C#

```
schedule.ScheduleType = ScheduleType.TimeLine;
this.schedule.CurrentTimeIndicatorVisibility = Visibility.Visible;
```



Customize current time indicator

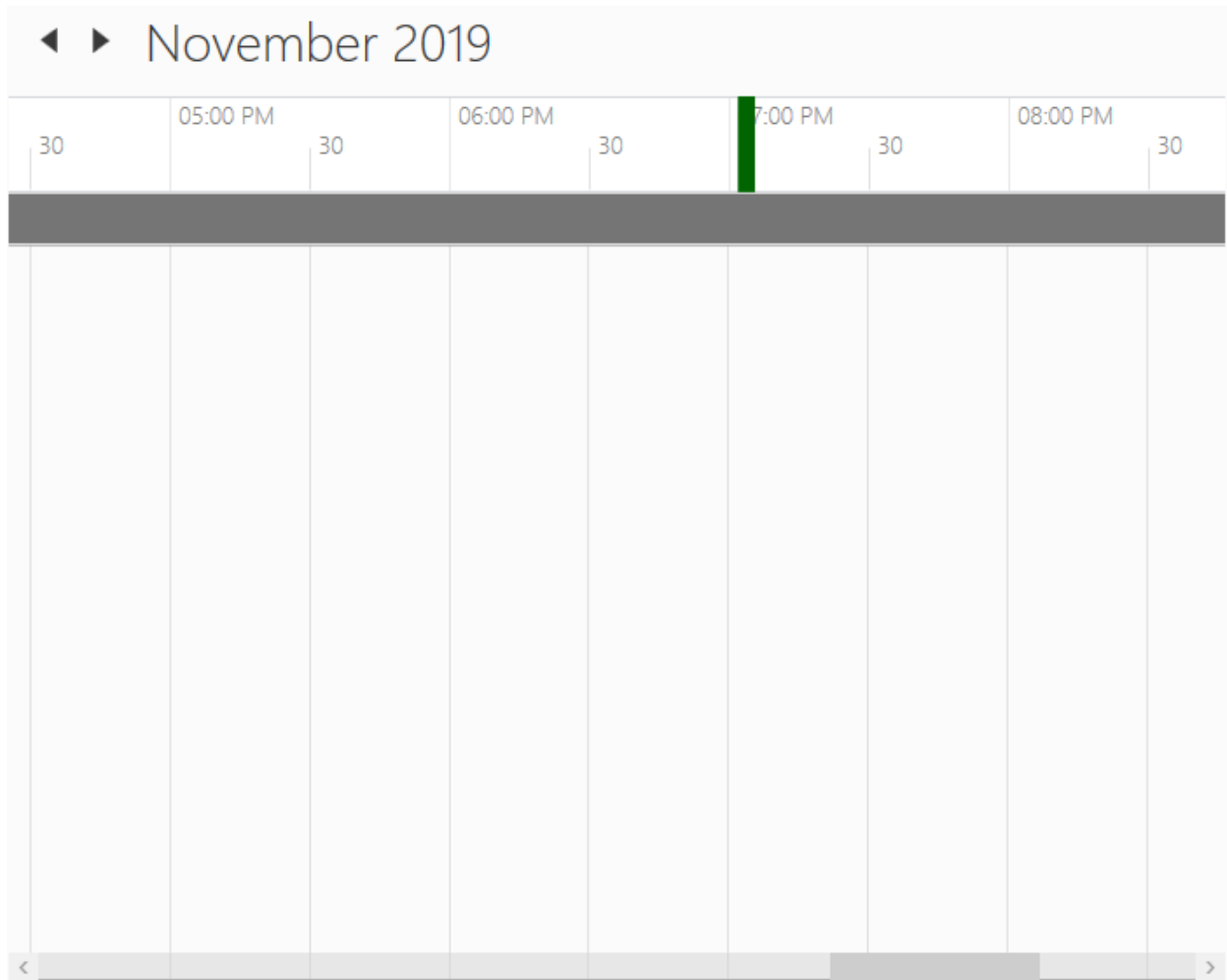
Scheduler supports to customize the current time indicator using [CurrentTimeIndicatorTemplate](#) property.

XML

```
<Schedule:SfSchedule x:Name="schedule" ScheduleType="TimeLine"
    CurrentTimeIndicatorVisibility="Visible">
    <Schedule:SfSchedule.CurrentTimeIndicatorTemplate>
    <DataTemplate>
    <Border Background="DarkGreen" Height="10" Width="100"></Border>
    </DataTemplate>
    </Schedule:SfSchedule.CurrentTimeIndicatorTemplate>
</Schedule:SfSchedule>
```

C#

```
schedule.ScheduleType = ScheduleType.TimeLine;
schedule.CurrentTimeIndicatorVisibility = Visibility.Visible;
schedule.CurrentTimeIndicatorTemplate =
    (DataTemplate) this.Resources["CurrentTimeIndicatorTemplate"];
```



Change hours or minutes time label visibility

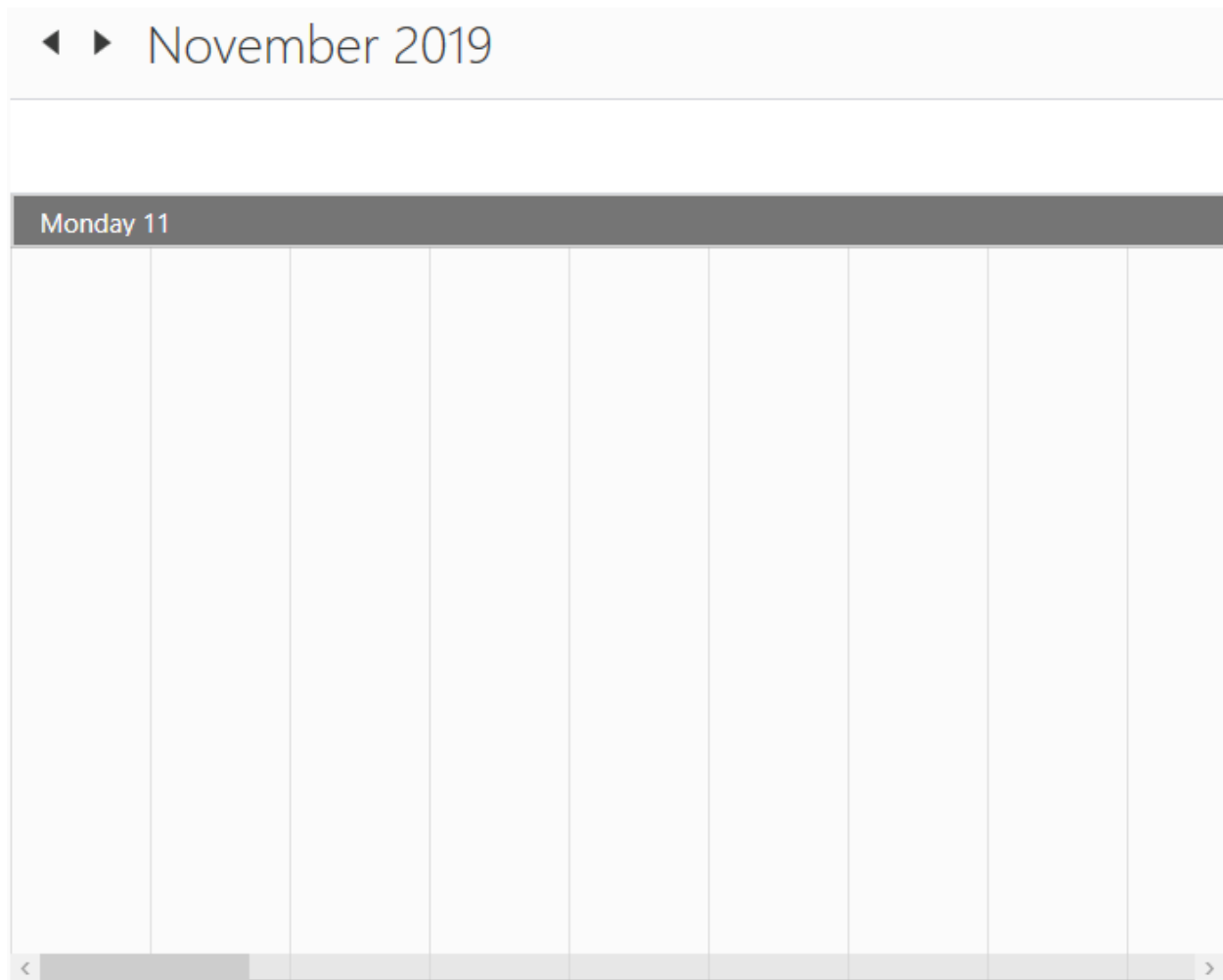
Scheduler supports to visible/collapse the hours and minutes time label visibility using [MajorTickVisibility](#) and [MinorTickVisibility](#) properties.

XML

```
<Schedule:SfSchedule ScheduleType="TimeLine" MajorTickVisibility="Collapsed"
MinorTickVisibility="Collapsed" />
```

C#

```
this.schedule.ScheduleType = ScheduleType.TimeLine;
this.schedule.MajorTickVisibility = Visibility.Collapsed;
this.schedule.MinorTickVisibility = Visibility.Collapsed;
```



Appearance

Changing time label background

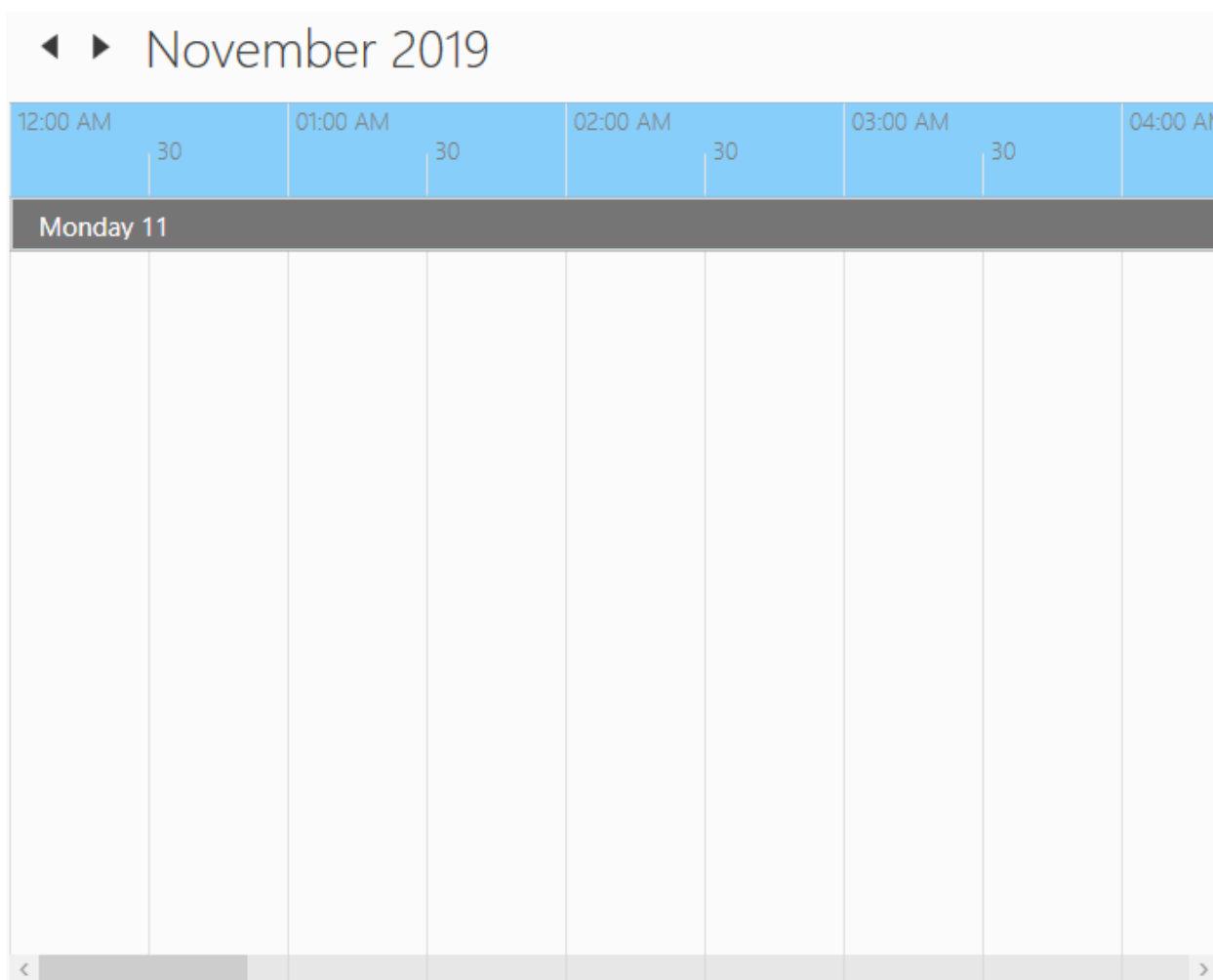
Scheduler supports to change the time slot label background using [HeaderBackground](#) property.

XML

```
<Schedule:SfSchedule HeaderBackground="LightSkyBlue" />
```

C#

```
this.schedule.HeaderBackground = Brushes.LightSkyBlue;
```

Stroke customization

In Scheduler control, major, minor horizontal and vertical lines drawn in the day, week, workweek and timeline views by using following properties,

Property Table

API Name	Data Type	Description
MajorTickStroke	Brush	Used to customize the major line stroke of the day, week, workweek and timeline views.
MinorTickStroke	Brush	Used to customize the minor line stroke of the day, week, workweek and timeline views.
MajorTickLabelStroke	Brush	Used to customize the major line label stroke in the day, week, workweek and timeline views.
MinorTickLabelStroke	Brush	Used to customize the minor line label stroke of the day, week, workweek and timeline views.
MajorTickStrokeDashArray	DoubleCollection	Used to customize the major line stroke dash array of the day, week, workweek and timeline views.

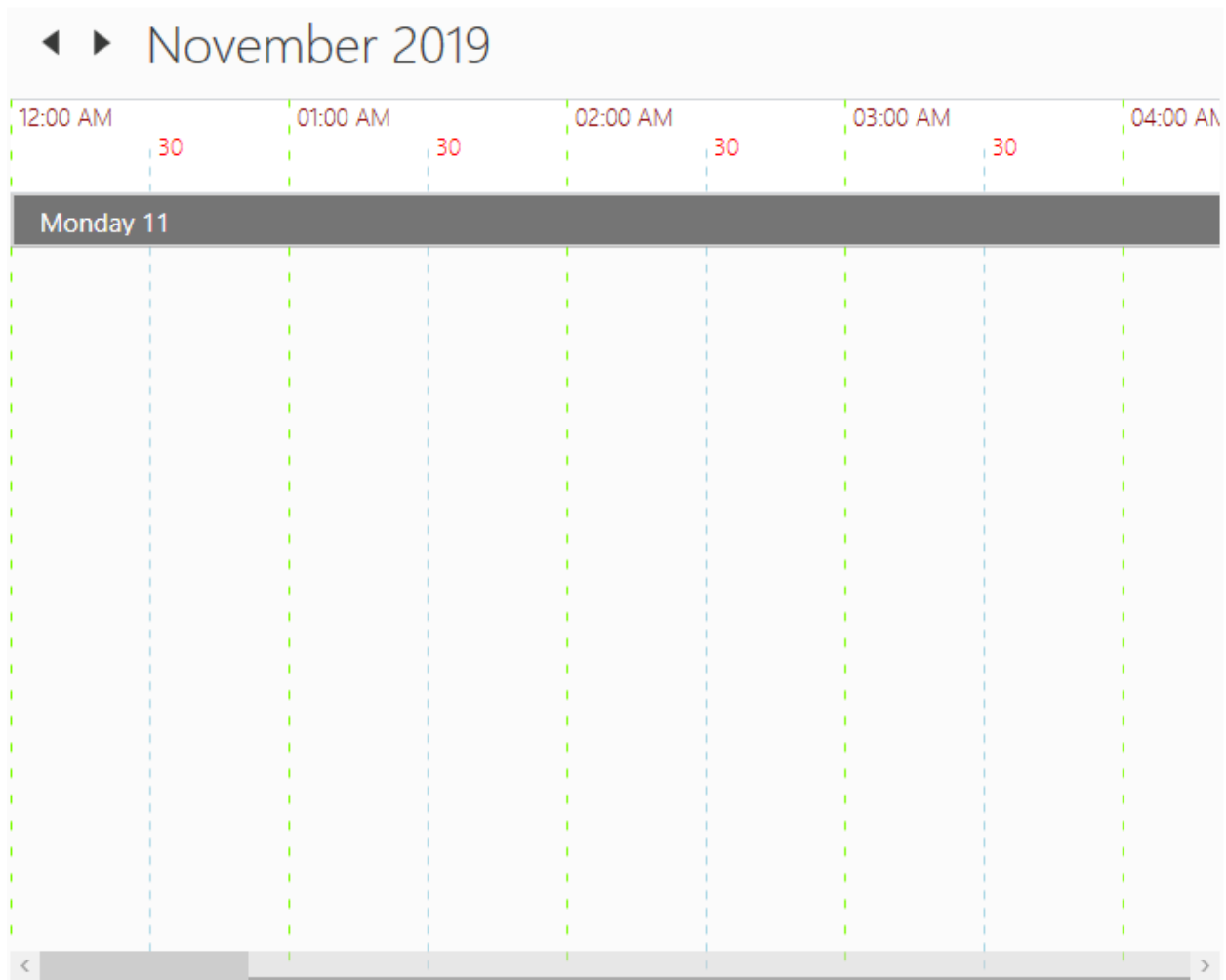
MinorTickStrokeDashArray	DoubleCollection	Used to customize the minor line stroke dash array of the day, week, workweek and timeline views.
DayViewVerticalLineStroke	Brush	Used to customize the vertical line stroke of the day, week and workweek view.

XML

```
<syncfusion:SfSchedule MajorTickLabelStroke="DarkRed"
MinorTickLabelStroke="Red"
MajorTickStroke="LawnGreen"
MinorTickStroke="LightBlue"
MajorTickStrokeDashArray="5,10"
MinorTickStrokeDashArray="5,5"
DayViewVerticalLineStroke="Brown"/>
```

C#

```
this.schedule.MajorTickLabelStroke = Brushes.DarkRed;
this.schedule.MinorTickLabelStroke = Brushes.Red;
this.schedule.MajorTickStroke = Brushes.LawnGreen;
this.schedule.MinorTickStroke = Brushes.LightBlue;
this.schedule.MajorTickStrokeDashArray = new DoubleCollection(new Double[] {
5, 10 });
this.schedule.MinorTickStrokeDashArray = new DoubleCollection(new Double[] {
5, 5 });
this.schedule.DayViewVerticalLineStroke = Brushes.Brown;
```



Current day highlighting

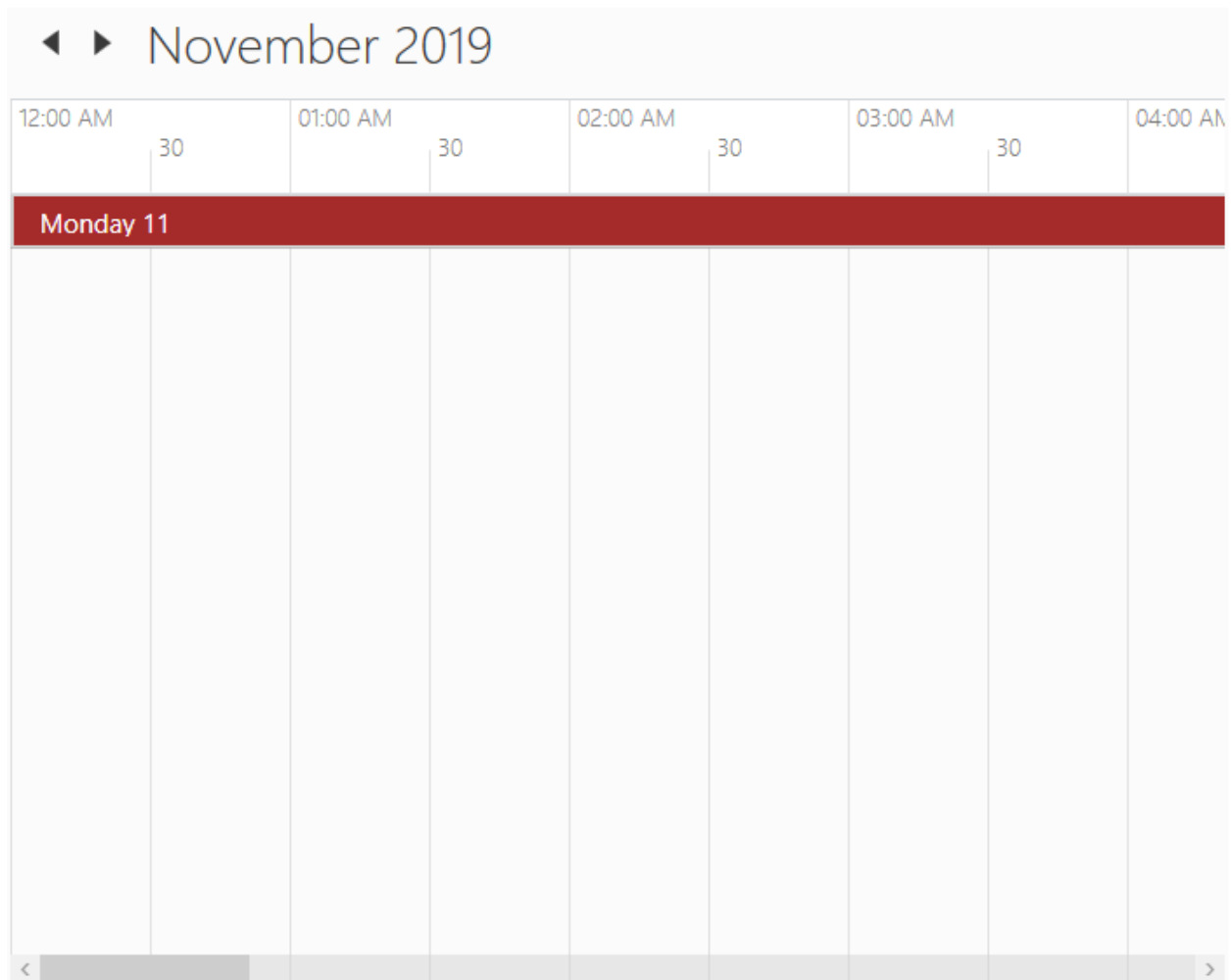
Scheduler supports to change current day background and foreground for all views using [CurrentDateBackground](#) and [CurrentDateForeground](#) property.

XML

```
<syncfusion:SfSchedule CurrentDateBackground="Brown"
CurrentDateForeground="White"/>
```

C#

```
this.schedule.CurrentDateBackground = Brushes.Brown;
this.schedule.CurrentDateForeground = Brushes.White;
```



Appointments in WPF Schedule (Classic)

Scheduler control displays appointments based on details such as `StartTime`, `EndTime`, `Subject` and etc in `ScheduleAppointment` class. Also, it support to bind any collection and allows to render appointments based on data object in collection using mapping concept which explained under data binding section.

Adding appointment

Scheduler support to add the `ScheduleAppointment` by using `Appointments` property. Schedule supports to render all day appointments, spanned appointment and recurring appointments.

XML

```
<syncfusion:SfSchedule.Appointments>
<syncfusion:ScheduleAppointment StartTime="05/08/2017 10:0:0"
EndTime="05/08/2017 11:0:0" Subject="Meeting" Location="Hutchison road"/>
</syncfusion:SfSchedule.Appointments>
```

C#

```
// Creating an instance for schedule appointment collection
ScheduleAppointmentCollection scheduleAppointmentCollection = new
ScheduleAppointmentCollection();
```

```
//Adding schedule appointment in schedule appointment collection
scheduleAppointmentCollection.Add(new ScheduleAppointment()
{
    StartTime = new DateTime(2017, 05, 08, 10, 0, 0),
    EndTime = new DateTime(2017, 05, 08, 11, 0, 0),
    Subject = "Meeting",
    Location = "Hutchison road",
});
//Adding schedule appointment collection to Appointments of SfSchedule
this.Schedule.Appointments = scheduleAppointmentCollection;
```

◀ ▶ May 2017						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
30	01	02	03	04	05	06
07	08 Meeting	09	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	01	02	03

AppointmentCollectionChanged

Scheduler notifies changes of **Appointments** collection by [AppointmentCollectionChanged](#) event.

This event triggers with default [NotifyCollectionChangedEventArgs](#).

C#

```
this.schedule.AppointmentCollectionChanged +=
Schedule_AppointmentCollectionChanged;
private void Schedule_AppointmentCollectionChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
    //To notify whenever make the changes in Appointments collection.
}
```

Data Binding

Scheduler supports to bind any collection that implements the `IEnumerable` interface to populate appointments. You can map properties in data object to `ScheduleAppointment` by configuring the [AppointmentMapping](#) property. Below table which property shows mapping property details to `ScheduleAppointment`.

Property Name	Description
StartTimeMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's StartTime .
StartTimeZoneMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's StartTimeZone .
EndTimeMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's EndTime .
EndTimeZoneMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's EndTimeZone .
SubjectMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's Subject .
AppointmentBackgroundMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's AppointmentBackground .
AllDayMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's AllDay . Refer the following link to know more about the <code>AllDay</code>
RecurrenceRuleMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's RecurrenceRule . Refer the following link to know more about the <code>RecurrenceRule</code>
RecurrenceTypeMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's RecurrenceType .
RecurrenceProperitesMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's RecurrenceProperites .
RecursiveExceptionDatesMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's RecursiveExceptionDates . Refer the following link to know more about the <code>RecursiveExceptionDates</code>
ReminderTimeMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's ReminderTime . Refer the following link to know more about the <code>ReminderTime</code>
IsRecursiveMapping	This property is intended to map the custom class property name that is identical to <code>ScheduleAppointment</code> 's IsRecursive .

NotesMapping	This property is intended to map the custom class property name that is identical to ScheduleAppointment's Notes .
LocationMapping	This property is intended to map the custom class property name that is identical to ScheduleAppointment's Location .
StatusMapping	This property is intended to map the custom class property name that is identical to ScheduleAppointment's Status .
ResourceCollectionMapping	This property is intended to map the custom class property name that is identical to ScheduleAppointment's ResourceCollection .
ResourceNameMapping	This property is intended to map the custom class property name that is identical to ScheduleAppointment's ResourceName .
DisplayNameMapping	This property is intended to map the custom class property name that is identical to ScheduleAppointment's DisplayName .
TypeNameMapping	This property is intended to map the custom class property name that is identical to ScheduleAppointment's TypeName .
ReadOnlyMapping	This property is intended to map the custom class property name that is identical to ScheduleAppointment's ReadOnly .

Note: Custom appointment class should contain two DateTime fields and a string field as mandatory.

Creating data object

Scheduler supports to create an appointment by using custom object. Appointments can be generated by creating a custom class `Event` with mandatory fields `From`, `To` and `EventName`.

C#

```

/// <summary>
/// Represents custom data properties.
/// </summary>
public class Event
{
    public string EventName { get; set; }
    public DateTime From { get; set; }
    public DateTime To { get; set; }
}

```

Note: You can inherit this class from `INotifyPropertyChanged` for dynamic changes in custom data.

You can map those properties of `Event` class with our Scheduler control by using [AppointmentMapping](#) and bind the mapping collection with scheduler control using [ItemSource](#) property.

XML

```

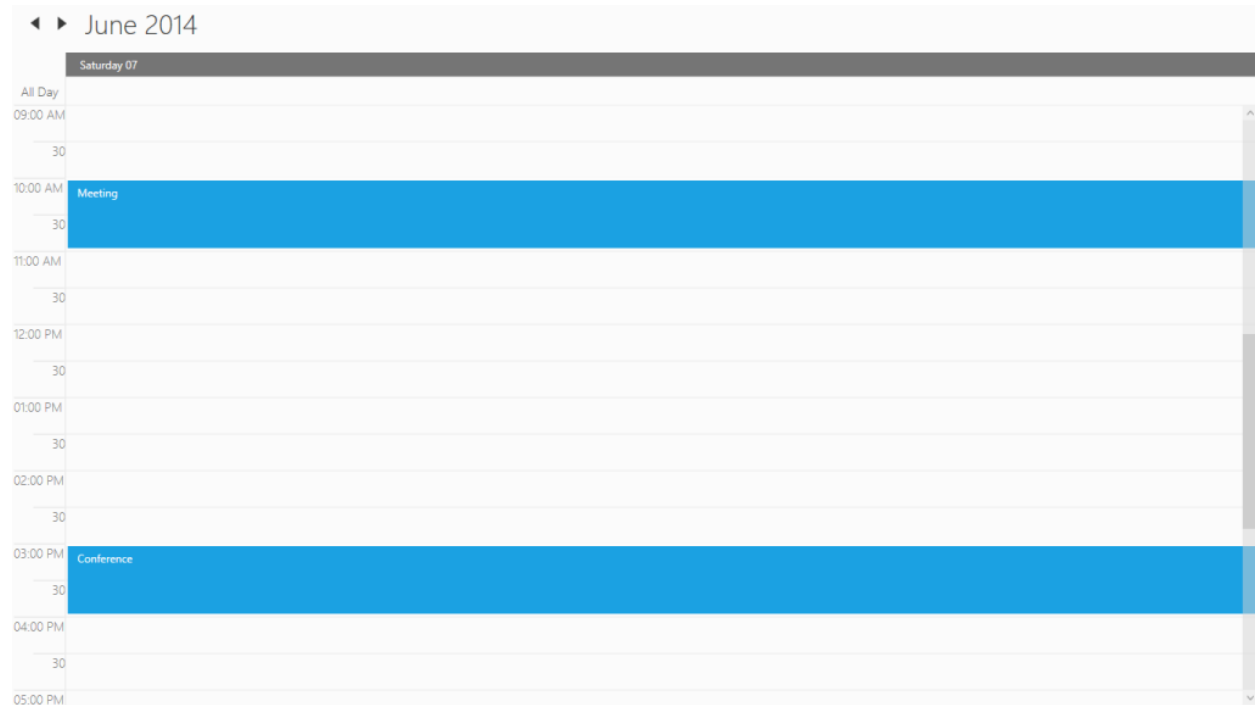
<Window x:Class="SfSch eduleWpf.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:schedule="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525"

```

```
WindowState="Maximized">
<Grid Name="grid">
<Schedule:SfSchedule Name="schedule" ScheduleType="Day"
ItemsSource="{Binding Events}">
<Schedule:SfSchedule.AppointmentMapping>
<Schedule:ScheduleAppointmentMapping
SubjectMapping="EventName"
StartTimeMapping="From"
EndTimeMapping="To"/>
</Schedule:SfSchedule.AppointmentMapping>
</Schedule:SfSchedule>
</Grid>
</Window>
```

C#

```
public partial class MainWindow : Window
{
    public ObservableCollection<Event> Events { get; set; }
    public MainWindow()
    {
        InitializeComponent();
        Events = new ObservableCollection<Event>
        {
            new Event{EventName = "Meeting", From = DateTime.Now.Date.AddHours(10), To =
            DateTime.Now.Date.AddHours(11)},
            new Event{EventName = "Conference", From = DateTime.Now.Date.AddHours(15),
            To = DateTime.Now.Date.AddHours(16)},
        };
        this.DataContext = this;
    }
}
```

ItemSourceChanged event

Scheduler notifies changes to the `ItemSource` by `ItemSourceChanged` event in custom binding.

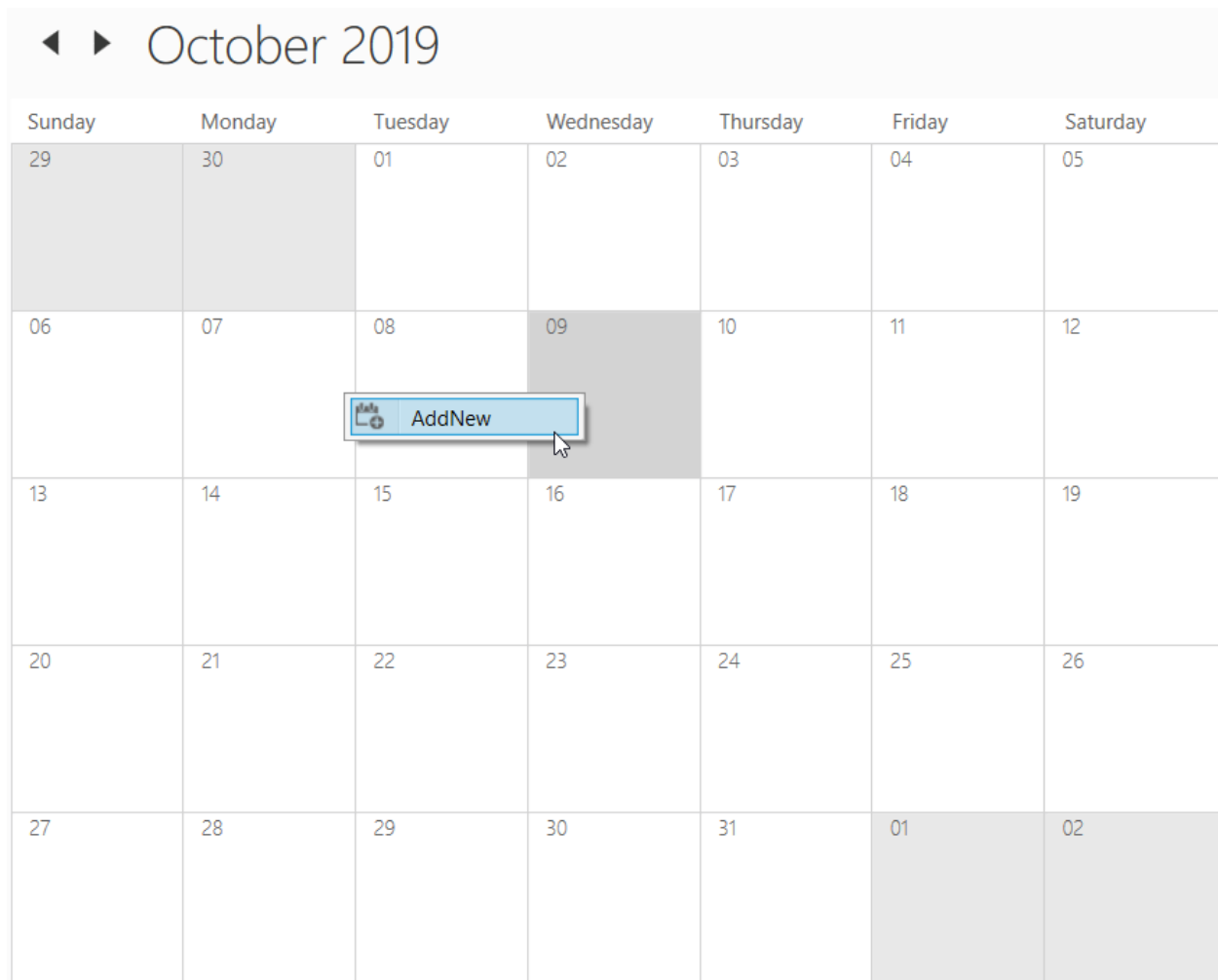
C#

```
this.schedule.ItemsSourceChanged += Schedule_ItemsSourceChanged;  
private void Schedule_ItemsSourceChanged(object sender, EventArgs e)  
{  
    //To notify when changing the ItemSource.  
}
```

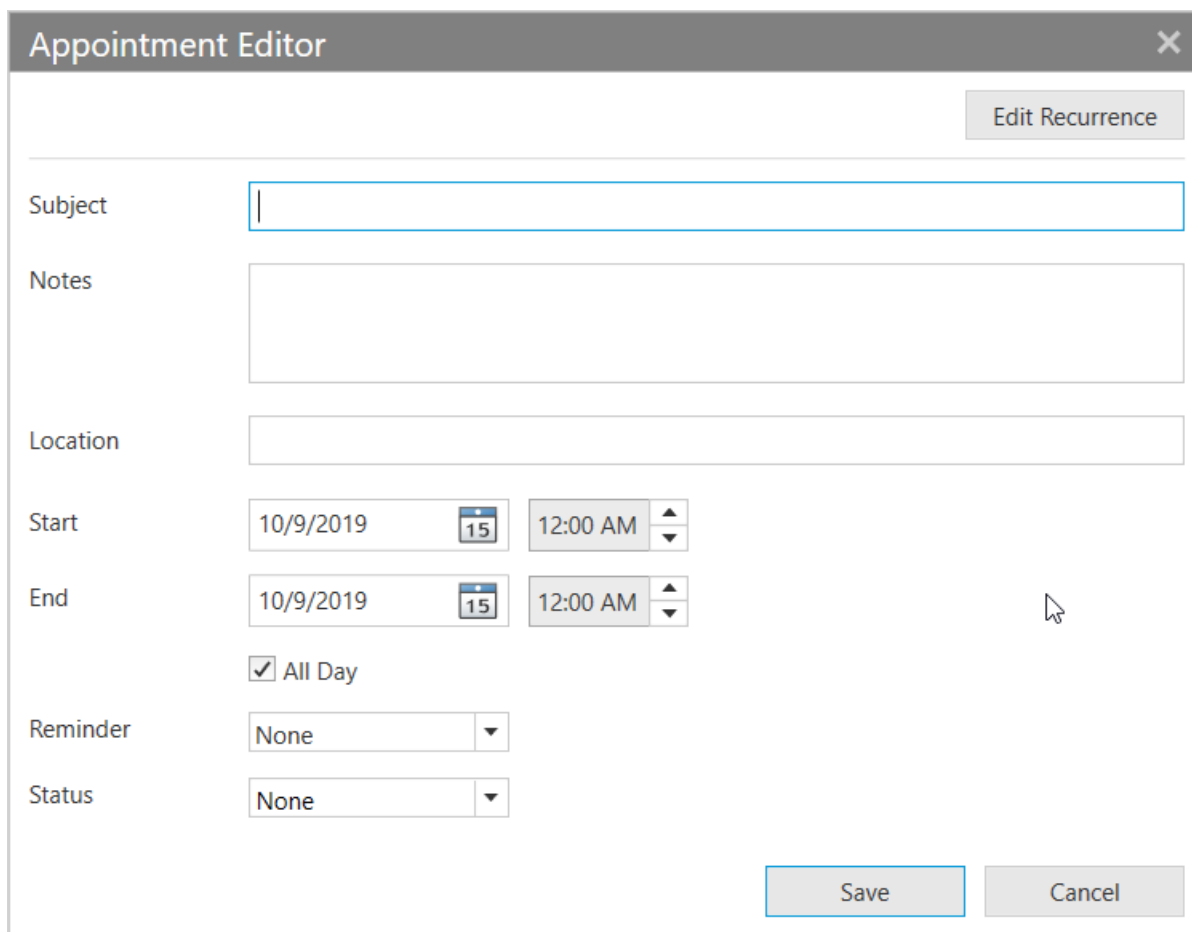
Add new appointment

Scheduler supports to add the new appointment by using the `Appointment Editor` window. It will appear when you double click or select the `AddNew` option from the `ContextMenu` on the time when you want the appointment occur.

Add new appointment using `ContextMenu`



Appointment editor window



The Appointment Editor dialog box is shown with the following fields and controls:

- Subject:** A text input field.
- Notes:** A large text area for notes.
- Location:** A text input field.
- Start:** A date picker set to 10/9/2019 and a time spinner set to 12:00 AM.
- End:** A date picker set to 10/9/2019 and a time spinner set to 12:00 AM.
- All Day:** A checked checkbox.
- Reminder:** A dropdown menu currently set to "None".
- Status:** A dropdown menu currently set to "None".
- Buttons:** "Edit Recurrence" (top right), "Save" (bottom right), and "Cancel" (bottom right).

Appointment template customization

Scheduler supports to customize the appointment appearance by using the [AppointmentTemplate](#) property. The `AppointmentTemplate` is a `ControlTemplate` type, used to customize or override the default template of the appointments.

XML

```
<Schedule:SfSchedule>
<Schedule:SfSchedule.AppointmentTemplate>
<ControlTemplate>
<Grid>
<Rectangle Fill="{Binding AppointmentBackground}"/>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="0.25*" />
<RowDefinition Height="0.75*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Margin="10,2,0,0" Text="{Binding Subject}" Grid.Row="0"
Grid.Column="0" Grid.ColumnSpan="2" HorizontalAlignment="Left"
VerticalAlignment="Center" FontSize="20" Foreground="White"
FontWeight="Light" FontFamily="Segoe UI" />
</Grid>
</ControlTemplate>
</Schedule:SfSchedule.AppointmentTemplate>
</Schedule:SfSchedule>
```

```

<Image Source="../../../Assets/Team.png" Grid.Row="1" HorizontalAlignment="Left"
VerticalAlignment="Center" Stretch="Fill" />
<TextBlock Text="{Binding StartTime}" Grid.Row="1" Grid.Column="1"
HorizontalAlignment="Left" VerticalAlignment="Center" FontSize="20"
Foreground="White" FontWeight="Light" FontFamily="Segoe UI"
TextWrapping="NoWrap"/>
</Grid>
</Grid>
</ControlTemplate>
</Schedule:SfSchedule.AppointmentTemplate>
</Schedule:SfSchedule>

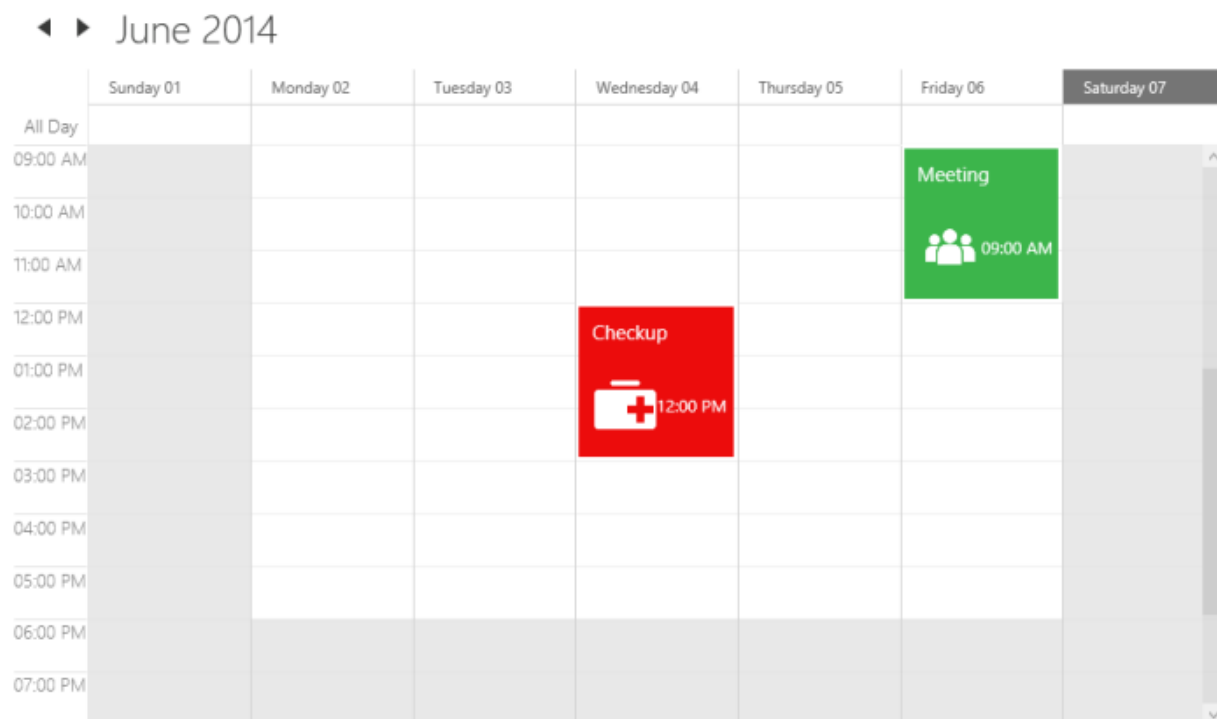
```

C#

```

schedule.Appointments.Add(new ScheduleAppointment()
{
    StartTime = DateTime.Now.Date.AddHours(9),
    EndTime = DateTime.Now.Date.AddHours(11),
    Subject = "Meet the doctor",
    Location = "Hutchison road",
});

```



Types of appointments

Spanned appointment

Spanned Appointment is an appointment which will generate the appointment when **StartTime** and **EndTime** day difference is more than 1.

C#

```

ObservableCollection<Meeting> Meetings = new
ObservableCollection<Meeting>();
// Creating instance for custom appointment class
Meeting meeting = new Meeting();
// Setting start time of an event
meeting.From = new DateTime(2017, 05, 08, 10, 0, 0);
// Setting end time of an event
meeting.To = meeting.From.AddDays(1).AddHours(1);
// Setting start time for an event
meeting.EventName = "Anniversary";
// Setting color for an event
meeting.Color = Brushes.Green;
// Adding a custom appointment in CustomAppointmentCollection
Meetings.Add(meeting);
this.schedule.ItemsSource = Meetings;

```

◀ ▶ May 2017						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
30	01	02	03	04	05	06
07	08 Anniversary	09	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	01	02	03

Recurrence Appointment

This section has briefly explained in following [link](#)

All day appointments

Appointments can be scheduled for an entire day by using [AllDay](#) property in `ScheduleAppointment`.

XML

```
<syncfusion:SfSchedule.Appointments>
<syncfusion:ScheduleAppointment StartTime="05/08/2017 10:0:0"
EndTime="05/08/2017 11:0:0" Subject="Meeting" AllDay="True"
Location="Hutchison road"/>
</syncfusion:SfSchedule.Appointments>
```

C#

```
// Creating an instance for schedule appointment collection
ScheduleAppointmentCollection scheduleAppointmentCollection = new
ScheduleAppointmentCollection();
//Adding schedule appointment in schedule appointment collection
scheduleAppointmentCollection.Add(new ScheduleAppointment()
{
StartTime = new DateTime(2017, 05, 08, 10, 0, 0),
EndTime = new DateTime(2017, 05, 08, 11, 0, 0),
Subject = "Meeting",
AllDay = true,
Location = "Hutchison road",
});
//Adding schedule appointment collection to Appointments of Scheduler
this.Schedule.Appointments = scheduleAppointmentCollection;
```

Note: Appointment which lasts through an entire day (exact 24 hours) will be considered as all day appointment without setting `AllDay` property. For example 06/09/2018 12:00AM to 06/10/2018 12:00AM.

Appointment generating behavior

Scheduler supports to change the all day appointment creation behavior by using [AppointmentBehavior](#) property.

Default - Appointments will be generated based on `StartTime` and `EndTime`. If `AllDay` property is enabled appointment will be generated in all day panel.

ExChangeBehavior - Appointments will be generated when appointment has scheduled for full day.

XML

```
<schedule:SfSchedule x:Name="Schedule"
AppointmentBehavior="ExchangeBehavior">
<syncfusion:SfSchedule.Appointments>
<syncfusion:ScheduleAppointment StartTime="10/09/2019 10:0:0"
EndTime="10/11/2019 11:0:0" Subject="Meeting" Location="Hutchison road"/>
</syncfusion:SfSchedule.Appointments>
</schedule:SfSchedule>
```

C#

```
// Creating an instance for schedule appointment collection
ScheduleAppointmentCollection scheduleAppointmentCollection = new
ScheduleAppointmentCollection();
//Adding schedule appointment in schedule appointment collection
scheduleAppointmentCollection.Add(new ScheduleAppointment()
{
StartTime = new DateTime(2019, 10, 09, 10, 0, 0),
EndTime = new DateTime(2019, 10, 11, 11, 0, 0),
Subject = "Meeting",
Location = "Hutchison road",
});
```

```

{
    StartTime = new DateTime(2019, 10, 09, 10, 0, 0),
    EndTime = new DateTime(2019, 10, 11, 11, 0, 0),
    Subject = "Meeting",
    Location = "Hutchison road",
});
this.Schedule.AppointmentBehavior = AppointmentBehavior.ExchangeBehavior;
//Adding schedule appointment collection to Appointments of Scheduler
this.Schedule.Appointments = scheduleAppointmentCollection;

```

Default

◀ ▶ October 2019					
	Monday 07	Tuesday 08	Wednesday 09	Thursday 10	Friday 11
All Day			Meeting	Meeting	Meeting
12:00 AM					
30					
01:00 AM					
30					
02:00 AM					
30					
03:00 AM					
30					
04:00 AM					
30					
05:00 AM					
30					

ExchangeBehavior

◀ ▶ October 2019					
	Monday 07	Tuesday 08	Wednesday 09	Thursday 10	Friday 11
All Day			Meeting	Meeting	
12:00 AM					
30					
01:00 AM					
30					
02:00 AM					
30					
03:00 AM					
30					
04:00 AM					
30					
05:00 AM					
30					

All day appointment panel

All-day appointment and Spanned appointment doesn't block out entire time slot in Scheduler, rather it will render in separate layout exclusively for all-day appointment. It can be disabled by setting [ShowAllDay](#) property.

XML

```
<Schedule:SfSchedule x:Name="schedule" ShowAllDay="False"/>
```

C#

```
this.Schedule.ShowAllDay = false;
```




Appointment tooltip

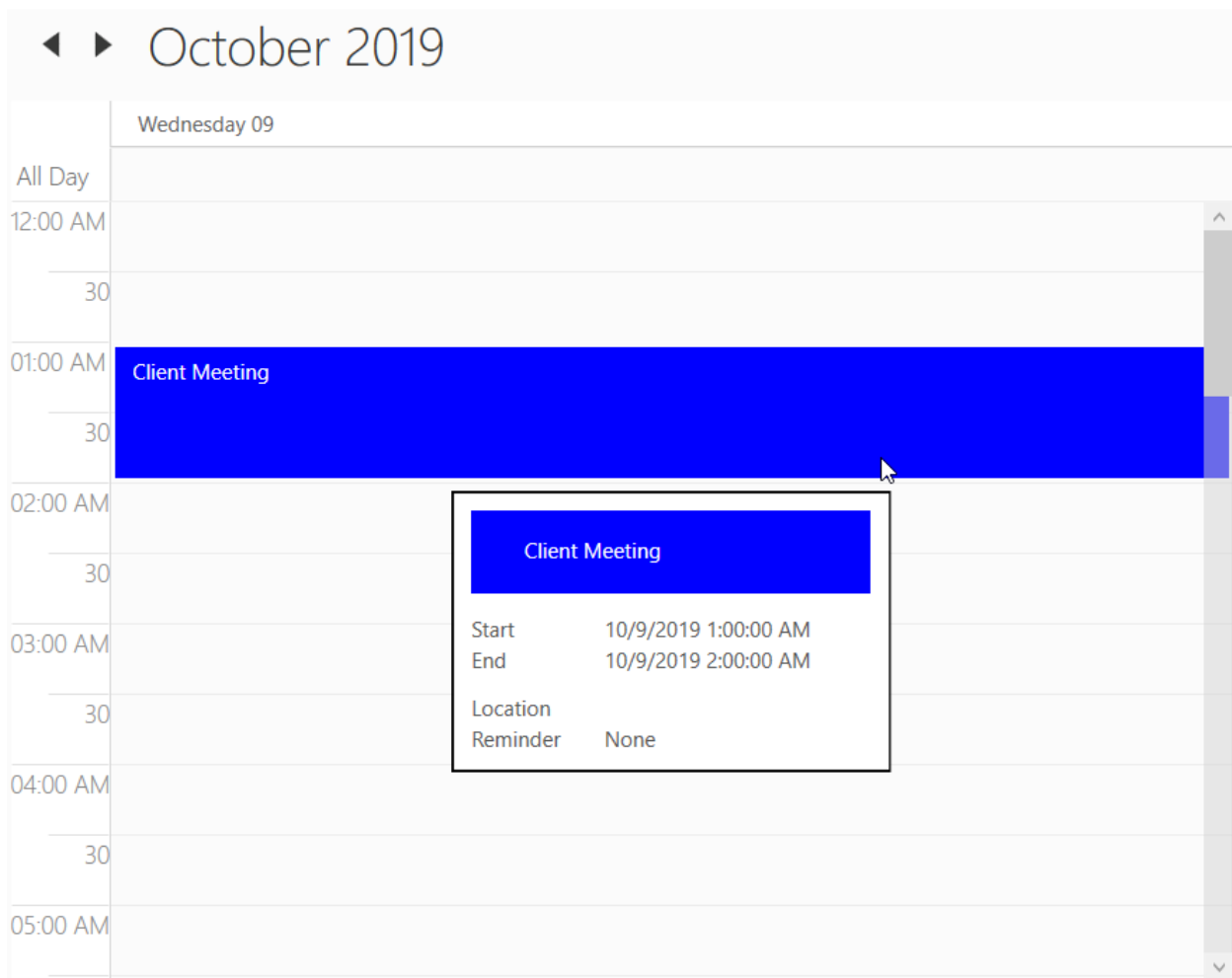
Scheduler supports to show the tooltip for appointment with the details of appointment. The appointment tooltip will be displayed by using the [AppointmentTooltipVisibility](#) property.

XML

```
<Schedule:SfSchedule x:Name="schedule"
AppointmentTooltipVisibility="Visible" ScheduleType="Day"/>
```

C#

```
this.Schedule.AppointmentTooltipVisibility = Visibility.Visible;
```



Appointment tooltip template customization

Scheduler supports to customize the tooltip by using [AppointmentToolTipTemplate](#) property.

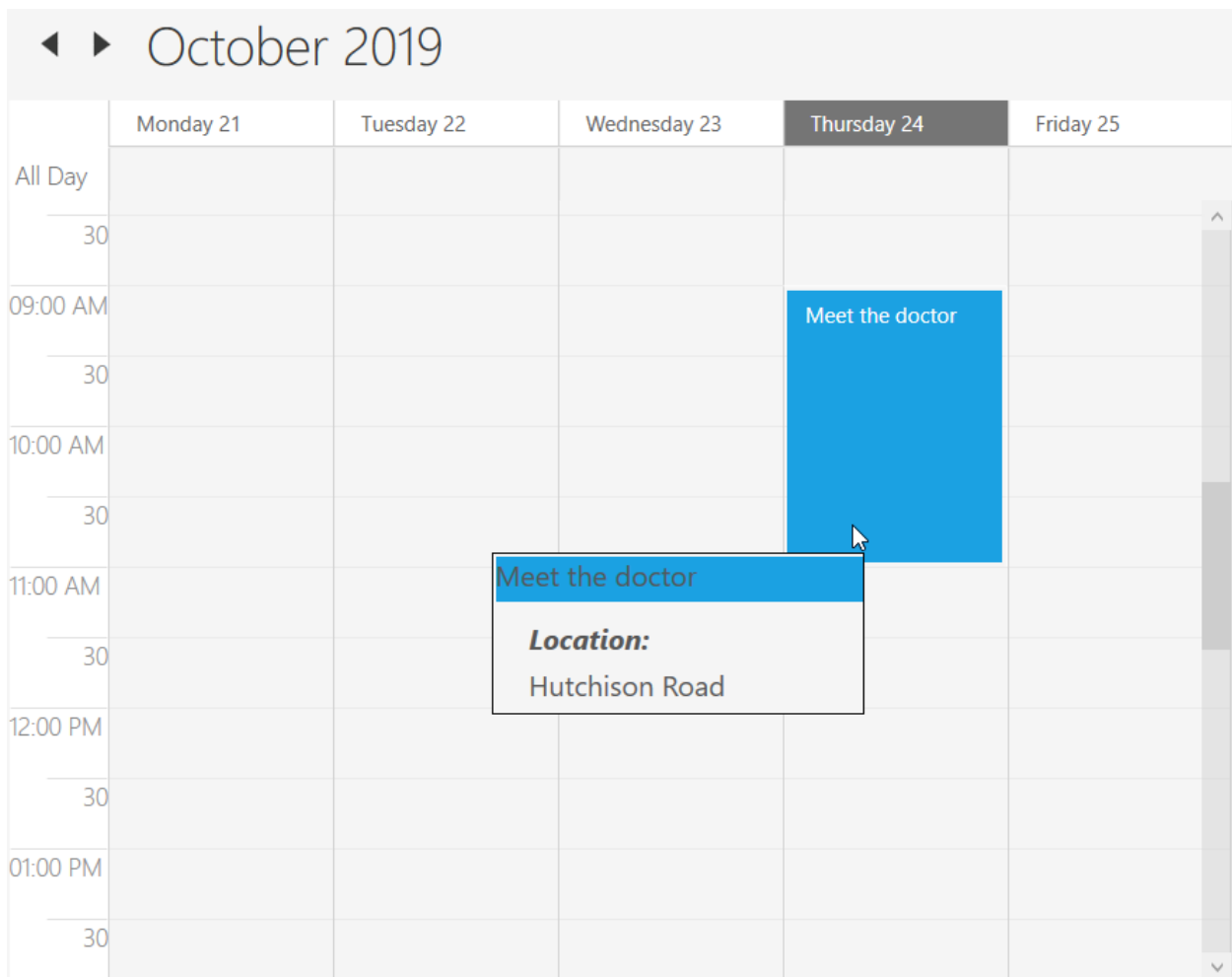
XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleType="WorkWeek"
AppointmentToolTipVisibility="Visible" Background="WhiteSmoke" >
  <schedule:SfSchedule.AppointmentToolTipTemplate>
    <ControlTemplate>
      <Border BorderBrush="Black" BorderThickness="1">
        <Grid Background="WhiteSmoke" Height="90" Width="210">
          <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition Height="10"/>
            <RowDefinition/>
            <RowDefinition />
          </Grid.RowDefinitions>
          <Border Grid.Row="0">
            <TextBlock Margin="1,1,0,0" Background="{Binding AppointmentBackground}"
              FontSize="16" Text="{Binding Subject}" Grid.Row="0"/>
          </Border>
          <TextBlock FontSize="16" FontWeight="Bold" FontStyle="Italic"
            Margin="20,0,0,0" Text="Location: " Grid.Row="2"/>
        </Grid>
      </Border>
    </ControlTemplate>
  </schedule:SfSchedule.AppointmentToolTipTemplate>
</schedule:SfSchedule>
```

```
<TextBlock FontSize="16" Margin="20,0,0,0" Text="{Binding Location}"
Grid.Row="3"/>
</Grid>
</Border>
</ControlTemplate>
</schedule:SfSchedule.AppointmentToolTipTemplate>
</schedule:SfSchedule>
```

C#

```
schedule.Appointments.Add(new ScheduleAppointment()
{
    StartTime = DateTime.Now.Date.AddHours(9),
    EndTime = DateTime.Now.Date.AddHours(11),
    Subject = "Meet the doctor",
    Location = "Hutchison Road"
});
```



Change the appointment selection color

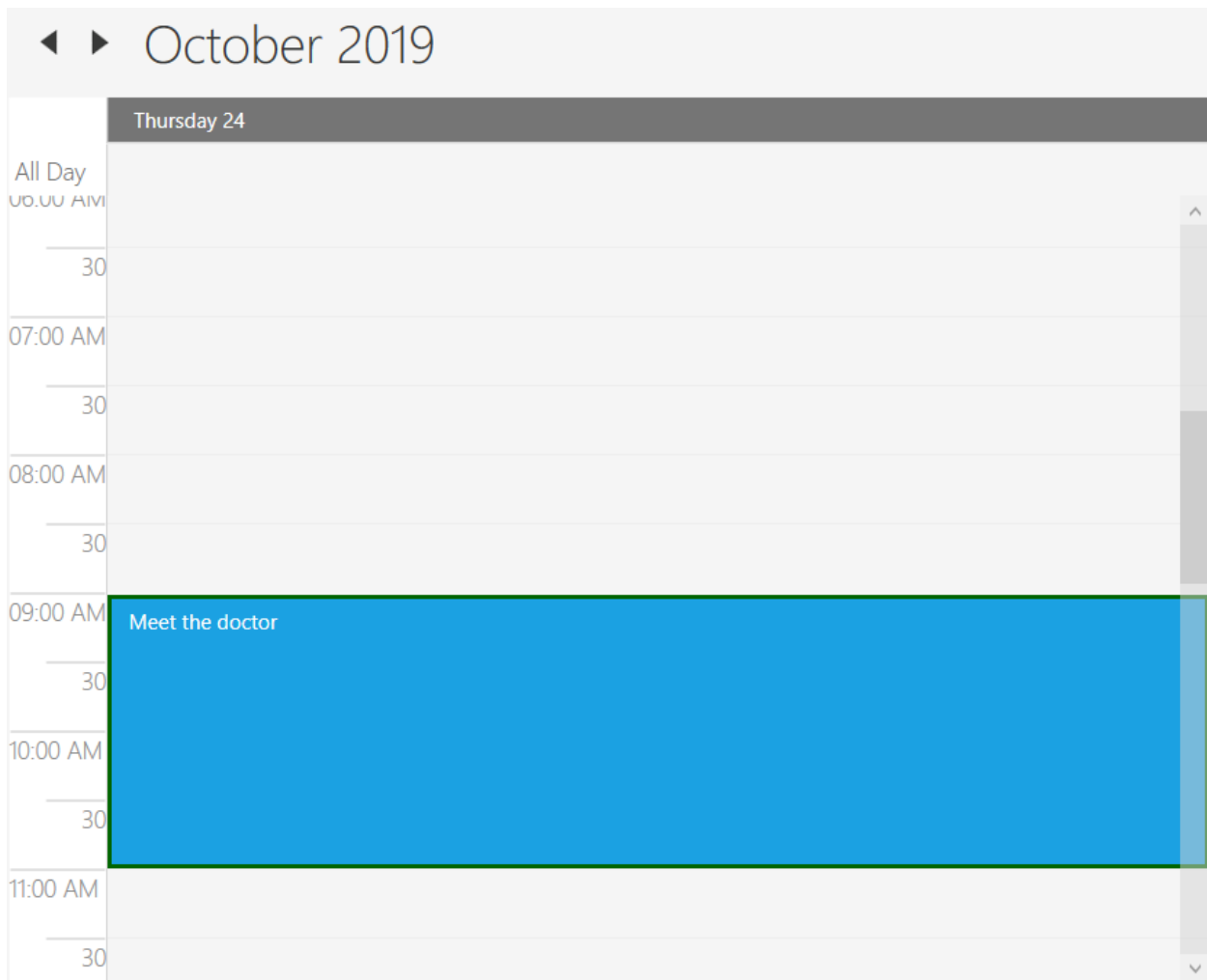
Scheduler supports to change the appointment selection background by using [AppointmentSelectionBrush](#) property.

XML

```
<Schedule:SfSchedule x:Name="schedule" AppointmentSelectionBrush
="DarkGreen"/>
```

C#

```
this.schedule.AppointmentSelectionBrush = Brushes.DarkGreen;
```



Customize the appointment status collection

Scheduler supports to customize the appointment status collection by using [AppointmentStatusCollection](#) property.

XML

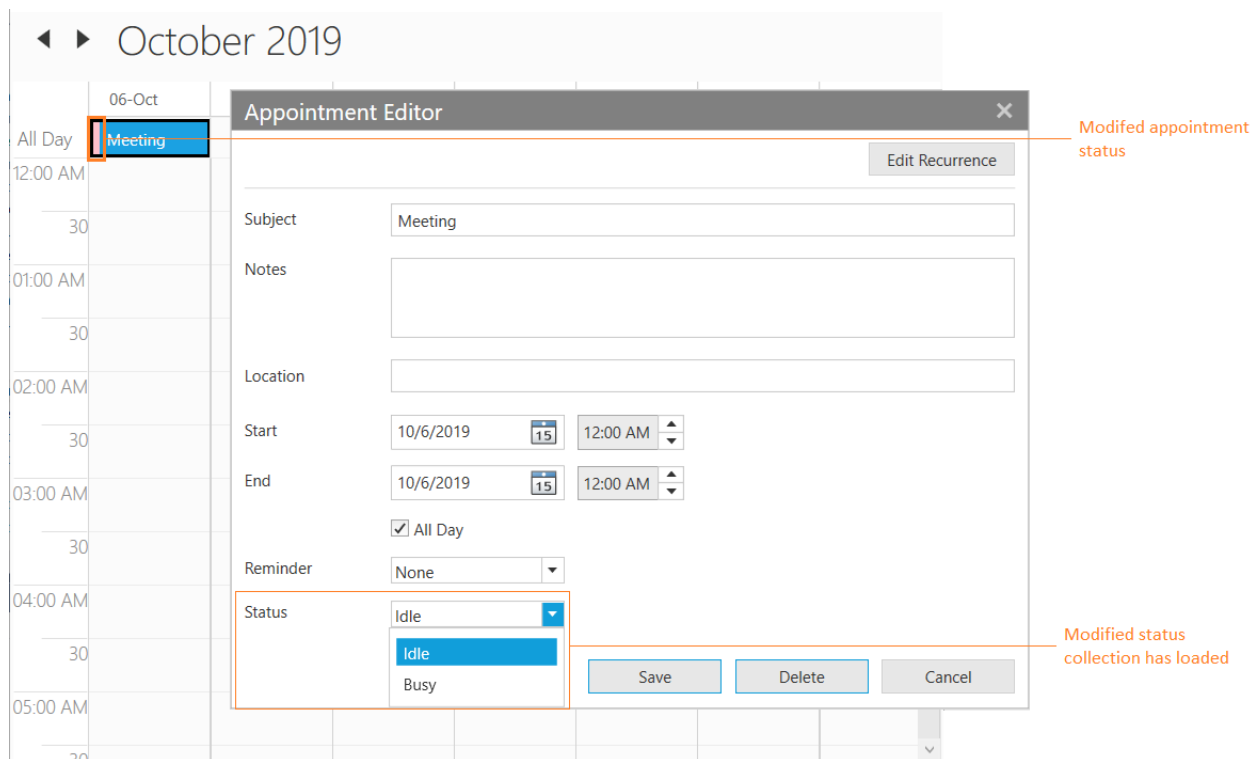
```
<schedule:SfSchedule x:Name="schedule">
  <schedule:SfSchedule.AppointmentStatusCollection>
    <schedule:ScheduleAppointmentStatus Status="Idle" Brush="Pink"/>
    <schedule:ScheduleAppointmentStatus Status="Busy" Brush="Red"/>
  </schedule:SfSchedule.AppointmentStatusCollection>
</schedule:SfSchedule>
```

C#

```

schedule.AppointmentStatusCollection = new
ScheduleAppointmentStatusCollection()
{
    new ScheduleAppointmentStatus { Status = "Idle", Brush = new
SolidColorBrush(Colors.Pink) },
    new ScheduleAppointmentStatus { Status = "Busy", Brush = new
SolidColorBrush(Colors.Red) }
};

```



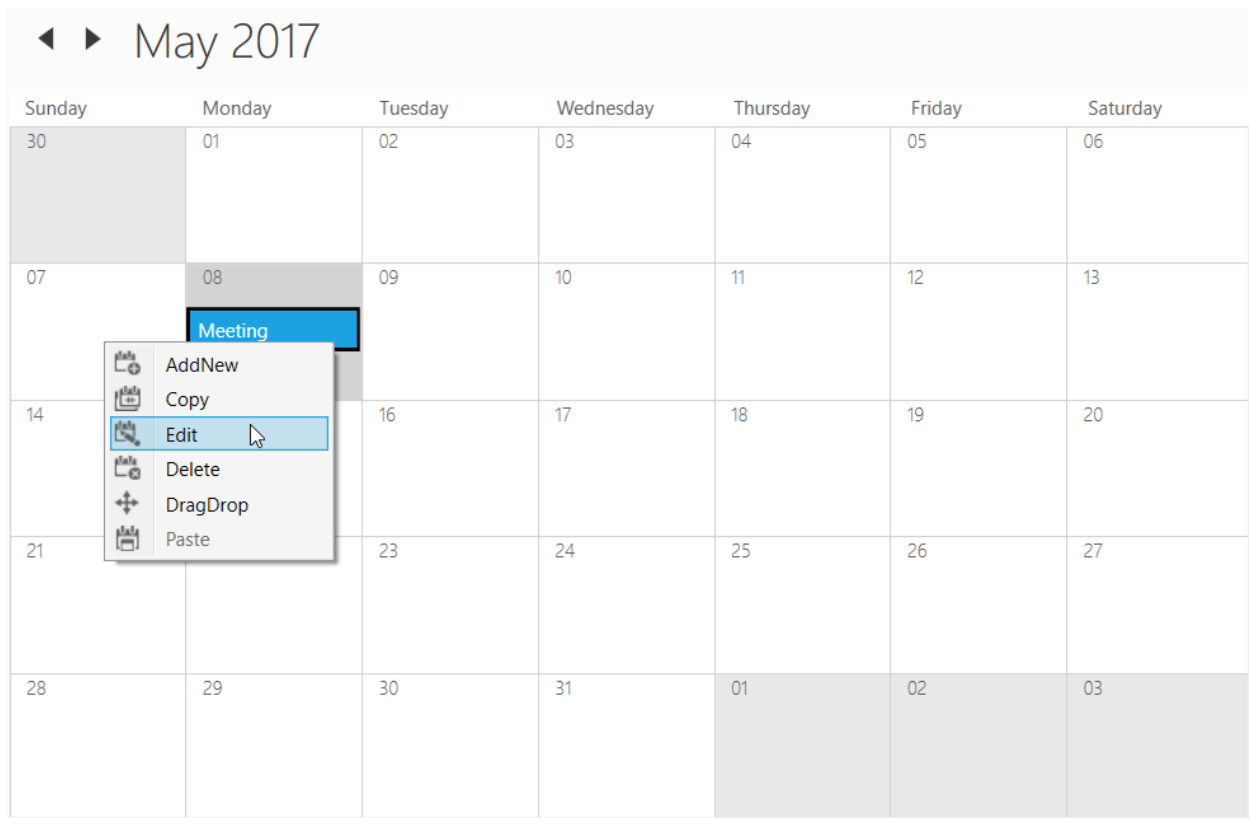
Appointment Editing in WPF Schedule (Classic)

This section explains how to handle appointment editing in WPF scheduler and also explains about the appointment resizing and drag drop operations.

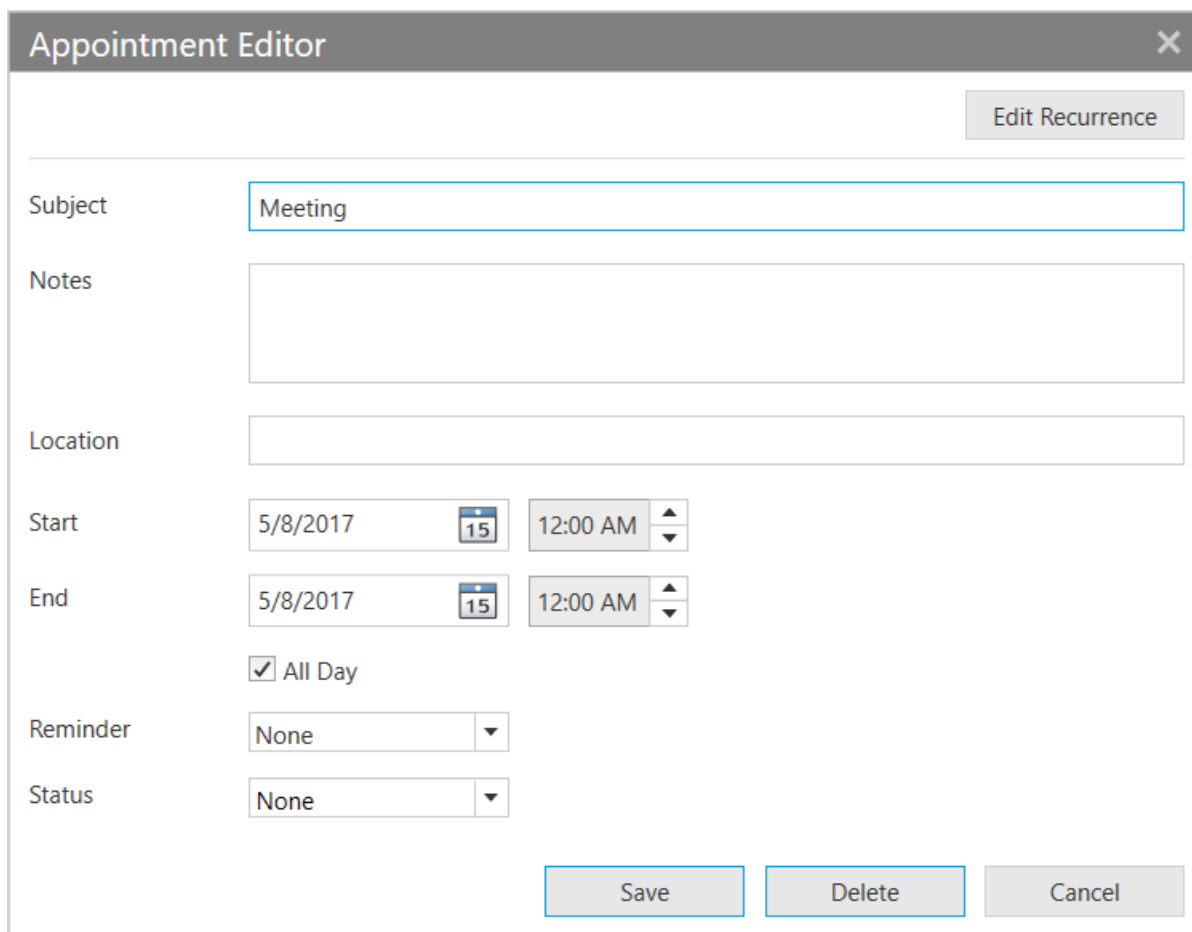
Editing Appointment

Scheduler supports to edit the appointment by using 'Appointment Editor' UI window. User can open this window by double click over the appointment or right click over the appointment and select the edit option from the `ContextMenu`.

ContextMenu edit option



Appointment editor window

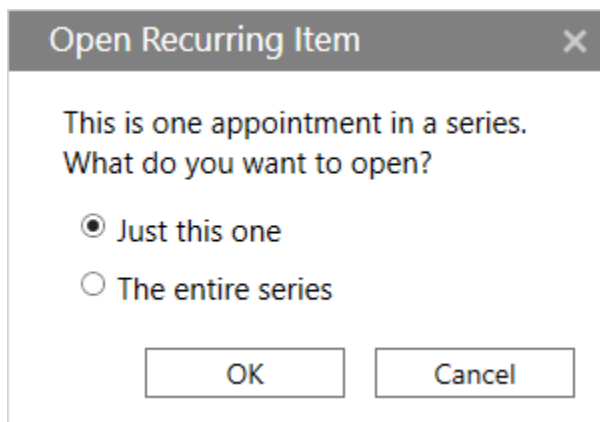


The Appointment Editor window is a dialog box with a title bar containing the text "Appointment Editor" and a close button (X). Inside the window, there is a button labeled "Edit Recurrence" in the top right corner. The main area contains several input fields: "Subject" with the text "Meeting", "Notes" (a large empty text area), "Location" (an empty text field), "Start" with a date picker showing "5/8/2017" and a time spinner showing "12:00 AM", "End" with a date picker showing "5/8/2017" and a time spinner showing "12:00 AM", a checkbox labeled "All Day" which is checked, "Reminder" with a dropdown menu showing "None", and "Status" with a dropdown menu showing "None". At the bottom right, there are three buttons: "Save", "Delete", and "Cancel".

User can edit the fields in appointment editor window. The changes will be saved back in appointment and mapped data object when using data binding.

Edit recurring appointment

Scheduler supports to edit the recurrence appointment. The following window will appear when user edit the recurrence appointment to select whether to edit only the particular occurrence or appointment series.



The Open Recurring Item dialog box has a title bar with the text "Open Recurring Item" and a close button (X). The main text reads "This is one appointment in a series. What do you want to open?". Below this text are two radio button options: "Just this one" (which is selected) and "The entire series". At the bottom, there are two buttons: "OK" and "Cancel".

You can also handle the opening of Open Recurring Item window using `RecurrenceEditMode` property in [AppointmentEditorOpeningEventArgs](#) by handling `AppointmentEditorOpening` event.

AppointmentEditorOpening event

When user opens the appointment editor UI window to edit the appointment, then scheduler notifies by [AppointmentEditorOpening](#) event.

[AppointmentEditorOpeningEventArgs](#) has following members which provides information for AppointmentEditorOpening event.

[Action](#) - Gets the action (add or delete or edit) for the selected appointment.

[Appointment](#) - Gets the selected appointment details.

[RecurrenceEditMode](#) - Get or Sets the edit mode to perform the edit option to edit the occurrence or series for recurrence appointment.

- User - Default window dialog will appear when editing a recurrence appointment to select the edit option from the end-user itself.
- Occurrence - Edit the particular occurrence alone in recurrence appointment. Default window dialog will not appear.
- Series - Edit the entire series in recurrence appointment. Default window dialog will not appear.

[SelectedResource](#) - Gets the selected appointment resource details if scheduler does have the resource.

[StartTime](#) - Gets the appointment start time

[Cancel](#) - To avoid the default appointment editor showing by enabling this property.

For example, to use custom the appointment editor window instead of default appointment editor window you can handle [AppointmentEditorOpening](#) event.

C#

```
this.schedule.AppointmentEditorOpening += Schedule_AppointmentEditorOpening;
private void Schedule_AppointmentEditorOpening(object sender,
AppointmentEditorOpeningEventArgs e)
{
    //To handle the default appointment editor window by setting the e.Cancel
    value as true.
    e.Cancel = true;
    if (e.Appointment != null)
    {
        //Display the custom appointment editor window to edit the appointment
    }
    else
    {
        //Display the custom appointment editor window to add new appointment
    }
}
```

AppointmentEditorClosed

Scheduler notifies by [AppointmentEditorClosed](#) when user close the appointment editor window.

[AppointmentEditorClosedEventArgs](#) event has following members which provides information for AppointmentEditorClosed event.

[Action](#) - Gets the action of appointment which is add or delete or edit.

[EditedAppointment](#) - Gets the edited appointment details if appointment editor closed with edit action.

[OriginalAppointment](#) - Gets the selected appointment details.

[IsNew](#) - Gets the appointment is new or not.

[Handled](#) - To handle appointment editor changes update into the Scheduler **Appointments** collection.

For example, to handle the appointment adding for today's date, you can handle the **AppointmentEditorClosed** event.

C#

```
this.schedule.AppointmentEditorClosed += Schedule_AppointmentEditorClosed;
private void Schedule_AppointmentEditorClosed(object sender,
AppointmentEditorClosedEventArgs e)
{
    var appointment = e.EditedAppointment as ScheduleAppointment;
    if (appointment != null)
    {
        if (appointment.StartTime.Day == DateTime.Now.Day)
        {
            e.Handled = true;
        }
    }
}
```

Disable appointment editing

Scheduler supports to prevent the editing of the appointments by using [AllowEditing](#) property.

XML

```
<Schedule:SfSchedule x:Name="Schedule" AllowEditing="False"/>
```

C#

```
this.Schedule.AllowEditing = false;
```

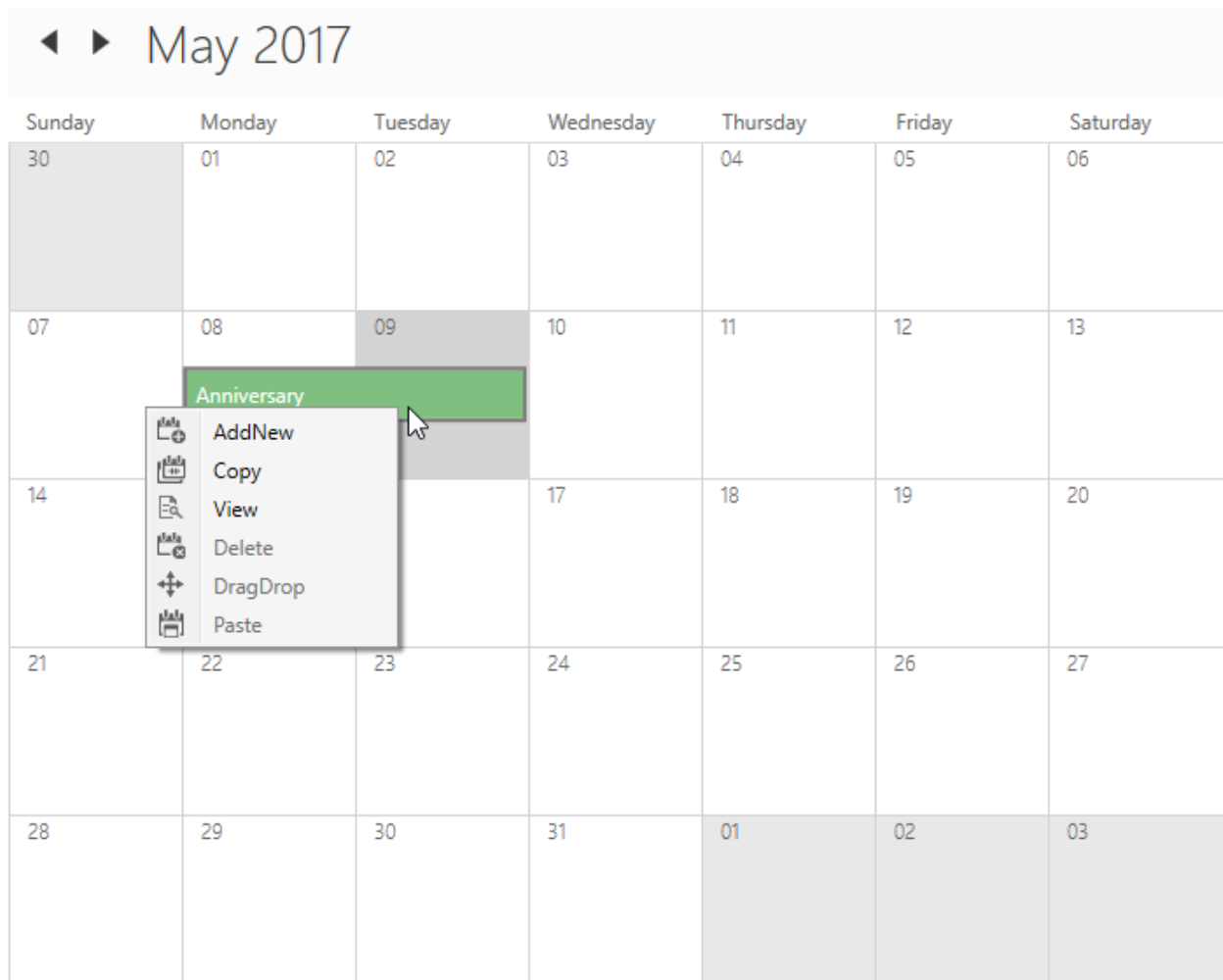
Create read only appointment

Scheduler supports to create the read only appointment by using [ScheduleAppointment.ReadOnly](#) property. If you enable this property, user will not be able to perform edit, resize and drag drop operations.

C#

```
//// Creating an instance for schedule appointment collection
ScheduleAppointmentCollection scheduleAppointmentCollection = new
ScheduleAppointmentCollection();
////Adding schedule appointment in schedule appointment collection
scheduleAppointmentCollection.Add(new ScheduleAppointment()
{
    StartTime = new DateTime(2017, 05, 08, 10, 0, 0),
    EndTime = new DateTime(2017, 05, 10, 10, 0, 0),
    Subject = "Anniversary",
    Location = "Hutchison road",
    ReadOnly = true,
    AppointmentBackground = Brushes.Green
});
```

```
//Adding schedule appointment collection to Appointments of SfSchedule
this.Schedule.Appointments = scheduleAppointmentCollection;
```



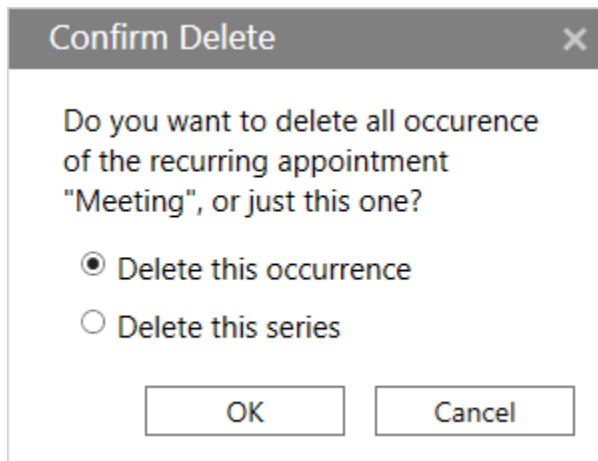
Appointment deleting

Scheduler supports three ways to remove the selected appointment

1. Pressing `Delete` key
2. Using `ContextMenu` delete option.
3. Using `AppointmentEditor`.

Delete recurring appointment

Scheduler supports to delete the recurrence appointment. The following window will appear when user deletes the recurrence appointment to select the delete option to make the changes for occurrence or appointment series.



AppointmentDeleting event

Scheduler notifies by [AppointmentDeleting](#) event when user delete the appointment.

[AppointmentDeletingEventArgs](#) has following members which provides information for AppointmentDeleting event.

Appointment - Gets the selected appointment

MappedObject - Gets the binding object details of selected appointment if schedule appointments are mapped with custom object.

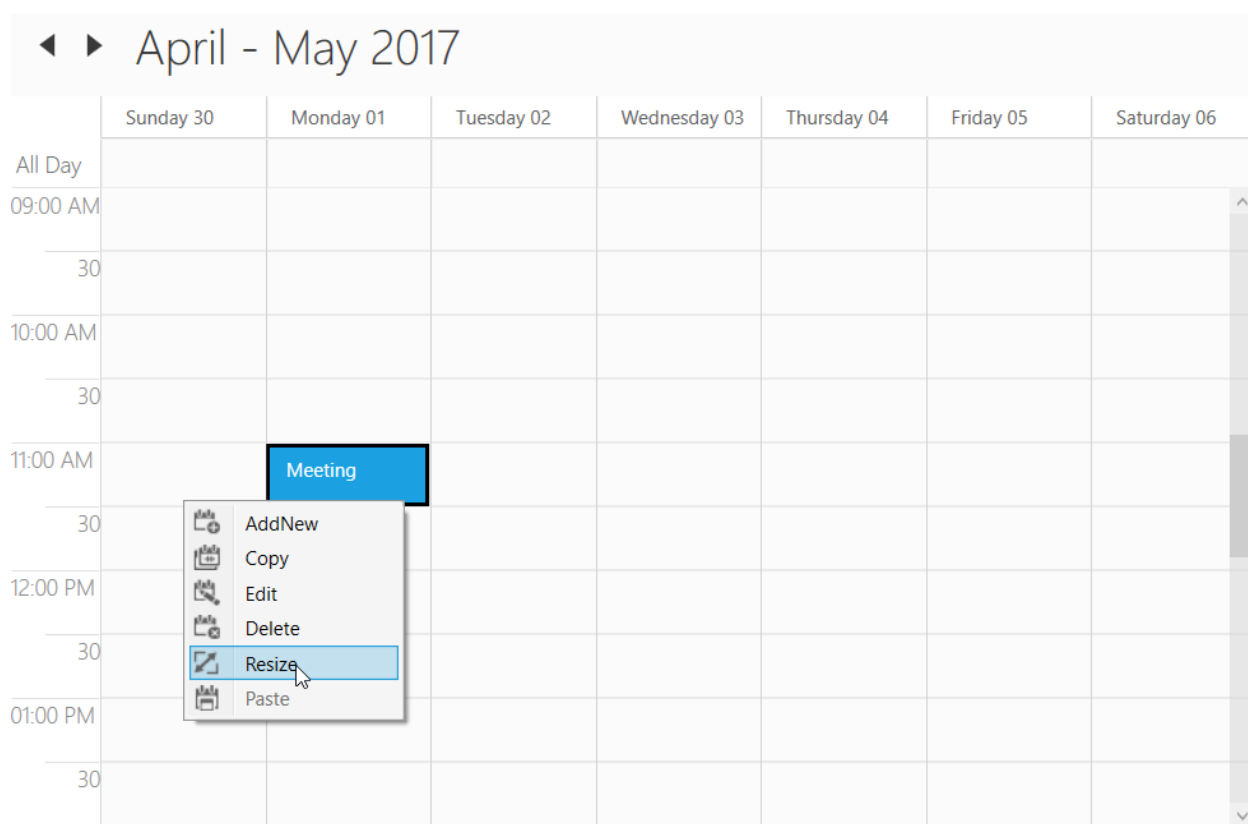
RecurrenceEditMode - Gets or sets whether to delete particular occurrence or appointment sequence when delete a recurrence appointment. You can let end-user handle this option (using built-in dialog shown in scheduler) or define it by yourself using [AppointmentDeleting](#) event.

- User - Default window dialog will appear when deleting a recurrence appointment to select the delete option from the end-user itself.
- Occurrence - Delete the particular occurrence alone in recurrence appointment. Default window dialog will not appear.
- Series - Delete the entire series in recurrence appointment. Default window dialog will not appear.

[Cancel](#) - By enabling this property, avoid deleting the appointment.

Appointment resizing

Scheduler supports to resize the appointment using the option **Resize** option from the **ScheduleAppointment** context menu. This support is available for all views except 'Month' view.



AppointmentStartResizing event

Scheduler notifies by [AppointmentStartResizing](#) event when user start to resize the appointment.

[AppointmentStartResizingEventArgs](#) has following members which provides information for AppointmentStartResizing event.

[Appointment](#) - Get the appointment details that is start to resize.

[Cancel](#) - Setting value to true, cancels the triggered action.

C#

```
this.schedule.AppointmentStartResizing += Schedule_AppointmentStartResizing;
private void Schedule_AppointmentStartResizing(object sender,
AppointmentStartResizingEventArgs e)
{
    //To notify when start to resize the appointment.
}
```

AppointmentResizing event

Scheduler notifies by [AppointmentResizing](#) event when user resizing the appointment.

[AppointmentResizingEventArgs](#) has following members which provides information for AppointmentResizing event.

[Appointment](#) - Gets the resizing appointment details.

[From](#) - Gets the appointment start time.

[To](#) - Gets the appointment end time.

[ResizeType](#) - Gets the resize type for appointment whether it is resizing from start or end.

[RefreshAppointment](#) - Get or Sets appointment need to be refresh or not.

C#

```
this.schedule.AppointmentResizing += Schedule_AppointmentResizing;
private void Schedule_AppointmentStartResizing(object sender,
AppointmentResizingEventArgs e)
{
    //To notify when resizing the appointment.
}
```

AppointmentEndResizing event

Scheduler notifies by [AppointmentEndResizing](#) event when user ends the appointment resizing.

[AppointmentEndResizingEventArgs](#) has following members which provides information for AppointmentEndResizing event.

[Appointment](#) - Gets the resizing appointment details.

[From](#) - Gets the appointment start time.

[To](#) - Gets the appointment end time.

[ResizeType](#) - Gets the resize type for appointment whether it is resizing from start or end.

[Cancel](#) - Setting value to true, cancels the triggered action.

C#

```
this.schedule.AppointmentEndResizing += Schedule_AppointmentResizing;
private void Schedule_AppointmentStartResizing(object sender,
AppointmentEndResizingEventArgs e)
{
    //To notify when resizing is completed for an appointment.
}
```

Drag and drop

Scheduler supports to reschedule the appointment by performing the drag and drop operation. This support is available for all views.

DragStarting event

Scheduler notifies by [DragStarting](#) when start to drag the appointment.

[DragStartingEventArgs](#) has following members which provides information for DragStarting event.

[Appointment](#) - Get the selected appointment.

[Cancel](#) - Setting the value as `true` to prevent the drag and drop operation.

C#

```
this.schedule.DragStarting += Schedule_DragStarting;
private void Schedule_DragStarting(object sender, DragStartingEventArgs e)
{
    //To notify when start to drag the appointment.
}
```

AppointmentDragging event

Scheduler notifies by [AppointmentDragging](#) when drag the appointment.

[AppointmentDraggingEventArgs](#) has following members which provides information for AppointmentDragging event.

[Appointment](#) - Gets the selected appointment.

[From](#) - Gets the selected appointment start time.

[RefreshAppointment](#) - Gets or sets the bool value to update the appointment [StartTime](#) and [EndTime](#) while dragging the appointment or not.

[Resource](#) - Gets the resource details if the selected appointment presenting the resource.

[To](#) - Gets the selected appointment end time.

C#

```
this.Schedule.AppointmentDragging += Schedule_AppointmentDragging;  
private void Schedule_AppointmentDragging(object sender,  
AppointmentDraggingEventArgs e)  
{  
    //To notify when dragging the appointment.  
}
```

AppointmentStartDragging event

Scheduler notifies by [AppointmentStartDragging](#) when user start to drag the appointment.

[AppointmentStartDraggingEventArgs](#) has following members which provides information for AppointmentStartDragging event.

[Appointment](#) - Gets the selected appointment.

C#

```
this.schedule.AppointmentStartDragging += Schedule_AppointmentStartDragging;  
private void Schedule_AppointmentStartDragging(object sender,  
AppointmentStartDraggingEventArgs e)  
{  
    //To notify when start to drag the appointment  
}
```

AppointmentEndDragging event

Scheduler notifies by [AppointmentEndDragging](#) when user ends to drag the appointment.

[AppointmentEndDraggingEventArgs](#) has following members which provides information for AppointmentEndDragging event.

[Appointment](#) - Gets the selected appointment.

[From](#) - Gets the selected appointment start time.

[Resource](#) - Gets the resource details if the selected appointment does have added in resource.

[To](#) - Gets the selected appointment end time.

C#

```

this.schedule.AppointmentEndDragging += Schedule_AppointmentEndDragging;
private void Schedule_AppointmentEndDragging(object sender,
AppointmentEndDraggingEventArgs e)
{
    //To notify when the appointment dragging is end.
}

```

Appointment-Navigation in WPF Schedule (Classic)

When there is no appointment in the current view, we can navigate to view the previous appointment or next appointment from the current view. This can be enabled or disabled using the property ShowNavigationTap. The default value of ShowNavigationTap property is False.

HTML

```

<Schedule:SfSchedule x:Name="schedule" ScheduleType="Week"
ShowAppointmentNavigationButtons ="True" />

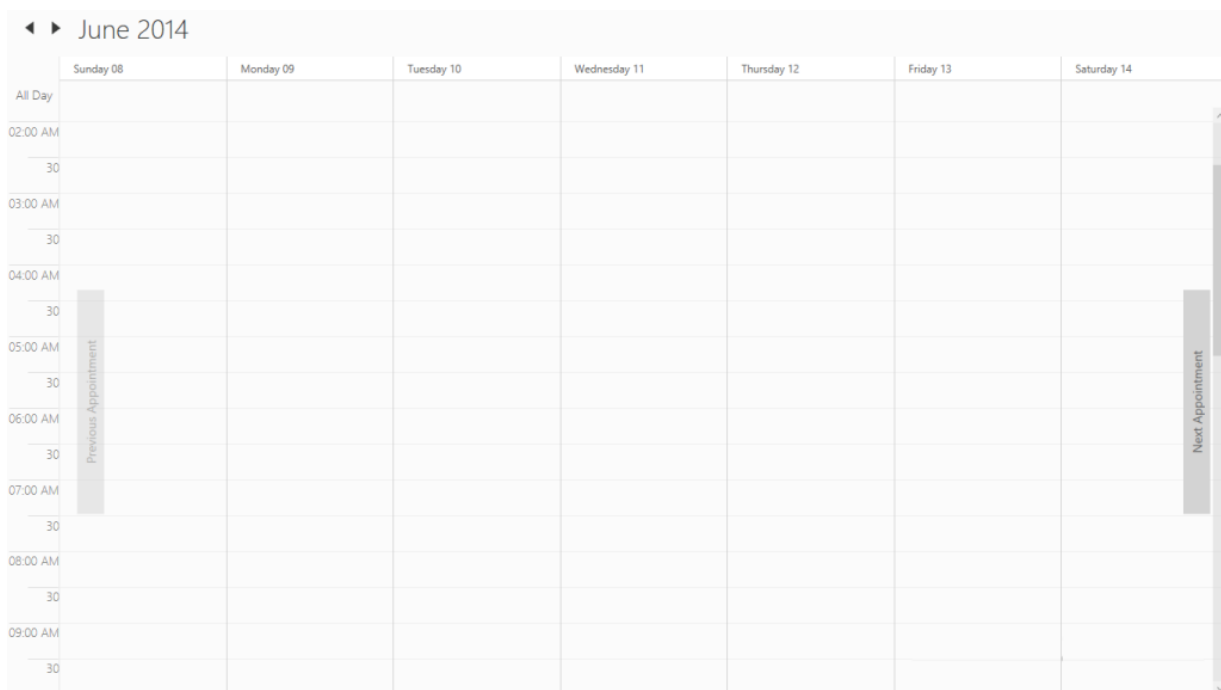
```

C#

```

SfSchedule schedule = new SfSchedule();
schedule.ScheduleType = ScheduleType.Week;
schedule.ShowAppointmentNavigationButtons = true;
this.grid.Children.Add(schedule);

```



PreviousNavigationButtonTemplate

Previous appointment navigation button can be customized by PreviousNavigationButtonTemplate property.

C#

```

<Schedule:SfSchedule x:Name="schedule" ScheduleType="Day"
ShowAppointmentNavigationButtons="True">
<Schedule:SfSchedule.PreviousNavigationButtonTemplate>
<DataTemplate>
<Border Height="200" Width="50" Background="Red"/>
</DataTemplate>
</Schedule:SfSchedule.PreviousNavigationButtonTemplate>
</Schedule:SfSchedule>

```



NextNavigationButtonTemplate

By using the `NextNavigationButtonTemplate` property the next appointment navigation button can be customized to view the next appointment.

HTML

```

<Schedule:SfSchedule x:Name="schedule" ScheduleType="Day"
ShowAppointmentNavigationButtons="True">
<Schedule:SfSchedule.NextNavigationButtonTemplate>
<DataTemplate>
<Border Height="200" Width="50" Background="Red"/>
</DataTemplate>
</Schedule:SfSchedule.NextNavigationButtonTemplate>
</Schedule:SfSchedule>

```




Reminder in WPF Schedule (Classic)

Schedule alerts you for particular appointment with reminder window when enable the [EnableReminderTimer](#) property. Reminder window supports to [Dismiss](#) or [DismissAll](#) or set the [SnoozeTime](#) for reminder appointments.

Setting reminder for an Appointment

Reminder can be set by setting the [EnableReminderTimer](#) property is `true`. The remainder time can be set using the [ReminderTime](#) property of `ScheduleAppointment`.

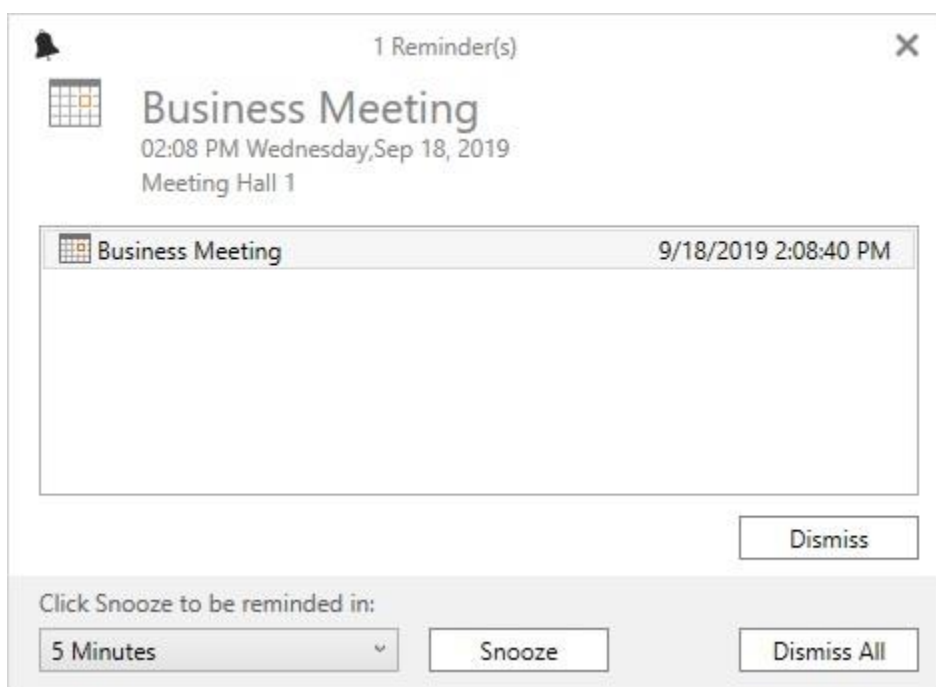
XML

```
<Grid>
<syncfusion:SfSchedule x:Name="schedule" ScheduleType="Month"
EnableReminderTimer="True"/>
</Grid>
```

C#

```
schedule.Appointments.Add(new ScheduleAppointment
{
    StartTime = DateTime.Now.Date.AddHours(9),
    EndTime    = DateTime.Now.Date.AddHours(12),
    AppointmentBackground = new SolidColorBrush(Color.FromArgb(0xFF, 0xA2, 0xC1, 0x39)),
    Subject = "Business Meeting",
    ReminderTime = ReminderTimeType.TenHours
});
schedule.Appointments.Add(new ScheduleAppointment
{
    StartTime = currentDate.Date.AddDays(1).AddHours(10),
    EndTime = currentDate.Date.AddDays(1).AddHours(16),
```

```
AppointmentBackground = new SolidColorBrush(Color.FromArgb(0xFF, 0xD8, 0x00,
0x73)),
Subject = "Auditing",
ReminderTime = ReminderTimeType.TwoDays
});
schedule.Appointments.Add(new ScheduleAppointment
{
StartTime = DateTime.Now.Date.AddDays(7).AddHours(10),
EndTime = DateTime.Now.Date.AddDays(7).AddHours(13),
AppointmentBackground = new SolidColorBrush(Color.FromArgb(0xFF, 0xF0, 0x96,
0x09)),
Subject = "Conference",
ReminderTime = ReminderTimeType.TwoWeeks
});
```



Download demo from [GitHub](#)

Configuring Reminder Duration

Scheduler supports to set the reminder duration time to remind the appointments by using the [ReminderTime](#) property of [ScheduleAppointment](#).

C#

```
schedule.Appointments[0].ReminderTime = ReminderTimeType.FifteenMin;
```

Create a custom binding for ReminderTime

[ReminderTime](#) supports to map your custom object with `ScheduleAppointment.ReminderTime`.

C#

```
/// <summary>
/// Represents custom data properties.
/// </summary>
```

```
public class Meeting
{
    public String Subject { get; set; }
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }
    public Brush AppointmentColor { get; set; }
    public ReminderTimeType ReminderTime { get; set; }
}
```

Note: You can inherit this class from `INotifyPropertyChanged` for dynamic changes in custom data.

You can map those properties of `Meeting` class with our [SfSchedule](#) control by using [AppointmentMapping](#) and [ScheduleAppointmentMapping](#).

XML

```
<syncfusion:SfSchedule x:Name="schedule" ScheduleType="Month"
    DataSource="{Binding Meetings}">
    <syncfusion:SfSchedule.AppointmentMapping>
    <syncfusion:ScheduleAppointmentMapping
        SubjectMapping="Subject"
        AppointmentBackgroundMapping="AppointmentColor"
        StartTimeMapping="StartTime"
        EndTimeMapping="EndTime"
        ReminderTimeMapping="ReminderTime">
    </syncfusion:ScheduleAppointmentMapping>
    </syncfusion:SfSchedule.AppointmentMapping>
</syncfusion:SfSchedule>
```

C#

```
// Schedule data mapping for custom appointments
ScheduleAppointmentMapping dataMapping = new ScheduleAppointmentMapping();
dataMapping.SubjectMapping = "Subject";
dataMapping.StartTimeMapping = "StartTime";
dataMapping.EndTimeMapping = "EndTime";
dataMapping.AppointmentBackgroundMapping = "AppointmentColor";
dataMapping.ReminderTimeMapping = "ReminderTime";
schedule.AppointmentMapping = dataMapping;
```

Download demo from [GitHub](#)

Handling Reminder events

[ReminderOpening event](#)

[ReminderOpening](#) – occurs when appearing the reminder window.

[ReminderControlOpeningEventArgs](#) has following property.

[RemindAppCollection](#) – Gets list of reminder appointments.

You can prevent the reminder window opening through [ReminderControlOpeningEventArgs.Cancel](#) property of [ReminderOpening](#) event.

C#

```
this.Schedule.ReminderOpening += Schedule_ReminderOpening;
```

```
private void Schedule_ReminderOpening(object sender,
ReminderControlOpeningEventArgs e)
{
    e.Cancel = true;
}
```

ReminderClosed event

[ReminderClosed](#) – occurs when closing the reminder window.

[ReminderControlClosedEventArgs](#) provides information for [ReminderClosed](#) event.

ReminderFormActionChanged event

[ReminderFormActionChanged](#) – occurs when change the reminder window action for the appointment.

[ReminderFormActionChangedEventArgs](#) has following properties which provides information for [ReminderFormActionChanged](#) event.

[Action](#) - Gets the action of schedule appointments.

[Appointments](#) – Gets list of appointments that are changed.

[SnoozeTime](#) – Gets the snooze time of action changed appointments.

Download demo from [GitHub](#)

Time-Zone in WPF Schedule (Classic)

In Schedule, appointments can be created at various time zones using the properties [StartTimeZone](#) and [EndTimeZone](#) of the [ScheduleAppointment](#).

C#

```
SfSchedule schedule = new SfSchedule();
schedule.Appointments.Add(new ScheduleAppointment() { StartTimeZone=
schedule.TimeZoneCollection[2], EndTimeZone= schedule.TimeZoneCollection[2],
StartTime= new DateTime(2013,6,5,5,0,0), EndTime= new
DateTime(2013,6,5,5,30,0), Subject="Meet the doc", Location="Hutchison
road", AllDay=false });
```

Recurrence Appointment

Recurring appointments can generate on a daily, weekly, monthly, or yearly interval. By setting [RecurrenceRule](#) and activating the [IsRecursive](#) property in [Schedule](#) appointment which will build recurring appointments.

Recurrence Rule

The [RecurrenceRule](#) is a string value that includes the details of the recurrence appointments such as repeat form - daily/weekly/monthly/yearly, how many times it needs to be repeated, the duration of the interval and also the time period to make the appointment, etc.

[RecurrenceRule](#) has the following properties and the Scheduler control makes the recurrence appointments with their respective time period.

PropertyName	Purpose
FREQ	Maintains the repeat type value of appointments. (Example: Daily, Weekly, Monthly, Yearly, Every week day) Example: FREQ=DAILY;INTERVAL=1

INTERVAL	Maintains the interval value of appointments. For example, when you create the daily appointment at an interval of 2, the appointments are rendered on the days Monday, Wednesday and Friday. (creates the appointment on all days by leaving the interval of one day gap) Example: FREQ=DAILY;INTERVAL=1
COUNT	It holds the appointment's count value. For example, when the recurrence appointment count value is 10, it means 10 appointments are created in the recurrence series. Example: FREQ=DAILY;INTERVAL=1;COUNT=10
UNTIL	This property is used to store the recurrence end date value. For example, when you set the end date of appointment as 6/30/2014, the UNTIL property holds the end date value when the recurrence actually ends. Example: FREQ=DAILY;INTERVAL=1;UNTIL=8/25/2014
BYDAY	It holds the "DAY" values of an appointment to render. For example, when you create the weekly appointment, select the day(s) from the day options (Monday/Tuesday/Wednesday/Thursday/Friday/Saturday/Sunday). When Monday is selected, the first two letters of the selected day "MO" is stored in the "BYDAY" property. When you select multiple days, the values are separated by commas. Example: FREQ=WEEKLY;INTERVAL=1;BYDAY=MO,WE;COUNT=10
BYMONTHDAY	This property is used to store the date value of the Month while creating the Month recurrence appointment. For example, when you create a Monthly recurrence appointment in the date 3, it means the BYMONTHDAY holds the value 3 and creates the appointment on 3rd day of every month. Example: FREQ=MONTHLY;BYMONTHDAY=3;INTERVAL=1;COUNT=10
BYMONTH	This property is used to store the index value of the selected Month while creating the yearly appointments. For example, when you create the yearly appointment in the Month June, it means the index value for June month is 6 and it is stored in the BYMONTH field. The appointment is created on every 6th month of a year. Example: FREQ=YEARLY;BYMONTHDAY=16;BYMONTH=6;INTERVAL=1;COUNT=10
BYSETPOS	This property is used to store the index value of the week. For example, when you create the monthly appointment in second week of the month, the index value of the second week (2) is stored in BYSETPOS. Example: FREQ=MONTHLY;BYDAY=MO;BYSETPOS=2;UNTIL=8/11/2014

Recurrence Pattern

Scheduler supports four types of recurrence patterns. You can set this recurrence pattern using [RecurrenceType](#) property of [RecurrenceRule](#).

RecurrenceType	RecurrenceProperties	Description
Daily	IsDailyEveryNDays	Gets or sets a value indicating whether the recurrence should be set based on specified day interval.
	DailyNDays	Gets or sets the day interval on which recurrence has to be set.

Weekly	WeeklyEveryNWeeks	Gets or sets the week interval on which recurrence has to be set.
	IsWeeklySunday	Gets or sets a value indicating whether the recurrence should be applied on Sundays with specified week interval.
	IsWeeklyMonday	Gets or sets a value indicating whether the recurrence should be applied on Mondays with specified week interval.
	IsWeeklyTuesday	Gets or sets a value indicating whether the recurrence should be applied on Tuesdays with specified week interval.
	IsWeeklyWednesday	Gets or sets a value indicating whether the recurrence should be applied on Wednesdays with specified week interval.
	IsWeeklyThursday	Gets or sets a value indicating whether the recurrence should be applied on Thursdays with specified week interval.
	IsWeeklyFriday	Gets or sets a value indicating whether the recurrence should be applied on Fridays with specified week interval.
	IsWeeklySaturday	Gets or sets a value indicating whether the recurrence should be applied on Saturdays with specified week interval.
Monthly	IsMonthlySpecific	Gets or sets a value indicating whether the recurrence has to be set for particular month day i.e. <code>MonthlySpecificMonthDay</code>
	MonthlyEveryNMonths	Gets or sets the month interval on which recurrence has to be set.
	MonthlyNthWeek	Gets or sets the week of month on which recurrence has to be set.
	MonthlySpecificMonthDay	Gets or sets the day on which recurrence has to be set for every month.
	MonthlyWeekDay	Gets or sets the day of week on which monthly recurrence has to be set.
Yearly	IsYearlySpecific	Gets or sets a value indicating whether the recurrence should be set based on specific year interval.
	YearlyEveryNYears	Gets or sets the year interval on which recurrence has to be set.

	YearlyGenericMonth	Gets or sets the generic month of year on which recurrence has to be set.
	YearlyNthWeek	Gets or sets the week of year on which recurrence has to be set.
	YearlySpecificMonth	Gets or sets the specific month of year on which recurrence has to be set.
	YearlySpecificMonthDay	Gets or sets the specific day of month on which yearly recurrence has to be set.
	YearlyWeekDay	Gets or sets the day of week on which yearly recurrence has to be set.
Common	IsRangeEndDate	Gets or sets a value indicating whether the date should be specified for ending the recurrence.
	IsRangeNoEndDate	Gets or sets a value indicating whether the recurrence should be ended.
	IsRangeRecurrenceCount	Gets or sets a value indicating whether the count of recurrence should be set.
	RangeEndDate	Gets or sets the date to end the recurrence.
	RangeStartDate	Gets or sets the date to start the recurrence.
	RangeRecurrenceCount	Gets or sets the count for recurring appointment.

Recurrence Rule Generator

[RRuleGenerator](#) method is used to construct the recurrence rule that can be found in the [ScheduleHelper](#) of the Scheduler control. Assign the generated recurrence rule to the appointment property called **RecurrenceRule** that assign the recurrence properties to the appointment. Or from any iCal directory, the client may apply recurrence rule directly.

Applying Recurrence to Appointments

Use [RRuleGenerator](#) method to generate the recurrence rule.

C#

```
// Daily Recursive Appointment
ScheduleAppointment SchApp = new ScheduleAppointment();
SchApp.Subject = "Team Meeting";
SchApp.Notes = "Daily Recurrence";
SchApp.Location = "Meeting Hall 1";
SchApp.StartTime = currentDate;
SchApp.EndTime = currentDate.AddHours(4);
SchApp.AppointmentBackground = new SolidColorBrush((Color.FromArgb(0xFF,
0xD8, 0x00, 0x73)));
SchApp.IsRecursive = true;
// Setting Recurrence Properties
RecurrenceProperties RecProp = new RecurrenceProperties();
RecProp.RecurrenceType = RecurrenceType.Daily;
```

```

RecProp.IsDailyEveryNDays = true;
RecProp.DailyNDays = 2;
RecProp.IsRangeRecurrenceCount = true;
RecProp.IsRangeNoEndDate = false;
RecProp.IsRangeEndDate = false;
RecProp.RangeRecurrenceCount = 100;
// Generating RRULE using ScheduleHelper
SchApp.RecurrenceRule = ScheduleHelper.RRuleGenerator(RecProp,
SchApp.StartTime, SchApp.EndTime);
AppCollection.Add(SchApp);
Schedule.Appointments = AppCollection;

```



Creating Custom Recurrence Appointment using Recurrence Builder

you need to create a custom class **Meeting** with mandatory fields **From**, **To**, **EventName** and **RecurrenceRule** to create custom recurrence appointment.

C#

```

/// <summary>
/// Represents custom data properties.
/// </summary>
public class Meeting
{
    public string EventName { get; set; }
    public DateTime From { get; set; }
    public DateTime To { get; set; }
    public Brush Color { get; set; }
    public string RecurrenceRule { get; set; }
    public bool IsRecursive { get; set; }
}

```

Note: You can inherit this class from **INotifyPropertyChanged** for dynamic changes in custom data.

You can map those properties of Meeting class with our SfSchedule control by using [ScheduleAppointmentMapping](#).

XML


```
<syncfusion:SfSchedule x:Name="schedule" ScheduleType="Month"
DataSource="{Binding Meetings}">
<syncfusion:SfSchedule.AppointmentMapping>
<syncfusion:ScheduleAppointmentMapping
SubjectMapping="EventName"
AppointmentBackgroundMapping="Color"
StartTimeMapping="From"
EndTimeMapping="To"
IsRecursiveMapping="IsRecursive"
RecurrenceRuleMapping="RecurrenceRule">
</syncfusion:ScheduleAppointmentMapping>
</syncfusion:SfSchedule.AppointmentMapping>
</syncfusion:SfSchedule>
```

C#

```
// Schedule data mapping for custom appointments
ScheduleAppointmentMapping dataMapping = new ScheduleAppointmentMapping();
dataMapping.SubjectMapping = "EventName";
dataMapping.StartTimeMapping = "From";
dataMapping.EndTimeMapping = "To";
dataMapping.AppointmentBackgroundMapping = "Color";
dataMapping.IsRecursiveMapping = "IsRecursive";
dataMapping.RecurrenceRuleMapping = "RecurrenceRule";
schedule.AppointmentMapping = dataMapping;
```

You can schedule recurring meetings for daily, weekly, monthly, or yearly interval by setting **RecurrenceRule** of **Meeting** class. Create meetings of type **ObservableCollection<Meeting>** and assign those appointments collection **Meetings** to the **DataSource** property which is of **IEnumerable** type.

C#

```
// Creating instance for custom appointment class
Meeting meeting = new Meeting();
// Setting start time of an event
meeting.From = new DateTime(2017, 06, 11, 10, 0, 0);
// Setting end time of an event
meeting.To = meeting.From.AddHours(2);
// Setting start time for an event
meeting.EventName = "Client Meeting";
// Setting color for an event
meeting.Color = new SolidColorBrush(Color.FromArgb(0xFF, 0xD8, 0x00, 0x73));
// Enable to generate the recurrence.
meeting.IsRecursive = true;
// Creating recurrence rule
RecurrenceProperties recurrenceProperties = new RecurrenceProperties();
recurrenceProperties.RecurrenceType = RecurrenceType.Weekly;
recurrenceProperties.IsWeeklyTuesday = true;
recurrenceProperties.IsWeeklyWednesday = true;
recurrenceProperties.IsWeeklyThursday = true;
recurrenceProperties.RangeRecurrenceCount = 10;
recurrenceProperties.RecurrenceRule =
ScheduleHelper.RRuleGenerator(recurrenceProperties, meeting.From,
meeting.To);
```

```
// Setting recursive rule for an event
meeting.RecurrenceRule = recurrenceProperties.RecurrenceRule;
// Creating instance for collection of custom appointments
var Meetings = new ObservableCollection<Meeting>();
// Adding a custom appointment in CustomAppointmentCollection
Meetings.Add(meeting);
// Adding custom appointments in ItemsSource of SfSchedule
schedule.ItemsSource = Meetings;
```

◀ ▶ June 2017						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
28	29	30	31	01	02	03
04	05	06	07	08	09	10
11	12	13 Client Meeting	14 Client Meeting	15 Client Meeting	16	17
18	19	20 Client Meeting	21 Client Meeting	22 Client Meeting	23	24
25	26	27 Client Meeting	28 Client Meeting	29 Client Meeting	30	01

Download demo from [GitHub](#).

How to get the Recurrence editor field values from recurrence rule?

You can get the recurrence properties from recurrence rule using the [RRuleParser](#) method of Scheduler.

C#

```
DateTime dateTime = new DateTime(2018, 5, 7, 9, 0, 0);
RecurrenceProperties recurrenceProperties =
schedule.RRuleParser("FREQ=DAILY; INTERVAL=1; COUNT=3", dateTime);
```

Recurrence properties retrieved from above method,

recurrenceProperties.RecurrenceType as RecurrenceType.Daily

recurrenceProperties.DailyNDays as 1

recurrenceProperties.IsDailyEveryNDays as true

recurrenceProperties.RangeRecurrenceCount as 3

How to get the occurrences date time list of recurring appointment from recurrence rule?

Use the [GetRecurrenceDateTimeCollection](#) feature of Scheduler to get the occurrence date time list of recurring appointment from the recurrence rule.

C#

```
DateTime dateTime = new DateTime(2018, 5, 7, 9, 0, 0);
IEnumerable<DateTime> dateCollection =
schedule.GetRecurrenceDateTimeCollection("FREQ=DAILY; INTERVAL=1; COUNT=3",
dateTime);
```

Following occurrence dates can be retrieved from the given RRULE,

```
var date0 = 5/7/2018;
```

```
var date1 = 5/8/2018;
```

```
var date2 = 5/9/2018;
```

[RecursiveExceptionDates](#)

[ScheduleAppointment](#) supports to delete any recurrence appointment which exception from the recurrence pattern appointment by adding exception dates in [RecursiveExceptionDates](#) collection to the schedule appointment.

C#

```
// Create the new exception date.
var exceptionDate = new DateTime(2017, 09, 07);
var recurrenceAppointment = new ScheduleAppointment()
{
    StartTime = new DateTime(2017, 09, 01, 10, 0, 0),
    EndTime = new DateTime(2017, 09, 01, 12, 0, 0),
    Subject = "Occurs Daily",
    Color=Color.Blue
};
// Creating recurrence rule
recurrenceAppointment.RecurrenceRule = "FREQ=DAILY;COUNT=20";
// Add RecursiveExceptionDates to appointment.
recurrenceAppointment.RecursiveExceptionDates = new
ObservableCollection<DateTime>()
{
    exceptionDate
};
this.schedule.Appointments.Add(recurrenceAppointment);
```

Note: You can also update the [RecursiveExceptionDates](#) collection dynamically.

Delete occurrence from recurrence pattern dynamically or add/remove exception dates to recurrence pattern dynamically

You can also delete or add deleted occurrence from the recurrence pattern appointment by adding/removing exception date from the [RecursiveExceptionDates](#) collection.

C#

```
var recurrenceAppointment = scheduleAppointmentCollection[0];
recurrenceAppointment.RecursiveExceptionDates.RemoveAt(0);
recurrenceAppointment.RecursiveExceptionDates.Add(new DateTime(2017, 09,
05));
```

Create recurrence exceptions for custom appointment

You can add/remove the recurrence exception appointments and recurrence exception dates to the CustomAppointment, You can create a custom class Meeting(refer [DataBinding](#)) with mandatory field RecurrenceExceptionDates.

C#

```

/// <summary>
/// Represents custom data properties.
/// </summary>
public class Meeting
{
    public string EventName { get; set; }
    public DateTime From { get; set; }
    public DateTime To { get; set; }
    public Brush Color { get; set; }
    public string RecurrenceRule { get; set; }
    public bool IsRecursive { get; set; }
    public ObservableCollection<DateTime> RecurrenceExceptionDates { get; set; }
}

```

Note: You can inherit this class from `INotifyPropertyChanged` for dynamic changes in custom data.

Using [AppointmentMapping](#) and [ScheduleAppointmentMapping](#) to map certain properties of the Meeting class with our Scheduler control.

XML

```

<syncfusion:SfSchedule x:Name="schedule" ScheduleType="Month"
DataSource="{Binding Meetings}">
  <syncfusion:SfSchedule.AppointmentMapping>
    <syncfusion:ScheduleAppointmentMapping
      SubjectMapping="EventName"
      AppointmentBackgroundMapping="Color"
      StartTimeMapping="From"
      EndTimeMapping="To"
      IsRecursiveMapping="IsRecursive"
      RecurrenceRuleMapping = "RecurrenceRule"
      RecursiveExceptionDatesMapping="RecurrenceExceptionDates">
    </syncfusion:ScheduleAppointmentMapping>
  </syncfusion:SfSchedule.AppointmentMapping>
</syncfusion:SfSchedule>

```

C#

```

// Schedule data mapping for custom appointments
ScheduleAppointmentMapping dataMapping = new ScheduleAppointmentMapping();
dataMapping.SubjectMapping = "EventName";
dataMapping.StartTimeMapping = "From";
dataMapping.EndTimeMapping = "To";
dataMapping.IsRecursiveMapping="IsRecursive";
dataMapping.RecurrenceRuleMapping = "RecurrenceRule";
dataMapping.RecursiveExceptionDatesMapping = "RecurrenceExceptionDates";
schedule.AppointmentMapping = dataMapping;
ObservableCollection<Meeting> Meetings = new
ObservableCollection<Meeting>();
// Create the new exception date.
var exceptionDate = new DateTime(2017, 09, 07);
var recurrenceAppointment = new Meeting()
{
    From = new DateTime(2017, 09, 01, 10, 0, 0),
    To = new DateTime(2017, 09, 01, 12, 0, 0),
    EventName = "Occurs Daily",
}

```

```

IsRecursive = true,
Color = new SolidColorBrush(Color.FromArgb(0xFF, 0xD8, 0x00, 0x73)),
RecurrenceExceptionDates = new ObservableCollection<DateTime>()
{
    exceptionDate
};
// Creating recurrence rule
recurrenceAppointment.RecurrenceRule = "FREQ=DAILY;COUNT=20";
Meetings.Add(recurrenceAppointment);
this.schedule.ItemsSource = Meetings;

```

◀ ▶ September 2017

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
27	28	29	30	31	01 Occurs Daily	02 Occurs Daily
03 Occurs Daily	04 Occurs Daily	05 Occurs Daily	06 Occurs Daily	07	08 Occurs Daily	09 Occurs Daily
10 Occurs Daily	11 Occurs Daily	12 Occurs Daily	13 Occurs Daily	14 Occurs Daily	15 Occurs Daily	16 Occurs Daily
17 Occurs Daily	18 Occurs Daily	19 Occurs Daily	20 Occurs Daily	21	22	23
24	25	26	27	28	29	30

Download demo from [GitHub](#).

Import-and-Export in WPF Schedule (Classic)

The schedule control allows users to export or import appointments and events as an .ics files. Appointments and events scheduled with the control can be exported as an .ics file and opened or imported in other schedulers such as Microsoft Outlook, Google Calendar, or any other scheduler supporting .ics files.

Similarly, files exported from other schedulers can also be imported into Essential Schedule. The code given below demonstrates how to import and export .ics files.

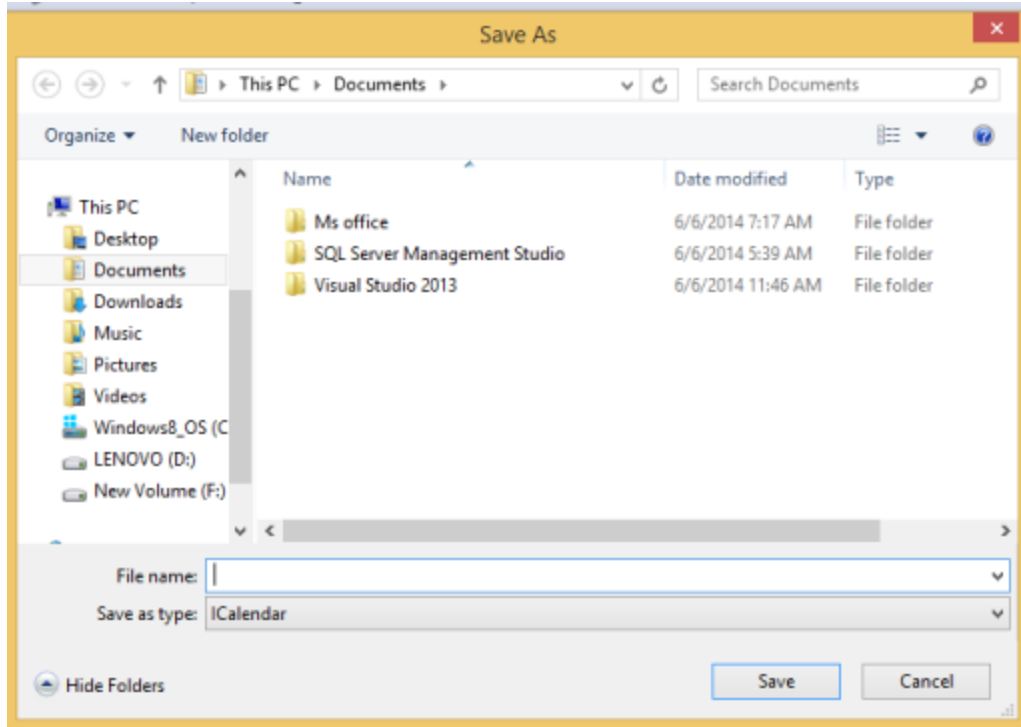
Exporting in the schedule control

C#

```
Schedule.ExportICS();
```

When the export function is called, a dialog box opens for selecting the location where the file has to be exported.

The exported .ics file will be saved in specified location. These exported files can then be imported in any other scheduler, such as Microsoft Outlook. Similarly, you can import .ics files from other schedulers.

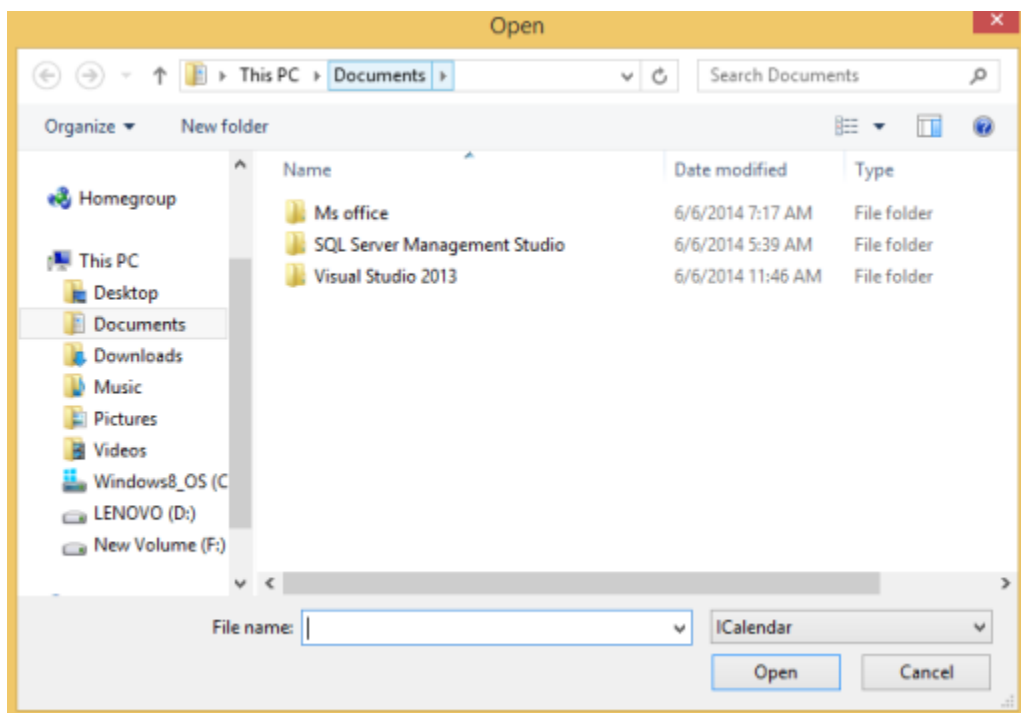


Importing in the schedule control

C#

```
Schedule.ImportICS();
```

If the imported calendar has already been given a name, it can be imported into the schedule control by using the method `ImportICS`, which will open a dialog for browsing to the location, as seen below.



Resources in WPF Schedule (Classic)

The Scheduler allows to define resources that can be assigned to appointments. Resources let you associate additional information with your appointments. The schedule can group appointments based on the resources associated with them. Appointments will be grouped based on the resource associated with them only when both the [Resource](#) and [ScheduleResourceTypeCollection](#) properties are set, and also when the value set for a Resource matches with any types specified in the [ScheduleResourceTypeCollection](#) property.

For example, end user can create appointments for different doctors.

XML

```
<Schedule:SfSchedule Name="Schedule1" ScheduleType="Week"
Background="WhiteSmoke" Resource="Doctors" >
<Schedule:SfSchedule.ScheduleResourceTypeCollection >
<Schedule:ResourceType TypeName="Doctors">
<Schedule:Resource DisplayName="Dr.Jacob John, M.D "
ResourceName="Dr.Jacob"/>
<Schedule:Resource DisplayName="Dr.Darsy Mascio, M.D"
ResourceName="Dr.Darsy"/>
</Schedule:ResourceType>
</Schedule:SfSchedule.ScheduleResourceTypeCollection>
</Schedule:SfSchedule>
```

C#

```
ScheduleAppointment app = new ScheduleAppointment() { StartTime =
currentDate, EndTime = currentDate.AddHours(ran.Next(0, 2)), Subject =
subject[count % subject.Length], Location = "Chennai", AppointmentBackground
= brush[m % 3] };
app.ResourceCollection.Add(new Resource() { ResourceName = "Dr.Jacob",
TypeName = "Doctors" });
ScheduleAppointment app1 = new ScheduleAppointment() { StartTime = nextDate,
EndTime = nextDate.AddHours(ran.Next(0, 2)), Subject = subject[count %
subject.Length], Location = "Chennai", AppointmentBackground = brush[(m + 2)
% 3] };
app1.ResourceCollection.Add(new Resource() { ResourceName = "Dr.Darsy",
TypeName = "Doctors" });
Schedule1.Appointments.Add(app);
Schedule1.Appointments.Add(app1);
```



Creating Resource for Schedule

Let's see the steps to add resources in Scheduler.

The first step is set the [Resource](#) in Scheduler

XML

```
<schedule:SfSchedule Resource="Doctors" >
```



```
...
</schedule:SfSchedule>
```

After this, we need to create a [ScheduleResourceTypeCollection](#), to assign the [ResourceType](#):

XML

```
<schedule:SfSchedule Resource="Doctors" >
...
<schedule:SfSchedule.ScheduleResourceTypeCollection>
...
</schedule:SfSchedule.ScheduleResourceTypeCollection>
</schedule:SfSchedule>
```

Adding ResourceType to a resource collection

After Creating the `ScheduleResourceTypeCollection` add the [ResourceType](#), here we create the example for `ResourceType` as Doctors.

XML

```
<schedule:SfSchedule Resource="Doctors" >
<schedule:SfSchedule.ScheduleResourceTypeCollection >
<schedule:ResourceType TypeName="Doctors">
. . .
</schedule:ResourceType>
</schedule:SfSchedule.ScheduleResourceTypeCollection>
</schedule:SfSchedule>
```

Adding a Resource to a ResourceType

The next step is create and assign resources to `ResourceType`.

XML

```
<schedule:SfSchedule Resource="Doctors" >
<schedule:SfSchedule.ScheduleResourceTypeCollection >
<schedule:ResourceType TypeName="Doctors">
<schedule:Resource DisplayName="Dr.Jacob John, M.D "
ResourceName="Dr.Jacob"/>
<schedule:Resource DisplayName="Dr.Darsy Mascio, M.D"
ResourceName="Dr.Darsy"/>
</schedule:ResourceType>
</schedule:SfSchedule.ScheduleResourceTypeCollection>
</schedule:SfSchedule>
```

Creating appointments by specifying the resource

You can add an appointments to group of added resources.

C#

```
ScheduleAppointment app = new ScheduleAppointment() { StartTime =
currentDate, EndTime = currentDate.AddHours(ran.Next(0, 2)), Subject =
subject[count % subject.Length], Location = "Chennai", AppointmentBackground
= brush[m % 3] };
```

```
app.ResourceCollection.Add(new Resource() { ResourceName = "Dr.Jacob",
TypeName = "Doctors" });
ScheduleAppointment appl = new ScheduleAppointment() { StartTime = nextDate,
EndTime = nextDate.AddHours(ran.Next(0, 2)), Subject = subject[count %
subject.Length], Location = "Chennai", AppointmentBackground = brush[(m + 2)
% 3] };
appl.ResourceCollection.Add(new Resource() { ResourceName = "Dr.Darsy",
TypeName = "Doctors" });
Schedule1.Appointments.Add(app);
Schedule1.Appointments.Add(appl);
```

Adding Resources in code behind

Refer to the following code to add a resources in the code behind

C#

```
SfSchedule schedule = new SfSchedule();
schedule.ScheduleType = ScheduleType.Week;
schedule.DayHeaderOrder = DayHeaderOrder.OrderByDate;
ResourceType resourceType = new ResourceType { TypeName = "Doctor" };
resourceType.ResourceCollection.Add(new Resource { DisplayName = "Dr.Jacob",
ResourceName = "Dr.Jacob", });
resourceType.ResourceCollection.Add(new Resource { DisplayName = "Dr.Darsy",
ResourceName = "Dr.Darsy" });
schedule.ScheduleResourceTypeCollection = new
ObservableCollection<ResourceType> { resourceType };
schedule.Resource = "Doctor";
this.grid.Children.Add(schedule);
```

SubResource Support

This feature enables users to view appointments based on their subcategory only in day and week views. Using this feature, the end user can group appointments under various subcategories (resources).

DayHeaderOrder property

[DayHeaderOrder](#) property is used to set the order by which resources have to be displayed

Property	Description
SubResourceType	Gets or sets the ResourceType value which is a SubResource type of its parent Resource type.

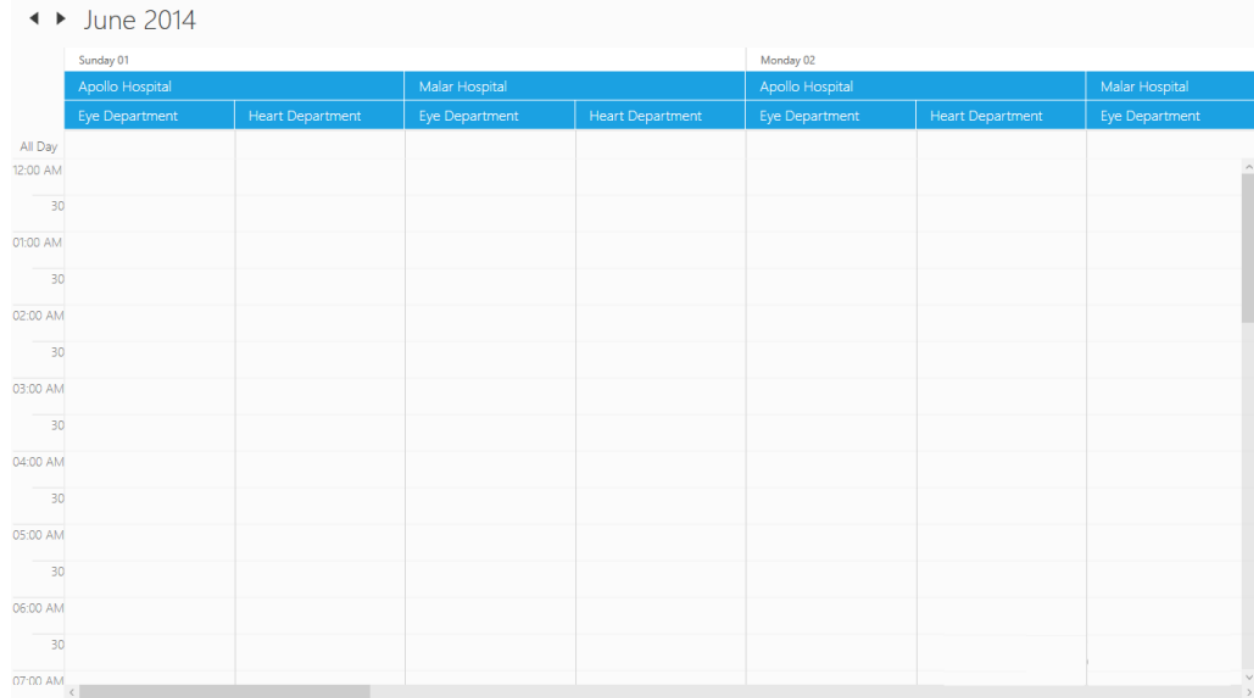
XML

```
<schedule:SfSchedule x:Name="schedule1" ScheduleType="Week"
Resource="Hospital" DayHeaderOrder="OrderByDate">
<schedule:SfSchedule.ScheduleResourceTypeCollection>
<schedule:ResourceType TypeName="Hospital">
<schedule:ResourceType.ResourceCollection>
<schedule:Resource DisplayName="Apollo
Hospital"ResourceName="ApolloHospital"/>
<schedule:Resource DisplayName="Malar
Hospital"ResourceName="MalarHospital"/>
</schedule:ResourceType.ResourceCollection>
<schedule:ResourceType.SubResourceType>
```

```
<schedule:ResourceType TypeName="Department">
  <schedule:ResourceType.ResourceCollection>
    <schedule:Resource DisplayName="Eye Department" ResourceName="Eye"/>
    <schedule:Resource DisplayName="Heart Department" ResourceName="Heart"/>
  </schedule:ResourceType.ResourceCollection>
</schedule:ResourceType>
</schedule:ResourceType.SubResourceType>
</schedule:ResourceType>
</schedule:SfSchedule.ScheduleResourceTypeCollection>
</schedule:SfSchedule>
```

C#

```
SfSchedule schedule = new SfSchedule();
schedule.ScheduleType = ScheduleType.Week;
schedule.DayHeaderOrder = DayHeaderOrder.OrderByDate;
ResourceType resourceType = new ResourceType { TypeName = "Hospital" };
ResourceType subResourceType = new ResourceType { TypeName = "Department" };
resourceType.ResourceCollection.Add(new Resource { DisplayName = "Apollo
Hospital", ResourceName = "ApolloHospital", });
resourceType.ResourceCollection.Add(new Resource { DisplayName = "Malar
Hospital", ResourceName = "MalarHospital" });
subResourceType.ResourceCollection.Add(new Resource { DisplayName = "Eye
Department", ResourceName = "Eye" });
subResourceType.ResourceCollection.Add(new Resource { DisplayName = "Heart
Department", ResourceName = "Heart" });
schedule.ScheduleResourceTypeCollection = new
ObservableCollection<ResourceType> { resourceType };
schedule.ScheduleResourceTypeCollection = new
ObservableCollection<ResourceType> { subResourceType };
schedule.Resource = "Hospital";
schedule.Resource = "Department";
this.grid.Children.Add(schedule);
```



N Number of Resources in Day View

This feature supports to display **N** number of resources in the Schedule view. You can achieve this by specifying the count of resources that needs to be displayed per view. This support is offered for **Day** view alone.

This support can be enabled by using property [DayViewColumnCount](#) in **SfSchedule**. By default, its value is “zero”.

Property	Type	Description
DayViewColumnCount	int	Gets or sets a value to specify the number of resources that need to be shown in the view.

Example:

In the following code example, [DayViewColumnCount](#) is “two”, so the Scheduler displays two resources in the view. This count is maintained while scrolling to view the other resources.

XML

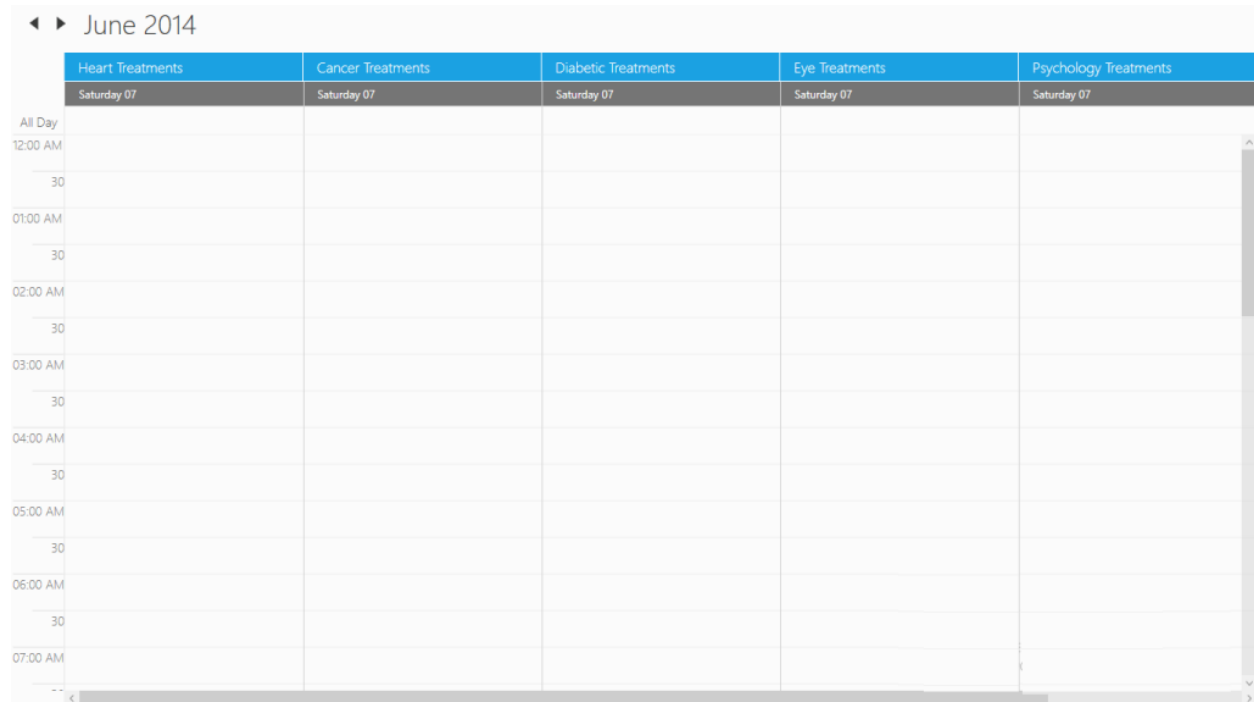
```
<syncfusion:SfSchedule x:Name="Schedule" ScheduleType="Day"
Resource="Doctors" DayViewColumnCount="2">
<syncfusion:SfSchedule.ScheduleResourceTypeCollection>
<syncfusion:ResourceType TypeName="Doctors">
<syncfusion:Resource ResourceName="Res1" DisplayName="Heart Treatments"/>
<syncfusion:Resource ResourceName="Res2" DisplayName="Cancer Treatments"/>
<syncfusion:Resource ResourceName="Res3" DisplayName="Diabetic Treatments"/>
<syncfusion:Resource ResourceName="Res4" DisplayName="Eye Treatments"/>
<syncfusion:Resource ResourceName="Res5" DisplayName="Psychology
Treatments"/>
<syncfusion:Resource ResourceName="Res6" DisplayName="Dermatology
Treatments"/>
</syncfusion:ResourceType>
</syncfusion:SfSchedule.ScheduleResourceTypeCollection>
</syncfusion:SfSchedule>
```

```
</syncfusion:ResourceType>
</syncfusion:SfSchedule.ScheduleResourceTypeCollection>
</syncfusion:SfSchedule>
```

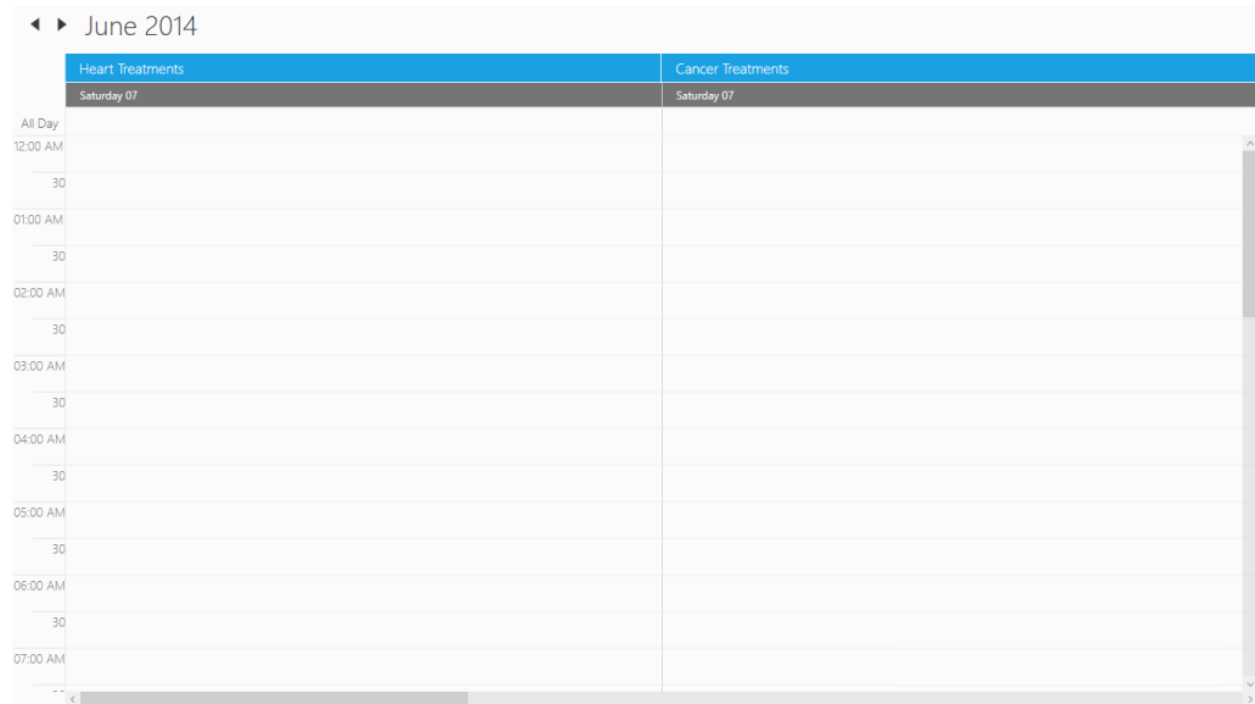
C#

```
SfSchedule schedule = new SfSchedule();
schedule.ScheduleType = ScheduleType.Day;
schedule.DayViewColumnCount = 0;
ResourceType resourceType = new ResourceType { TypeName = "Doctors" };
resourceType.ResourceCollection.Add(new Resource { ResourceName = "Res1",
DisplayName = "Heart Treatments" });
resourceType.ResourceCollection.Add(new Resource { ResourceName = "Res2",
DisplayName = "Cancer Treatments" });
resourceType.ResourceCollection.Add(new Resource { ResourceName = "Res3",
DisplayName = "Diabetic Treatments" });
resourceType.ResourceCollection.Add(new Resource { ResourceName = "Res4",
DisplayName = "Eye Treatments" });
resourceType.ResourceCollection.Add(new Resource { ResourceName = "Res5",
DisplayName = "Psychology Treatments" });
resourceType.ResourceCollection.Add(new Resource { ResourceName = "Res6",
DisplayName = "Dermatology Treatments" });
schedule.ScheduleResourceTypeCollection = new
ObservableCollection<ResourceType> { resourceType };
schedule.Resource = "Doctors";
this.grid.Children.Add(schedule);
```

The following screenshot shows a view with default value of `DayViewColumnCount` property set to "Zero".



The following screenshot shows a view with default value of `DayViewColumnCount` property set to "Two".



Localization in WPF Schedule (Classic)

Localization is the process of customizing the user interface based on a culture specific to a particular country or region in order to display regional data. The culture is represented by a unique string, for example, —en-US|| for U.S. English and —fr|| for French (common).

Localization is the key feature that provides solutions to global customers with the help of localized resource files provided by the control. The Scheduler supports localization, and you can create a resource file for any culture to be applied in the schedule.

Set Current UI Culture to the Application

Application culture can be changed by setting [CurrentUICulture](#)

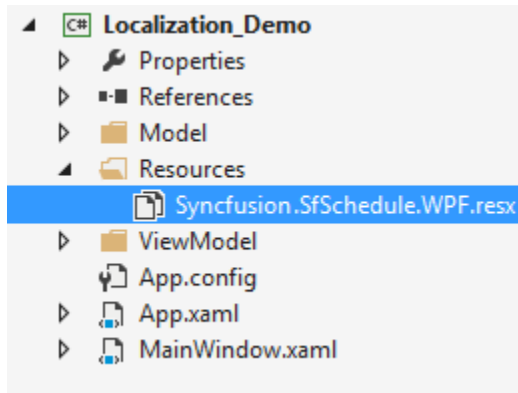
C#

```
public MainWindow()
{
    this.InitializeComponent();
    //Download resx files from GitHub. https://github.com/syncfusion/wpf-
    //demos/tree/master/Schedule/Localization/CS/Resources
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("fr-FR");
    System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-
    FR");
}
```

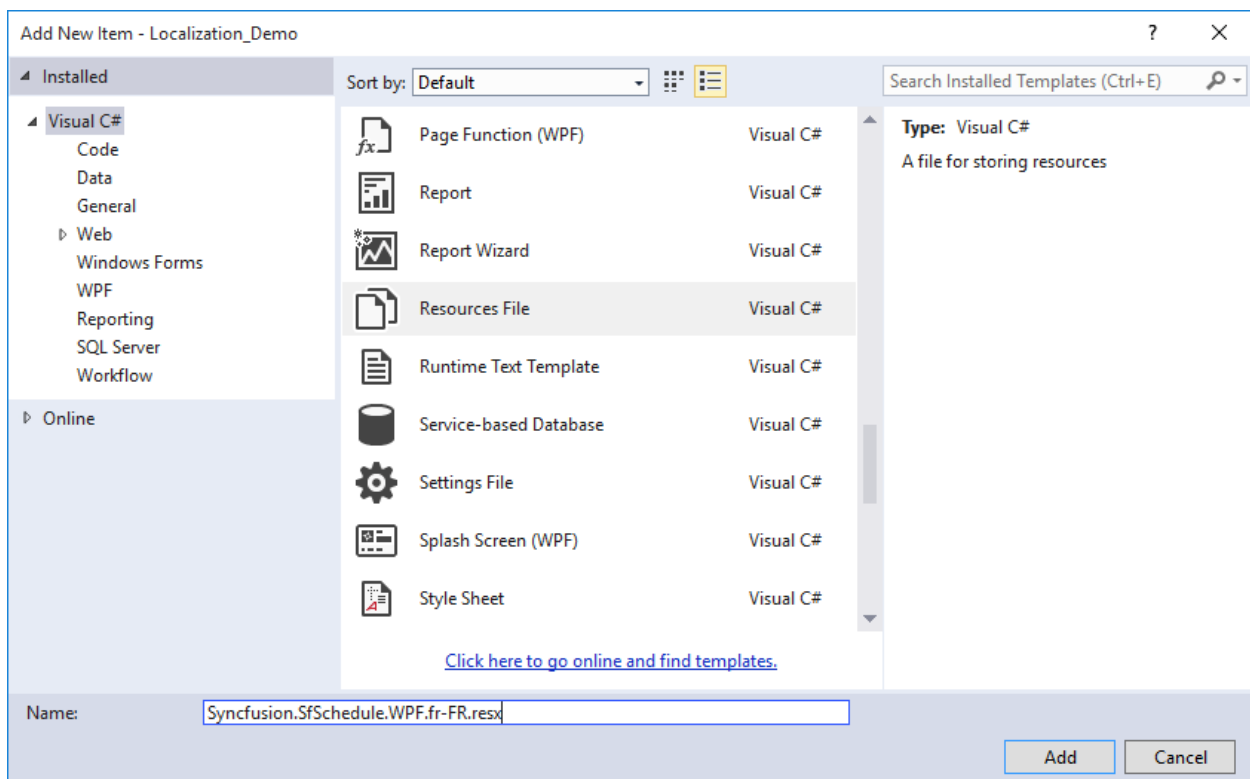
Localization using Resource file

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

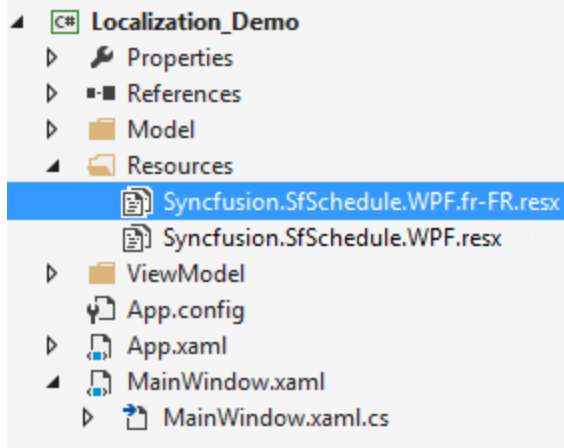
1.Create new folder and named as **Resources** in your application. 2.Add the default resource file of Scheduler into **Resources** folder. You can download the Syncfusion.SfSchedule.WPF.resx [here](#).



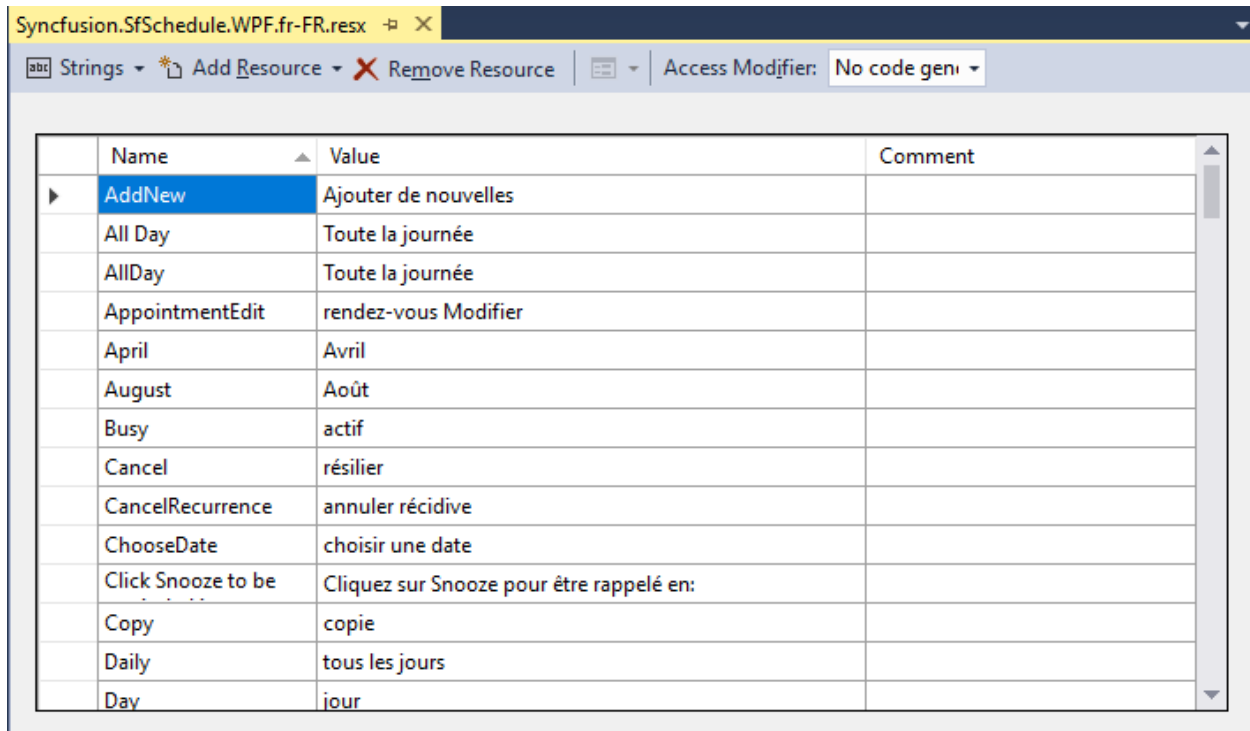
3.Right-click on the Resources folder, select **Add** and then **NewItem**. 4.In **Add New Item** wizard, select the **Resource File** option and name the filename as **Syncfusion.SfSchedule.WPF.<culture name>.resx**. For example, you have to give name as **Syncfusion.SfSchedule.WPF.de.resx** for German culture. 5.The culture name that indicates the name of language and country.

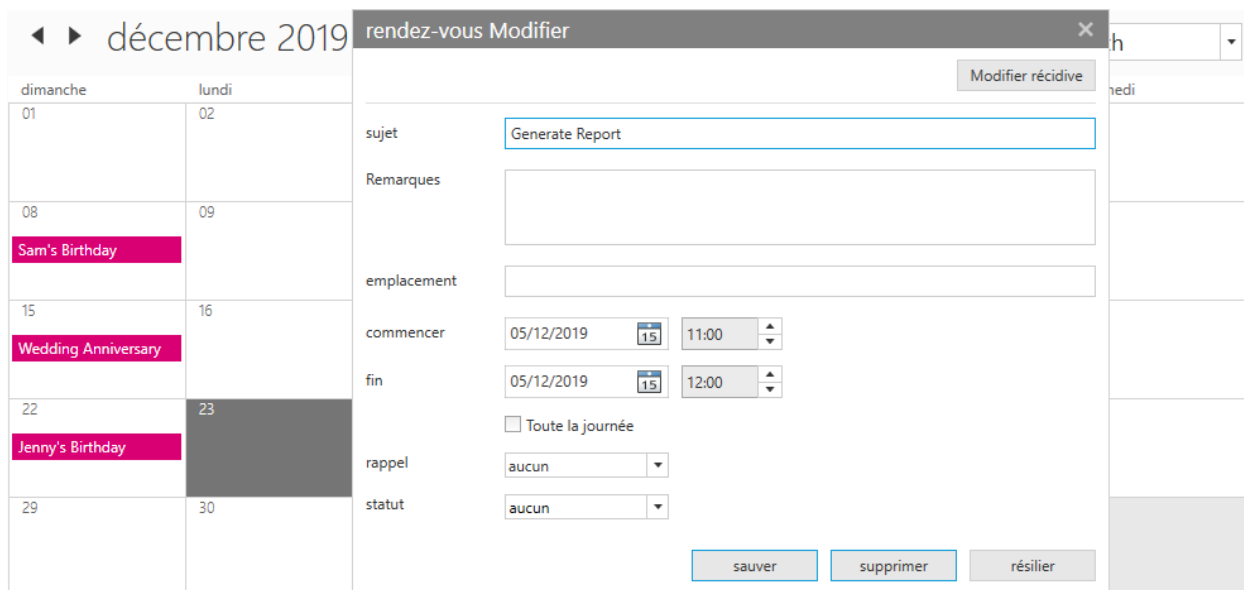


6.Now, select **Add** option to add the resource file in **Resources** folder.



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





You can get the sample from [here](#)

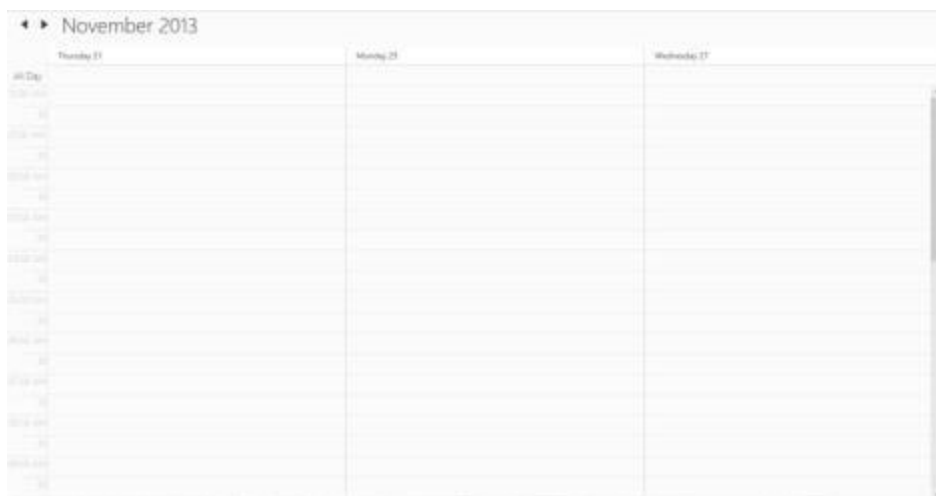
Visible-Dates-customization in WPF Schedule (Classic)

All views in the schedule have their own number of visible dates. The SfSchedule control allows users to view multiple dates in the day and time line views.

If users want to view particular dates in a single view, users can provide a DateTime collection to the ScheduleDateRange property to view the particular dates in the day and time line view types.

C#

```
ObservableCollection<DateTime> visibleDates
= newObservableCollection<DateTime>();
DateTime Date1 = new DateTime(2013, 9, 1);
DateTime Date2 = new DateTime(2013, 9, 22);
visibleDates.Add(Date1);
visibleDates.Add(Date2);
SfSchedule schedule = new SfSchedule();
schedule.ScheduleDateRange= visibleDates;
```





ContextMenuType in WPF Schedule (Classic)

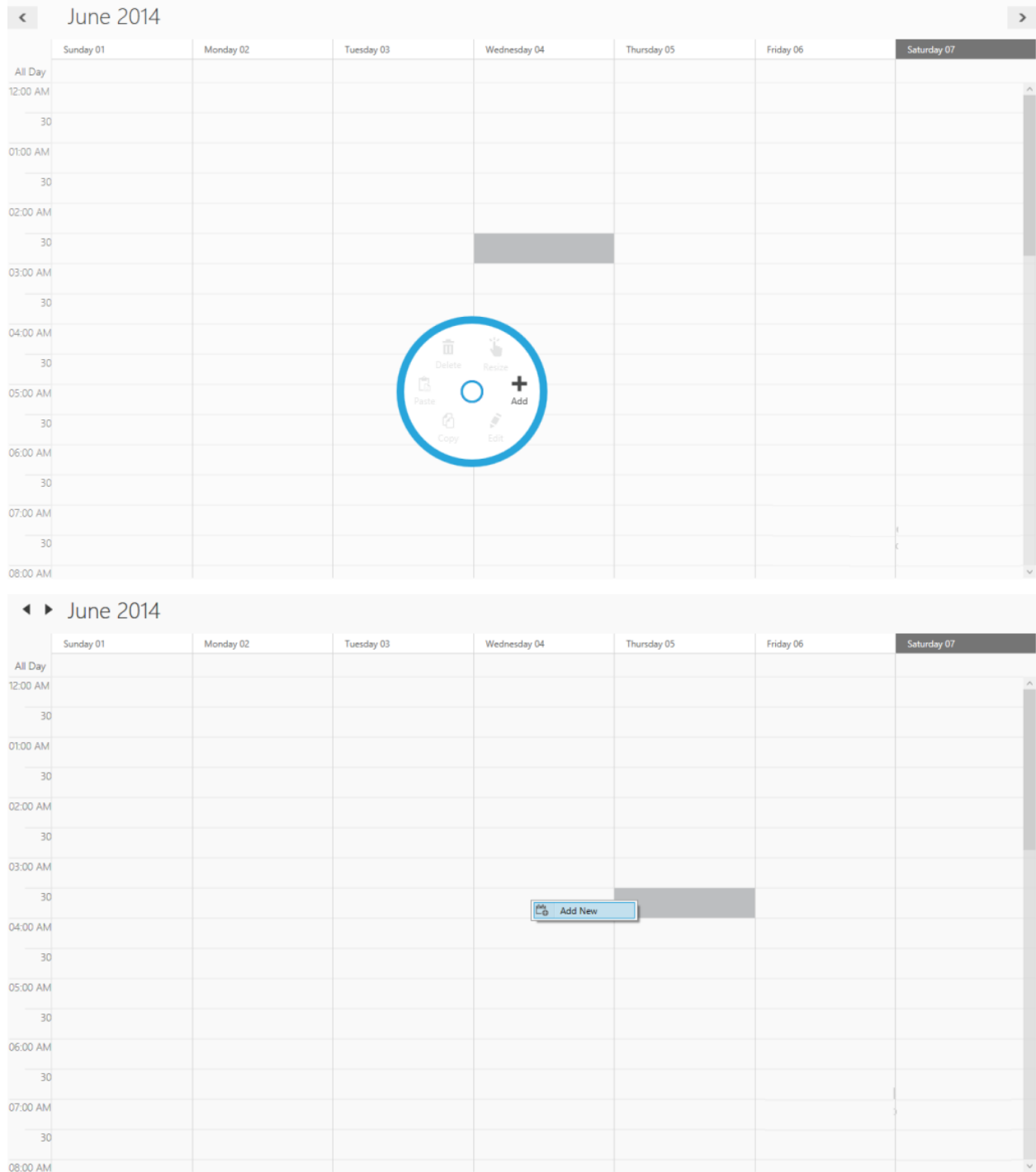
The collection of MenuItem elements can be organized by ContextMenuType property.

HTML

```
<Schedule:SfSchedule x:Name="schedule" ScheduleType="Week"
ContextMenuType="RadialMenu">
</Schedule:SfSchedule>
```

C#

```
SfSchedule schedule = new SfSchedule();
schedule.ScheduleType = ScheduleType.Week;
schedule.EnableTouch = true;
this.grid.Children.Add(schedule);
```



Commands in WPF Schedule (Classic)

Operations that are done by context menu can be performed using Schedule commands. The following actions of context menu can be handled by executing schedule commands in the application.

- Adding new appointment
- Editing appointment
- Deleting appointment

- Copying appointment
- Pasting appointment
- Drag and drop appointment

AddNewCommand

Using AddNewCommand of ScheduleCommands, a new appointment can be added in SfSchedule.

C#

```
ScheduleCommands.AddNewCommand.Execute(this.Schedule);
```

EditCommand

Using EditCommand of ScheduleCommands, an existing appointment of SfSchedule can be modified.

C#

```
ScheduleCommands.EditCommand.Execute(this.Schedule);
```

DeleteCommand

DeleteCommand can be used to delete an appointment in SfSchedule.

C#

```
ScheduleCommands.DeleteCommand.Execute(this.Schedule);
```

CopyCommand

Using CopyCommand of ScheduleCommands, a schedule appointment can be copied.

C#

```
ScheduleCommands.CopyCommand.Execute(this.Schedule);
```

PasteCommand

Using PasteCommand of ScheduleCommands, a copied schedule appointment can be pasted in SfSchedule.

C#

```
ScheduleCommands.PasteCommand.Execute(this.Schedule);
```

DragAndDropCommand

Using DragAndDropCommand of ScheduleCommands, an appointment can be dragged and dropped from one timeslot to another timeslot. Resizing operation can also be performed using this command.

C#

```
ScheduleCommands.DragAndDropCommand.Execute(this.Schedule);
```

Event in WPF Schedule (Classic)

Event can be used for various operation such as Appointment Editor Opening and Closed or while opening or closing the Context Menu and events ItemsSourceChanged, ScheduleTapped, ScheduleDoubleTapped are used for customization purposes.

- AppointmentEditorOpening - occurs when the appointment editor is opening. The AppointmentEditorOpening event handler receives two arguments:
- The sender argument contains the SfSchedule. This argument is of type object, but can be cast to the SfSchedule type.
- An AppointmentEditorOpeningEventArgs is a class. Via the AppointmentEditorOpeningEventArgs you can access the following properties:

Appointment - gets the appointment, this argument is of type object.

StartTime - gets the appointment start time.

Action – gets the Editor action such as Add or Edit or Delete.

SelectedResource – gets the list of Resources.

Cancel- set this Boolean property to True to canceling the event.

- AppointmentEditorClosed- occurs when the appointment editor is closed. The AppointmentEditorClosed event handler receives two arguments:
- The sender argument contains the SfSchedule. This argument is of type object, but can be cast to the SfSchedule type.
- An AppointmentEditorClosedEventArgs a class. Via the AppointmentEditorClosedEventArgs you can access the following properties:

OriginalAppointment - gets the Original appointment, this argument is of type object.

EditedAppointment - gets the edited appointment, this argument is of type object.

IsNew – set this Boolean property to True, when you Updating or Setting new appointment property.

Action – gets the Editor action such as Add or Edit or Delete.

- AppointmentCollectionChanged - occurs when the appointment collection changed. The AppointmentCollectionChanged event handler receives two arguments:
- The sender argument contains the SfSchedule. This argument is of type object, but can be cast to the SfSchedule type.
- Via the NotifyCollectionChangedEventArgs gets the collection are affected by the change.
- ContextMenuOpening- occurs when opening Context Menu. The ContextMenuOpening event handler receives two arguments:
- The sender argument contains the SfSchedule. This argument is of type object, but can be cast to the SfSchedule type.
- A ContextMenuOpeningEventArgs is a class. Via the ContextMenuOpeningEventArgs you can access the following properties:

Appointment - gets the appointment, this argument is of type object.

CurrentSelectedDate - gets the appointment current selected Date.

CurrentEventArgs – gets the Editor action such as Add or Edit or Delete.

SelectedResource – gets the list of Resources.

Cancel- set this Boolean property to True to disable the context menu.

- ContextMenuClosed- occurs when the Context Menu closed. The ContextMenuClosedEvent handler receives two arguments:
 - The sender argument contains the SfSchedule. This argument is of type object, but can be cast to the SfSchedule type.
 - A ContextMenuClosedEventArgs is a class.
- ItemsSourceChanged- occurs when item source changed. The ItemsSourceChanged event handler receives two arguments:
 - The sender argument contains the SfSchedule. This argument is of type object, but can be cast to the SfSchedule type.
 - Via the PropertyChangedEventArgs gets the name of the property that changed.
- ScheduleClick- occurs when tapping the schedule. The ScheduleClickEvent handler receives two arguments:
 - The sender argument contains the SfSchedule. This argument is of type object, but can be cast to the SfSchedule type.
 - ScheduleClickEventArgs is a class. Via the ScheduleClickEventArgs you can access the following properties:

Appointment - gets the appointment, this argument is of type object.

SelectedDate - gets the appointment current selected Date.

SelectedResource – gets the list of Resources.

- ScheduleDoubleClick- occurs when double tapping the schedule. The ScheduleDoubleClick event handler receives two arguments:
 - The sender argument contains the SfSchedule. This argument is of type object, but can be cast to the SfSchedule type.
 - ScheduleClickEventArgs is a class. Via the ScheduleClickEventArgs you can access the following properties:

Appointment - gets the appointment, this argument is of type object.

SelectedDate - gets the appointment current selected Date.

SelectedResource – gets the list of Resources.

SkinStorage (Classic)

WPF SkinStorage (Classic) Overview

The Skin Manager Framework provides a convenient way to give the appealing appearance to the WPF controls as well as the Syncfusion controls.

Feature summary

- 9 Built-In skins support for the Syncfusion controls as well as the Microsoft controls.
- Applying Custom color for the WPF controls by setting the Single property.
- Applying styles for dynamically added controls and derived controls.
- Overridden and non-overridden styles can be dynamically switched.
- Styles can be applied and overridden at the application level.

Built-in skins

Skins can be applied to the control by setting the VisualStyle property defined in the Skin Storage class. Set the VisualStyle property to the corresponding theme. This property can be set either in XAML or in C#.

Visual style property

Property	Description	Type	Data Type	Reference links
VisualStyle	Used to set Skins for the controls. The Built-In-Skins are as follows. <i>Office2003 Office2007Blue Office2007Black Office2007Silver ShinyRed Blend ShinyBlue SyncOrange VS2010 Office2010Blue Office2010Black Office2010Silver Metro Transparent</i>	Attached Property	String	Setting VisualStyle in XAMLSetting VisualStyle in C#

Setting VisualStyle in XAML

The following code snippet explains how to set the VisualStyle property in XAML.

1. Add the following namespace in the sample.

XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
```

2. Set the VisualStyle property for the control as shown below.

XML

```
<syncfusion:CalendarEdit syncfusion:SkinStorage.VisualStyle="Blend">
</syncfusion:CalendarEdit>
```

Setting VisualStyle in C#

You can also set the VisualStyle property in C# using SetVisualStyle.

The following code snippet explains how to set the VisualStyle property in C#.

1. Name the control using the Name attribute.

XML

```
<syncfusion:CalendarEdit Name="calendar">
</syncfusion:CalendarEdit>
```

2. Add the following line in code behind file.

C#

```
SkinStorage.SetVisualStyle(calendar, "Blend");
```

The output is displayed as shown below.



Calendar with Blend Style

Active color scheme

You can set the custom color for the WPF controls by using the `ActiveColorScheme` property defined in the Skin Manager class. This property can be set either in XAML or in C#.

ActiveColorScheme property

Property	Description	Type	Data Type	Reference links
ActiveColorScheme	Sets the custom color for the controls.	Attached Property	SolidColorBrush	Setting ActiveColorScheme property in XAML Setting ActiveColorScheme property in C#

Setting ActiveColorScheme property in XAML

The following code snippet explains how to set the `ActiveColorScheme` property in XAML.

1. Add the following namespace in the sample.

XML

```
xmlns:syncfusion=http://schemas.syncfusion.com/wpf
```

2. Set the ActiveColorScheme property for the control as shown below.

XML

```
<syncfusion:CalendarEdit Name="calendar"  
syncfusion:SkinManager.ActiveColorScheme="Red">  
</syncfusion:CalendarEdit>
```

Setting ActiveColorScheme property in C#

You can set the custom color for the WPF controls in C# using *SetActiveColorScheme*.

The following code snippet explains how to set the ActiveColorScheme property in C#.

1. Name the control using the Name attribute.

XML

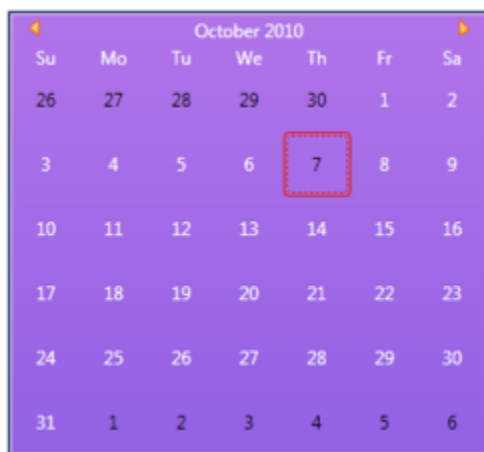
```
<syncfusion:CalendarEdit Name="calendar">  
</syncfusion:CalendarEdit>
```

2. Add the following line in code behind file.

C#

```
SkinManager.SetActiveColorScheme(calendar, Brushes.Red);
```

The output is displayed as shown below.



Calendar with Custom Color

Metro theme customization

Our well sophisticated metro theme will support a complete customization over the brushes and fonts. Each and every brushes of Metro Theme can be changed and customized based on the user needs.

The following are the brushes that can be customized in Metro Theme.

- MetroBrush.
- MetroBackgroundBrush.
- MetroPanelBackgroundBrush.
- MetroBorderBrush.
- MetroForegroundBrush.
- MetroFontFamily.
- MetroHoverBrush.
- MetroFocusedBorderBrush.
- MetroHighlightedForegroundBrush.

Setting MetroBackgroundBrush property in XAML

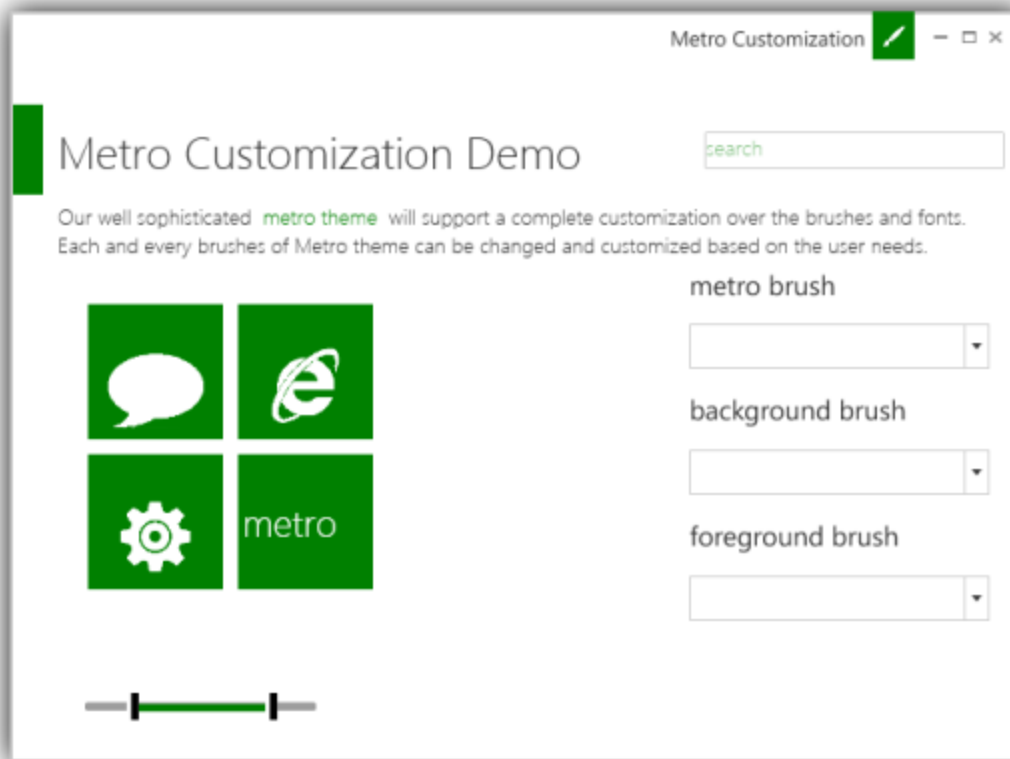
XML

```
<syncfusion:ChromelessWindow x:Class="WpfApplication18.MainWindow"
Title="Window1" Height="350" Width="525"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
syncfusion:SkinStorage.VisualStyle="Metro"
syncfusion:SkinStorage.MetroBackgroundBrush="Green">
</syncfusion:ChromelessWindow>
```

Setting MetroBackgroundBrush property in C#

C#

```
SkinStorage.SetMetroBrush(this, Brushes.Green);
```



Metro Customization Demo

Performance

The performance of SkinStorage can be improved by setting the [EnableOptimization](#) property. So, all themes resource dictionaries merged to `Application.Resources` instead of merging resource dictionaries to each individual controls in the application.

XML

```
<syncfusion:ChromelessWindow x:Class="Sample.MainWindow" x:Name="window"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:sfSample="clr-namespace:Sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow"
syncfusion:SkinStorage.EnableOptimization="True"
syncfusion:SkinStorage.VisualStudio="Office2010Blue" >
```

C#

```
SkinStorage.SetEnableOptimization(window, true);
SkinStorage.SetVisualStyle(DependencyObject Object, "Office2010Blue");
```

ResourceDictionary path for Syncfusion themes

Resource Dictionary path for Syncfusion themes are tabulated below:

Replace "< CurrentVisualStyle>" with the required VisualStyle name

Ex:

To merge the Office2010Blue Theming Dictionary for MicrosoftControls into the application:

XML

```
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.Shared.WPF;Component/SkinManager/Office2010BlueStyle.xaml" />
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

Controls table

Control Name	Theming Resource Dictionary Path
MSControls	/Syncfusion.Shared.WPF;component/SkinManager/.xaml
AutoComplete	/Syncfusion.Tools.WPF;component/Controls/AutoComplete/Themes/.xaml
Busy Indicator	/Syncfusion.Shared.WPF;component/Controls/BusyIndicator/Themes/.xaml
ButtonAdv	/Syncfusion.Shared.WPF;component/Controls/ButtonControls/Button/Themes/.xaml
DropDownButtonAdv	/Syncfusion.Shared.WPF;component/Controls/ButtonControls/DropDownButton/Themes/.xaml
SplitButtonAdv	/Syncfusion.Shared.WPF;component/Controls/ButtonControls/SplitButton/Themes/.xaml
Calendar	/Syncfusion.Shared.WPF;component/Controls/Calendar/Themes/.xaml
CardView	/Syncfusion.Tools.WPF;component/Controls/CardView/Themes/.xaml
Clock	/Syncfusion.Shared.WPF;component/Controls/Clock/Themes/.xaml
CheckListBox	/Syncfusion.Tools.WPF;component/Controls/CheckListBox/Themes/.xaml
ColorPicker	/Syncfusion.Shared.WPF;component/Controls/ColorPicker/Themes/.xaml
ColorPickerPalette	/Syncfusion.Shared.WPF;component/Controls/ColorPickerPalette/Themes/.xaml
ComboBoxAdv	/Syncfusion.Shared.WPF;component/Controls/ComboBoxAdv/Themes/.xaml
ChromelessWindow	/Syncfusion.Shared.WPF;component/Controls/ChromelessWindow/Themes/.xaml
DateTimeEdit	/Syncfusion.Shared.WPF;component/Controls/DateTimeEdit/Themes/.xaml
DockingManager	/Syncfusion.Tools.WPF;component/Framework/DockingManager/Themes/.xaml
DocumentContainer	/Syncfusion.Tools.WPF;component/Framework/DocumentContainer/Themes/.xaml

Editors(for All TextBoxControls)	/Syncfusion.Shared.WPF;component/Controls/Editors/Themes/.xaml
FontListBox	/Syncfusion.Tools.WPF;component/Controls/FontListBox/Themes/.xaml
FontListComboBox	/Syncfusion.Tools.WPF;component/Controls/FontComboBox/Themes/.xaml
Gallery	/Syncfusion.Tools.WPF;component/Controls/Gallery/Themes/.xaml
GroupBar	/Syncfusion.Tools.WPF;component/Controls/GroupBar/Themes/.xaml
HierarchyNavigator	/Syncfusion.Tools.WPF;component/Controls/HierarchyNavigator/Themes/.xaml
MenuAdv	/Syncfusion.Shared.WPF;component/Controls/MenuAdv/Themes/.xaml
NotifyIcon	/Syncfusion.Tools.WPF;component/Controls/NotifyIcon/Themes/.xaml
PinnableListBox	/Syncfusion.Shared.WPF;component/Controls/PinnableListBox/Themes/.xaml
RangeSlider	/Syncfusion.Tools.WPF;component/Controls/RangeSlider/Themes/.xaml
Ribbon	/Syncfusion.Tools.WPF;component/Framework/Ribbon/Themes/.xaml
TabControlExt	/Syncfusion.Tools.WPF;component/Controls/TabControlExt/Themes/.xaml
TabNavigationControl	/Syncfusion.Tools.WPF;component/Controls/TabNavigationControl/Themes/.xaml
TabSplitter	/Syncfusion.Tools.WPF;component/Controls/TabSplitter/Themes/.xaml
TaskBar	/Syncfusion.Tools.WPF;component/Controls/TaskBar/Themes/.xaml
TileView	/Syncfusion.Shared.WPF;component/Controls/TileView/Themes/.xaml
TimeSpanEdit	/Syncfusion.Shared.WPF;component/Controls/TimeSpanEdit/Themes/.xaml
ToolBarAdv	/Syncfusion.Shared.WPF;component/Controls/ToolBarAdv/Themes/.xaml
TreeViewAdv	/Syncfusion.Tools.WPF;component/Controls/TreeViewAdv/Themes/.xaml
UpDown	/Syncfusion.Shared.WPF;component/Controls/Updown/Themes/.xaml
WizardControl	/Syncfusion.Tools.WPF;component/Controls/WizardControl/Themes/.xaml

How to

Override Syncfusion Themes in WPF SkinStorage (Classic)

All Syncfusion styles can be overridden by a common Naming Convention. A unique key is given to each and every style, so that you can override the styles using the BasedOn property.

Naming Convention of a key

VisualStyle-ControlName-Style

Example: ShinyRedCalendarEditStyle

The following steps explain how to override the Syncfusion Themes.

1. Add the corresponding resource dictionary in the sample.

XML

```
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.Shared.WPF;Component/Controls/Calendar/themes/ShinyRedSt
yle.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

2. Define the new style using the BasedOn property.

The following code snippet overrides the Syncfusion style for the Calendar Control.

XML

```
<Grid>
<Grid.Resources>
<Style x:Key="CalendarEditStyle" TargetType="syncfusion:CalendarEdit"
BasedOn="{StaticResource ShinyRedCalendarEditStyle}" >
<Setter Property="Foreground" Value="Blue"/>
<Setter Property="HeaderForeground" Value="Blue"/>
</Style>
</Grid.Resources>
<syncfusion:CalendarEdit Name="calendar" Style="{StaticResource
CalendarEditStyle}"></syncfusion:CalendarEdit>
</Grid>
```

The output is displayed as shown below.



Switch between Skins at Run time in WPF SkinStorage (Classic)

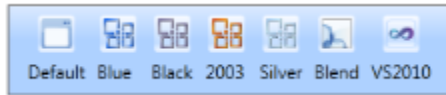
Themes can be dynamically switched.

Example 1:

You can use the Syncfusion Skin Picker Control to switch between the skins. Just add the Skin Picker Control to your application as shown below.

XML

```
<syncfusion:SkinPicker Height="60"/>
```



Example 2:

You can switch between the skins at run-time by using the ComboBox Selection Changed event.

Below is the code snippet to explain how to switch between the skins at run-time by using the ComboBox SelectionChanged event.

XML

```
<Grid Name="grid">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <ComboBox Name="themecombobox" Grid.Column="0"
    SelectionChanged="ComboBox_SelectionChanged" Width="150" Height="30">
    <ComboBoxItem> Blend </ComboBoxItem>
    <ComboBoxItem> ShinyBlue </ComboBoxItem>
  </ComboBox>
  <syncfusion:CalendarEdit Name="calendar"
    Grid.Column="1"></syncfusion:CalendarEdit>
</Grid>
```

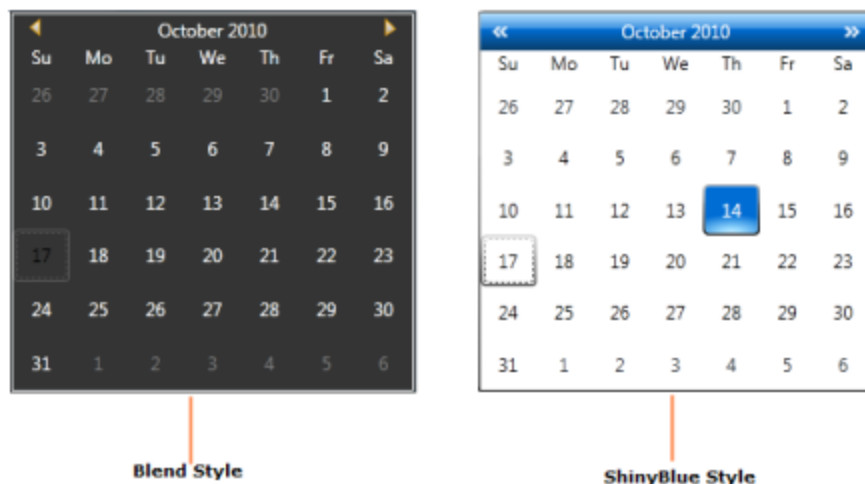
On ComboBox SelectionChanged event, particular VisualStyle should be set to the control.

The following code snippet explains how to set the switch between the skins.

C#

```
Private void ComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
  if (themecombobox.SelectedIndex == 0)
  {
    SkinStorage.SetVisualStyle(calendar, "ShinyBlue");
  }
  Else if (themecombobox.SelectedIndex == 1)
  {
    SkinStorage.SetVisualStyle(calendar, "Blend");
  }
}
```

The output is displayed as shown below.



Switch between Overridden Styles in WPF SkinStorage (Classic)

Switching between the overridden styles should be done manually. The overridden styles should be merged into the Resource Dictionary.

The following steps explain how to switch between the overridden styles.

1. Add the corresponding Resource Dictionaries in the sample.

XML

```
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.Shared.WPF;Component/Controls/Calendar/themes/ShinyRedSt
yle.xaml"/>
<ResourceDictionary
Source="/Syncfusion.Shared.WPF;Component/Controls/Calendar/themes/BlendStyle
.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

2. Define the new styles using the BasedOn property.

The following code snippet explains how to override the Syncfusion style for the Calendar Control.

XML

```
<Grid Name="grid">
<Grid.Resources>
<Style x:Key="ShinyRedStyle" TargetType="syncfusion:CalendarEdit"
BasedOn="{StaticResource ShinyRedCalendarEditStyle}" >
<Setter Property="Background" Value="PaleGoldenRod"/>
</Style>
<Style x:Key="BlendStyle" TargetType="syncfusion:CalendarEdit"
BasedOn="{StaticResource BlendCalendarEditStyle}" >
<Setter Property=" Background" Value="Green"/>
</Style>
```



```
</Style>
</Grid.Resources>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<ComboBox Name="themecombobox" Grid.Column="0"
SelectionChanged="ComboBox_SelectionChanged">
<ComboBoxItem>ShinyRed</ComboBoxItem>
<ComboBoxItem>Blend</ComboBoxItem>
</ComboBox>
<syncfusion:CalendarEdit Name="calendar"
Grid.Column="1"></syncfusion:CalendarEdit>
</Grid>
```

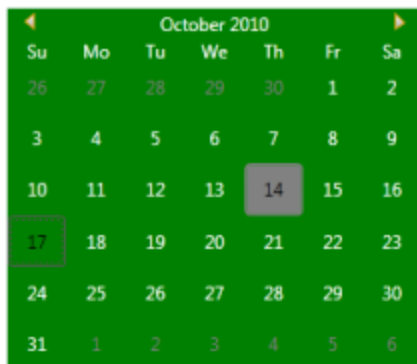
3. On ComboBox SelectionChanged event, particular overridden style should be set to the control depending on the current visual style.

The following code snippet explains how to set the overridden styles to the controls.

XML

```
Private void ComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
if (themecombobox.SelectedIndex == 0)
{
SkinStorage.SetVisualStyle(this, "ShinyRed");
System.Windows.Style style = grid.Resources["ShinyRedStyle"] as Style;
calendar.Style = style;
}
else
{
SkinStorage.SetVisualStyle(this, "Blend");
System.Windows.Style style = grid.Resources["BlendStyle"] as Style;
calendar.Style = style;
}
}
```

The output is displayed as shown below.



Blend Overridden Style



ShinyRed Overridden Style

Set and Override Visual Style in SkinStorage

This page explains Set and Override Visual Style in SkinStorage and more details.

Set and Override Visual Style at Application Level in WPF SkinStorage (Classic)

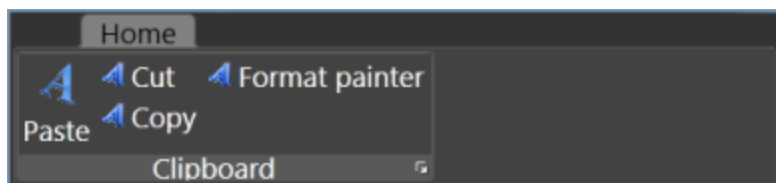
The Visual Styles can also be defined at application level. The VisualStyle property cannot be set at the application level. You have to merge the appropriate Resource Dictionary on Application resources which will cause all the controls to pick up the particular style.

The following code snippet explains how to override the Syncfusion Blend Style for the Ribbon Control at the application level.

XML

```
<Application x:Class="WpfApplication2.App"
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"
StartupUri="MainWindow.xaml">
  <Application.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary
Source="/Syncfusion.Tools.WPF;component/Framework/Ribbon/Themes/BlendStyle.x
aml"/>
      </ResourceDictionary.MergedDictionaries>
      <Style TargetType="syncfusion:Ribbon" BasedOn="{StaticResource
BlendRibbonStyle}">
        <Setter Property="FontSize" Value="19"/>
      </Style>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

The output is displayed as shown below.



SetVisualStyle for Derived in SkinStorage

This page explains SetVisualStyle for Derived in SkinStorage and more details.

Set Visual Style for dynamically added Derived Controls in WPF SkinStorage (Classic)

Normally, a control added to an application will dynamically pick up the existing style using the SkinStorage. But when an user control derived from the existing control is added to an application, a style based on the Base class should be defined in the application.

The following code snippet explains the scenario where an user control of Button type is exposed here.

XML

```
<Button x:Class="WpfApplication2.TestButton"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Height="300" Width="300" xmlns:theme="http://schemas.syncfusion.com/wpf"
theme:SkinStorage.VisualStyle="Blend">
</Button>
```

C#

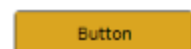
```
public partial class TestButton : Button
{
    public TestButton ()
    {
        InitializeComponent();
    }
}
```

Styles based on the button style should be defined in the application as follows. You have to merge the corresponding Resource Dictionary when overriding the style in the application.

XML

```
<Application.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.Shared.WPF;component/SkinManager/BlendStyle.xaml"/>
</ResourceDictionary.MergedDictionaries>
<Style TargetType="local:TestButton" BasedOn="{StaticResource
BlendButtonStyle}">
<Setter Property="Background" Value="GoldenRod"/>
</Style>
</ResourceDictionary>
</Application.Resources>
```

The output is displayed as shown below.



Resource ID for Syncfusion Themes in WPF SkinStorage (Classic)

The WPF Skin Manager provides a swatch of theme colors that contains all Syncfusion theme brushes. These brushes are used by getting key names for the given particular theme. Brush names are stored as properties in the ThemeColors class.

Note: ThemeColors class has been referred at application level by adding Syncfusion.Shared.WPF assembly.

Use case scenarios

Users can customize brushes easily to get unique colors for all controls in the application.

Properties

Property	Description	Type	Data Type
BackgroundBrush	Gets the background color of the control	CLR	String
HoverBrush	Gets the mouse hover color of the control	CLR	String
SelectedBrush	Gets the foreground color of selected control	CLR	String
BorderBrush	Gets the border color of the control	CLR	String
Foreground	Gets the foreground color of text	CLR	String
ActiveWindowTitleBrush	Gets the background color of active window header	CLR	String
InActiveWindowTitleBrush	Gets the background color of inactive window header	CLR	String
RibbonPanelBrush	Gets the background color of the ribbon panel	CLR	String
MenuBackgroundBrush	Gets the background color of Menu	CLR	String
MenuHoverBrush	Gets the mouse hover color of the MenuItem	CLR	String
GalleryHeaderSelectedBrush	Gets the foreground color of selected GalleryHeader	CLR	String
TrackBrush	Gets the background color of scrollbar track	CLR	String
HorizontalThumbBrush	Gets the background color of horizontal thumb	CLR	String
VerticalThumbBrush	Gets the background color of vertical thumb	CLR	String
HorizontalThumbHoverBrush	Gets the background color of the horizontal thumb when mouse is hovered	CLR	String
VerticalThumbHoverBrush	Gets the background color of the vertical thumb when mouse is hovered	CLR	String
HorizontalThumbPressedBrush	Gets the background color of pressed horizontal thumb	CLR	String

VerticalThumbPressedBrush	Gets the background color of pressed vertical thumb	CLR	String
ColumnHeaderBrush	Gets the background color of column header	CLR	String
TabHoverBrush	Gets the background color of hovered TabItem	CLR	String
TabHoverBorderBrush	Gets the border color of hovered TabItem	CLR	String
CardViewGroupingBarBrush	Gets the background color of the CardView grouping header	CLR	String

Adding resource ID to an application

Setting through XAML

The following code snippet explains how to set the Resource ID through XAML

1. Add the VisualStyle in the sample.

XML

```
syncfusion:SkinStorage.VisualStudio="Office2013"
```

2. Set Resource ID as shown below.

XML

```
<Grid x:Name="grid" Background="{DynamicResource {x:Static syncfusion:ThemeColors.BackgroundBrush}}">
```

Setting through C#

C#

```
SkinStorage.SetVisualStyle(this, "Office2013");
grid.SetResourceReference(BackgroundProperty, ThemeColors.BackgroundBrush);
```

Touch UI in WPF SkinStorage (Classic)

The touch support allows users to interact with some Syncfusion WPF controls with finger gestures on touchscreen devices.

The following controls have touch styles implemented:

- Editors
- ComboBox
- ListBox Controls
- TreeView
- TabControl
- Menu
- GroupBar

- RangeSlider

How To apply touch styles

Touch styles can be applied to a control by setting the EnableTouch property defined in the SkinStorage class to True. This property can be set in either XAML or C#.

Setting the EnableTouch property in XAML

XML

```
<syncfusion:IntegerTextBox  
syncfusion:SkinStorage.EnableTouch="True" >  
</syncfusion:IntegerTextBox >
```

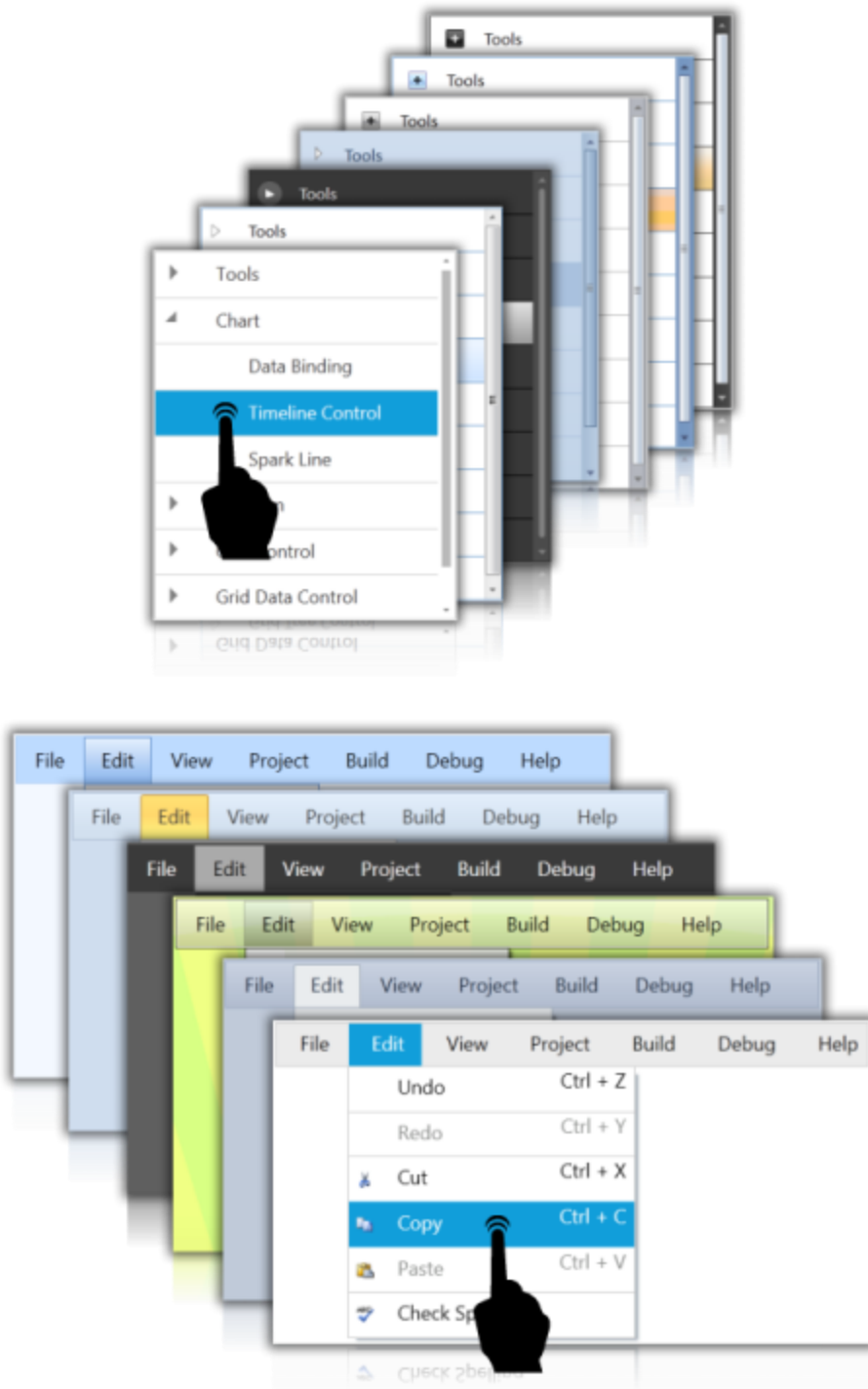
Setting the EnableTouch Property in C#

C#

```
SkinStorage.SetEnableTouch(this, true);
```

Controls with touch UI





SpreadsheetControl (Classic)

WPF SpreadsheetControl (Classic) Overview

Spreadsheet control enables you to create, manipulate and save an Excel document with the native excel format, without depending on Microsoft Excel. It does not require Microsoft Excel to be installed in the machine.

You can open and modify any Excel document that is compatible with the following Excel versions:

- Excel 97 (.xls)
- Excel 2000 (.xls)
- Excel 2002 (.xls)
- Excel 2003 (.xls)
- Excel 2007 (.xlsx)
- Excel 2010 (.xlsx)

Use Case Scenarios

Spreadsheet control enables you to create, view and manipulate large Excel documents in your application quickly and efficiently.

You can use this in sales tracking application to generate and customize a report on sales figures.

You can use this in a banking application to allow the users to customize monthly statements and save it in Microsoft Excel format.

Feature Summary

This section provides the basic information about some important features of Essential Grid with their definition and usage.

Create, Open and Save Excel Document

Spreadsheet control supports you to create the Excel documents without having the Microsoft Excel in your system. Also you can able to open the existing Excel documents in Spreadsheet control and modify and save the contents in the Excel documents.

Formatting

Conditional Formatting

Conditional Formatting feature allows you to apply formats to a cell or range of cells depending on the value of cells or formula that meet specific criteria.

Number formatting

Using the number formatting feature, you can change the appearance of a number or value in a cell. This number formatting does not change the actual number, but just change the way it appears in the cell. This feature improves the appearance of the spreadsheet and makes the numbers easier to read and understand.

Cell formatting

Cell formatting feature allows you to change font style for the data you entered into the cells. The color and size of the fonts, cell backgrounds, and cell borders can also be changed using this feature.

Merge cells

Using this feature, you can merge two or more adjacent cells into a single cell and display the contents of one cell in the merged cell. You can also merge the contents of multiple cells and display them in one cell.

Data Validation

Data validation feature helps you to ensure the data integrity by enforcing end users to enter valid data into the cells. You can also write your own input message that appears before entering data; it helps the users to acquaint with valid data. You can also write an error message that appears in case of any invalid data entered.

Copy and Paste

Spreadsheet control allows you to copy the data or formulas from one or more cells and paste them to other cells in the same worksheet, or in different worksheets.

Fill series

Spreadsheet control has the support for fill series as like in Microsoft Excel. By using the fill series feature, you can automatically fill the data based on the previous cells. This control also features copy series, and fill with/without formatting supports as like in Excel.

Formulas

Spreadsheet control supports most commonly used mathematical formulas and also supports all the formulas that are supported by the Essential Grid control. These formulas can be used to manipulate data and calculate strings and numbers.

Name ranges

Spreadsheet control also supports the name ranges in the formulas. By using the name ranges, you can specify the name of the cell range, and then you can use it in the formula more easily without hassling of remembering cell locations.

Review Options

Comments

Spreadsheet control also supports comments that provide additional information about a cell such as what the value represents. And it would be useful for you if you want the end users to understand the data cells more deeply.

Protect workbook

Spreadsheet control allows you to protect the workbook. By using the workbook protection level, you can only lock-down the structure and worksheet window, which enables you to prevent workbook from any structural change or from any change in size.

Protect sheet

Spreadsheet control also allows you to protect worksheets. With the worksheet protection, you can have total control of particular worksheet by protecting each element in the worksheet. In this level of protection you can prevent users from modifying values, cells, formula, name ranges, etc.

Encrypt and Decrypt

Spreadsheet control supports you to encrypt and decrypt the Excel document by using the passwords.

Add or Remove Worksheet

Spreadsheet control also supports you to add or remove the worksheets in the Excel document.

Hide or Show Worksheet

Spreadsheet control also supports you to hide or show the worksheet in the Excel document.

View Options

Freeze panes

It enables you to freeze certain rows or columns of the spreadsheet so that they remain visible at all times while scrolling to the right or down. Freezing headings on the screen makes it easier to read your data in the spreadsheet.

Zooming

By using this feature, you can change the zoom level of the Spreadsheet control. This allows you to zoom in the spreadsheet to get a close-up view of the cells or zoom out to see more number of cells in the spreadsheet.

Formula Bar

The formula bar displays the data or formula stored in the selected active cell. The formula bar can be used to enter or edit a formula or data in a cell. You can also show or hide the formula bar similar to Microsoft Excel.

Grid lines

Grid lines are the faint lines that appear around cells. They are used to distinguish cells on the spreadsheet. By default, grid lines are displayed in the Spreadsheet control. You can also disable the visibility of grid lines in spreadsheet.

Headings

You can also enable and disable the heading in the Spreadsheet control similar to Microsoft Excel.

Hyperlink

Spreadsheet control supports the hyperlink navigation. The hyperlink is a convenient way to navigate or browse data within a worksheet or other worksheets in a workbook.

SpreadsheetRibbon Support

Spreadsheet control provides SpreadsheetRibbon support to format the cells and workbook. It gives Microsoft Excel like appearance to the Spreadsheet control.

Commands Support

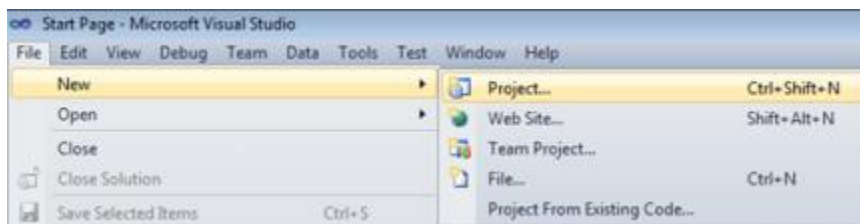
Spreadsheet Control also provides the support for the Commands, to format the cells and workbook.

Getting Started with WPF SpreadsheetControl (Classic)

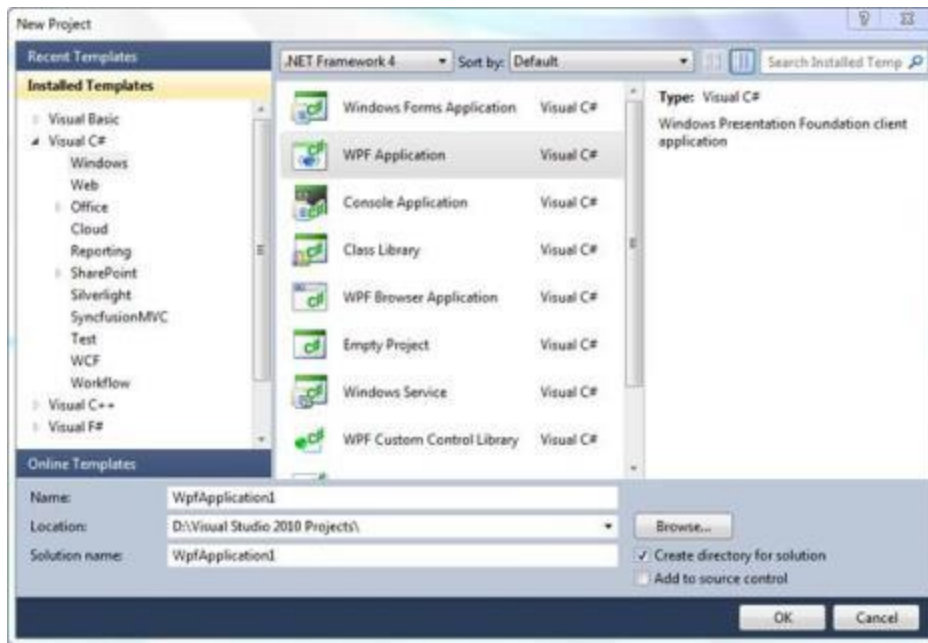
Creating a WPF Application

The following are steps to create a new WPF application in Visual Studio:

1. Open Visual Studio.
2. Click File tab and navigate to New > Project.



3. In the New Project dialog box, select WPF Application, enter a name for the application in the Name field and click OK.



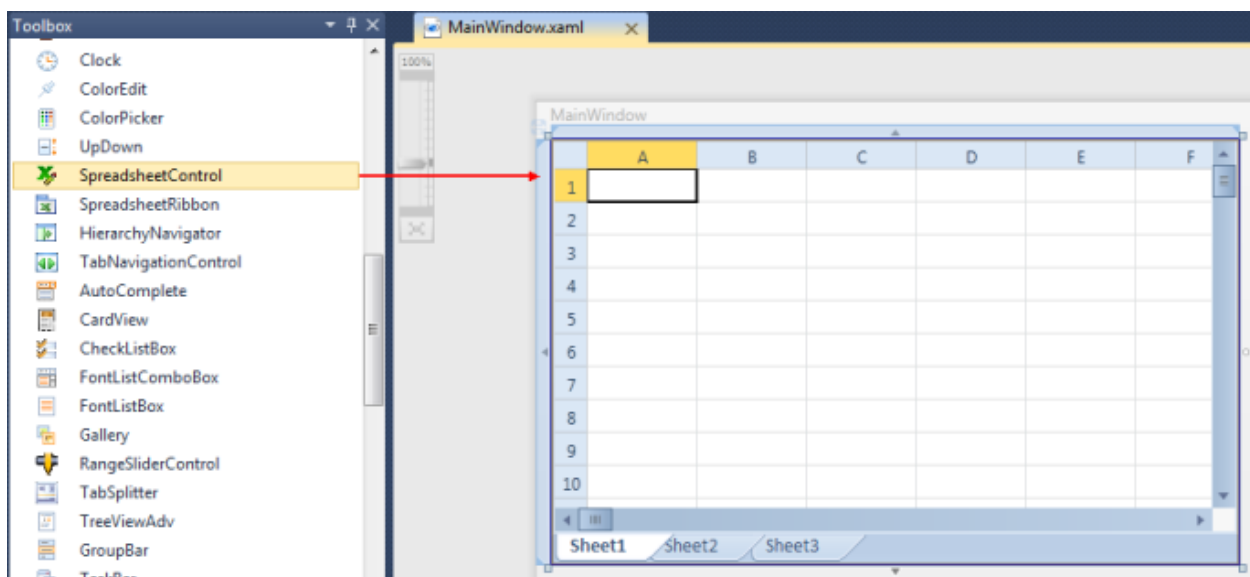
4. Now, a new WPF application will be created.

Adding Spreadsheet Control to WPF Application

Through Visual Studio

Following are the steps to add the Spreadsheet control to WPF application using Visual Studio.

1. Create a new WPF application in Visual Studio.
2. In Visual Studio Toolbox, click Syncfusion WPF Toolbox tab.



3. Drag SpreadsheetControl to the Designer area.
4. Customize the properties of SpreadsheetControl using Properties window.

Note: To add the SpreadsheetRibbon control to your application, drag SpreadsheetRibbon to the Designer area and set the Spreadsheet control as a DataContext as shown the following code.

XML

```
<syncfusion:SpreadsheetRibbon DataContext="{Binding ElementName=spreadsheetControl1}" />
```

Through XAML and C#

You can also add the Spreadsheet control to a WPF application through XAML and C#. The following code example illustrates this.

XML

```
<syncfusion:SpreadsheetControl HorizontalAlignment="Left"
Name="spreadsheetControl1" VerticalAlignment="Top" />
```

C#

```
SpreadsheetControl Spreadsheet1 = new
SpreadsheetControl(); LayoutRoot.Children.Add(Spreadsheet1);
```

VB.NET

```
Dim Spreadsheet1 As SpreadsheetControl = New
SpreadsheetControl() LayoutRoot.Children.Add(Spreadsheet1)
```

Loading Excel Files in Spreadsheet Control

You can open the Excel document in the Spreadsheet control using *ImportFromExcel* method. The following code illustrates this.

C#

```
FileStream stream = new FileStream(@"..\..\Data\DefaultSheet.xlsx", FileMode
.Open);
spreadsheet.ImportFromExcel(stream);
```

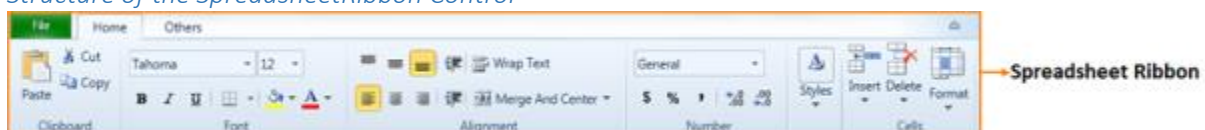
VB.NET

```
Dim stream As FileStream = New FileStream(@"..\..\Data\DefaultSheet.xlsx",
FileMode.Open) spreadsheet.ImportFromExcel(stream)
```

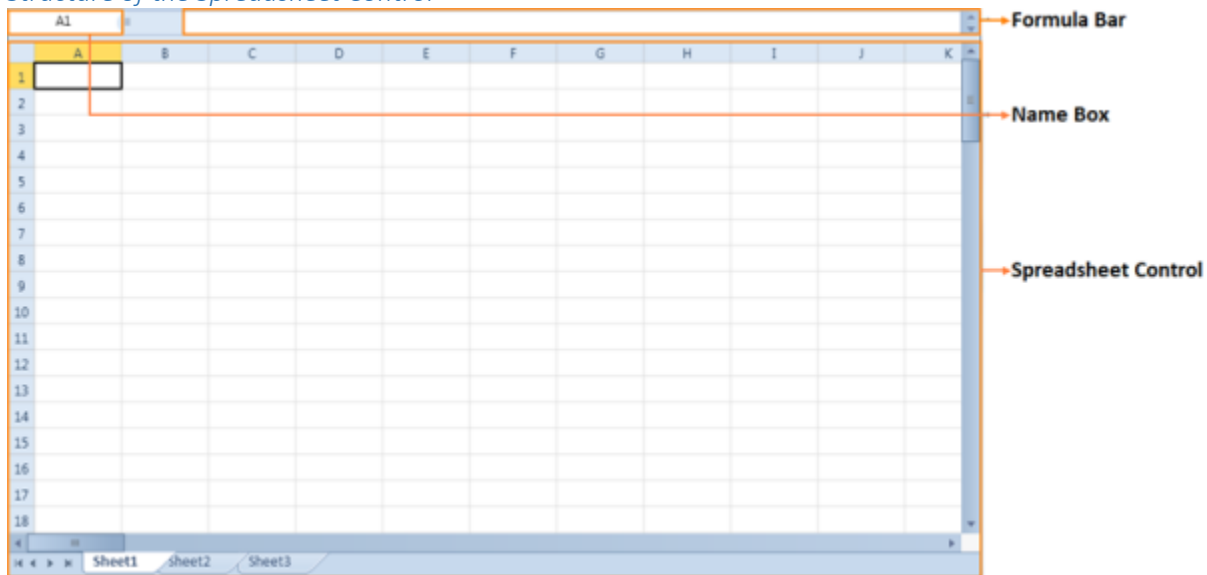
Note: You can also open the Excel document using *ImportFromExcelCommand*. When you execute the *ImportFromExcelCommand* it will display the Open dialog box. Using this Open dialog, you can open the Excel document in the Spreadsheet control.

Appearance and Structure of the Controls

Structure of the SpreadsheetRibbon Control



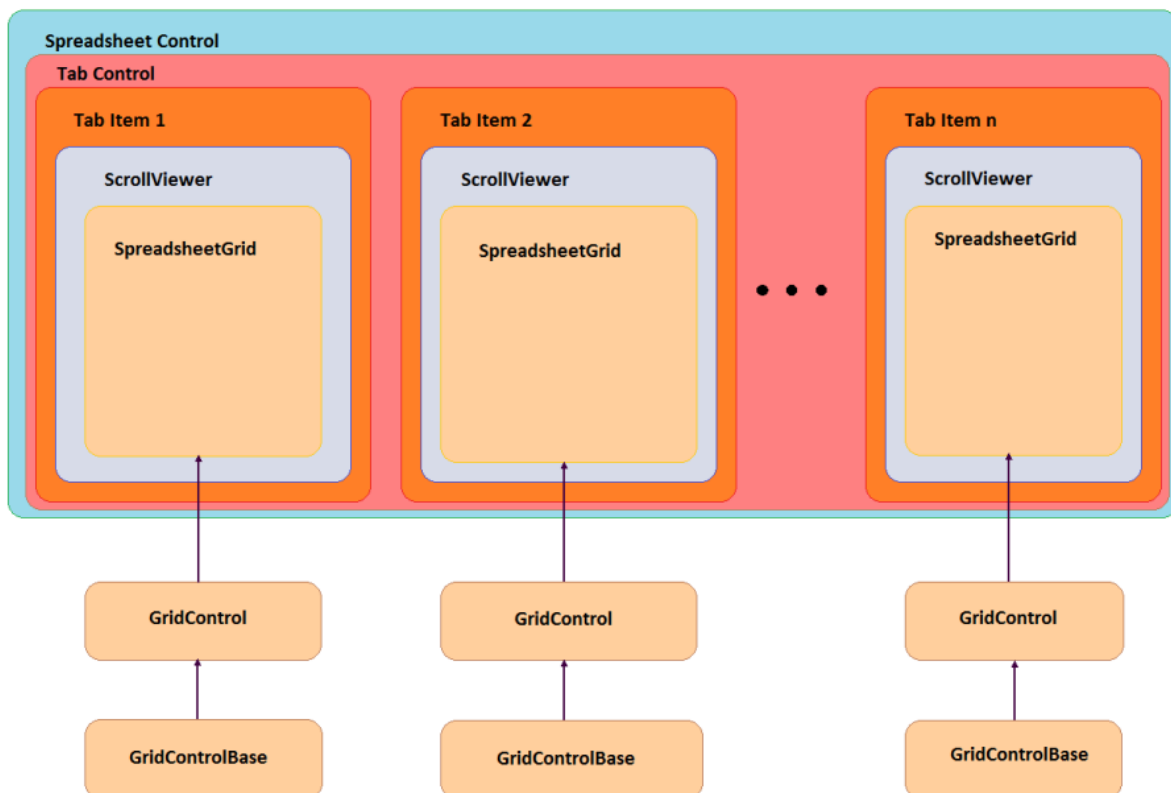
Structure of the Spreadsheet Control



Architecture

The Spreadsheet control supports ControlTemplate to define its content. By default, its content includes a TabControlExt object that contains number of TabItemExt based on sheet count. The TabItemExt contains a ScrollViewer object that contains a SpreadsheetGrid object.

The following sketch illustrates the Spreadsheet control architecture.



Accessing the Underlying Grid control

The Spreadsheet control is a control derived class that has its own properties. You can use Grid control derived property namely `ActiveSheetGrid` to get its grid-like behavior. To access the underlying Grid control associated with the Spreadsheet control, you can use the `SpreadsheetControl.GridProperties.ActiveActiveSheetGrid` property.

Creating an Excel Document in WPF SpreadsheetControl (Classic)

When you add the Spreadsheet control to the application, it will be loaded with a blank workbook. You can save this as an Excel document. You can also create new workbooks if required.

To create a new workbook, call the `New` method. New workbook will be created with three worksheets by default. The following code illustrates this:

C#

```
spreadControl.New();
```

VB.NET

```
spreadControl.New()
```

You can also specify the number of worksheet you want to add in workbook by passing the sheet count to the `New` method. The following code illustrates this:

C#

```
spreadControl.New(3);
```

VB.NET

```
spreadControl.New(3)
```

Using Command

You can also use the `NewCommand` to create the Excel document. By default when you execute the `NewCommand` it will create the workbook with three worksheets and you can also specify the number of worksheet you want to add in workbook by passing the sheet count as `Command` parameter.

The following code illustrates how to bind the `NewCommand` to a button:

XML

```
<Button Command="{Binding Path=NewCommand}"/>
```

Open and Save Excel Document in WPF SpreadsheetControl (Classic)

You can open and save the Excel document that is compatible with following Excel versions:

- Excel 97 (.xls)
- Excel 2000 (.xls)
- Excel 2002 (.xls)
- Excel 2003 (.xls)
- Excel 2007 (.xlsx)
- Excel 2010 (.xlsx)

Open Excel Document

You can open the Excel document using the one of the following override methods:

The following code illustrates how to open Spreadsheet control with file stream:

C#

```
spreadsheet.ImportFromExcel(stream);
```

VB.NET

```
spreadsheet.ImportFromExcel(stream)
```

The following code illustrates how to open Spreadsheet control with file stream and ExcelOpenType:

C#

```
spreadsheet.ImportFromExcel(stream,  
    Syncfusion.XlsIO.ExcelOpenType.Automatic);
```

VB.NET

```
spreadsheet.ImportFromExcel(stream,  
    Syncfusion.XlsIO.ExcelOpenType.Automatic)
```

The following code illustrates how to open Spreadsheet control with file stream and Excel Version:

C#

```
spreadsheet.ImportFromExcel(stream,  
    Syncfusion.XlsIO.ExcelVersion.Excel2010);
```

VB.NET

```
spreadsheet.ImportFromExcel(stream, Syncfusion.XlsIO.ExcelVersion.Excel2010)
```

The following code illustrates how to open Spreadsheet control with file stream, ExcelOpenType and Excel Version:

C#

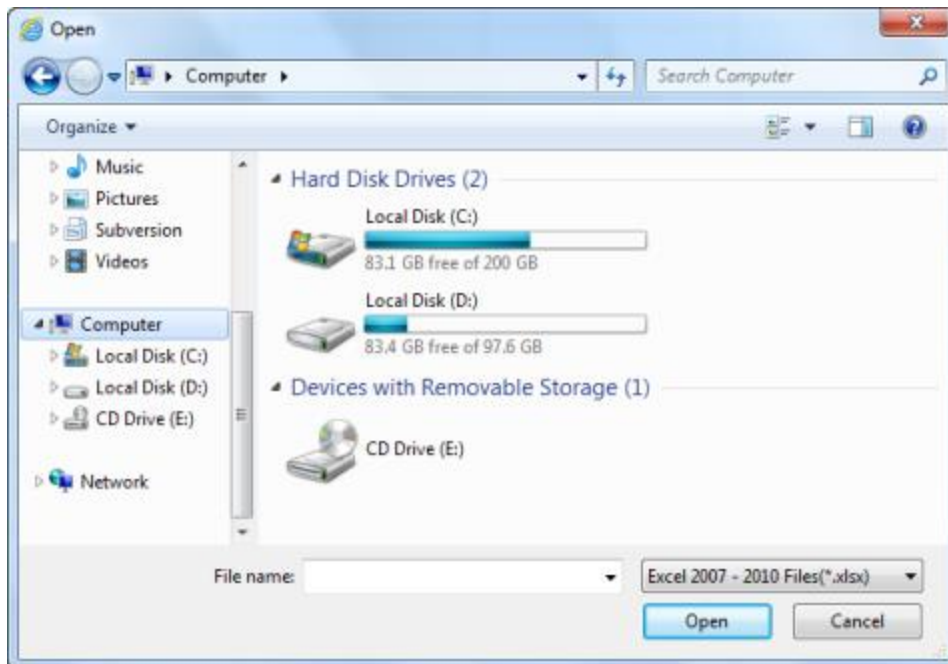
```
spreadsheet.ImportFromExcel(stream,  
    Syncfusion.XlsIO.ExcelOpenType.Automatic,  
    Syncfusion.XlsIO.ExcelVersion.Excel2010);
```

VB.NET

```
spreadsheet.ImportFromExcel(stream, Syncfusion.XlsIO.ExcelVersion.Excel2010)
```

Using Command

You can also open the Excel document by using the ImportFromExcelCommand. When you execute the ImportFromExcelCommand it will open the Open dialog box. Using that dialog you can open the Excel document.



The following code illustrates how to bind the `ImportFromExcelCommand` to a button:

XML

```
<Button Command="{Binding Path= ImportFromExcelCommand}"/>
```

Save Excel Documents

To save the current Workbook, call the `SaveAs` method as given in the following code:

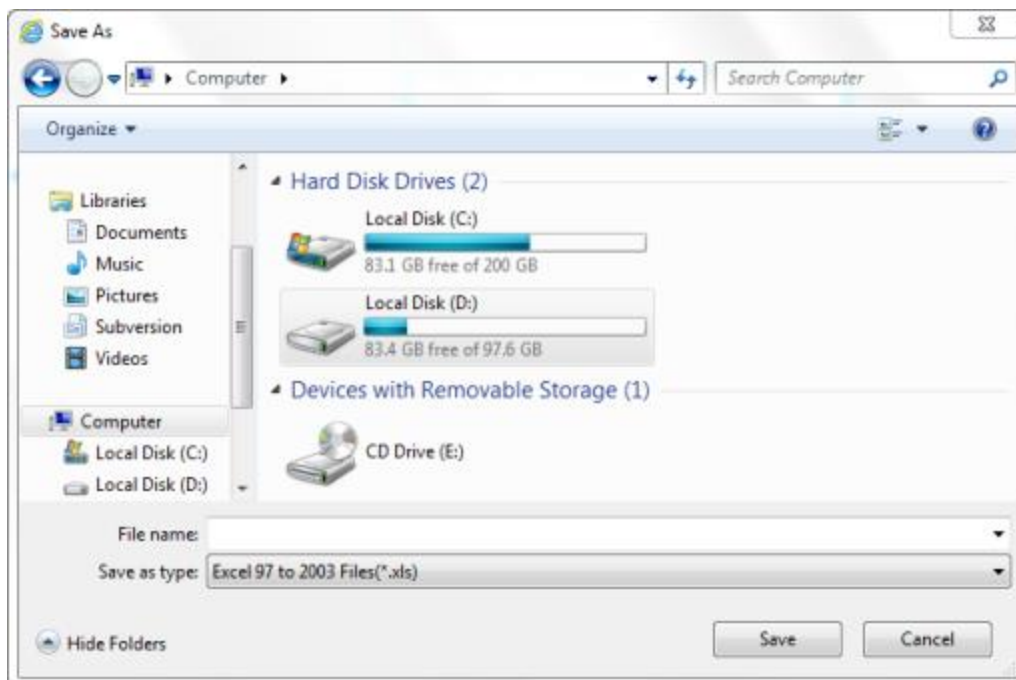
C#

```
spreadControl.SaveAs();
```

VB.NET

```
spreadControl.SaveAs()
```

Save As dialog box will open. Save the document at the required location.



Using Command

You can save the Excel document using the `ExportToExcelCommand`. When you execute the `ExportToExcelCommand`, it will open the SaveAs dialog box. Save the content of the Spreadsheet control as Excel document.

The following code illustrates how to bind the `ExportToExcelCommand` to a button:

XML

```
<Button Command="{Binding Path= ExportToExcelCommand}"/>
```

Formatting in WPF SpreadsheetControl (Classic)

Conditional Formatting

Conditional formatting is a feature by which cell styles will be dynamically formatted based on its value and specified condition. It enables you to apply formatting to the cell through a dialog window.

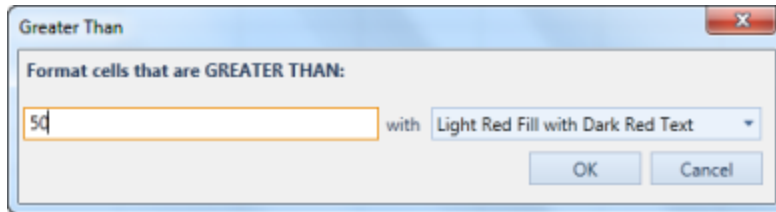
Use Case Scenarios

This feature enables you to categorize the cell for analysis. For example, you can format a time sheet, to point out the overtime obtained by an employee. You can also use it to track the best sales employees in the company, by setting a quota that makes a cell range particular.

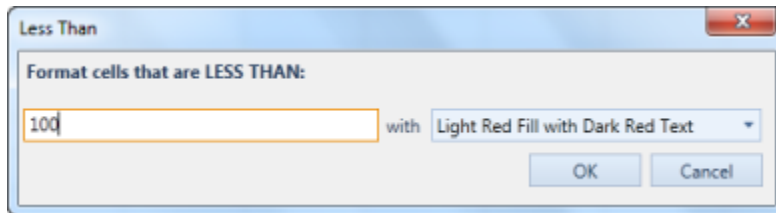
Defining Condition for Spreadsheet Cells

You can define the condition for formatting the Spreadsheet cells using a conditional formatting dialog box. Specify the conditional value and style format in the dialog box. You can open the conditional formatting dialog using the `ConditionalFormatCommand`. Based on the Command parameter one of the following dialogs will open:

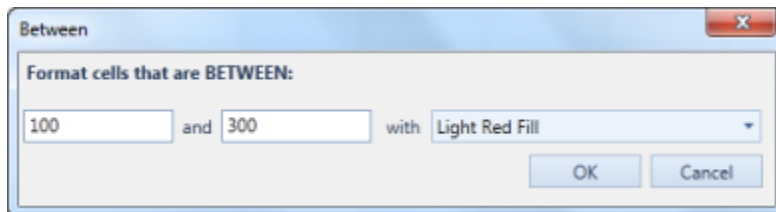
- Greater than



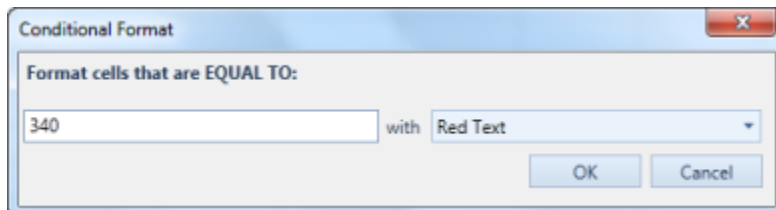
- Less Than



- Between



- Equal to



The following code illustrates how to bind the ConditionalFormatCommand to a button:

XML

```
<Button Command="{Binding Path=ConditionalFormatCommand}">
  <Button.CommandParameter>
    <XlsIO:ExcelComparisonOperator>Greater</XlsIO:ExcelComparisonOperator>
  </Button.CommandParameter>
</Button>
```

	A	B	C
1			
2	Greater than 50	350	
3			
4	Less than 50	23	
5			
6	Equal to 100	100	
7			

Number formatting

Number formatting helps you to control the format of the number in Spreadsheet cells. Number in the cell will be displayed based on the number format applied.

Defining number formatting for Spreadsheet Cells

You can define the number formatting using the NumberFormattingCommand. Number formatting will be applied to the Spreadsheet cells based on the command parameter. The following code illustrates this:

XML

```
<Button Command="{Binding Path=NumberFormatCommand}" Grid.Row="2">
  <Button.CommandParameter>
    <System:String>0.00%</System:String>
  </Button.CommandParameter>
</Button>
```

	A	B	C	D
1	Data	Number Format	Result	
2	12	0.00	12.00	
3	123678	\$#,##0.00	\$123,678.00	
4	40880	d-mmm-yy	23-Jan-00	
5	45	0.00%	4500.00%	
6	-23	(* #,##0);(* (#,##0);(* "-");(@)	(23)	
7				

Center Across Selection

Overview

This feature center aligns the cell value in Spreadsheet, across the range of selected cells.

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							

Properties, Methods and Events

Methods

Method	Description	Parameters	Type	Return Type
SetCenterAcrossSelection()	This method sets the Center Across Selection on the selected cells. It should	setCenterAcrossSelection()	null	void

	be explicitly called to set Center Across Selection.			
--	--	--	--	--

Center aligning cell values in Spreadsheet

The following code snippets manually set the center across selection by calling the method:

C#

```
SpreadsheetControl.GridProperties.CurrentExcelGridModel.setCenterAcrossSelection();
```

VB.NET

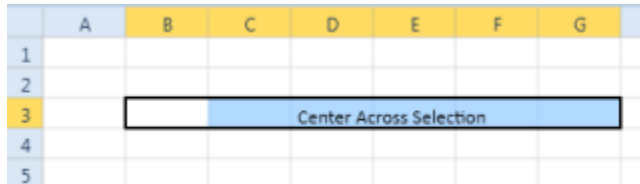
```
SpreadsheetControl.GridProperties.CurrentExcelGridModel.setCenterAcrossSelection()
```

The following code snippet manually sets the center across selection by calling the command:

XML

```
<Button Content="CentreAcrossSelection" Margin="5" Width="200"
Command="{Binding ElementName=spreadSheetControl,
Path=CenterAcrossSelectionCommand}"/>
```

The output displayed is shown in the following screenshot:



Appearance in WPF SpreadsheetControl (Classic)

Fonts

You can customize the font setting of Spreadsheet cells using this feature. You can apply various font family, font color and font size.

Font Settings in Spreadsheet Control

Spreadsheet control provides API support for specifying the font style for the cells text. You can apply various font settings using the *CellStyle* property. Use the *CurrentExcelRangeStyle* property, to apply style for the current selected cells. You can also specify the worksheet range for applying style. The following code illustrates this:

C#

```
//Applying font settings to a worksheet range
spreadControl.ExcelProperties.WorkBook.Worksheets[0]
["A1"].CellStyle.Font.FontName = "Arial Black";
spreadControl.ExcelProperties.WorkBook.Worksheets[0]
["A2"].CellStyle.Font.Size = 15;
spreadControl.ExcelProperties.WorkBook.Worksheets[0]
["A3"].CellStyle.Font.Color = ExcelKnownColors.Blue;
```

```
spreadControl.ExcelProperties.WorkBook.Worksheets[0]
["A4"].CellStyle.Font.Bold = true;
spreadControl.ExcelProperties.WorkBook.Worksheets[0]
["A5"].CellStyle.Font.Italic = true;
//Applying font setting to the selected cell
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.FontName
= "Arial Black";
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.Size =
18;
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.Color =
ExcelKnownColors.Red;
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.Bold =
true;
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.Italic =
true;
```

VB.NET

```
' Applying font settings to a worksheet range
spreadControl.ExcelProperties.WorkBook.Worksheets(0)
("A1").CellStyle.Font.FontName = " Arial Black"
spreadControl.ExcelProperties.WorkBook.Worksheets(0)
("A2").CellStyle.Font.Size =
15spreadControl.ExcelProperties.WorkBook.Worksheets(0)
("A3").CellStyle.Font.Color = ExcelKnownColors.Blue
spreadControl.ExcelProperties.WorkBook.Worksheets(0)
("A4").CellStyle.Font.Bold = True
spreadControl.ExcelProperties.WorkBook.Worksheets(0)
("A5").CellStyle.Font.Italic = True' Applying font setting to the selected
cell
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.FontName
= "Arial Black"
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.Size =
18
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.Color =
ExcelKnownColors.Red
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.Bold =
True
spreadControl.ExcelProperties.CurrentExcelRangeStyle.CellStyle.Font.Italic =
True
```

	A	B
1	FontName - Arial Black	
2	Font Size - 18	
3	Font color - Blue	
4	Bold text	
5	Italic Text	
6		

Using Command

You can set the font family to the spreadsheet cells using the *FontFamilyCommand*. To set the font family, specify the font name as command parameter. The following code illustrates this:

XML

```
<Button Command="{Binding Path= FontFamilyCommand}" CommandParameter ="
Arial Black">
</Button>
```

Merge cells

Spreadsheet control provides support to merge two or more cells. When a group of cells is merged, the contents of the upper-left cell will be taken as the content of the merged cell, rest will be deleted.

You can merge/unmerge the group of cell using the MergeCommand. To merge the cell, pass the command parameter as MergeAndCenter. To unmerge the cells pass the command parameter as UnMerge.

The following code illustrates how to merge the selected cells in Spreadsheet:

XML

```
<Button Command="{Binding Path= MergeCommand}" CommandParameter
="MergeAndCenter">
</Button>
```

The following code illustrates how to unmerge the selected cells in Spreadsheet:

XML

```
<Button Command="{Binding Path= MergeCommand}" CommandParameter =" UnMerge">
</Button>
```

	A	B	C	D	E
1					
2		Merged Cell			
3					
4					
5					
6					

Borders

This feature enables you to add or remove cell borders as required.

To add border to an individual cell use the BorderCommand. You can also specify the border type through the command parameter.

The following code illustrates, how to add and remove the cell border in Spreadsheet Control

XML

```
<!--To draw the Bottom Border-->
<Button Command="{Binding Path= BorderCommand}" CommandParameter
="BottomBorder">
</Button>
<!--To draw the Top Border-->
<Button Command="{Binding Path= BorderCommand}" CommandParameter
="TopBorder">
</Button>
<!--To draw all Border-->
```

```
<Button Command="{Binding Path= BorderCommand}" CommandParameter
="AllBorder">
</Button>
```

	A	B	C	D
1				
2				
3				
4				
5				

Freeze Panes

This feature enables you to freeze rows or columns of the spreadsheet, while scrolling. Freezing the header row or column makes it easier to read the data in the spreadsheet.

To apply the freeze panes, use the FreezePaneCommand. You can pass one of the following command parameter.

- FreezePanes—To freeze the particular row and column
- FreezeTopRow—To freeze top row only
- FreezeFirstColumn—To freeze the first column only

The following code illustrates this:

XML

```
<Button Command="{Binding Path= FreezePaneCommand}" Grid.Row="2">
<Button.CommandParameter>
<Syncfusion:Freeze>FreezePanes</Syncfusion:Freeze>
</Button.CommandParameter>
</Button>
```

A	B	C	D	E	F	G
1	Access Asia					
2	Inventory/Cost of Goods Sold Analysis					
3	8/4/11					
4						
5	need to enter anything into them.					
6						
7		Product 1	Product 2	Product 3	Product 4	Total
10	Production	700	800	600	600	2,700
11	Units available for sale	1,900	1,800	1,800	1,900	7,400
12	Units sold	800	600	500	750	2,650
13	Number of units in inventory—end of period	1,100	1,200	1,300	1,150	4,750

Wrap Text Support

Overview

You can manually wrap text and resize cells when the text or cell value exceeds the column width, by calling the SetWrapText method or WrapTextCommand in Spreadsheet control.

Use Case Scenario

Users can wrap text for a selected range of cells by just calling a method.

	A	B
1		
2	Spreadsheet control enables you to create, manipulate and save an Excel document with the native excel format, without depending on Microsoft Excel.	
3		

Methods

Method	Description	Parameters	Type	Return Type
SetWrapText()	This Method is called to set wrapText on selected cells. This method is called when we manually invoke the method or when we call wrapText command. This method changes the float cell to WrapText cell and calls the method AutoSizeToFit().	Overloads: nil	Nil	void

Data Management in WPF SpreadsheetControl (Classic)

Formulas

Formulas are entries in Excel document that have an equation, which calculates the value to display. A typical formula might contain cells, constants, and even functions.

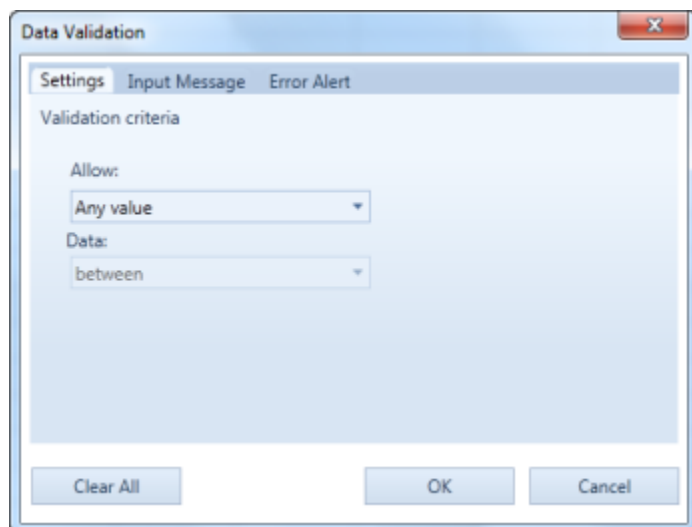
Spreadsheet control supports various built-in functions that make large calculations in large sheets easier.

Spreadsheet control supports all the formulas that are supported by the Grid control. For more details about the

supported formula refer to [Grid Control UG](#).

Data Validation

The Data Validation enables you to dynamically validate the data entered in a cell. You can specify the validation rules in the Data Validation dialog box. Spreadsheet control supports the various validation types for each data type. This also enables you to show the error message for invalid data.



Data Validation in Spreadsheet control

Essential Spreadsheet control enables you to edit the existing data validation. You can also create new data validation. Spreadsheet control supports the following validation types:

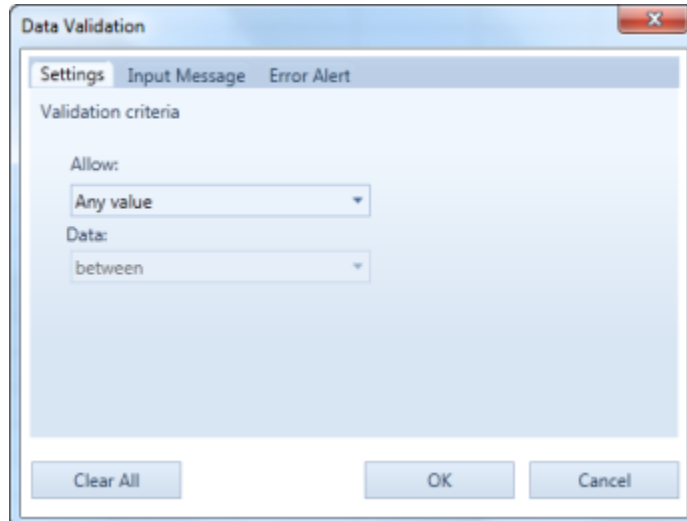
- Text Length Validation
- Time Validation
- List Validation
- Number Validation
- Date Validation

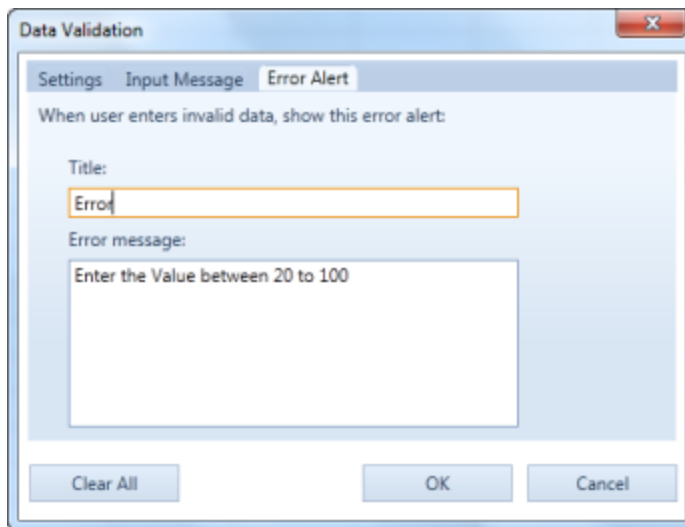
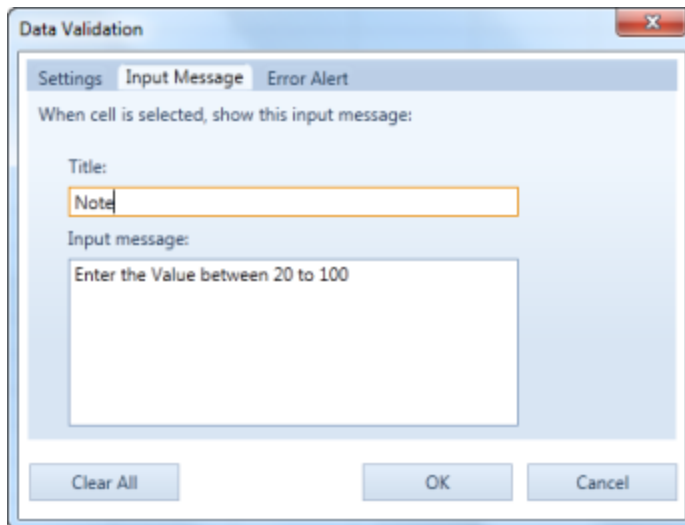
Events

Event	Description	Arguments	Type	Reference links
CurrentCellValidating	Will be triggered when the cell value is committed. This can be canceled. Error alert can be customized.	void CurrentCellValidating(object sender, CurrentCellValidatingEventArgs e)	Routed	NA

Defining Condition for Validating Cells

You can define the data validation to the Spreadsheet cells using the Data Validation dialog box. You have to specify the validation rule in the Settings tab, tooltip content in the Input Message tab and error message in Error Alert tab. You can open the data validation dialog using the `DataValidationCommand`.

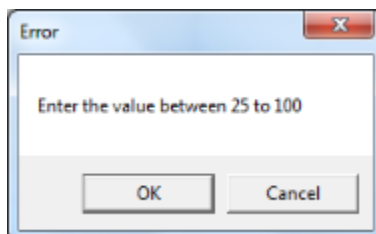




The input message will be displayed as tooltip, when the particular cell is in active state.

	A	B	C	D
1	50			
2	Note : Enter the value between 25 to 100			
3				
4				

The error message will be display only when you enter the value beyond the data validation limit.



When you click OK, the cell value will not be committed and when you click Cancel, it will revert the cell value.

The following code illustrates how to bind the `DataValidationCommand` to a button:

XML

```
<Button Command="{Binding Path=DataValidationCommand}">
</Button>
```

Comments

Comments provide additional information about a cell such as, what the value represents. Spreadsheet control allows you to insert, edit and delete comments in the Excel document.

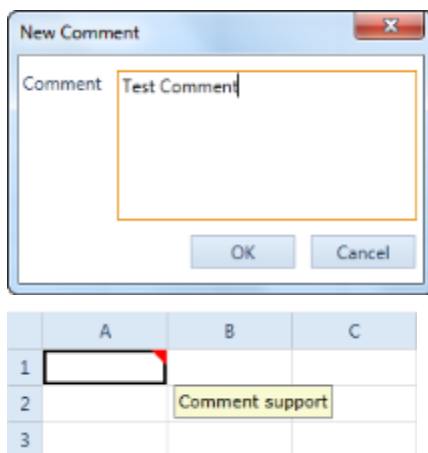
Insert comment

You can insert comment to the Spreadsheet cells using the New Comment dialog box. You can open the New Comment dialog box using the `InsertCommentCommand`.

The following code illustrates how to bind the `InsertCommentCommand` to a button:

XML

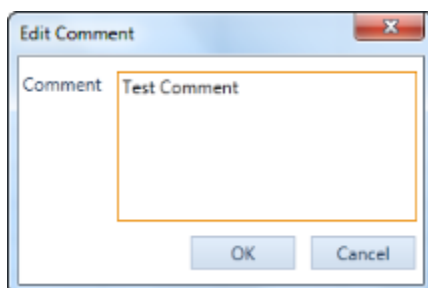
```
<Button Command="{Binding Path= InsertCommentCommand}">
</Button>
```



Note: You can insert only one comment for a cell. The `InsertCommentCommand` will open the Edit comment dialog box if the specific cell already has a comment.

Edit Comment

You can edit the existing comments using the Edit Comment dialog box. You can open the Edit comment dialog using the `InsertCommentCommand`.



The following code illustrates how to bind the `InsertCommentCommand` to a button:

XML

```
<Button Command="{Binding Path= InsertCommentCommand}">
</Button>
```

Delete Comment

You can delete the comment in the spreadsheet cells using the DeleteCommentCommand. When you execute this command it will delete the comment from the selected cells.

The following code illustrates how to bind the DeleteCommentCommand to a button:

XML

```
<Button Command="{Binding Path= DeleteCommentCommand}">
</Button>
```

Import and Export from Data Table

Spreadsheet offers some helper methods that enable you to import and export data form ADO.NET data sources very easily. The ImportDataTable and ExportDataTable methods allow you to use one line of code to import data from a DataTable to a SpreadSheet and export data from a SpreadSheet to a DataTable respectively.

	A	B	C	D	E	F	G	H
	Company Name	Contact Name	Contact Title	Address	City	Region	Postal Code	Country
1	ALFKI	Alfreda Pütt...	Maria Anders	Salto Kopp...	Oslo St. 57	Berlin	12209	Germany
2	ANATR	Ana Tringhi...	Ana Tringhi	Owner	Auda, de la	Mexico D.F.	5021	Mexico
3	ANTON	Antonio Mor...	Antonio Mor...	Owner	Matadero...	Mexico D.F.	5023	Mexico
4	AROUT	Arnaud Be...	Thomas Mar...	Sales Repre...	123 rueve...	London	W41 1DP	UK
5	BENGS	Berglund S...	Christina B...	Order Adm...	Berggröns...	Luleå	S-596 22	Sweden
6	BLAIS	Blauer Ste...	Hanna Moos	Sales Repre...	Friedenstr...	Mannheim	68309	Germany
7	BOLNP	Bonaldi p...	Fredérique...	Marketing M...	24, place M...	Strasbourg	67001	France
8	BOLID	Boldu Com...	Hartmut Sem...	Owner	C/ Aragall...	Madrid	28023	Spain
9	BONAP	Bonapace L...	Laurence L...	Owner	12, rue des...	Marseille	13008	France
10	BOTTM	Bottom-Doll...	Elizabeth G...	Accounting...	22 Tuusvas...	Tuusmasen	02101	Canada
11	BSBEV	B's Beverages	Victoria Ash...	Sales Repre...	Fawcett St...	London	EC2A 3HT	UK
12	CACSU	Cactus Com...	Patricia Sem...	Sales Agent	Centro 323	Buenos Aires	1010	Argentina
13	CENTC	Centro com...	Francisco C...	Marketing M...	Sierras de G...	Mexico D.F.	5022	Mexico
14	CHOPS	Chop-sum...	Yang Wang	Owner	Hakatastr...	Bern	3012	Switzerland
15	COMPX	Comptech M...	Pedro Afonso	Sales Agent...	Air. Rio Lan...	São Paulo	05453-043	Brazil
16	CONSH	Consolidate...	Elizabeth B...	Sales Repre...	Berkley Ca...	London	9012 4LT	UK
17	DRACD	Drachendoll...	Joan Ottob...	Order Adm...	Hallenweg 21	Aachen	52069	Germany
18	DURON	Durand e...	Julie Lab...	Owner	67, rue des...	Nantes	44000	France
19	ELACTE	Elactech Com...	Ann Dron...	Order Adm...	25 King's Cr...	London	W1P 4AD	UK

Samples Link

The samples for Importing from data table are located at:

[Essential Studio Dashboard > Classic > Spreadsheet Control > Data Management > Import Data Table.](#)

Refer to section 2.2 Samples and Location to access the samples location.

Methods

Method	Description	Parameters	Type	Return Type
ImportFromDataTable	This method imports data from the DataTable into the Spreadsheet.	ImportDataTable(DataTable dataTable)	N/A	void
ImportFromDataTable	This method imports data from a DataTable into a Spreadsheet with parameters Row and Column of the first cell, where DataTable should be imported.	ImportFromDataTable(DataTable table, bool isFieldNameShow, int startRow, int startCol)	N/A	void

ImportFromDataTable	This method imports data from a DataTable into a Spreadsheet with parameters Row index and column index of SpreadSheet, where DataTable should be imported and preserve types (This Indicates whether Spreadsheet should try to preserve types in DataTable)	ImportFromDataTable(DataTable table, bool isFieldNameShow, int startRow, int startCol, bool preserveTypes)	N/A	void
ImportFromDataTable	This method imports data from a DataTable into a Spreadsheet with parameters starting row index and column index and maximum number of rows and columns to import.	ImportFromDataTable(DataTable table, bool isFieldNameShow, int startRow, int startCol, int maxRow, int maxCol)	N/A	void
ImportFromDataTable	This method imports data from a DataTable into a Spreadsheet with parameters starting row index and column index and maximum number of rows and columns to import and preserve types.	ImportFromDataTable(DataTable table, bool isFieldNameShow, int startRow, int startCol, int maxRow, int maxCol, bool preserveTypes)	N/A	void

Importing from a Data Table

The following lines of code are used to import data from DataTable to Spreadsheet control.

C#

```
this.spreadsheetControl.ImportFromDataTable(datatable);
```

VB.NET

```
Me.spreadsheetControl.ImportFromDataTable(datatable)
```

Exporting to a Data Table

Similarly, you can to export the Spreadsheet data to a data table by using the ExportDataTable method of Worksheet. The following code demonstrates this:

C#

```
IWorksheet sheet =  
this.spreadsheetControl.ExcelProperties.WorkBook.Worksheets[0];  
IRange range = sheet.Range["A1:K50"];
```

```
DataTable Dt = sheet.ExportDataTable(range,
ExcelExportDataTableOptions.ColumnNames);
```

VB.NET

```
Dim sheet As IWorksheet =
Me.spreadsheetControl.ExcelProperties.WorkBook.Worksheets(0)
Dim range As IRange = sheet.Range("A1:K50")
Dim Dt As DataTable = sheet.ExportDataTable(range,
ExcelExportDataTableOptions.ColumnNames)
```

Clipboard Support

The Spreadsheet control provides support for clipboard operations. When you copy and paste the cells within the Spreadsheet control, it will paste the cell data and cell style to the Spreadsheet cells and also Spreadsheet also having the support for pasting formula with relative reference. When you copy and paste the cells from Spreadsheet control to other application, it will only copy the cell text.

Use Case Scenarios

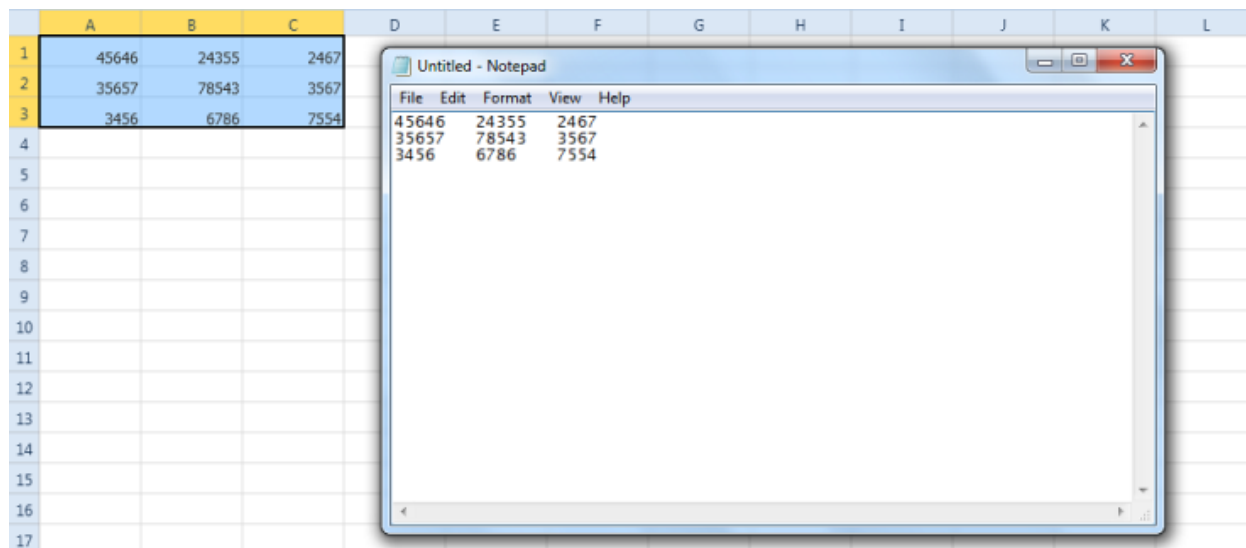
1. It is easier and faster to copy the data than to rewrite it every time you need it.
2. Spreadsheet control support Excel-like relative reference copy and paste operations. It also allows you to do clipboard operations like those available in Excel.

Copy and Paste Options

1. Normal copy paste
2. Relative reference copy and paste

Normal Copy and Paste

By default, the normal copy and paste option is enabled in the Spreadsheet control. When you copy the cells from Spreadsheet control, it will copy the cell text to the clipboard. Then you can paste the text in other applications. While copying the cell text to clipboard spreadsheet will add the tab symbol ("/t") as a delimiter.



Relative reference Copy and paste

If you copy and paste cells within the Spreadsheet control, then the cell styles will be preserved, and also the formula references will be automatically updated in the pasted cells.

If you are pasting the data in the spreadsheet cells, it will show this small pop-up with two options: one for pasting the formula, and one for pasting the value. By default, when you paste the copied formula cell, it will paste the formula in that cell. By using this pop-up, you can change this formula to a value.

	A	B	C	D	E	F	G	H
1	579	123	456					
2	1800	789	1011					
3								
4				1491	789	702		
5				3600	1011	2589		
6								
7								
8								
9								
10								

How to Disable the Relative Reference Copy and Paste

By default, this feature is enabled in the Spreadsheet control. You can disable this copy paste feature by using the following code:

C#

```
spreadsheetControl.GridProperties.ActiveSpreadsheetGrid.Model.GridCopyPaste
= null;
```

Events

The Spreadsheet control provides the following events for customizing the clipboard data.

- ClipboardCanCopy
- ClipboardCanCut
- ClipboardCanPaste
- ClipboardCopy

- ClipboardCut
- ClipboardPaste

You can get all grid control objects in the WorkbookLoaded event of the Spreadsheet control. By using these objects, you can handle the grid events.

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 the data.

1. The grid cell data and range of cells that are going to be moved to the clipboard can be accessed by using the DataObject and RangeList properties.
2. If you don't want to move the data to the clipboard, you can cancel the operation by setting e.Cancel to true.

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.

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.

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.

ClipboardCanPaste

This event is 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.

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.

ClipboardCopy

This event is 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.

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.

ClipboardCut

This event is 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.

Properties

Property	Description
DataObject	Data to be 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.

ClipboardPaste

This event is 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.

Properties

Property	Description
<code>DataObject</code>	Data to be pasted.
<code>RangeList</code>	List of cell ranges that are selected for pasting.
<code>Handled</code>	When true, indicates that the event has been handled and no further processing of the event should happen.

How to Copy and Paste the Display Text

By default, when you copy and paste the formula cell, the Spreadsheet control will copy and paste the formula from the particular cell. To copy and paste the display text you have to handle the `ClipboardCopy` event of the underling grid control. You can get all the Grid controls in the `WorkbookLoaded` event of the spreadsheet control.

The following code used to copy and paste only the display text.

C#

```
void spreadsheetControl_WorkBookLoaded(object sender, WorkbookLoadedEventArgs args)
{
    foreach (var grid in args.GridCollection)
    {
        grid.Model.GridCopyPaste = null;
        grid.Model.ClipboardCopy += new GridCutPasteEventHandler(Model_ClipboardCopy);
    }
}

void Model_ClipboardCopy(object sender, GridCutPasteEventArgs e)
{
    GridRangeInfoList rowRanges = e.RangeList.GetRowRanges(GridRangeInfoType.Cells | GridRangeInfoType.Rows);
    GridRangeInfoList colRanges = e.RangeList.GetColRanges(GridRangeInfoType.Cells | GridRangeInfoType.Cols);
    var sb = new StringBuilder();
    int nrowdone = 0; int ncolsdone = 0;
    string tabDelim = "\t";
    for (int rowindex = 0; rowindex < rowRanges.Count; rowindex++)
    {
        for (int nrow = rowRanges[rowindex].Top; nrow <= rowRanges[rowindex].Bottom; nrow++)
        {
            if (nrowdone > 0)
            {
                sb.Append(Environment.NewLine);
                ncolsdone = 0;
            }
            bool firstCol;
            firstCol = true;
            for (int colindex = 0; colindex < colRanges.Count; colindex++)
            {
```

```

for (int ncol = colRanges[colindex].Left; ncol <= colRanges[colindex].Right;
    ncol++)
{
    if (!firstCol)
    sb.Append(tabDelim);
    string text = this.spreadsheetControl.GridProperties.CurrentExcelGridModel[n
row, ncol].FormattedText;
    if (!e.RangeList.AnyRangeContains(GridRangeInfo.Cell(nrow, ncol)))
    {
        ncoldone++;
        continue;
    }
    text = new StringBuilder(text).ToString().Trim();
    // Append the Cell value to buffer text
    sb.Append(text);
    firstCol = false;
    ncoldone++;
}
}
nrowsdone++;
}
}
Clipboard.SetText(sb.ToString());
e.DataObject = null;
e.Handled = true;
}

```

Documents Settings in WPF SpreadsheetControl (Classic)

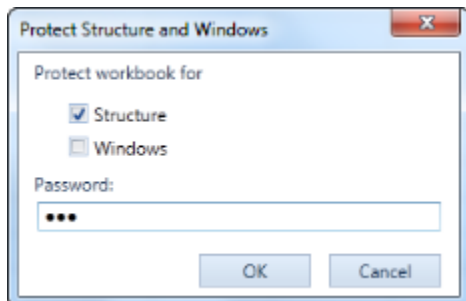
Protect and unprotect workbook

Spreadsheet control provides support to protect and unprotect the workbook with a password. When you protect the workbook the following option will be disabled:

- Insert Worksheet
- Delete Worksheet
- Hide Worksheet
- Unhide Worksheet

Protect workbook

You can protect the workbook using the Protect Structure and Windows dialog box. You can open the Protect Structure and Windows dialog using the ProtectWorkbookCommand.



The following code illustrates how to bind the ProtectWorkbookCommand to a button:

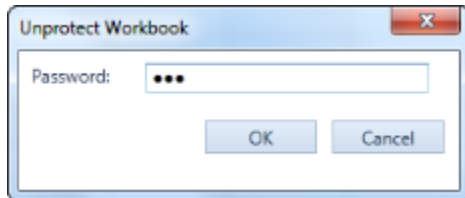
XML

```
<Button Command="{Binding Path= ProtectWorkbookCommand}">
</Button>
```

Note: The ProtectWorkbookCommand will open the Unprotect Workbook dialog box if the workbook is already protected.

Unprotect workbook

You can unprotect the workbook using the Unprotect Workbook dialog box. You can open the Unprotect Workbook dialog using the ProtectWorkbookCommand.



Protect and unprotect worksheet

Spreadsheet control supports to protect and unprotect a worksheet. You can set the worksheet as read only. There are two methods to protect and unprotect the worksheet. They are:

- Using method
- Using command

Protect/Unprotect worksheet using method

Protect

To set a worksheet as password protected, pass the sheet name and the password as String to ProtectSheet method. This prevents unwanted changes to the worksheet.

Following code illustrates this:

C#

```
spreadControl.ProtectSheet("Sheet1", "asd123");
```

VB.NET

```
spreadControl.ProtectSheet("Sheet1", "asd123")
```

UnProtect

To unprotect a worksheet, pass the sheet name and the password as String to the UnProtectSheet method.

C#

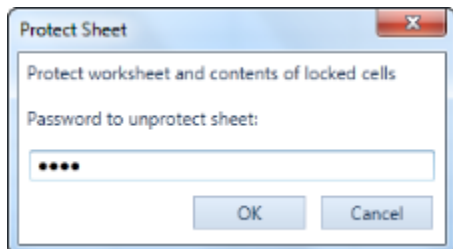
```
spreadControl.UnProtectSheet("Sheet1", "asd123");
```

VB.NET

```
spreadControl.UnProtectSheet("Sheet1", "asd123")
```

Protect/Unprotect worksheet using Command

You can protect/unprotect the current worksheet using the Protect Sheet dialog box. You can open the Protect Sheet dialog using the `ProtectCurrentSheetCommand`. *If the current worksheet is already protected* Protect Sheet dialog box will unprotect the worksheet after confirming the password.



The following code illustrates how to bind the `ProtectCurrentSheetCommand` to a button:

XML

```
<Button Command="{Binding Path= ProtectCurrentSheetCommand}">
</Button>
```

Encrypt workbook

Encryption

Encryption is used to protect the workbook data with password. It converts the data into an encrypted format (unreadable). This restricts anonymous users to access the data. You can achieve this by two methods. They are:

- Using method
- Using command

Method

To encrypt a workbook, pass the password in the `EncryptWorkBook` method. The following code illustrates this:

C#

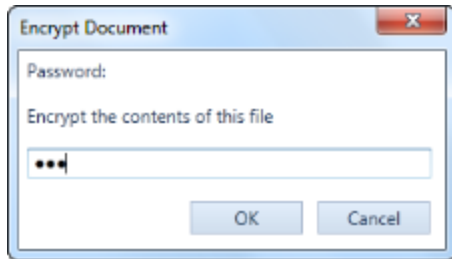
```
spreadControl.EncryptWorkBook("asd123");
```

VB.NET

```
spreadControl.EncryptWorkBook("asd123")
```

Encrypt workbook using the Commands

You can also encrypt the workbook using the Encrypt Document dialog box. You can open the Encrypt Document dialog using the `EncryptCommand`.



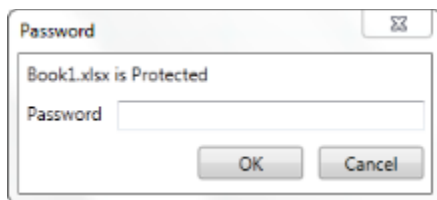
The following code illustrates how to bind the EncryptCommand to a button:

XML

```
<Button Command="{Binding Path= EncryptCommand}">
</Button>
```

Open Encrypted Document

When you open encrypted document Password dialog box will open. Enter password to decrypt the data.



Hide and Unhide Worksheet

This feature enables you to hide/unhide a worksheet. You can achieve this in two methods:

- Using method
- Using command

Hide worksheet using method

To hide a worksheet, pass the sheet name to the HideSheet method:

C#

```
spreadControl.HideSheet("Sheet1");
```

VB.NET

```
spreadControl.HideSheet("Sheet1")
```

Unhide worksheet using method

To unhide a worksheet, pass the sheet name to the UnHideSheet method:

C#

```
spreadControl.UnHideSheet("Sheet1");
```

VB.NET

```
spreadControl.UnHideSheet("Sheet1")
```

Hide worksheet using comment

You can hide the current worksheet by using the HideCurrentSheetCommand. When you execute this command based on the command parameter it will hide or unhide the worksheet.

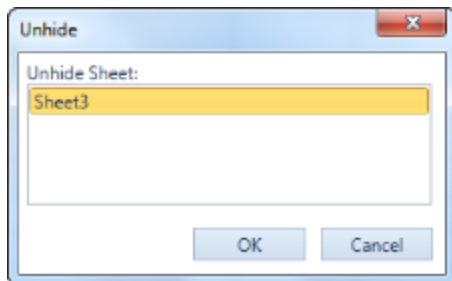
The following code illustrates how to bind the HideCurrentSheetCommand to a button for hide current worksheet:

XML

```
<Button Command="{Binding Path= HideCurrentSheetCommand}">
<System:bool>true</System:bool>
</Button>
```

Unhide worksheet using command

You can unhide the worksheet by using the Unhide dialog box. To open the Unhide dialog box, pass the command parameter as false in the HideCurrentSheetCommand.



The following code illustrates how to bind the HideCurrentSheetCommand to a button for unhide worksheet:

XML

```
<Button Command="{Binding Path= HideCurrentSheetCommand}">
<System:bool>>false</System:bool>
</Button>
```

Add or Remove Worksheet

This feature enables you to add or remove a worksheet from a workbook.

Add a worksheet using method

To add a worksheet, pass the sheet name to the AddSheet method.

C#

```
spreadControl.AddSheet("Sheet4");
```

VB.NET

```
spreadControl.AddSheet("Sheet4")
```

You can also specify the position of the added worksheet by passing the sheet name and position to the AddSheet method.

C#

```
spreadControl.AddSheet("Sheet4", 3);
```

VB.NET

```
spreadControl.AddSheet("Sheet4", 3)</td></tr>
```

Remove a worksheet using method

To remove a worksheet, pass the sheet name to the RemoveSheet method.

C#

```
spreadControl.RemoveSheet("Sheet4");
```

VB.NET

```
spreadControl.RemoveSheet("Sheet4")
```

Add a worksheet using command

You can add a worksheet by using the InsertSheetCommand. When you execute this command, it will add a new worksheet at the end of the workbook.

The following code illustrates how to bind the InsertSheetCommand to a button:

XML

```
<Button Command="{Binding Path= InsertSheetCommand}">
</Button>
```

Remove a worksheet using Command

You can delete current worksheet using the DeleteCurrentSheetCommand. When you execute this command, it will delete the current worksheet.

The following code illustrates how to bind the DeleteCurrentSheetCommand button:

XML

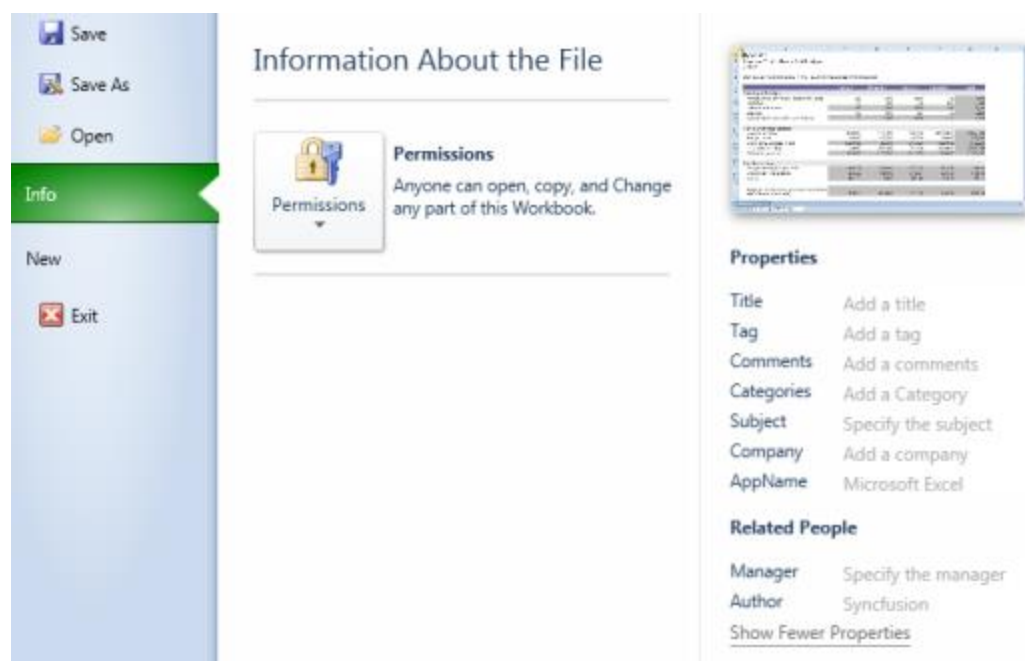
```
<Button Command="{Binding Path= DeleteCurrentSheetCommand}">
</Button>
```

Document Settings Options

Document properties are named values that provide information about the document namely author, manager, title, tags, commands, subject, company and AppName. Values for these properties can be changed.

Use Case Scenario

When importing and exporting and the Excel sheet, users can get and set the values of the Excel properties.



Properties

Property	Description	Type	Data Type
Author	Used to get or set the author of the Excel	Dependency	String
ApplicationName	Used to get or set the application name of the Excel	Dependency	String
Category	Used to get or set the CategoryName of the Excel	Dependency	String
Comments	Used to get or set the comments of an Excel	Dependency	String
Company	Used to get or set the company name of the Excel		
Subject	Used to get or set the subject of the Excel	Dependency	String
Keywords	Used to get or set the keywords of the Excel	Dependency	String
Manager	Used to get or set the Manager's name	Dependency	String
Title	Used to get or set the Title	Dependency	String

Setting Values in Document Properties

The following code snippets are used to set values in Document properties:

C#

```

this.spreadSheetControl.ExcelProperties.Author = "Syncfusion";
this.spreadSheetControl.ExcelProperties.Manager = "Syncfusion Manager";
this.spreadSheetControl.ExcelProperties.Category = "Spreadsheet control";
this.spreadSheetControl.ExcelProperties.Comments = "This document generated by spreadsheet control";
this.spreadSheetControl.ExcelProperties.Company = "Syncfusion";
this.spreadSheetControl.ExcelProperties.Title = "Spreadsheet control";
this.spreadSheetControl.ExcelProperties.Keywords = "Sync";

```

```
this.spreadSheetControl.ExcelProperties.Subject = "Native Excel generator";
this.spreadSheetControl.ExcelProperties.ApplicationName = " Spreadsheet ";
```

VB.NET

```
Me.spreadSheetControl.ExcelProperties.Author = "Syncfusion"
Me.spreadSheetControl.ExcelProperties.Manager = "Syncfusion Manager"
Me.spreadSheetControl.ExcelProperties.Category = "Spreadsheet control"
Me.spreadSheetControl.ExcelProperties.Comments = "This document genera ted
by spreadsheet control"
Me.spreadSheetControl.ExcelProperties.Company = "Syncfusion"
Me.spreadSheetControl.ExcelProperties.Title = "Spreadsheet control"
Me.spreadSheetControl.ExcelProperties.Keywords =
"Sync"Me.spreadSheetControl.ExcelProperties.Subject = "Native Excel
generator"
Me.spreadSheetControl.ExcelProperties.ApplicationName = "Spreadsheet "
```

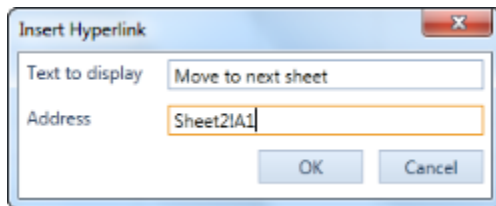
Links in WPF SpreadsheetControl (Classic)

Bookmarks and Hyperlinks

A hyperlink is a convenient way to navigate from one workbook to another workbook. Links created in the native Excel will be imported to the Spreadsheet Control. You can also add new links.

Adding hyperlink to Cell

You can add the hyperlink to the spreadsheet cell using the Insert Hyperlink dialog box. You can open the Insert Hyperlink dialog using the HyperlinkCommand.



The following code illustrates how to bind the HyperlinkCommand button:

XML

```
<Button Command="{Binding Path= HyperlinkCommand}">
</Button>
```

Command Support in WPF SpreadsheetControl (Classic)

The Spreadsheet control provides the built-in commands to show case the features available in Spreadsheet. It can be executed in the Spreadsheet whenever it is bound to Command Property.

Commands Table

Property	Description	Type	Data Type	Reference links
BoldCommand	Command which will toggle bold.	Dependency Property	CommandBase	NA

ItalicCommand	Command which will toggle Italics.	Dependency Property	CommandBase	NA
UnderlineCommand	Command to change the underline.	Dependency Property	CommandBase	NA
CopyCommand	Command to copy the selected cells.	Dependency Property	CommandBase	NA
CutCommand	Command to cut the selected cells.	Dependency Property	CommandBase	NA
PasteCommand	Command to paste the cells from the clipboard.	Dependency Property	CommandBase	NA
ColumnWidthCommand	Command to customize the column width.	Dependency Property	CommandBase	NA
ConditionalFormatCommand	Command to apply conditional formatting for the selected cells.	Dependency Property	CommandBase	NA
DataValidationCommand	Command to apply data validation for the selected cells.	Dependency Property	CommandBase	NA
DeleteCommentCommand	Command to delete the comment.	Dependency Property	CommandBase	NA
DeleteColumnCommand	Command to delete column.	Dependency Property	CommandBase	NA
DeleteRowCommand	Command to delete row.	Dependency Property	CommandBase	NA
DeleteCurrentSheetCommand	Command to delete current sheet.	Dependency Property	CommandBase	NA
EncryptCommand	Command to encrypt the workbook.	Dependency Property	CommandBase	NA
ExportToExcelCommand	Command to save the workbook as Excel.	Dependency Property	CommandBase	NA

FormatAsTableCommand	Command to format as table.	Dependency Property	CommandBase	NA
HyperlinkCommand	Command to insert hyperlink.	Dependency Property	CommandBase	NA
ImportFromExcelCommand	Command to import the Excel document to Spreadsheet.	Dependency Property	CommandBase	NA
InsertCommentCommand	Command to insert comment.	Dependency Property	CommandBase	NA
InsertPictureCommand	Command to insert picture.	Dependency Property	CommandBase	NA
InsertRowCommand	Command to insert row.	Dependency Property	CommandBase	NA
InsertSheetCommand	Command to insert worksheet.	Dependency Property	CommandBase	NA
ProtectCurrentSheetCommand	Command to protect current worksheet.	Dependency Property	CommandBase	NA
ProtectWorkbookCommand	Command to protect workbook.	Dependency Property	CommandBase	NA
RowHeightCommand	Command to customize row height.	Dependency Property	CommandBase	NA
ShowGridLinesCommand	Command to show or hide gridlines.	Dependency Property	CommandBase	NA
RowColumnHeadersVisibilityCommand	Command to show or hide row and column header.	Dependency Property	CommandBase	NA

Parameterized Commands

Property	Description	Type	Data Type	Reference links	
BorderCommand	Command to customize the cell border.	Dependency Property	CommandBase	String	NA

FreezePaneCommand	Command to freeze rows and column.	Dependency Property	CommandBase	Freeze	NA
HideColumnCommand	Command to hide the column.	Dependency Property	CommandBase		NA
HideRowCommand	Command to hide the row.	Dependency Property	CommandBase	Boolean	NA
HideCurrentSheetCommand	Command to hide the current sheet.	Dependency Property	CommandBase	Boolean	NA
VerticalAlignmentCommand	Command to align the cells vertically.	Dependency Property	CommandBase	String	NA
MergeCommand	Command to merge the cells.	Dependency Property	CommandBase	String	NA
NewCommand	Command to add new workbook.	Dependency Property	CommandBase	Int	Creating an Excel Document
NumberFormatCommand	Command to apply number formatting.	Dependency Property	CommandBase	String	NA
IncreaseDecimalCommand	Command to increase decimal places.	Dependency Property	CommandBase	Boolean	NA
IncreaseIndentCommand	Command to increase text margin.	Dependency Property	CommandBase	Boolean	NA
InsertColumnCommand	Command to insert column.	Dependency Property	CommandBase	Boolean	NA
HorizontalAlignmentCommand	Command to align the cells horizontally.	Dependency Property	CommandBase	String	NA
CellStyleCommand	Command to apply cell style.	Dependency Property	CommandBase	BuiltInStyles	NA

FontFamilyCommand	Command to specify the font family.	Dependency Property	CommandBase	String	NA
FontSizeCommand	Command to specify the font size.	Dependency Property	CommandBase	Double	NA

Localization in WPF SpreadsheetControl (Classic)

Localization is the process of customizing the application to culture-specific. This involves configuring the application for the specific languages. Culture is the combination of Language and the Location (e.g. En-US is the Culture for English spoken at United States; En-GB is the Culture for English spoken at Great Britain).

Syncfusion Spreadsheet allows you to set custom resource through Resx file. You can give the string values in resource file for a specific Culture and set the Culture in the application. The given string values will be set to the Grid which does not affect the Code Block of the Grid.

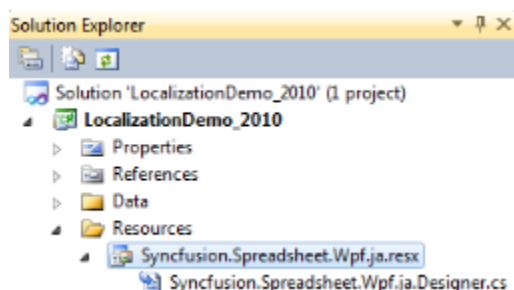
Adding Localization to an Application

The following are steps to implementation Localization support to an application:

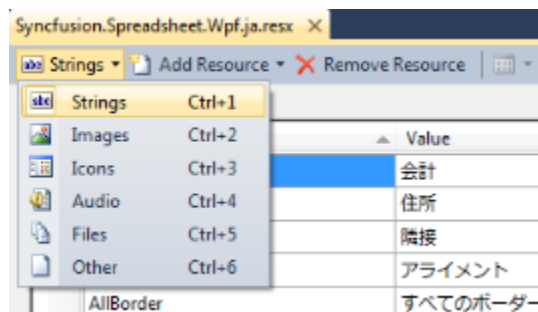
1. Create a WPF application and add spreadsheet control to it.
2. Create a folder named Resources in the application.
3. Create a resource file (Resx file) and name it as Syncfusion.Spreadsheet.WPF. .resx

Example: Syncfusion.Spreadsheet.WPF.ja.resx.

Note: It is mandatory to use this naming convention.



4. Select the String option in the Resource file.



5. Resource table will open.
6. Enter the UI name in the Name column and the equivalent term you want in the Value column.

	Name	Value	Comment
▶	Accounting	会計	
	Address	住所	
	Adjacency	隣接	
	Alignment	アライメント	
	AllBorder	すべてのボーダー	
	Allow	許可	
	And	や	
	Angels	天使たち	
	AnyValue	任意の値	
	Apex	頂点	
	Apothecary	薬割師	
	Aspect	側面	
	Austin	オースティン	
	Author	著者	
	Bad		
	Between	間	
	Between_Lower	間	
	BetweenWindowDescription	の間にあるセルの書式設定 :	
	BetweenWindowTitle	間	
	Blacktie	黒のネクタイ	
	BlankWorkbook	空白のワークブック	
	BlankWorkbookDescription	空のブックを作成します。	

localizing the application to the Japanese Culture

7. Assign the CultureInfo to the application before the InitializeComponent() method is being called.

The following code illustrates localizing the application to the Japanese CultureInfo.

CSHARP

```
public MainPage ()
{
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("ja-JP");
    InitializeComponent();
}
```

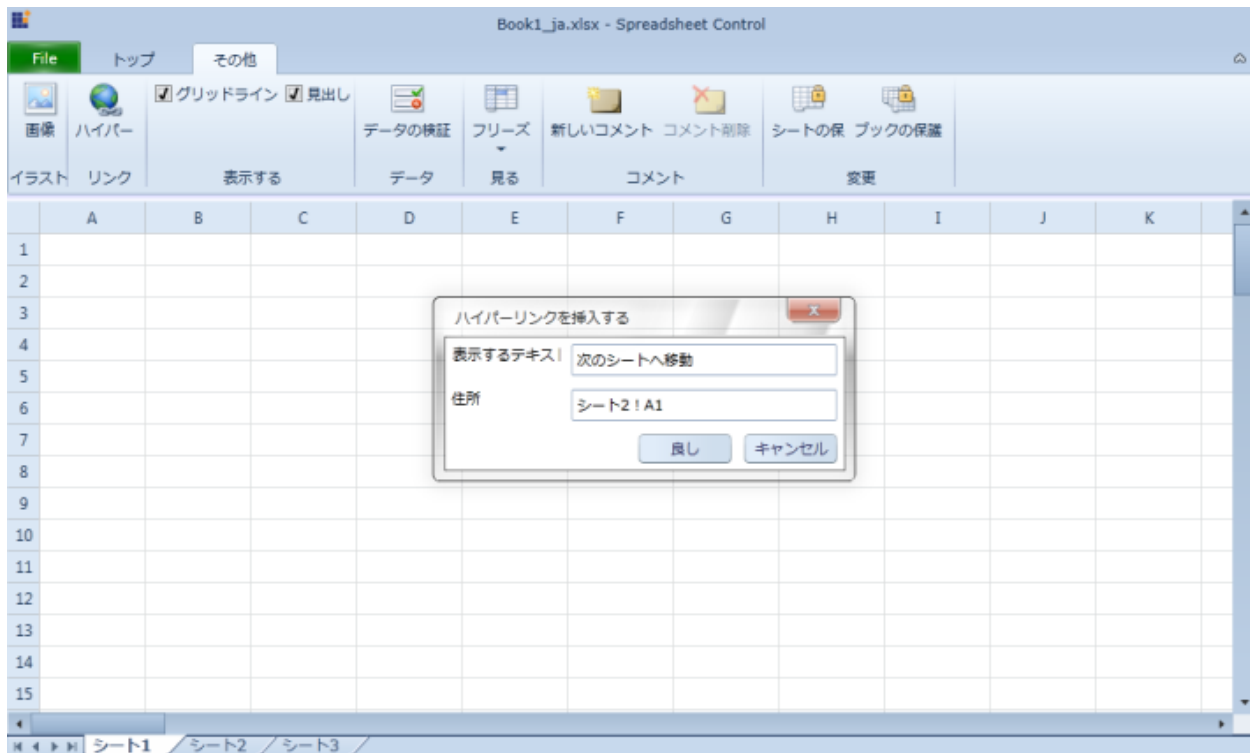
VB.NET

```
Public Sub New ()
    System.Threading.Thread.CurrentThread.CurrentUICulture = New
    System.Globalization.CultureInfo("ja-JP")
    InitializeComponent()
End Sub
```

8. Add Supported Cultures to the Application

The following are the steps to add Supported Cultures:

1. In the Solution Explorer, right-click application project and choose Unload Project from the Context Menu. The project will be unloaded.
2. Right click the project again, and select the Edit .csproj option.
3. Example: LocalizationSample WPF.csproj
4. In the .csproj file, find the tags. Default the tags will be empty. So, add the required cultures. Use semicolon to separate if you want to add multiple culture.
5. Example: en-GB;de;hi;es;it;Ja
6. Save the project.
7. Right click the .csproj and choose Reload .csproj. Project will be added with specific culture.



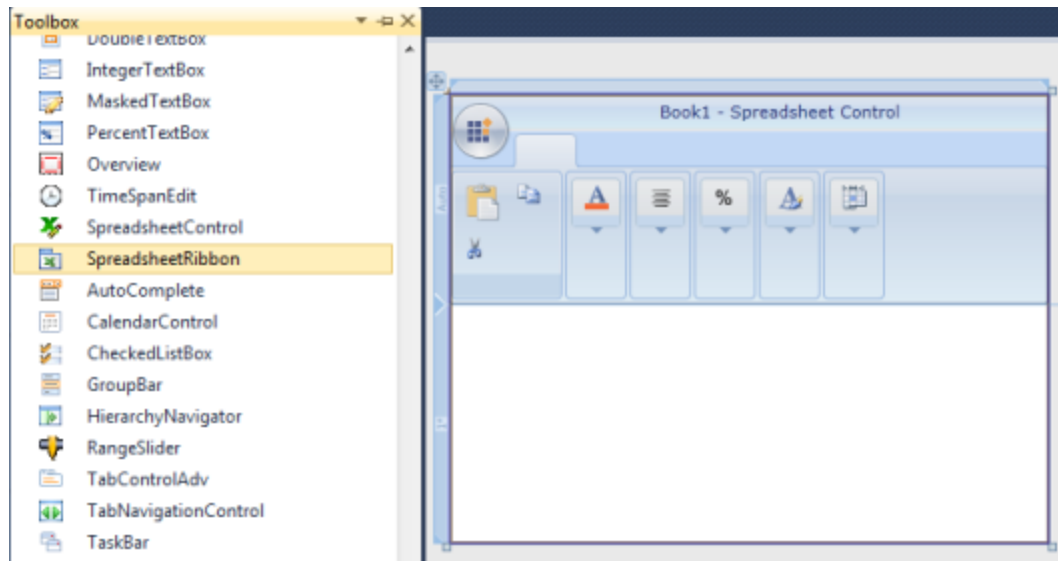
[Sample Link](#)

A demo of localization is available at the following location:

Essential Studio WPF Sample Browser - Spreadsheet - Localization - Localization Demo.

[SpreadsheetRibbon Support in WPF SpreadsheetControl \(Classic\)](#)

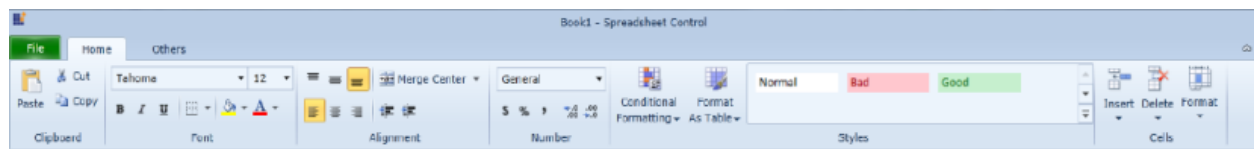
Spreadsheet control provides SpreadsheetRibbon support to format the spreadsheet cell and so on. It gives Microsoft Excel like look and feel. You can add SpreadsheetRibbon by drag and drop the SpreadsheetRibbon from Toolbox.



The following XAML code will be automatically added to the XAML viewer. You can also add this manually to add the SpreadsheetRibbon:

XML

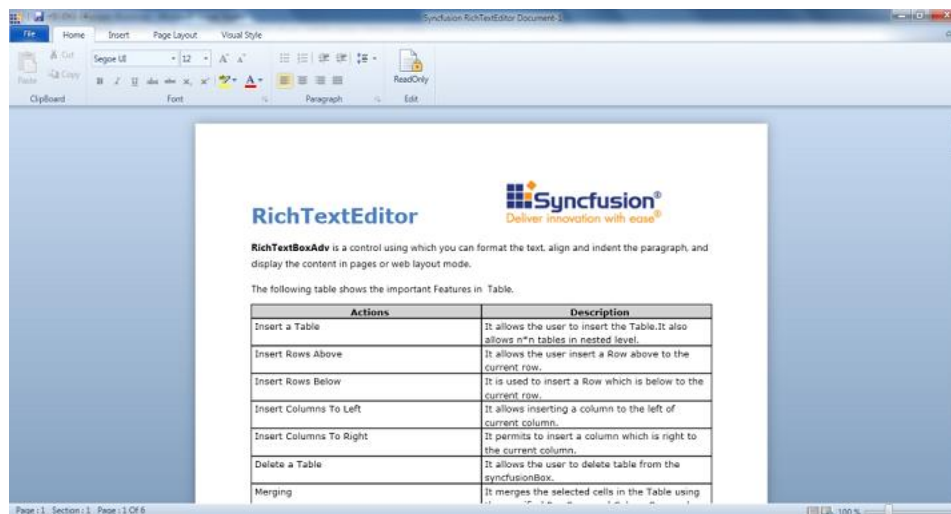
```
<Syncfusion:SpreadsheetRibbon DataContext="{Binding
ElementName=spreadControl}"/>
```



RichTextBoxAdv (Classic)

WPF RichTextBoxAdv (Classic) Overview

RichTextBoxAdv is a control for formatting text, such as aligning and indenting paragraphs, and displaying the content in pages or in a Web layout mode.



The features of the RichTextBoxAdv control include:

1. Clipboard support
2. Text formatting
3. Paragraph alignment and indentation features like text alignment, line spacing, left indent, right indent, before spacing, and after spacing
4. Command support
5. Insert an image or UIElement into RichTextBoxAdv
6. Page layout
7. Import and export feature for .doc, .docx, .html, .xaml, and .txt file formats
8. Printing and zooming
9. Keyboard shortcuts
10. Undo and redo support
11. Disable editing using the IsReadOnly property

Use Case Scenarios

This control helps users with rich text formatting such as bold, italic, highlighting, aligning, indenting paragraphs, and so on. It can also be used in forums, blogs, and so on.

1. It can be used like Microsoft Word in rich content applications.
2. It can be used as an HTML editor control as it has the HTML export feature.
3. The built-in commands help to create customized toolbar applications.

Sample Link

The access the samples:

- Select Start > Programs > Syncfusion > Essential Studio xx.x.x.xx > Dashboard.
- Select Run Locally Installed Samples.
- Now expand the RichTextBoxAdv item in the sample browser.
- Choose the OfficeUI Demo.

Getting Started with WPF RichTextBoxAdv (Classic)

Adding RichTextBoxAdv to an Application

The RichTextBoxAdv control can be added to an application by using the following applications:

1. Microsoft Visual Studio
2. Microsoft Expression Blend

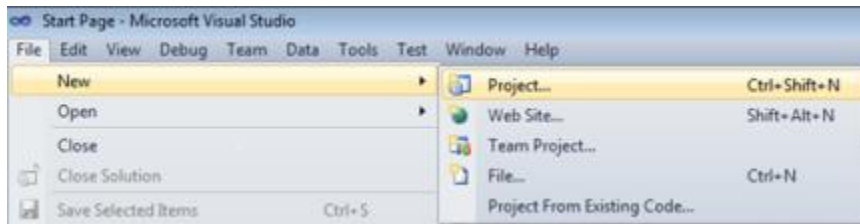
Note: Starting with v16.2.0.41 (2018 Vol 2), if you reference Syncfusion assemblies from trial setup or from the NuGet feed, you also have to add "Syncfusion.Licensing" assembly reference and include a license key in your projects. Please refer to this [link](#) to know about registering Syncfusion license key in your WPF application to use our components.

Creating the RichTextBoxAdv Control in Visual Studio

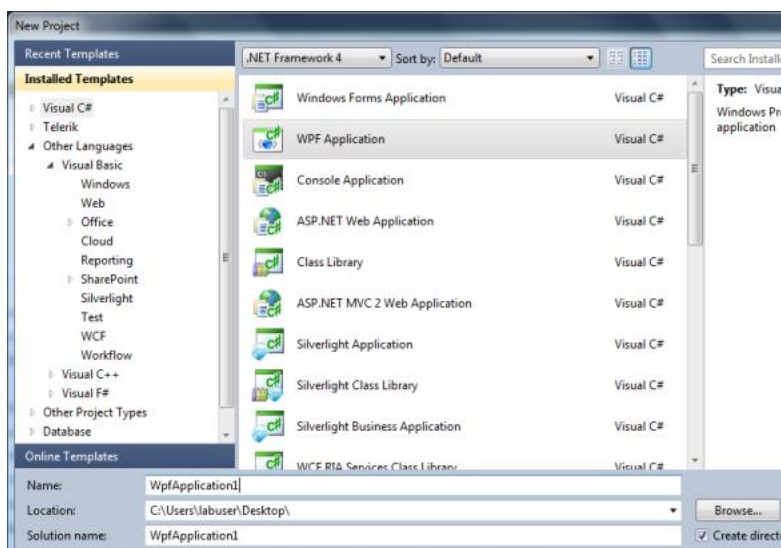
To create a RichTextBoxAdv instance in Visual Studio:

1. Open Visual Studio.

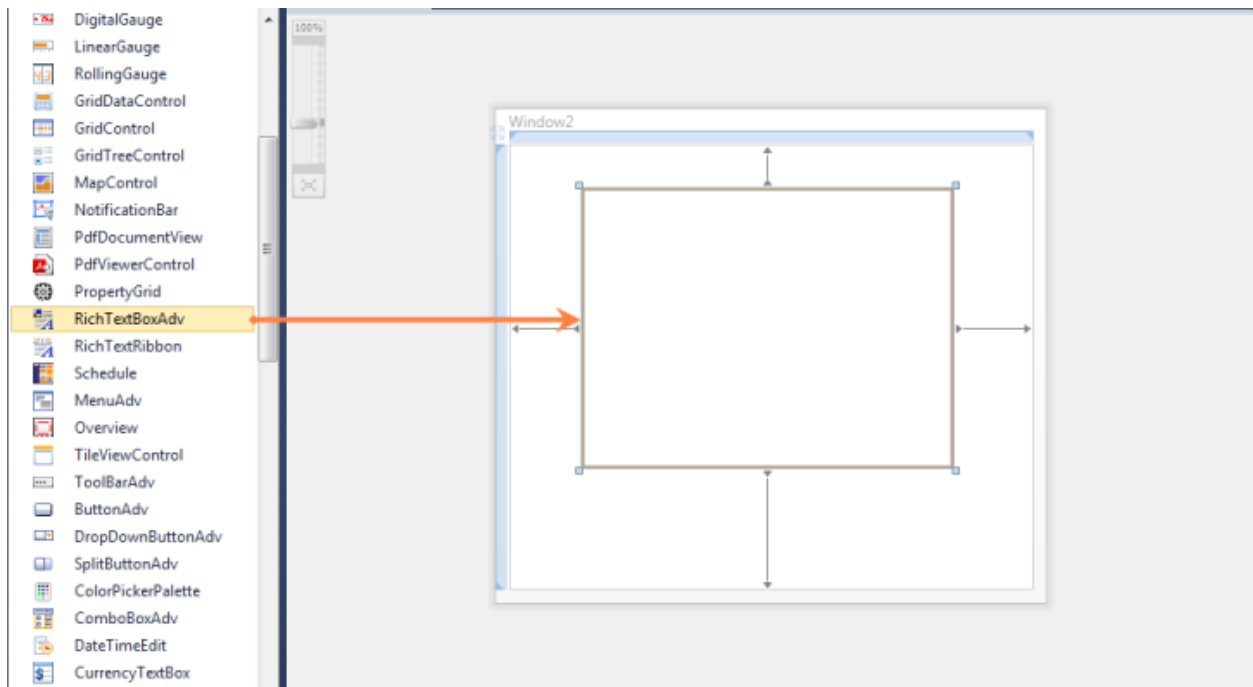
2. On the File menu, select New, and then select Project. The New Project dialog box is displayed.



3. In the New Project dialog box, select WPF Application.
4. In the Name field, type the name of the project.
5. Click OK.



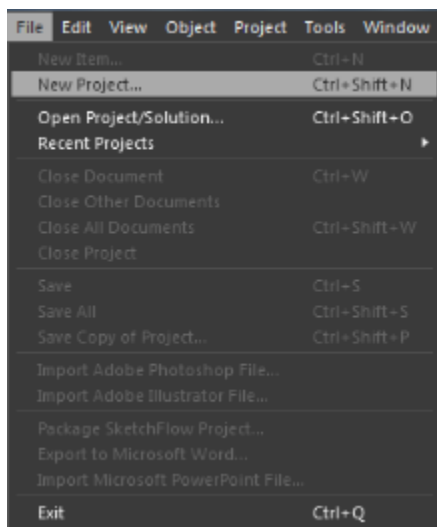
6. Drag the RichTextBoxAdv control from the Toolbox window to the Design View. An instance of the RichTextBoxAdv control is created in Design view.



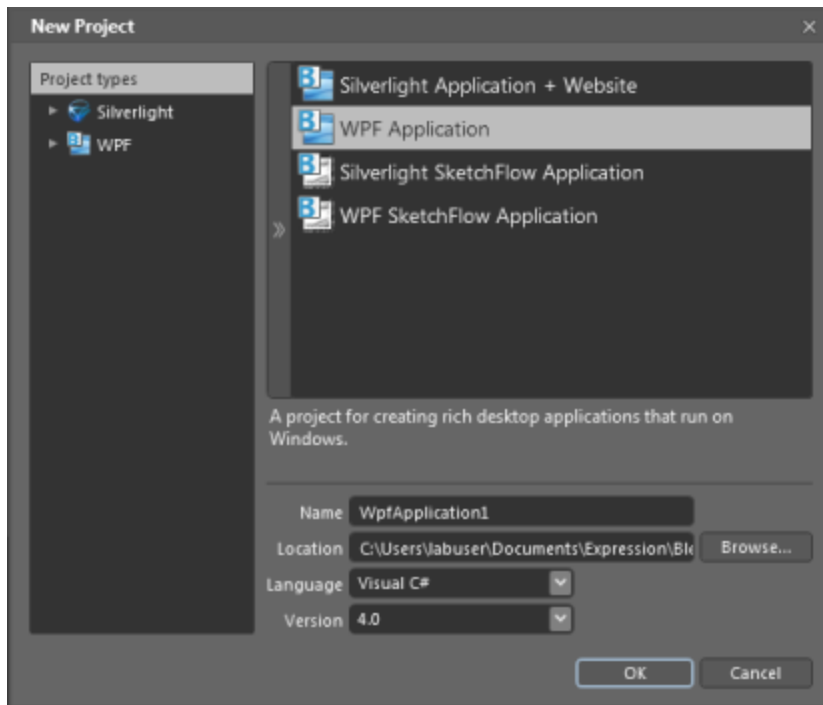
Creating the RichTextBoxAdv Control in Expression Blend

To create a RichTextBoxAdv instance in Expression Blend:

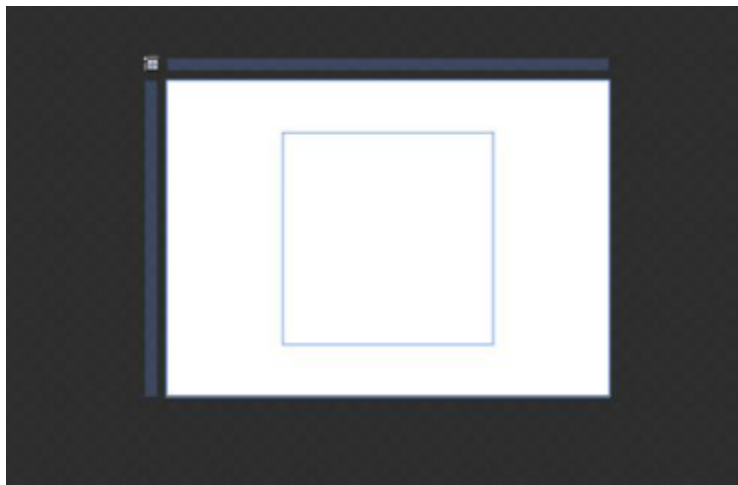
1. Open Expression Blend.
2. On the File menu, select New Project. The New Project dialog box opens.



3. In the Project types pane, select WPF, and then select WPF Application.
4. In the Name field, type the name of the project, and then click OK.



5. On the Window menu, select Assets. The Assets Library dialog box opens.
6. In the Search box, type RichTextBoxAdv. The search results are displayed.
7. Drag the RichTextBoxAdv control to the Design view. An instance of the RichTextBoxAdv control is created.



Appearance

You can customize the appearance of the RichTextBoxAdv control by editing the style of the control in ExpressionBlend or by using the properties exposed by the RichTextBoxAdv control.

Blendability

You can edit the style of the RichTextBoxAdv control by using Expression Blend. To edit the RichTextBoxAdv control's style in Expression Blend:

1. Drag the RichTextBoxAdv control to the Design view. The RichTextBoxAdv control will appear as shown in the screen shot displayed below. 2. Right-click the RichTextBoxAdv control, select Edit Template, and then select Edit a Copy.



Paragraph Alignment and Indentation in WPF RichTextBoxAdv (Classic)

The paragraph alignment and indentation features allow you to define content in paragraphs. They can be added only inside the sections. We can keep n number of paragraphs inside SectionAdv. Each paragraph may contain inline elements such as SpanAdv, HyperlinkAdv, ImageContainerAdv, and UIContainerAdv.

Properties

Property	Description	Type	Data Type
TextAlignment	Aligns the paragraph to left, right, center, and justified.	Dependency Property	TextAlignment
AfterSpacing	Changes the spacing between the current paragraph and the next paragraph.	Dependency Property	Double
BeforeSpacing	Changes the spacing between the current paragraph and previous paragraph.	Dependency Property	Double
LineSpacing	Changes the spacing between lines of text.	Dependency Property	Double
LeftIndent	Specifies the indent at the left side of the paragraph.	Dependency Property	Double
RightIndent	Specifies the indent at the right side of the paragraph.	Dependency Property	Double
ListType	Indicates whether the current paragraph is bulleted or numbered.	Dependency Property	ListType
Inlines	Contains the inline elements to be added in the paragraph.	Dependency Property	InlineCollection

Adding Paragraph Alignment to an Application

ParagraphAdv can be added directly to an application using the following code snippet:

XML

```
<syncfusion:RichTextBoxAdv Height="300" Width="400" x:Name="richtext">
<syncfusion:DocumentAdv>
<syncfusion:SectionAdv><syncfusion:ParagraphAdv TextAlignment="Center"
AfterSpacing=".3" ListType="Bulleted">
</syncfusion:ParagraphAdv>                                </syncfusion:SectionAdv>
</syncfusion:DocumentAdv>
</syncfusion:RichTextBoxAdv>
```

C#

```
RichTextBoxAdv richtext = new RichTextBoxAdv();
DocumentAdv document = new DocumentAdv();
SectionAdv section = new SectionAdv();
ParagraphAdv paragraph = new ParagraphAdv();
paragraph.TextAlignment = TextAlignment.Center;
paragraph.AfterSpacing = .3;
paragraph.ListType = ListType.Bulleted;
section.Blocks.Add(paragraph);
document.Sections.Add(section);
richtext.Document = document;
```

Text Formatting Using SpanAdv in WPF RichTextBoxAdv (Classic)

As Inline will be the content property of ParagraphAdv, SpanAdv can be added only inside the ParagraphAdv. Every ParagraphAdv can keep *n* number of inline elements inside it. It allows you to display the formatted text using advanced features like font size, font family, strikethrough, and baseline.

Properties

Property	Description	Type	Data Type
Text	Contains the text information.	Dependency Property	String
Baseline	Indicates whether the inline's text is subscript or superscript.	Dependency Property	Baseline
FontFamily	Specifies the font family of the text.	Dependency Property	FontFamily
FontSize	Specifies the size of the font.	Dependency Property.	Double
Foreground	Designates the font color.	Dependency Property	Color
HighlightColor	Designates the highlight color of the text.	Dependency Property	Color
Strikethrough	Shows whether the text has SingleStrikeThrough or DoubleStrikeThrough.	Dependency Property	StrikeThrough
FontWeight	Indicates the font weight of the text.	Dependency Property	FontWeight
Underline	Indicates whether the text should be underlined.	Dependency Property	Underline

Adding SpanAdv to an Application

SpanAdv can be added directly to an application using the following code snippet:

XML

```
<syncfusion:RichTextBoxAdv Height="300" Width="400" x:Name="richtext">
  <syncfusion:DocumentAdv>
    <syncfusion:SectionAdv>
      <syncfusion:ParagraphAdv>
        <syncfusion:SpanAdv Text="This is Span text"/>
      </syncfusion:ParagraphAdv>
    </syncfusion:SectionAdv>
  </syncfusion:DocumentAdv>
</syncfusion:RichTextBoxAdv>
```

C#

```
RichTextBoxAdv richtext = new RichTextBoxAdv();
DocumentAdv document = new DocumentAdv();
SectionAdv section = new SectionAdv();
ParagraphAdv paragraph = new ParagraphAdv();
SpanAdv span = new SpanAdv();
span.Text = "This is span text";
paragraph.Inlines.Add(span);
section.Blocks.Add(paragraph);
document.Sections.Add(section);
richtext.Document = document;
```

Text Formatting Using HyperlinkAdv in WPF RichTextBoxAdv (Classic)

HyperlinkAdv can be kept only inside the ParagraphAdv since the content property of ParagraphAdv is Inline. It allows you to display the formatted text using advanced features like `NavigationUrl` and `TargetType`. It simply differs by these properties when compared to SpanAdv.

Properties

Property	Description	Type	Data Type
Text	Contains the text information.	Dependency Property	String
Baseline	Indicates whether the inline text is subscript or superscript.	Dependency Property	Baseline
NavigateURL	Holds the URL to which the page navigates to.	Dependency Property	String
TargetType	Indicates whether to open the link in the same window or in a new window.	Dependency Property	HyperlinkTargetType
FontFamily	Specifies the font family of the text.	Dependency Property	FontFamily
FontSize	Specifies the size of the font.	Dependency Property	Double
Foreground	Designates the font color.	Dependency Property	Color
HighlightColor	Designates the highlight color of the text.	Dependency Property	Color

Strikethrough	Shows whether the text has SingleStrikeThrough or DoubleStrikeThrough.	Dependency Property	StrikeThrough
FontWeight	Indicates the font weight of the text.	Dependency Property	FontWeight
Underline	Indicates whether the text should be underlined.	Dependency Property	Underline

Add HyperlinkAdv to an Application

HyperlinkAdv can be added directly to an application by using the following code snippet:

XML

```
<syncfusion:RichTextBoxAdv Height="300" Width="400" x:Name="richtext">
  <syncfusion:DocumentAdv>
    <syncfusion:SectionAdv>
      <syncfusion:ParagraphAdv>
        <syncfusion:HyperlinkAdv Text="This is Hyperlink text"/>
      </syncfusion:ParagraphAdv>
    </syncfusion:SectionAdv>
  </syncfusion:DocumentAdv>
</syncfusion:RichTextBoxAdv>
```

C#

```
RichTextBoxAdv richtext = new RichTextBoxAdv();
DocumentAdv document = new DocumentAdv();
SectionAdv section = new SectionAdv();
ParagraphAdv paragraph = new ParagraphAdv();
HyperlinkAdv hyperlink = new HyperlinkAdv();
hyperlink.Text = "This is Hyperlink text";
paragraph.Inlines.Add(hyperlink);
section.Blocks.Add(paragraph);
document.Sections.Add(section);
richtext.Document = document;
```

Inserting-Images in WPF RichTextBoxAdv (Classic)

In order to insert an image in the document, ParagraphAdv provides an inline called ImageContainerAdv.

Properties

Property	Description	Type	Data Type
Height	Specifies the height of the image.	Dependency Property	Double
Width	Specifies the width of the image.	Dependency Property	Double
ImageSource	Source of the image. It can be bytes or URL.	Dependency Property	ImageSource.
ImageBytes	Image source can be represented as byte array.	Dependency Property	Byte[]
FitToContent	Fits the image inside the layout when it exceeds the size limit.	Dependency Property	Boolean

Adding Images to an Application

ImageContainerAdv can be used to add an image directly to an application using the following code snippet:

XML

```
<syncfusion:RichTextBoxAdv Height="300" Width="400" x:Name="richtext">
  <syncfusion:DocumentAdv>
    <syncfusion:SectionAdv>
      <syncfusion:ParagraphAdv >
        <syncfusion:ImageContainerAdv
          ImageSource="/RibbonAndRichTextBox;component/OfficeUI/calender.png"
          Width="100" Height="100"/>
      </syncfusion:ParagraphAdv>
    </syncfusion:SectionAdv>
  </syncfusion:DocumentAdv>
</syncfusion:RichTextBoxAdv>
```

C#

```
RichTextBoxAdv richtext = new RichTextBoxAdv();
DocumentAdv document = new DocumentAdv();
SectionAdv section = new SectionAdv();
ParagraphAdv paragraph = new ParagraphAdv();
ImageContainerAdv imagecontainer = new ImageContainerAdv();
BitmapImage bitmap = new BitmapImage();
bitmap.UriSource = new
Uri("/RibbonAndRichTextBox;component/OfficeUI/calender.png");
imagecontainer.ImageSource = bitmap;
paragraph.Inlines.Add(imagecontainer);
section.Blocks.Add(paragraph);
document.Sections.Add(section);
richtext.Document = document;
```

Limitations

Inserting an image has the following limitations:

1. It does not have a separate context menu.
2. It does not deal with brightness, reflection, or glow effects of the image.

Inserting-a-UI-Element in WPF RichTextBoxAdv (Classic)

In order to insert a UI element in the document, ParagraphAdv provides an inline called UIContainerAdv.

Properties

Property	Description	Type	Data Type
Height	Specifies the height of the image.	Dependency Property	Double
Width	Specifies the width of the image.	Dependency Property	Double
UIElement	Specifies the UI element to be added.	Dependency Property	UIElement

FitToContent	Fits the UIElement based on the size of a page.	Dependency Property	Boolean
--------------	---	---------------------	---------

Adding UIElement to an Application

UIContainerAdv can be used to add a UI element directly to an application by using the following code snippet:

XML

```
<syncfusion:RichTextBoxAdv Height="300" Width="400" x:Name="richtext">
  <syncfusion:DocumentAdv>
    <syncfusion:SectionAdv>
      <syncfusion:ParagraphAdv >
        <syncfusion:UIContainerAdv >
          <Button />
        </syncfusion:UIContainerAdv>
      </syncfusion:ParagraphAdv>
    </syncfusion:SectionAdv>
  </syncfusion:DocumentAdv>
</syncfusion:RichTextBoxAdv>
```

C#

```
RichTextBoxAdv richtext = new RichTextBoxAdv();
DocumentAdv document = new DocumentAdv();
SectionAdv section = new SectionAdv();
ParagraphAdv paragraph = new ParagraphAdv();
UIContainerAdv uicontainer = new UIContainerAdv();
uicontainer.UIElement = new Button();
paragraph.Inlines.Add(uicontainer);
section.Blocks.Add(paragraph);
document.Sections.Add(section);
richtext.Document = document;
```

Limitations

Cut, copy, and paste operations are not permitted for the UIElement.

Insert-Tables in WPF RichTextBoxAdv (Classic)

Table support for the RichTextBoxAdv control has been implemented as in Microsoft Word. This is used to insert tables with user-defined rows and columns, and it also allows the user to insert multiple tables for every cell.

XML

```
<syncfusion:TableAdv>
  <syncfusion:TableRowAdv>
    <syncfusion:TableCellAdv>
      <syncfusion:ParagraphAdv>
        <syncfusion:SpanAdv Text="Table support"/>
      </syncfusion:ParagraphAdv>
    </syncfusion:TableCellAdv>
  </syncfusion:TableRowAdv>
</syncfusion:TableAdv>
```

C#

```
TableAdv table = new TableAdv();
TableRowAdv row = new TableRowAdv();
TableCellAdv cell = new TableCellAdv();
ParagraphAdv paragraph = new ParagraphAdv();
SpanAdv span = new SpanAdv();
span.Text = "Table support";
paragraph.Inlines.Add(span);
cell.Blocks.Add(paragraph);
row.Cells.Add(cell);
table.Rows.Add(row);
```

Editing Support

Editing is done in the table cell. Table width and height will be calculated for every change in the block of cells. We can insert an image, hyperlink, and UIElement in the table cells.

Selection Support

The table, table cells, table rows, and table columns have separate selection behavior to select the corresponding element in the blocks. Table cell selection can be achieved by double-clicking when inside the table cells. Table row and table column selection can be achieved by the SelectRow, SelectCell, and SelectTable commands.

Insert n*n Tables

We can insert n*n tables inside a single table cell. It can be achieved by InsertTable.

Insert n*n Columns

We can insert n* n columns in a table. It can be achieved by the InsertColumn command. This is possible only when a number of cells or the current position inside the table is selected.

Insert n*n Rows

We can insert n* n rows in a table. It can be achieved by the InsertRow command. This is possible only when a number of cells or the current position inside the table is selected.

Delete Table

We can delete the table from the document. It can be achieved by the DeleteTable command. This is possible only when a number cells or the current position inside the table is selected.

Delete n*n Rows

We can delete n*n rows at a single attempt. It can be achieved by the DeleteRow command. This is possible only when a number of cells or the current position inside the table is selected.

Delete n*n Columns

We can delete n*n columns at a single attempt. It can be achieved by the DeleteColumn command. This is possible only when a number of cells or the current position inside the table is selected.

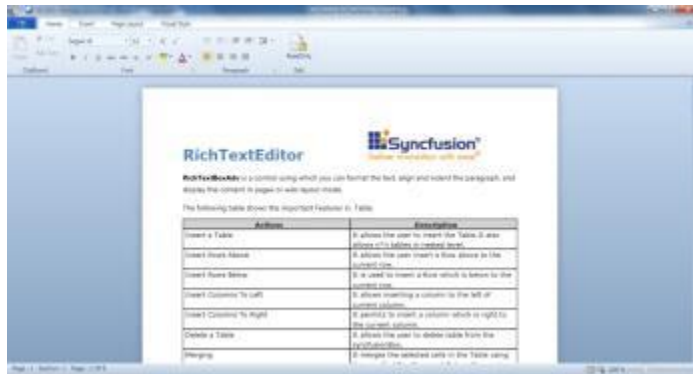
Merging

The user can merge the n*n cells into a single cell. It can be achieved by the MergeSelectedCells command. This is possible only when a number of cells are selected. The RowSpan and ColumnSpan properties in the table cell take responsibility for this.

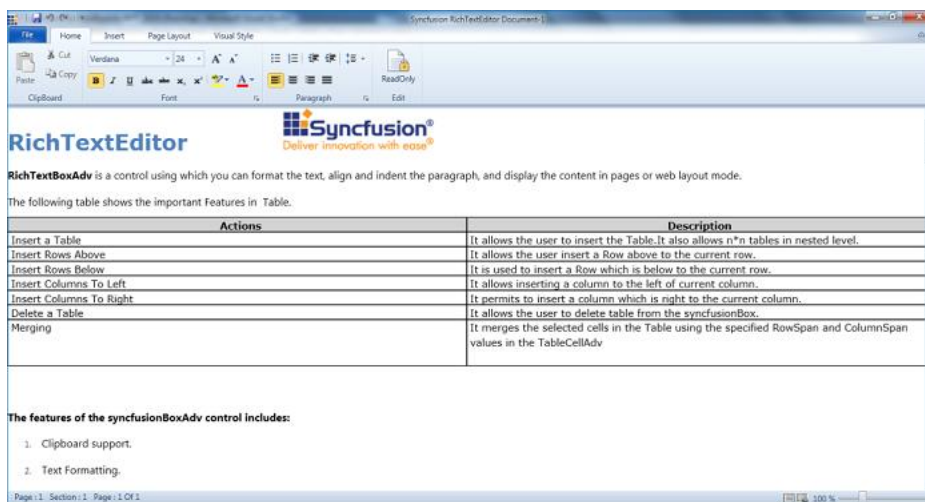
Layout-Modes in WPF RichTextBoxAdv (Classic)

The layout mode feature allows choosing several layouts in the document. They are:

1. **PageLayout**—A single document in the edited content can be separated into n number of pages depending upon the capacity of the content. We can customize the page margin and size using the properties which are available in SectionAdv.



2. **ContinuousLayout**—This layout mode looks like the Web Layout mode in Microsoft Word. Instead of having pages of content, it will have a single page that displays all of the content.



Import and Export in WPF RichTextBoxAdv

This feature lets you use several file formats, namely .doc, .docx, .html, .xaml, and .txt, inside the RichTextBoxAdv control. Using any one format allows the use of the other formats.

HTML Import/Export

The HTML import feature allows the user to import an .html file into the RichTextBoxAdv. It renders the HTML tags like a browser and displays the text in the format of RichTextBoxAdv's content model. The HTML export feature actually exposes the RichTextBoxAdv's document as an .html file. The following methods clearly show this use case.

C#

```
DocumentAdv document = HTMLImporting.ConvertToDocumentAdv(stream);
DocumentAdv document = HTMLImporting.ConvertToDocumentAdv(htmlstring);
string html = HTMLExporting.ConvertToHtml(RichTextBox.Document, stream);
```

Methods

Method	Description	Parameters	Type	Return Type
ConvertToDocumentAdv()	Converts HTML stream into DocumentAdv.	(Stream stream)(string htmlstring)	NA	DocumentAdv
ConvertToHtml()	Converts the DocumentAdv to an HTML string.	NA	NA	String

Limitations

The HTML import/export feature has the following limitations:

1. Script support has not been provided.
2. It does not provide support for tables.

DOC Import/Export and DOCX Import/Export

This feature allows you to open and save the Word DOC and DOCX format (.doc, .docx) documents, and to view, edit, and print the contents by using the RichTextBoxAdv control.

The following assembly references are required to use this feature and its namespace.

WPF 3.5 and 4.0 Framework:

- Assembly: Syncfusion.RichTextBoxAdv.Wpf
- Dependent Assemblies: Syncfusion.Compression.Base, Syncfusion.Core, Syncfusion.DocIO.ClientProfile, Syncfusion.RichTextBoxAdv.Wpf, and Syncfusion.Shared.Wpf
- Namespace: Syncfusion.Windows.Tools.Controls

WPF 4.5 and 4.5.1 Framework:

- Assembly: Syncfusion.RichTextBoxAdv.Wpf
- Dependent Assemblies: Syncfusion.Compression.Base, Syncfusion.Core, Syncfusion.DocIO.Base, Syncfusion.RichTextBoxAdv.Wpf, and Syncfusion.Shared.Wpf
- Namespace: Syncfusion.Windows.Tools.Controls

Silverlight platform:

- Assembly: Syncfusion.RichTextBoxAdv.Silverlight
- Dependent Assemblies: Syncfusion.Compression.Silverlight, Syncfusion.DocIO.Silverlight, Syncfusion.RichTextBoxAdv.Silverlight, and Syncfusion.Shared.Silverlight
- Namespace: Syncfusion.Windows.Tools.Controls

DOC and DOCX Import

You can convert the Word document stream to RichTextBoxAdv document by invoking the ConvertToDocumentAdv method from the extension class DocxImporting.

Method	Return Type	Parameters	Description
--------	-------------	------------	-------------

ConvertToDocumentAdv	DocumentAdv	1. Stream documentStream2. String documentExtension (.doc, .docx)	Converts the Word document stream to DocumentAdv instance.
----------------------	-------------	---	--

C#

```
//Initializes the new RichTextBoxAdv control.
RichTextBoxAdv richTextBoxAdv = new RichTextBoxAdv();
//Loads the DOC format document in RichTextBoxAdv control from document
stream.
richTextBoxAdv.Document = DocxImporting.ConvertToDocumentAdv(documentStream,
".doc");
//Loads the DOCX format document in RichTextBoxAdv control from document
stream.
richTextBoxAdv.Document = DocxImporting.ConvertToDocumentAdv(documentStream,
".docx");
```

VB.NET

```
'Initializes the new RichTextBoxAdv control.
Dim richTextBoxAdv As New RichTextBoxAdv()
'Loads the DOC format document in RichTextBoxAdv control from document
stream.
richTextBoxAdv.Document = DocxImporting.ConvertToDocumentAdv(documentStream,
".doc")
'Loads the DOCX format document in RichTextBoxAdv control from document
stream.
richTextBoxAdv.Document = DocxImporting.ConvertToDocumentAdv(documentStream,
".docx")
```

DOC and DOCX Export

You can convert the RichTextBoxAdv document to Word document stream by invoking the ConvertToDocument method from the extension class DocxExporting.

Method	Return Type	Parameters	Description
ConvertToDocument	void	1. DocumentAdv documentadv 2. Stream documentStream 3. String documentExtension (.doc, .docx)	Converts the DocumentAdv instance to Word document stream.

C#

```
//Saves the RichTextBoxAdv document to Word DOC format document stream.
DocxExporting.ConvertToDocument(richTextBoxAdv.Document, documentStream,
".doc");
//Saves the RichTextBoxAdv document to Word DOCX format document stream.
DocxExporting.ConvertToDocument(richTextBoxAdv.Document, documentStream,
".docx");
```

VB.NET

```
'Saves the RichTextBoxAdv document to Word DOC format document stream.
DocxExporting.ConvertToDocument (richTextBoxAdv.Document, documentStream,
".doc")
'Saves the RichTextBoxAdv document to Word DOCX format document stream.
DocxExporting.ConvertToDocument (richTextBoxAdv.Document, documentStream,
".docx")
```

Limitations

The .doc import/export and .docx import/export features have the following limitations:

1. Hyperlinks do not work for tables of contents.

XAML Import/Export

The XAML import feature allows users to import a .xaml file into the RichTextBoxAdv. It renders the XAML elements as XamlReader and displays the text in the format of RichTextBoxAdv's content model. The XAML export feature actually exposes the RichTextBoxAdv's document as a .xaml file. The following methods clearly show this use case.

C#

```
RichTextBox.Document=XAMLImporting.ConvertToDocumentAdv (xamlStream)
string xaml = XAMLExporting.ConvertToXAML (RichTextBox.Document, xamlstream);
```

Methods

Method	Description	Parameters	Type	Return Type
ConvertToDocumentAdv()	It converts XAML stream into DocumentAdv.	NA	NA	DocumentAdv
ConvertToXaml()	It returns the RichText content as XAML.	NA	NA	Void

Text Import/Export

The text import feature allows the user to import a .txt file into the RichTextBoxAdv. It renders the text in Notepad format and displays the text in the format of RichTextBoxAdv's content model. The text export feature actually exposes the RichTextBoxAdv's document as a .txt file. The following methods clearly show this usage.

C#

```
RichTextBox.Document = TextImporting.ConvertToDocumentAdv (textstream);
```

C#

```
string txtstring= TextExporting.ConvertToText (RichTextBox.Document,
textStream);
```

Methods

Method	Description	Parameters	Type	Return Type
--------	-------------	------------	------	-------------

ConvertToDocumentAdv()	Converts text stream into DocumentAdv.	NA	NA	DocumentAdv
ConvertText()	Returns the rich-text content as a text file.	NA	NA	String

Clipboard-Support in WPF RichTextBoxAdv (Classic)

Clipboard support in the RichTextBoxAdv control allows you to cut, copy, and paste the selected text inside the RichTextBoxAdv. These can be executed by the built-in commands.

1. Cut—Performs the cut operation over the selected text.
2. Copy—Copies the selected text from the RichTextBoxAdv.
3. Paste—Pastes the copied text into RichTextBoxAdv.

Methods

Method	Description	Parameters	Type	Return Type
Cut()	Slices the selected text.	NA	NA	Void
Copy()	Copies the selected text.	NA	NA	Void
Paste()	Pastes the copied text.	NA	NA	Void

Commands-Support in WPF RichTextBoxAdv (Classic)

The RichTextBoxAdv control provides built-in static commands that showcase the features available in RichTextBoxAdv.

Properties

Property	Description	Type	Data Type
AfterSpacing	Command to change the spacing after.	Command	RoutedUICommand
BeforeSpacing	Command to change the spacing before.	Command	RoutedUICommand
ToggleBold	Command which will toggle bold.	Command	RoutedUICommand
ChangeFontFamily	Command to change font family of the selected text.	Command	RoutedUICommand
ChangeFontSize	Command to change font size of the selected text.	Command	RoutedUICommand
HighlightColor	Command to change highlight color of the selected text.	Command	RoutedUICommand
TextColor	Command to change font color.	Command	RoutedUICommand
Copy	Command to copy the selected text.	Command	RoutedUICommand

Cut	Command to cut the selected text.	Command	RoutedUICommand
ToggleStrikeThrough	Command to change the strikethrough of the text.	Command	RoutedUICommand
FontDialog	Command to show the built-in Font dialog box.	Command	RoutedUICommand
ToggleItalic	Command which will toggle italics.	Command	RoutedUICommand
HyperlinkDialog	Command to show the built-in Hyperlink dialog box.	Command	RoutedUICommand
LeftIndent	Command to change the left indent of the paragraph.	Command	RoutedUICommand
RightIndent	Command to change the right indent of the paragraph.	Command	RoutedUICommand
ParagraphDialog	Command to show the built-in Paragraph dialog box.	Command	RoutedUICommand
Open	Command to show the Open dialog and open a document.	Command	RoutedUICommand
Save	Command to open the Save dialog and save the document.	Command	RoutedUICommand
Paste	Command to paste the copied text.	Command	RoutedUICommand
ToggleSubScript	Command which will toggle subscript.	Command	RoutedUICommand
ToggleSuperScript	Command which will toggle superscript.	Command	RoutedUICommand
TextAlignment	Command to change the text alignment of the paragraph.	Command	RoutedUICommand
ToggleUnderline	Command to change the underline.	Command	RoutedUICommand
New	Command to create a new document.	Command	RoutedUICommand
Print	Command to show the built-in Print dialog box.	Command	RoutedUICommand
ListType	Command to apply bullets and numbering to lists.	Command	RoutedUICommand
Undo	Command to undo the previous operation.	Command	RoutedUICommand
Redo	Command to redo the previous operation.	Command	RoutedUICommand
ChangeLayout	Command to change the layout mode.	Command	RoutedUICommand

ToggleDoubleStrikeThrough	Command to apply a double strikethrough.	Command	RoutedUICommand
LineSpacing	Command to change the line spacing.	Command	RoutedUICommand
InsertTable	Command to insert a table.	Command	RoutedUICommand
InsertRow	Command to insert a row.	Command	RoutedUICommand
InsertColumn	Command to insert a column.	Command	RoutedUICommand.
DeleteRow	Command to delete the row.	Command	RoutedUICommand.
DeleteColumn	Command to delete the column.	Command	RoutedUICommand.
DeleteTable	Command to delete the table.	Command	RoutedUICommand.
InsertTableDialog	Command to open the built-in Table dialog.	Command	RoutedUICommand.
SelectCell	Command to select a cell.	Command	RoutedUICommand.
SelectRow	Command to select a row.	Command	RoutedUICommand
SelectColumn	Command to select a column.	Command	RoutedUICommand.
SelectTable	Command to select a table.	Command	RoutedUICommand.

Code-Snippet in WPF RichTextBoxAdv (Classic)

XML

```

<Window x:Class="Testing.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525">
  <Grid>
    <DockPanel>
      <Menu DockPanel.Dock="Top">
        <MenuItem Command="{x:Static syncfusion:RichTextBoxAdv.ToggleBold}"
Header="Bold"/>
      </Menu>
      <syncfusion:RichTextBoxAdv x:Name="richtextbox">
        <syncfusion:DocumentAdv>
          <syncfusion:SectionAdv>
            <syncfusion:ParagraphAdv>
              <syncfusion:SpanAdv Text="I am the Text...."/>
            </syncfusion:ParagraphAdv>
          </syncfusion:SectionAdv>
        </syncfusion:DocumentAdv>
      </syncfusion:RichTextBoxAdv>
    </DockPanel>
  </Grid>
</Window>

```

RichTextBoxAdv Methods

Methods Table

Method	Description	Parameters	Type	Return Type
Copy	Called to copy the selected text.	NA	NA	Void
Paste	Called to paste the text from the clipboard.	NA	NA	Void
Cut	Called to cut the selected text.	NA	NA	Void
ChangeTextAlignment	Changes the specified text alignment.	(object commandparameter)	NA	Void
Bold	Toggles bold for the selected text.	NA	NA	Void
Italic	Toggles italics for the selected text.	NA	NA	Void
ChangeLeftIndent	Changes the left indent of the current paragraph.	(object commandparameter)	NA	Void
ChangeRightIndent	Changes the right indent of the current paragraph.	(object commandparameter)	NA	Void
IncreaseIndent	Increases the indent.	NA	NA	Void
DecreaseIndent	Decreases the indent.	NA	NA	Void
ChangeLineSpacing	Changes the line spacing.	(object commandparameter)	NA	Void
ChangeFontFamily	Changes the font family.	(object commandparameter)	NA	Void
ChangeAfterSpacing	Changes the AfterSpacing of the paragraph to the specified value.	NA	NA	Void
ChangeBeforeSpacing	Changes the BeforeSpacing of the paragraph to the specified value.	NA	NA	Void
ChangeFontSize	Changes the font size of the selected text.	(object commandparameter)	NA	Void
ChangeSingleStrikeThrough	Toggles single strikethrough.	NA	NA	Void

ChangeDoubleStrikeThrough	Toggles double strikethrough.	NA	NA	Void
ChangeHighlightColor	Changes the highlight color to the specified color.	NA	NA	Void
ChangeUnderline	Toggles underline for the selected text.	NA	NA	Void
ChangeSuperscript	Toggles the superscript.	NA	NA	Void
ChangeSubscript	Toggles the subscript.	NA	NA	Void
Undo	Undo the operation.	NA	NA	Void
Redo	Redo the operation.	NA	NA	Void
ChangePageLayout()	Changes the layout of the document.	(object commandparameter)	NA	Void
InsertInlineInParagraph	Inserts the inline.	(Inline inline)	NA	Void
InsertTableInBlocks	Inserts the table.	(int rowcount,int columncount)	NA	Void
InsertRowInTable	Inserts a row in a table.	(RowPlacement rowplacement)	NA	Void
InsertColumnInTable	Inserts a column in a table.	(Columnplacement placement)	NA	Void
DeleteTableFromBlocks	Deletes the table.	NA	NA	Void
DeleteRowFromTable	Deletes the row from the table.	NA	NA	Void
DeleteColumnFromTable	Deletes the column from the table.	NA	NA	Void
SelectCellInTable	Selects the cell.	NA	NA	Void
SelectRowInTable	Selects the row in a table.	NA	NA	Void
SelectColumnInTable	Selects the column in a table.	NA	NA	Void
SelectTableInBlocks	Selects the table.	NA	NA	Void
MergeSelectedCellsInTable	Merges the selected cells in the table.	NA	NA	Void

Keyboard-Support in WPF RichTextBoxAdv (Classic)

RichTextBoxAdv provides some shortcut keys for ease of use. The following table gives the list of supported shortcuts.

Keyboard shortcuts table

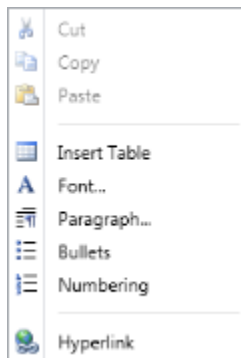
Hot keys	Description
CTRL + A	Selects the entire document
CTRL + C	Copy the selected text
CTRL + Down	Moves the cursor to the start of the next paragraph
CTRL + End	Moves the cursor to the end of the document
CTRL + Home	Moves the cursor to the start of the document
CTRL + Left	Moves the cursor to the start of the previous word
CTRL + Right	Moves the cursor to the start of the next word
CTRL + Shift + Left	Selects word by word in left direction
CTRL + Shift + Right	Selects word by word in right direction
CTRL + Up	Moves the cursor to the start of the previous paragraph
CTRL + V	Paste the text
CTRL + X	Cut the selected text
CTRL + Y	Redo the action
CTRL + Z	Undo the action
CTRL + B	Bold the selection
CTRL + I	Italic the selection
CTRL + U	Underline the selection
CTRL + L	Change the TextAlignment to left
CTRL + E	Change the TextAlignment to center
CTRL + R	Change the TextAlignment to right
CTRL + S	Save the document
CTRL + P	Print the document
End	Moves the cursor to the end of the line
Home	Moves the cursor to the beginning of the line
Shift + Down	Selects line by line in the down direction
Shift + Insert	Pastes the text
Shift + Left	Selects letter by letter in the left direction
Shift + Right	Selects letter by letter in the right direction

Shift + Up	Selects line by line in the up direction
------------	--

Context-Menu-Support in WPF RichTextBoxAdv (Classic)

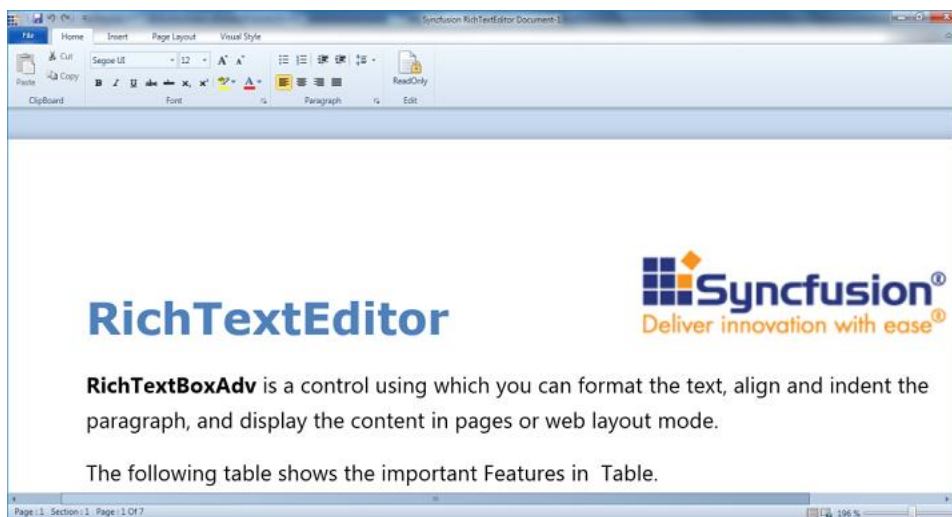
RichTextBoxAdv supports a context menu similar to Microsoft Word. It supports the following list of features in its context menu:

1. Cut
2. Copy
3. Paste
4. Insert Table
5. Font
6. Paragraph
7. Bullets
8. Numbering
9. Hyperlink



Zooming-and-Printing in WPF RichTextBoxAdv (Classic)

RichTextBoxAdv allows you to zoom in and out on the content of the document. CTRL + mouse wheel will zoom in and zoom out the content of the RichTextBoxAdv control.



The printing feature permits users to print the content of the document using `PrintDocument()`.

We can also use the Print command which will execute the PrintDocument method whenever it is hooked to the command property.

C#

```
RichTextBox.PrintDocument();
```

Properties

Property	Description	Type	Data Type
IsZoomEnabled	Decides whether the content of the RichTextBoxAdv can be zoomed or not.	Dependency Property	Boolean
ZoomFactor	Factor to show the zoomed value. It ranges from 0.1 to 1.	Dependency Property	Double

Methods

Method	Description	Parameters	Type	Return Type
ResetZooming()	Resets the zoom.	NA	NA	Void
PrintDocument()	Prints the content of the RichTextBoxAdv.	NA	NA	Void

Disable-Editing in WPF RichTextBoxAdv (Classic)

You can disable editing in RichTextBoxAdv by enabling the property called IsReadOnly. It will not allow the user to edit the content.

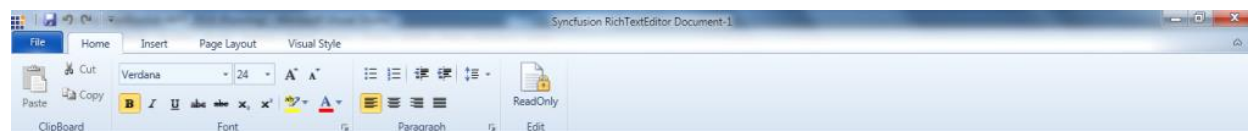
Properties

Property	Description	Type	Data Type
IsReadOnly	Enables/disables the editing.	Dependency Property	Boolean

RichTextRibbon-Support in WPF RichTextBoxAdv (Classic)

RichTextBoxAdv has RichTextRibbon support to format the text, paragraph, and so on. It provides a Microsoft Word look and feel.

The Syncfusion.RichTextRibbon.WPF assembly is required to add this RichTextRibbon in the application. RichTextRibbon has been inherited from the Ribbon control. When this control is dragged and dropped over the designer, items will be automatically serialized. Setting the DataContext to RichTextBoxAdv will automatically bind the commands.



Touch-Events in WPF RichTextBoxAdv (Classic)

The RichTextBoxAdv control provides support for touch manipulation like panning, zooming, and selecting the content of the document.

Panning: With the touch manipulation event, we can interpret the multi-touch input to simulate directly manipulating the viewer and allow users to use their finger to scroll over the contents of the RichTextBoxAdv control.

Zooming: With the touch manipulation event, we can interpret multi-touch input to simulate directly manipulating the viewer and allow users to use their fingers to zoom in and out the contents of the RichTextBoxAdv control.

Selection: With the touch manipulation event, we can interpret the multi-touch input to simulate directly manipulating the contents rendered on the viewer and allow users to use their fingers to perform select content similar to using a mouse.

Note: By default, the `IsManipulationEnabled` property of the RichTextBoxAdv control is set to false and touch manipulations are disabled. In order to enable touch manipulation in the RichTextBoxAdv control, set the `IsManipulationEnabled` property to true.

Properties

Property	Description	Type	Data Type
IsManipulationEnabled	Gets or sets a value that indicates whether manipulation events are enabled in the RichTextBoxAdv.	Dependency Property	Boolean

How to

Bind the content of the WPF RichTextBoxAdv (Classic) By Using MVVM

You can bind the content of a RichTextBoxAdv control in XAML (designer) by defining a MVVM extension for the XAMLText or HTMLText property. You cannot update the XAMLText or HTMLText property dynamically by modifying content in the RichTextBoxAdv, but you can update it by accessing the XAMLText or HTMLText property (In Getter method).

You can explicitly update the XAMLText or HTMLText property to view model for content binding by using the LayoutUpdated event that is triggered when content is modified in the RichTextBoxAdv control.

The following code example illustrates defining MVVM extension for the XAMLText property of the RichTextBoxAdv control.

XML

```
<syncfusion:RichTextBoxAdv x:Name="richTextBoxAdv" XAMLText="{Binding
DescriptionXaml}" />
```

C#

```
/// <summary>
/// Handles the LayoutUpdated event of the richTextBoxAdv control.
/// </summary>
/// <param name="sender">The source of the event.</param>
/// <param name="e">The <see cref="EventArgs"/> instance containing the
event data.</param>
void richTextBoxAdv_LayoutUpdated(object sender, EventArgs e)
{
    if ((this.DataContext as ViewModel).XamlContent != richTextBoxAdv.XAMLText)
```

```

//Updates the XAMLText for changes made in RichTextBoxAdv to ViewModel.
(this.DataContext as ViewModel).XamlContent = richTextBoxAdv.XAMLText;
}
/// <summary>
/// Sample View Model class
/// </summary>
public class ViewModel : INotifyPropertyChanged
{
    private String descriptionXaml;
    /// <summary>
    /// Initializes a new instance of the <see cref="ViewModel"/> class.
    /// </summary>
    public ViewModel()
    {
        // Sample description xaml.
        DescriptionXaml = "<RichText:DocumentAdv
xmlns=\"http://schemas.microsoft.com/winfx/2006/xaml/presentation\"
xmlns:RichText=\"clr-
namespace:Syncfusion.Windows.Tools.Controls;assembly=Syncfusion.RichTextBoxA
dv.Silverlight\" >\r\n<RichText:SectionAdv><RichText:ParagraphAdv
ListType=\"None\" TextAlignment=\"Left\" LeftIndent=\"0\" RightIndent=\"0\"
BeforeSpacing=\"0\" AfterSpacing=\"13\" >\r\n<RichText:SpanAdv
Text=\"RichTextBoxAdv is a control using which you can format the text,
align and indent the paragraph, and display the content in pages or web
layout mode.\" FontStyle=\"Normal\" Baseline=\"Normal\"
HighlightColor=\"#00000000\" Foreground=\"#FF000000\" FontSize=\"12\"
FontFamily=\"Segoe UI\" FontWeight=\"Normal\"
>\r\n</RichText:SpanAdv>\r\n</RichText:ParagraphAdv>\r\n</RichText:SectionAd
v></RichText:DocumentAdv>";
    }
    /// <summary>
    /// Gets or sets the description xaml.
    /// </summary>
    /// <value>
    /// The description xaml.
    /// </value>
    public String DescriptionXaml
    {
        get
        {
            return descriptionXaml;
        }
        set
        {
            descriptionXaml = value;
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs("DescriptionXaml"));
        }
    }
    /// <summary>
    /// Gets or sets the content of the xaml.
    /// </summary>
    /// <value>
    /// The content of the xaml.
    /// </value>
    internal String XamlContent
    {

```

```
get
{
    return descriptionXaml;
}
set
{
    descriptionXaml = value;
}
}
/// <summary>
/// Occurs when a property value changes.
/// </summary>
public event PropertyChangedEventHandler PropertyChanged;
}
```

Sample

To view a sample on binding the XAMLText property of RichTextBoxAdv control by using MVVM Framework, see [Sample](#).